Characterization of Performance, Robustness, and Behavior Relationships in a Directly Connected Material Handling System

Roger J. Anderson

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Mechanical Engineering

Robert Sturges, Chair
Michael Deisenroth
Douglas Nelson
Charles Reinholtz
John Shewchuk

February 13, 2006
Blacksburg, Virginia

Keywords: Complexity, Chaos, Optimization, Algorithmic Complexity, Genetic Algorithm

Characterization of Performance, Robustness, and Behavior Relationships in a Directly Connected Material Handling System

Roger J. Anderson

(ABSTRACT)

In the design of material handling systems with complex and unpredictable dynamics, conventional search and optimization approaches that are based only on performance measures offer little guarantee of robustness. Using evidence from research into complex systems, the use of behavior-based optimization is proposed, which takes advantage of observed relationships between complexity and optimality with respect to both performance and robustness. Based on theoretical complexity measures, particularly algorithmic complexity, several simple complexity measures are created. The relationships between these measures and both performance and robustness are examined, using a model of a directly connected material handling system as a backdrop. The fundamental causes of the relationships and their applicability in the proposed behavior-based optimization approach are discussed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Complex systems are characterized by non-linear interactions and interdependencies among system elements. In these types of systems, the global system behavior may not be predictable from the set of rules dictating local interactions among elements, although these interactions may be simple in nature, making the collective behavior 'greater than the sum of the parts'.

These characteristics of complex systems present difficulties with respect to optimization using traditional techniques. These techniques utilize some measure or measures of performance through the identification and manipulation of relevant system variables, largely ignoring the behavior of systems during operation. When behavior is taken into account, only the systems exhibiting the simplest forms of behavior are typically considered, as they represent the majority of systems expressible in a closed form. But for complex systems, optimality may arguably be defined not only with respect to performance, but adaptability, especially in dynamic environments.

The very same attributes that make a system complex, or result in complex behavior, and present difficulties for conventional techniques are theoretically applicable to finding optimal solutions - optimal with respect to performance and with respect to adaptability or robustness. Evidence suggests associations exist among behavior, performance, and robustness. Most notably, when a system has a certain combination of order and randomness, it is in a state that offers good performance and, perhaps more importantly, the ability to adapt to changes or variability in its environment.

The objective of this work is to identify the existence and then examine the useful relationships between behavior (in terms of complexity) and performance and between behavior and robustness as a first step towards utilizing behavior as an optimization tool. A significant hurdle implicit in this task is the definition of quantitative applied theoretical complexity measures of system behavior or architecture. The relationships are examined in the context of a naval weapons elevator system, a system characterized by near deterministic cycle times,

fixed operational logic sets, and deterministic input streams. Although important attributes of this particular system that are applicable towards its optimization are revealed in the course of analyses of the relationships between its behavior, performance, and robustness, the objective of this work should not be interpreted as an attempt to optimize this particular system. The naval weapons elevator system is utilized as a means to an end - an analysis of potential relationships in a system from which we can draw general conclusions, not the relationships in this particular system. Similarly, while exhaustive simulations for assumed control logics and a spectrum of input streams of the illustrative weapons elevator system are performed for complete characterization of the relationships between complexity measures, performance, and robustness, the use of exhaustive simulations is not intended to imply exhaustive methods are a requisite for behavior-based optimization methods.

1.1 Outline of Approach

The foundation for the idea that the merits of a system can be based on its behavior, not on measures of performance in particular situations alone, lies in evidence of relationships between both complexity and performance and between complexity and adaptability in complex systems. In many natural systems in particular, optimality with respect to both performance (survivability) and robustness (evolution) is often observed to correspond to a complex regime of behavior, with properties representing a mixture of simplicity and chaos. We will investigate these relationships in Chapter 2. But before we do, we get a qualitative feel for complexity and some of its attributes, using a class of simple and discrete, yet interesting, networks known as cellular automata.

While it is fairly straightforward to quantify performance and robustness, defining a universal quantitative measure of behavior or complexity has been more elusive. In Chapter 3, we identify some theoretical quantitative complexity measures, including algorithmic complexity. Because of their theoretical nature, it is not necessarily apparent how some measures are directly applicable in the quantification of complexity in real systems, specifically for material handling systems. Drawing mainly from the concepts of algorithmic complexity and component counting, we create system-specific measures, both dynamic and static, based on simulation attributes that are readily obtainable, for the purpose of identifying correlations between system behavior and performance and between system behavior and robustness in the context of an aircraft carrier naval weapons elevator system. These modified complexity measures are describable in terms of their foundations, accuracy with respect to theoretical measures, and their advantages and disadvantages.

Since our complexity measures are system-specific, we first have to describe the system in which they will be applied. Therefore, prior to the definition of system specific measures in the sixth chapter, we present the problems associated with the design and optimization of general vertical transportation systems and how they represent a complex environment in Chapter 4. In Chapter 5, we then describe the specific case of naval weapons elevator

systems, which has a large design space and is describable as a deterministic system. Here, we also present how a configuration is defined in terms of its physical relationships between system elements, the variety of input streams that are used in simulations, and the control logic used in system evolutions. We also introduce concepts used throughout this work, such as the definitions for valid configurations, encoding of configurations, and visualization techniques used to interpret evolutions.

Using the results of exhaustive simulations for a set of system sizes, we construct a picture of the relationships present between our complexity measures, performance, and adaptability in Chapters 7 through 12 for conventional naval weapons elevators. The use of exhaustive simulations is not intended to suggest that this approach is required when using behavior towards optimization, but only as a means for full characterization of the possible relationships. In Chapters 7 to 10, we look at dynamic measures of complexity, which rely on the explicit evolutions of configurations. We begin with small, simple systems, then expand to larger systems to create generalizations regarding the interrelationships between connectivity, input streams, and logic and their effects on performance and robustness. Throughout, we examine in detail how the modified measures compare to their theoretical counterparts and how they may be applied in an optimization technique.

The importance of connectivity is re-examined in particular in Chapter 11, in the discussion of static complexity measures. These measures represent a powerful approach toward optimization, since they are not dependent on explicit simulations and indicate the potential behavior of a system based on its defined attributes.

With a well-defined characterization of the relationships among complexity measures, performance, and robustness, based on variations with respect to physical connectivity and input streams, we turn our attention to the effect of changes in control logic on these relationships in Chapters 12 and 13. While maintaining a conventional format, logic specifying the operation of shafts is varied in Chapter 12. In Chapter 13, we explore a fundamentally different system, in which carriages are not dedicated to any particular uni-directional shaft, resulting in mobile carriage circuits.

Finally, we examine the conclusions regarding the relationships between complexity measures and performance and robustness, for both dynamic and static measures. In summarizing the results for conventional and mobile carriage systems, we see the importance of physical connectivity, input streams, and operational logic in complexity/performance and complexity/robustness relationships. Also offered are some thoughts regarding the potential of complexity-based optimization techniques and general applicability.

Chapter 2

Background

2.1 Cellular Automata

The advent of electronic computers both raised and enabled the solution of many new problems, including the possibility of artificial self-reproduction, or the ability of a machine to replicate itself. To address this question and avoid the costs associated with hardware, John von Neumann, often considered the father of the modern computer, created an abstract mathematical representation of a machine. Von Neumann demonstrated that the abstract representation, known as automata, was capable of self-replication if given the proper logic and initial conditions [60].

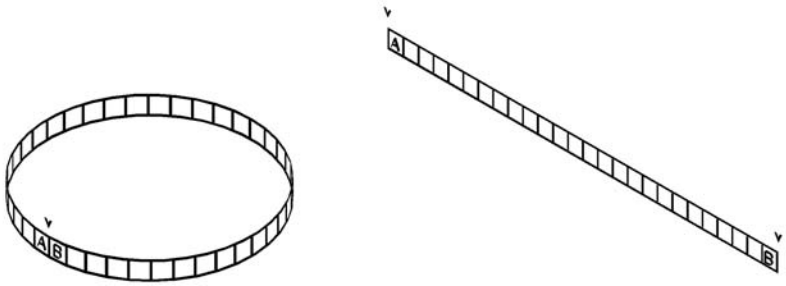
Cellular automata consist of a network of elements, or cells, that exist on some lattice structure. Cellular automata networks operate by the application of a set of transition functions, Θ, to each cell in the network. The network evolves as the transition functions are recursively applied.

To further explain cellular automata systems, we will use one-dimensional, two state, nearest single neighbor, closed networks with periodic boundary conditions, commonly referred to as elementary cellular automata. This class of cellular automata represents the simplest configuration capable of all forms of possible behavior [64, 63, 62].

2.2 Elementary Cellular Automata

One dimensional cellular automata networks consist of a line of cells, with each cell having a neighbor to the left and right. In a closed network with periodic boundary conditions, the cells therefore form a ring. When represented in two-dimensions, the ring must be "broken" and unfolded to capture all information about the state of the network. In this representa-

tion however, the left and right-most cells remain conceptually connected, as illustrated in Figure 2.1.



(a) (b)

Figure 2.1: A one dimensional closed network with periodic boundary conditions forms (a) a closed ring. Expressed in one dimension, the network is broken as in (b), but the ends A and B remain neighbors

As a one-dimensional netwok is evolved, we can visualize the current state of the system in two dimensions as in Figure 2.1, which is updated with each successive application of the transition functions on all cells. To visualize a history of an evolution, we simply add on a new row for each complete mapping. The result is a two-dimensional grid n columns wide and $t + 1$ rows long, where n is the number of cells involved in the network and t is the number of evolution steps. Evolution of the actual topography of the network shown in Figure 2.1 results in a hollow cylinder. Since the term evolution is used to describe the recursive mapping of the transition functions, we refer to each row in the evolution in integer values of time, so that the state of the system at time step t_1 evolves to the system state at time step t_2.

Cells in elementary cellular automata networks can exist in one of two states. In binary terms, these values are 0 and 1, equivalent to the Boolean values of $FALSE$ and $TRUE$. In our visual representations, the 0 and 1 states are equivalent to white and black cells, respectively.

2.2.1 Transition Functions

The transition functions associated with a cellular automata represent the conditional logic that control the operation of the system - the set of 'rules' that map a neighborhood of cells into a single state. While it is possible to have cellular automata networks with complete connectivity, where a cell evolves based on the states of all other cells in the network, elementary cellular automata have a neighborhood with a radius of one cell, $r = 1$. With this type of connectivity, a cell evolves to a state at time $t + 1$ based on its own state and the states of the nearest adjacent cells at time t.

In general, for automata with k possible states and a neighborhood of r adjacent cells to the left and right, there are $k^{(2r+1)}$ combinations of cell states that form the input to the transition functions. For elementary cellular automata with two possible states and a neighborhood of three cells ($r = 1$), the number of combinations of cell states is $2^3 = 8$, which are presented in Figure 2.2.

Figure 2.2: The eight combinations of 3 cells with 2 possible states which define elementary cellular automata transition function inputs

Each of these sequences can evolve to one of two states. In general, there are $k^{k^{2r+1}}$ possible sets of transition functions. For the case of elementary cellular automata, there are $2^{2^3} = 256$ possible rule sets, or combinations of evolved states. This evolution structure leads to a convenient method for expressing a rule set without explicit description of cell digit sequences [62].

Each combination of cell sequences is assigned a decimal value equivalent to the binary representation of cells, letting black and white equal 1 and 0, respectively. Each of these decimal equivalents represents a single bit in an eight bit byte, with the bit evolving from the sequence of 3 black cells representing the most significant bit and the bit evolving from the sequence of 3 white cells representing the least significant bit. The eight bit binary sequence of evolved cell states, converted to its decimal equivelent yields a single coded "rule number" that completely describes the evolution logic. An example of the coding process is illustrated in Figure 2.3 for elementary cellular automata rule 22.

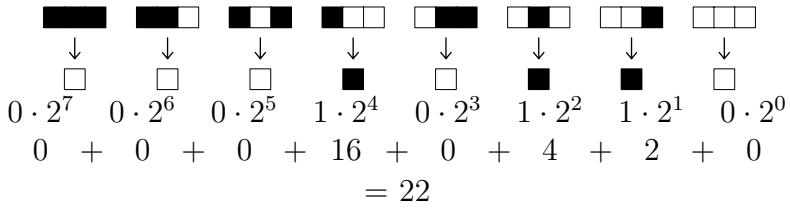


Figure 2.3: Rule numbers for cellular automata are based on the evolved binary value (white = 0 and black = 1) of a transition function and the corresponding bit value of the transition function, assuming transition functions are arranged in some standard order. For elementary rule 22, the 1^{st}, 2^{nd}, and 4^{th} transition functions bits evolve to a black, or 1, state, meaning the evolved "byte" is (00010110). The decimal representation of this binary value is 22.

2.2.2 Elementary Cellular Automata Behaviors

It might initially appear that elementary cellular automata are capable of producing only simple behaviors because of the apparent simplicity of the rule sets. Despite their simple

construction, elementary cellular automata are capable of producing all types of behavior qualitatively possible in any system. Furthermore, the initial conditions required to produce this range of behavior need not be complicated. A certain form of behavior is often inherent to a given rule set, regardless of the initial conditions.

Depending on the source, there range from four to six qualitative classes of behavior that have been identified in elementary cellular automata [64, 40, 38, 39]. In Wolfram's classification system, the four classes of behavior are as follows:

Class I: Trivial behavior. System states rapidly evolve to a uniform quiescent state.

Class II: Simple behavior. The system evolves to a non-uniform quiescent state or alternates periodically among a limited number of states. This is the equivalent of fixed point or limit cycle attractors.

Class III: Chaotic behavior. There are no attractors and the states are essentially random.

Class IV: Complex behavior. A mix of Class II and Class III, where similar patterns can be found throughout the system evolution, but not according to any predictable distribution.

To provide a better illustration of each type of behavior, we will look at examples of elementary cellular automata evolutions for each class of behavior. While some elementary cellular automata rule sets appear to behave in an inherent manner, inherency is by no means guaranteed [2]. It is therefore possible for a given rule set to behave differently based on the initial conditions of the system. For the examples shown, the behavior illustrated may not always be associated with the rule set described.

2.2.3 Class I: Trivial

An example of trivial behavior is found in rule 0. The rule set for rule 0, shown in Figure 2.4, maps all possible three cell neighborhoods to a white cell. As a result, rule 0 creates evolutions of all white cells in one evolution step, regardless of the initial conditions. Rule 0 is inherently trivial and, as a result, has very predictable evolutions. Rule 255, the 'negative' of rule 0 is another example of rule set that inherently yields trivial behavior.

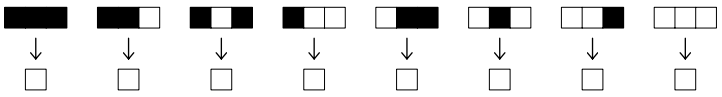


Figure 2.4: The rule set for elementary cellular automata rule 0. All possible combinations of three cells maps to a white cell.

An example of a rule set that is normally regarded as producing simple behavior, but is capable of trivial behavior, is rule 90. Rule 90, shown in Figure 2.5, is an additive rule. An additive rule has the property that the evolution is the spatial superposition of the independent evolutions of each initial cell. Because of this property, rule 90 is defined as simple despite the ability to produce apparently random evolutions. The additive property also leads to trivial behavior in rule 90. When the number of cells, N, in the network is equal to $2^n|(n = 1, 2, 3, \ldots)$, the evolution reaches a trivial state in exactly 2^{n-1} steps, regardless of the initial conditions. Example evolutions illustrating this behavior using various system sizes and initial conditions are shown in Figure 2.6.

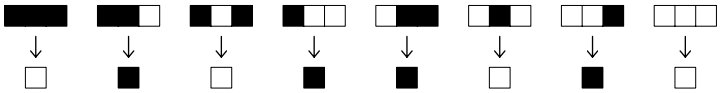
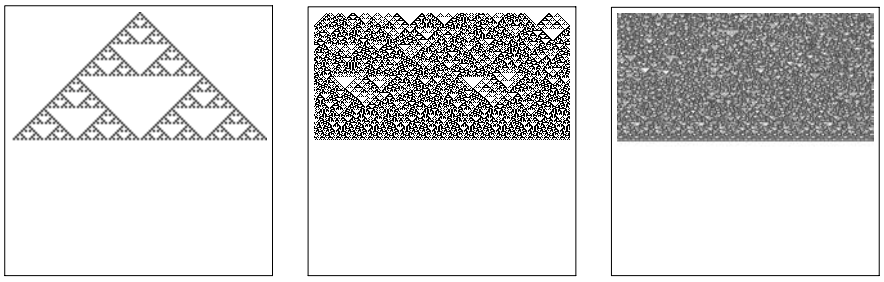


Figure 2.5: The rule set for elementary cellular automata rule 90, an additive rule.



(a) (b) (c)

Figure 2.6: Trivial behavior in a non-inherently trivial rule set. Rule 90, an additive rule, produces trivial behavior after 2^{n-1} evolution steps for a network with 2^n cells, regardless of the initial conditions. The evolution in (a) shows rule 90 starting from a single black cell and $N = 128$, (b) from random initial conditions with a 20% black cell density and $N = 256$, and (c) from random initial conditions with a 50% black cell density and $N = 512$

2.2.4 Class II: Simple

Many of the 256 elementary cellular automata rule sets produce only simple behavior, yielding a repetitive pattern of finite limit cycles or streaks resulting from fixed point attractors. Rule 4 provides an example of a fixed point attractor. Rule 4, shown in Figure 2.7, is known as a 'filter' rule, identifying the number of white-black-white cell sequences in the initial conditions. An example evolution of rule 4 is shown in Figure 2.8. After the first evolution step, only rule bits 0, 1, 2, 4, and 5 are applied. The relative ratio of their application is based on the number of streaks and the amount of white space in the evolution.

Since the number of streaks depends on the number of initial white-black-white cell sequences in the initial conditions, the evolution of the network is very predictable if the initial condi-

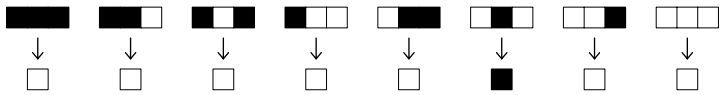


Figure 2.7: The rule set for elementary cellular automata rule 4. Rule 4 is a filtering rule, identifying the number of white-black-white sequences.

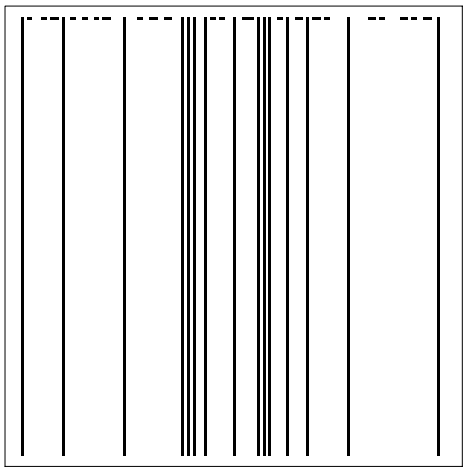


Figure 2.8: The first 150 evolution steps of rule 4 from random initial conditions. There are 150 cells in the network.

tions are known. While the behavior of the rule is consistent with all intial conditions, the actual states of the system in each evolution will vary with different intial conditions.

Figure 2.9 shows the transition functions for rule 94. Like rule 4, the evolution of rule 94 is dependent on the initial conditions. However, rule 94 typically has limit cycle attractors rather than fixed point attractors. In the example evolution shown in Figure 2.10, rule 94 has a limit cycle with a period of 6 evolution steps, the first common multiple of the period 3 sequence in the left-most band and the period 2 sequences on the right bands.

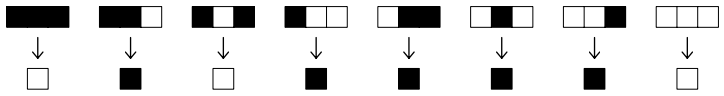


Figure 2.9: The rule set for elementary cellular automata rule 94. Rule 94 often has limit cycle attractors rather than fixed point attractors.

While the example evolution of rule 94 is more complicated than that for rule 4, the behavior in Figure 2.10 is still considered simple. Beyond the transient response, the network falls into a repetitive sequence of network states. Knowing the period and the states involved, it is straightforward to predict the state of any cell in the system at any evolution step.

It should be noted that, for a given set of initial conditions, the set of network states that

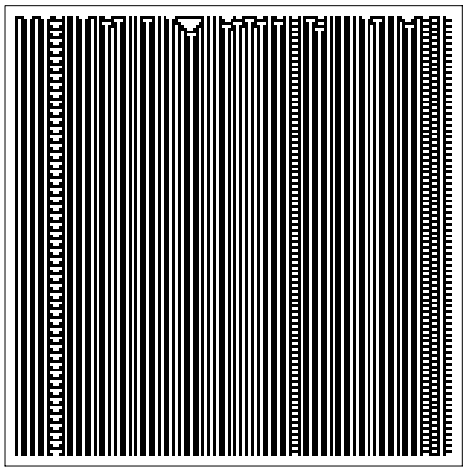


Figure 2.10: The first 150 evolution steps of rule 94 from random initial conditions. There are 150 cells in the network.

is occupied by a simple evolution is a tiny fraction of the total network state space. In the evolution of rule 94 in Figure 2.10, only 6 distinct sets of network states are used. However, for a network consisting of 150 cells, there are $2^{150} \approx 10^{45}$ possible states. This means that the system visits approximately $4 \cdot 10^{-43}$ percent of the available state space.

Nesting represents a special form of simple behavior. In a nested pattern, such as the one shown in Figure 2.11 created from rule 90, structures are self-similar on all scales proportional to a fractal dimension. Rule 90 has a fractal dimension of $log_2 3$ [62].

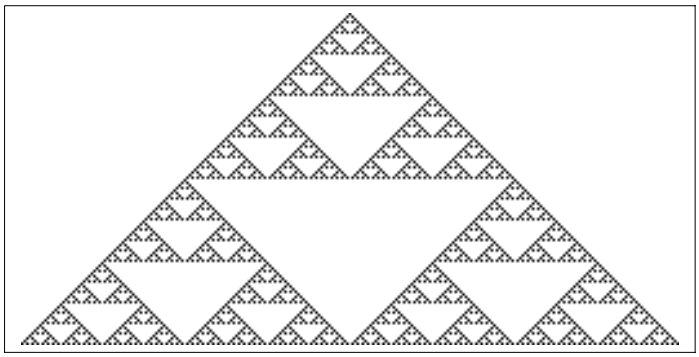


Figure 2.11: Nested behavior in elementary cellular automata rule 90, started from a single black cell.

Self-similar structures are considered simple because of the predictability of the evolution. Knowing the fractal dimension of the evolution, it is possible to determine the state of any cell within the evolution for those initial conditions that result in nested patterns.

2.2.5 Class III: Chaotic

Since cellular automata are deterministic systems, it is guaranteed that any cellular automata evolution is periodic. The behavior of a rule that creates chaotic behavior is therefore technically not random. However, the evolution may be required to evolve through 2^N distinct states for a network consisting of N cells. For even modest size N, this means that a rule set exhibiting random behavior locally can essentially be treated as globally random. The period of repetition has been shown to follow $2^{0.63N}$ in rule 30 and 2^N in rule 45. The assumption of global randomness is applied by Wolfram towards a random number generator in *Mathematica*, which uses chaotic rule 30 in a network of a few hundred cells [64].

The rule set for Rule 30 is shown in Figure 2.12. Despite the simplicity of the 8 rules defining the rule set, rule 30 inherently produces chaotic behavior for essentially all but a trivial set of initial conditions [2]. Example evolutions of rule 30 starting from a single black cell are shown in Figure 2.13. It should be noted that these evolutions are slightly different from the previous evolutions as the boundary conditions are aperiodic and the left and right edges are not connected. Figure 2.13(a) shows the evolution after 100 steps. Looking at this evolution, we see some periodic structure on the left of the triangle. However, on the right side, there is no discernible pattern, although there are similar structures, inverted triangles, throughout. These local structures are of different sizes and follow no global structure. We might at first assume that we have not evolved the network sufficiently to observe a pattern or attractor with a longer cycle than we have seen in simpler evolutions. But when the system is evolved further, as in Figure 2.13(b), the same behavior is apparent. And if we were to continue to evolve the network with no communication between the left and right edges of the network, the result would be the same. Some structure on the left side of the evolution with locally similar structures in no apparent pattern on the right. When periodic boundary conditions are applied to the network started from a single black cell, all structure would eventually disappear and the evolution would appear like the right side of the evolution in Figure 2.13 throughout. This behavior would also be qualitatively similar to a network with periodic boundary conditions started from random initial conditions.

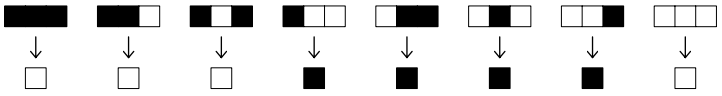
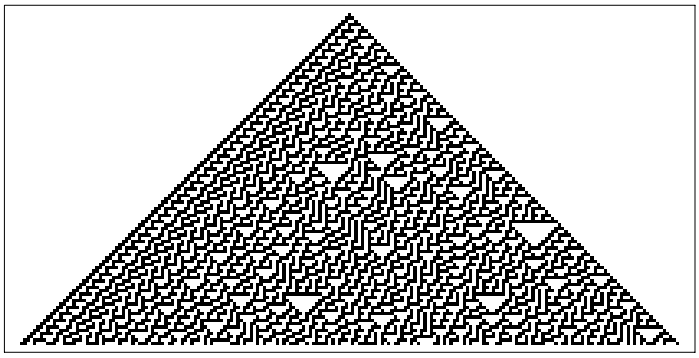
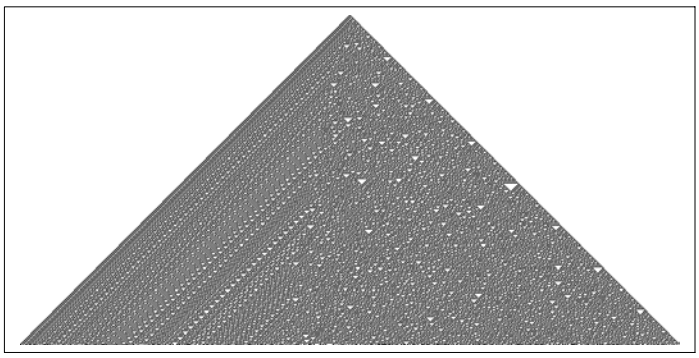


Figure 2.12: The rule set for elementary cellular automata rule 30. Despite the simplicity of the rule set, rule 30 inherently produces chaotic behavior.

As with evolutions that have simple behavior, chaotic rules are sensitive to initial conditions. The difference is that, while the state of a given cell within the simple network can be predicted at any evolution step if the initial state of the network is known, explicit evolution of a chaotic system is required to determine a cell state. Additionally, the rules in chaotic systems result in information passing throughout the network - the state of one cell will eventually affect all other cells within the network. This is the foundation for chaos theory,



(a)



(b)

Figure 2.13: Evolutions of rule 30 starting from a single centered black cell for (a) 100 steps and (b) 500 steps. While it may appear that the evolution created from a simple rule set should eventually settle down to some predictable pattern, especially with simple initial conditions, longer evolutions do not reveal any regularities.

illustrated with the classic Butterfly Effect - a butterfly flapping its wing in Australia can result in a hurricane in the North Atlantic [20].

2.2.6 Class IV: Complex

Complex behavior represents a mixture of simple and random behaviors, and has been described as a transition region between the two. Like a phase transition from solid to liquid, the complex regime is a transition regime from order to chaos. This form of behavior is illustrated in rule 110, described in Figure 2.14.

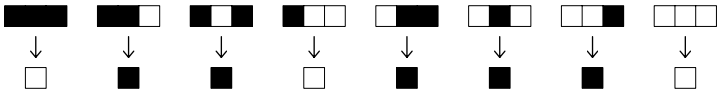


Figure 2.14: The rule set for elementary cellular automata rule 110. Rule 110 results in complex behavior, a mixture of order and chaos.

Figure 2.15 illustrates an example evolution of rule 110 from random initial conditions for the first 300 evolution steps. After a transient period, the system settles down to a regular background, through which random structures meander and collide to form new structures with different trajectories. As with chaotic evolutions, the system is deterministic and will therefore have a repetition period. Like chaotic systems, this limit cycle is often very long, making complex evolutions unpredictable. To determine the state of a given cell at a specific evolution step, explicit evolution of the network from initial conditions is therefore required.

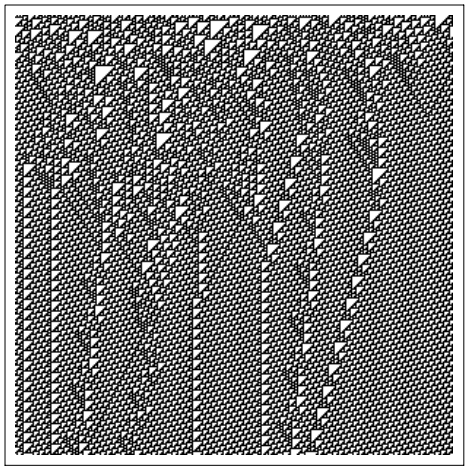


Figure 2.15: The first 300 evolution steps of rule 110 from random initial conditions. There are 300 cells in the network.

John Conway discovered a two-dimensional cellular automata with similar characteristics to rule 110 which is commonly known as the Game of Life [18]. The Game of Life has

been shown to be capable of self-reproduction, exactly the objective of John von Neumann. Closely associated with this is the concept of universality, or the ability of a system to emulate any other system. A familiar example of a universal system is a general purpose computer program, such as *C* or *Mathematica*. Given the proper code and initial conditions, a computer program can emulate any real system, including itself.

Despite its apparent simplicity, with only eight rules and two states, it has also been shown that rule 110 is universal [64]. It has also been suggested that all systems that exhibit class IV behavior share the property of universality, resulting from the ability to transfer information throughout a network without the degradation associated with randomness. The characteristic of universality suggests a certain degree of adaptability, through the ability to connect remote network components meaningfully, that other types of behaviors lack.

To provide additional evidence for the hypothesis that complex behavior is somehow associated with adaptability because of the universality that is believed to be inherent in this class of systems, we turn to the biological sciences, particularly evolutionary processes. We are not without precedent in looking to evolutionary processes towards application in engineering. Genetic algorithms, originally intended as a model for natural evolution, rely on the complex interactions of "schema" to search through intractable design spaces [42, 24].

2.3 Complexity in Natural Systems

The real reason for looking at natural systems is that they show a strong correlation between complex behavior and adaptability. In natural systems, we will see that adaptability, not necessarily performance in a given environment, implies optimality. Biological systems are the area where arguably the most work with complexity theory has occurred. This is largely a result of the fact that biological processes involve dynamic systems that almost always consist of, for all practical purposes, an intractable number of interacting elements that often yield non-linear behavior. To illustrate the association between complex behavior, adaptability and optimality in biological systems, we will investigate evolutionary theory and the models used to study evolution. Fundamental to the relationships of these attributes is the context of the network topology.

2.3.1 Co-evolution

We will discuss two models that both indicate that the process of co-evolution - the simultaneous evolution of interdependent species - evolves to a complex regime, at the transition region between order and chaos. More importantly, we will see that the complex regime corresponds to optimal performance in terms of fitness in one of the models. The first model, described by Kauffman [30], shows that by tuning the parameters that determine how genes affect the fitness within an organism and among species, it is possible to create a continuum

of evolutionary strategy behaviors, from stable to chaotic. In addition, the model shows that when the co-evolution is in a complex regime, the aggregate fitness of all co-evolving species is at a maximum. The second model, created by Holland [42, 17] uses a simulated ecosystem with no external measures of fitness, but still exhibits evolution towards complex regimes.

Coupled Fitness Landscapes

Kauffman describes two ultimate behaviors that can occur in a co-evolving system of interdependent species. The first is known as the Evolutionary Stable Strategy (ESS) regime of behavior. An ESS describes a Nash equilibrium [43], where each agent will always have at least one 'Nash' strategy. The Nash strategy has the property that an agent will be better off following it as long as all other agents involved follow their own Nash strategy. In terms of fitness landscapes, all species have found a suitable local peak from which they have no incentive to move.

One drawback often associated with Nash equilibria is that the overall fitness of agents at an equilibrium point may not be very good - the classic example being the Prisoner's Dilemma [42]. Furthermore, little exploration of the landscape occurs, since all strategies and performance remain unchanged. In evolutionary terms, species remain fixed, barring external inputs, and overall fitness remains low, with all species at local peaks.

The second region of behavior is the Red Queen, or chaotic regime. In this regime, the co-evolutionary process continually changes as a species deforms the fitness landscape of another. This second species in turns deforms the landscape of the first species, forming a feedback loop. This process becomes even more complicated as more species are added to the web of interactions. This form of co-evolution manifests itself as a 'biological arms race' as a species reacts to changes in competitors/adversaries that result in shifts in the balance of advantages. This form of behavior is known as Red Queen behavior in reference to Lewis Carrolls, *Through the Looking Glass*, where the Queen of Hearts instructs Alice, "it takes all of one's effort just to stay in one place".

Both the ESS and chaotic regimes result in low overall fitness for all species. In ESS behavior, species remain frozen at low local peaks on their fitness landscapes while in chaotic co-evolution, the landscape of each species constantly changes and a species that has found a peak is soon knocked off as the landscape is deformed by changes in other species. Whether a co-evolutionary process is ESS or chaotic depends on the movement of a species on its own landscape relative to the deformability of the landscape resulting from the interdependence with other species.

Kauffman uses a four parameter model to explore the relation between the relative rates of fitness landscape deformability and movement within a landscape and the resulting co-evolutionary behavior with coupled fitness landscapes. The first parameter, N, describes the number of genes (or traits) that defines each organism. The values for each trait or gene are

taken to be binary.

The second parameter, K, describes the number of epistatic couplings between genes within an organism's genome. Epistatic coupling describes the fitness contribution of one allele (0 or 1) of one gene in relation to other specific genes in the genome. That is, one gene alone does not determine behavior or fitness. Fitness is a function of several genes acting collectively. Each gene in the collective is assumed to contribute a certain fraction of the overall fitness however. And this fitness contribution can vary for different combinations of alleles (binary values) of the group of genes. The fitness contribution of each gene for each combination of alleles is determined through assignment of a random decimal value from 0.0 to 1.0.

A key assumption of the model is that a gene of one species impacts on the fitness of another species through several of the N genes. To define how genes interact between species and how fitness landscapes are coupled, the model uses a co-evolution coupling parameter C which indicates how each of the N genes in one species makes a fitness contribution that depends on C genes, again modeled as binary values, in each of the other connected species.

Fitness landscapes are coupled by assuming the fitness of one gene depends on the gene itself, K other traits in that organism, and C genes of another species. Like the fitness contributions with intra-species epistatic couplings, the fitness contribution of a gene for each combination of alleles of the C genes in another species is determined using a random decimal value between 0.0 and 1.0.

The final parameter necessary in the model describes the number of species found in the ecosystem, S. Closely associated with this parameter is the connectivity between species, or how many other species affect the fitness landscape of a given species. Kauffman uses both complete connectivity, where all species within an ecosystem are interdependent, and finite connectivity, where a species is affected only by species in a finite neighborhood on an arbitrary lattice.

The entire population of each species is considered genetically identical and all species are "updated" each generation/evolution step. At each generation, the genotype of each species is changed by random mutation of a single gene to its opposite allele (0 to a 1, 1 to a 0). If the new genotype results in a fitter organism, the mutation is adopted. Otherwise, the species remains unchanged. Only one mutation occurs per species per generation.

Varying the values of the number of genes in each species, the number of intra-species epistatic couplings, the number of inter-species gene couplings, and the species population in an ecosystem, it is possible to produce different evolutionary behaviors. When either the number of epistatic couplings, K, are high, or the number of inter-species gene couplings, C, are low, the ecosystem tends to settle to an evolutionary stable strategy where all species have found peaks in their fitness landscapes such that no mutation leads to increased fitness. When K is high, the landscape is rugged, resulting in many peaks for a species to get trapped on. When C is low, landscape deformation of species A that results from an adaptive walk of species B is minimal, so that the peak that A had found remains a peak. Behavior in

the ESS regime also arises when the number of species in the ecosystem, S, is low. This result follows the logic that, the fewer the species that can deform the landscape, the less the landscape has the potential to change. Kauffman identifies ESS behavior by plotting fitness of species through successive generations. When an ecosystem reaches a steady state condition, no changes in fitness occur throughout the population which indicates no changes in genotypes. The ecosystem has found a Nash equilibrium, where all species have found their optimal configurations with respect to the configurations of the rest of the population.

Chaotic behavior results when K is low (when there are few peaks to get trapped on), when C is high (many interdependencies exist between species so that a change in one species can have a significant effect on another species' landscape), or when S is high (so that each ftiness landscape is directly affected by many other species). In this regime of behavior, the peaks on each fitness landscape move away faster than the species can chase them and overall fitness is low.

The model indicates that the number of inter-species gene couplings and the number of species comprising an ecosystem has the greatest effect on the behavior of the co-evolutionary process. When C is high, a single move by one species has significant impact on the fitness landscapes of other species because the chance that the genotype of a species is affected by a mutated gene in another species is greater than if C is low. Similarly, when the number of species in an ecosystem is high and connectivity is complete so that many species affect the fitness of a single species, there is a greater chance of a single landscape being deformed and, the landscape is also subject to the collective effects of multiple deformations.

These results indicate that both stable and chaotic regimes can exist in the same system, with the type of behavior dependent on the values of the input parameters. Tuning the parameters results in a trasition between regimes, much like the change from simple to chaotic behavior observed when the value of the λ parameter, a characteristic of a rule set that indicates the level of activity and discussed later, is increased in cellular automata systems. In this transition regime, as in cellular automata networks, co-evolution is a mixture of ESS and chaotic behaviors.

To illustrate this mixture of behaviors, Kauffman uses a set of ecosystems co-evolved with the same model parameter values. Multiple sets are used to demonstrate the effect of varying the number of epistatic couplings. The observed values are the number of ecosystems, evolved with the same number of epistatic couplings, that have found a Nash equilibrium within a specified number of generations. Each set of 50 ecosystems is assumed to contain 25 species, arranged in an arbitrary 5 by 5 lattice. Connectivity is not complete, but is restricted to the nearest neighbors to the north, east, south, and west. Boundary conditions are not periodic and species in the corners and on the edges of the lattice have 2 and 3 neighbors, respectively. The number of inter-species couplings is fixed to a single gene.

The results of varying the number of epistatic couplings indicate that, for low values of K, the co-evolution process is always chaotic, with all 50 ecosystems in a state of perpetual variation at the end of 200 generations. Conversely, with K sufficiently high, all 50 ecosystems will

find Nash equilibria within the 200 generations available. The rate at which the species approach their equilibria increases with increasing values of K. When K is at a value of 10 however, a percentage of the 50 trial ecosystems has not reached a stable strategy in the 200 generations available. This indicates a transition region from ordered to chaotic behavior for this set of parameters with respect to a time scale as the number of epistatic couplings increases.

As parameters are changed, the aggregate fitness also varies. As K is increased from low values, the average fitness of an ecosystem at first increases, and then decreases. Absolute values of fitness are also affected by the number of epistatic couplings. The most important result here, and key to the assumptions in this work, is that, unlike in cellular automata networks, where tuning the λ parameter corresponds to qualitative changes in behavior, a correlation exists between the behavior of the co-evolution process and the performance of the species within the ecosystem, measured in terms of fitness. The highest average fitness occurs when the co-evolution process is in a complex regime, in the transition region between ESS and chaotic behavior.

ECHO

Echo, short for eco-system, is a model created by Holland to capture the fundamental dynamics of ecologies involving interacting species. Echo is characterized by the absence of endogenous fitness measures. Agents move about a discrete two-dimensional landscape based on environmental pressures. Located throughout the landscape are resources, which provide agents with "nutrients" necessary for survival and shape the genotypes and behaviors of agents. If an agent lives through an evolution step without gaining resources, it randomly moves to a nearby location. In their journeys, agents eventually encounter each other, and interact in a way that depends on their combined genotypes and what traits each agent can see of another. Agents interact through combat, trade, or mating, all of which involve some transfer of resources and a resultant change in the genotype of the interacting agents.

Since there is no external aggregate fitness measure for agents in Echo as for species in coupled fitness landscapes, it is not clear if the agents in Echo evolve to an optimal ecology in the sense of optimally fit species. However, the agents in Echo tend to evolve to a complex regime, in which they are optimally adaptive - continuously evolving to respond to changes in the genotypes of interacting agents in the ecosystem, but not to the point where a single species dominates or where each agent represents a distinct species. The complex and adaptive behavior of the model is evident in the distribution of populations of distinct species, which decreases logarithmically when species are ranked according to their abundance. This distribution is similar to the shape of rank-abundance plots in some real ecologies [42].

2.3.2 Organizational Size

In another model addressing "externalities", or interconnections between agents, the optimal organizational size of groups is examined [37]. Like the co-evolution model, intra-connections and inter-connections exist. Instead of connected genes however, connections between members of the same group and between different groups are considered. The configurations of group members are described by sequences of binary variables. The collective set of these member configurations determines the group payoff, a generic term intended to capture any measure of group welfare (cost, effort, time, profit, etc...). Payoffs for each configuration are defined by a randomly generated look-up table. Group members change their configurations through random movements in their configuration space in the search for optimal group payoffs.

The model shows that optimal group size is related to both the number of externalities and the length of the search period. When the number of externalities is low, individuals acting in groups eventually benefit, but optimal group sizes are small. As the number of externalities increases, optimal group size also increases, largely attributed the ability of large group sizes to buffer inputs from a large amount of external sources. A large amount of external inputs also provides a means for escaping local optima.

For long search periods, smaller organizations are desirable, as long as the number of inter-group connections is relatively low. With a small number of members, each group has a relatively small number of possible states to search through and a long search results in a larger fraction of searched states and a greater possibility of finding optimal configurations as compared to larger organizations. Conversely, larger sized organizations are optimal when short search periods are available as a large number of initial experiments in the configuration space are possible. With longer searches however, large organizations have a greater potential to remain in sub-optimal local optima.

The most critical result shows that, regardless of the constraints (search periods, group size, and the number of externalities), the optimal organizational configuration always evolves to a regime of behavior bordering between order and chaos. As the search period increases with large group sizes, the groups adopt stable strategies, reducing the uncertainty associated with complexity. Smaller groups operate well with small numbers of inter-group connections. As this number of interconnections increases, the complexity increases and the groups have the potential to stray into chaotic behavior, unless the group size increases.

2.3.3 Adaptation to a Complex Regime

Implicit in the co-evolution models described above is the process of selection. If a mutation results in a less fit variant, it is not adopted as the new genotype. This process of selection is common to adaptive processes, and is the basis for the genetic algorithm. In a more abstract model, Packard applies a genetic algorithm to find cellular automata that best perform

a computational task [45]. The result is adaptation to rules that are capable of complex behavior.

Packard defines the process of adaptation as the constant changing of the makeup of a population which is subject to selection where only the fittest members of the population survive. Given this definition, the ecosystem is defined by a description of the individuals comprising the population, the means by which they change, and how the fitness of the population is measured.

The population considered is a set of cellular automata rules. Under the application of a genetic algorithm, using mutation and crossover as adaptive agents, the cellular automata rules change to best accomplish a computational task. In the case of this model, the computational task is to determine whether a sequence of binary values has a density of 1's greater or less than 0.5. This task is modeled after the Gacs cellular automata rule (a radius 3, 2 state rule), which indicates the density of the initial set of states through the final state of the network. If the initial density is less than 0.5, the network evolves to all 0's and, if greater than 0.5, it evolves to all 1's. The fitness of a rule is measured by observing if the density of 1's in the network increases or decreases for various initial conditions.

Packard begins with a population of rules with a uniform mixture of λ values, where λ is defined as the fraction of transition functions that evolve to a quiescent state (this concept is discussed further in the following section). As the genetic algorithm is applied to the population of rules, they tend to cluster to areas corresponding to critical λ values. These critical values are associated with qualitatively complex behavior. Two transition regions corresponding to critical values of λ exist. The first as λ is increased from 0 and the second as λ is decreased from 1 and represents the 'negative' of the first. The transitions from order to chaos are based on the difference spreading rate, a measure of the chaos in an evolution.

Packard suggests that the evolved rules are independent of the computational goal. Instead, rules evolve to a population that is the most computationally effective for a fixed task. The relevant result to this work is that the computationally effective rules correspond to complex behavior. As with co-evolution, there exists a correlation between performance and behavior. The explanation for adaptation towards a complex regime to complete a computational objective is attributable to information propagation. In simple evolutions, no information in the form of patterns is transmitted across boundaries in the network. In chaotic evolutions, too much information is spread, making it impossible to maintain coherent structures capable of storing and passing information. In network evolutions showing complex behavior, information can prpogate throughout the network while maintaining its structure as boundaries are flexible yet defined.

Similar applications of genetic algorithms to the evolution of cellular automata rules for the purpose of performing computational tasks have also been performed [25, 16]. As in the evolution of cellular automata rules as density calculators, these models also yield cellular automata rules that correspond to a complex regime of behavior.

Genetic algorithms are applicable in other complex adaptive systems, including market based models. In a resource economics model describing how selfish agents with no information regarding the state of other agents or the resource regeneration rate, genetic algorithms bring systems to a range of behaviors [19]. Results indicate that shorter application periods of the genetic algorithm (which affects the potential of resource consumption) leads to chaotic behavior. Conversely, longer application periods result in more stable behavior with near optimal payoff.

The results from a virtual market model indicate similar results [29]. Frequent changes in investment stategies for all market participants result in a chaotic and inefficient market, with frequent bubbles and crashes in prices. As investment strategies are changed less often, the aggregate profit of market members increases and ordered, efficient markets result with longer term trends. The results also indicate a market equilibrium point corresponding to a Nash equilibrium, where no change in any market participant's strategy update period results in additional benefits. This point, which represents a stable strategy of all members, however, corresponds to a sub-optimal market. This result is analogous to the sub-optimal Nash equilibrium that can be achieved in the classic game theory Prisoner's Dilemma problem, where both participants defect.

2.3.4 Scale-Free and Distributed Networks

Strongly related to the behavior of a system is the structure of the connections between entities. Two network topologies that have been associated with complex behaviors while also exhibiting optimality with respect to either adaptability or performance are scale-free networks and distributed networks.

Network topologies are defined in part by the connectivity between nodes, or the probability $P(k)$ that a network is connected to k other nodes. In random networks, the connectivity distribution, $P(k)$, peaks at an average value of links $\langle k \rangle$ and decays exponetially with increases in k. For this reason, this class of networks is also known as exponential networks.

On the other end of the topology spectrum are structured networks (like cellular automata networks) which have homogeneous connectivity, where each node has the same number of connections. In addition, the path length between nodes is a constant.

In between these two topologies are scale-free networks, which have a connectivity distribution that follows a power law given in Equation 2.1, where $\phi(k/\xi)$ results in a cut-off at some characteristic scale ξ and k represents the number of connections.

$$P(k) \sim k^{-\gamma}\phi(k/\xi) \tag{2.1}$$

In these networks, most nodes are sparsely connected but a limited number of "hubs", or

highly connected nodes exist resulting in an inhomogeneous distribution of connections. Scale-free networks have been described as bearing the stamp of natural selection, sharing the charateristic of adaptability [56]. Mutations are typically considered random and natural selection favors systems that tolerate these random variations. Since the majority of nodes in scale-free networks are not highly connected, these networks are robust with respect to random failures of nodes [4, 1, 56, 54]. Evidence of this robustness has been shown in protein interactions in yeast cells [27], within the World Wide Web [4], in linguistics [10], and in electronic circuit design [9]. Scale-free networks have been shown to represent an optimal structure as well as being created through local optimization procedures [59]. Scale-free networks exhibit minimization of both the link density and the path length between nodes, resulting in the minimization of the costs associated with connections and optimal communication among system units.

By definition, the connectivity of a scale-free network exhibits a power law relationship. The power law relationship has been demonstrated to exhibit a close association with the concept of self-organization and arise in fractals, cellular automata, and statistical measures of the frequency and intensity of natural phenomena such as earthquakes, avalanches, floods, and forest fires [51]. Self-organization is a characteristic that requires meaningful information transfer through a network, which is associated with complex behavior. Evidence of this relationship exists in cellular automata [64, 36, 35] and in autocatalytic sets, networks of interactions that are self-sustaining [30, 31].

Closely related to the topology of scale-free networks is the small-world network structure, which exhibits large clustering of nodes and minimal vertex-vertex distance and link length. The characteristics of a small-world structure mean that nodes may not be directly connected, but the number of indirect connections between any two nodes is typically low, resulting in short communication distances. Networks exhibiting the small-world structure have been found in both natural and artificial systems including the neural network of the worm *Caenorhabditis elegans*, which is the sole example of a completely mapped natural neural network, the power grid of the western United States, and the collaboration graph of film actors, which is popularly referred to as "six degrees of Kevin Bacon" and serves as a model for a social network. Because of their complex architecture, small-world networks demonstrate enhanced signal-propagation speed, computational power, and synchronizability [61].

Distributed networks are another topology that has been shown to exhibit complex behavior and adaptability. In a distributed network, nodes are connected only to nearby neighbors and a node's actions are dictated by the state of these neighbors. The elementary cellular automata described earlier are a classic example of a distributed network, with each cell connected to neighbors within a radius r. Although this type of connectivity is not inherently associated with any one type of behavior, cellular automata networks exhibit a complete range of behaviors, even with small local networks.

In natural systems, distributed networks have been used as an accurate model of flocking

and herding behavior [48, 6, 8, 23, 49, 57]. Despite the limited connectivity of flock and herd members, these systems are capable of exhibiting complex global behaviors not necessarily predictable from the simple,local rules dictating the reaction of members to neighbors. At the same time, these distributed networks are robust and adaptable with respect to failure for the same reasons as those for scale-free networks. The random failure of a single entity does not cause the entire system to fail because no single entity controls the actions of the entire network.

2.4 Summary

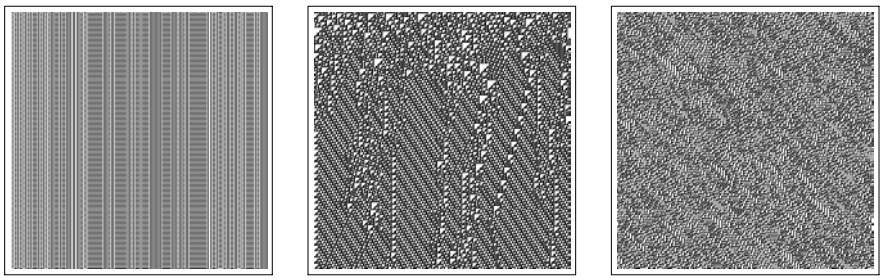
The models described above are evidence of some relationship present between behavior and optimality, where optimality can be defined with respect to either performance, adaptability, or robustness. Given the freedom to evolve their characteristics, systems with complex interactions have been shown in many cases to evolve to a complex regime, which offers a combination of stability to take advantage of niche environments and adaptability to respond to environmental changes, which can be indirectly self-induced. The same characteristics of complex behavior that result in adaptability also result in good performance. The stability allows a system to remain at optimal or near optimal fitness peaks while sufficient flexibility permits state space exploration in search of greater performance.

The indication of a meaningful relationship between system behavior and optimality suggests that behavior can provide a means for searching for solutions in large design spaces of complex systems. However, this relationship must be quantitative, not qualitative in order to take advantage of it. While performance measures are easily quantified, a universal quantitative definition of behavior is more elusive, requiring an examination of various complexity measures.

Chapter 3

Measures of Complexity

Thus far, all examples of behavior we have seen have had a qualitative nature associated with them. We have been able to classify the behavior mainly through our intuitive perceptive abilities [64, 26]. In the example cellular automata evolutions shown in Figure 3.1, it is apparent that (b) is the most complex of the three evolutions. Figure 3.1(a) is simple to represent by describing the fixed point attractor state while Figure 3.1(c) is indistinguishable from a series of states generated from a simple pseudorandom state generator.



(a) (b) (c)

Figure 3.1: Three qualitative behaviors from elementary cellular automata rule sets. Evolution (a) is from rule 4 and represents simple Class II behavior. Evolution (b) is Class IV complex behavior from rule 110. Class III random behavior is in (c) from rule 45.

Intuitively, we sense a continuum of complexity with a maximum lying in a region between maximum order and chaos. In addition, our intuition indicates the greatest complexity in systems where many components interact to create coherent but unpredictable behavior. In real systems, we can judge that the modern economy is more complex than a fiefdom, or that a jet engine is more complex than a vacuum cleaner. This perception of complexity, while powerful, is often subjective and only an objective measure of complexity provides an absolute description of a system that can validate intuition.

To date, there exists no universal quantitative definition of complexity. However, several

measures, both static and dynamic, have been proposed. A static measure of complexity is based on the architecture of the system or the strengths of the associations between components. Dynamic complexity measures the describe the computational effort required to describe the information content of a state or entity in a system. In practice, we can say that static complexity measures describe the potential types of behavior that a system can support while dynamic complexity measures describe the behavior that has emerged from a system. Static measures have the advantage of quantifying complexity with relatively little computational effort since large scale system simulations are not required for analysis. However, the "rules" of the system must be known, which are not always available. The dynamic measures act independently of known rules and use the actual behavior of the system, making them a truer measure of system behavior.

While many measures of complexity have been proposed, we will limit ourselves to the most commonly known and applied measures presented in Table 3.1.

Table 3.1: Proposed static and dynamic measures of complexity

Static measures	Shannon's Information
	λ parameter
	Hierarchical complexity
	Simplicial complexes
	Component counting
Dynamic measures	Mutual information
	Algorithmic complexity
	Computational complexity
	Logical depth

3.1 Shannon's Information

Shannon's information, or Shannon's entropy, originates from information theory as a means for measuring the information content in telecommunications [53]. The information content of a system of N parts is defined in Equation 3.1.

$$I = -K \sum_{i=1}^{N} p_i log_2 p_i \tag{3.1}$$

where p_i is the probability of the i^{th} part/event/state occurring and K is a constant that accounts for units of measure. The logarithm is used because it satisfies the property that information is additive for independent events. With two independent sets of events, N_1 and N_2, the total set of possible outcomes is $N = N_1 \cdot N_2$. The information content of N, $I(N)$ is the sum of the information in each independent set of events, as in Equation 3.2.

$$I(N) = I(N_1) + I(N_2) \tag{3.2}$$

As a simple example, we can consider the case of a system with two possible outcomes. The entropy of two outcomes with probabilities p and $q = 1 - p$ is given by Equation 3.3.

$$I = -(plog_2p + qlog_2q) \tag{3.3}$$

This simple case describes the entropy in the elementary cellular automata described in the previous chapter with $k = 2$ states. Entropy is maximal when $p_{black} = 0.5$ for a given cell, as is the case for evolutions of rules 30 and 45 (Figure 3.1(c), both chaotic rules. Entropy is 0 when either $p = 0(q = 1)$ or $p = 1(q = 0)$. In general, entropy is maximal for random systems and minimal for ordered ones.

Shannon shows the following attributes as support for entropy as a measure of information [53]. First, that $I = 0$ if and only if all p_i but one are 0. This p_i has a value of 1 (like evolution to a trivial quiescent state (all 0's or all 1's) in a cellular automata) and implies a certainty in the outcome. For all other combinations of p_i, information is positive. Second, for a set of events, n, the information is maximal and equal to $log(n)$ when all $p_i = \frac{1}{n}$ (like chaotic cellular automata evolutions) since the most uncertainty exists for all outcomes.

Entropy can be defined in both spatial and temporal dimensions, the results of which are not necessarily equivalent. A cellular automata evolution exhibiting repetitive behavior for instance may have minimal spatial entropy and maximal temporal entropy.

Some criticisms of using entropy as a measure of complexity are related to its additive properties, its inability to measure a single state, and the fact that it defines not what is known, but what information is missing from a system [26]. Furthermore, entropy as a measure of complexity runs counter to intuition regarding what qualitative form of behavior is most complex. Maximum entropy occurs when a system exhibits randomness while intuition suggests that random behavior has minimal complexity.

3.2 The λ parameter

The λ parameter, introduced by Langton, is intended as a method for parameterizing the cellular automata rule space [36, 35, 65, 39]. The λ parameter describes the number of cell neighborhoods, or transition functions, of the 2^{2r+1} possible neighborhoods, that evolve to a given state and acts as an indicator of the potential activity of evolutions. The λ parameter is defined formally in Equation 3.4, where m is the number of active cell neighborhoods.

$$\lambda = \frac{m}{2^{2r+1}} \tag{3.4}$$

The λ parameter represents an extremely powerful measure of behavior because it is determined independent of any explicit evolution. Since it is based on the evolution rules, not the evolution itself, behavior can be (approximately) determined with much less computation and provides the potential for designing behavior with only knowledge of the rules governing the evolution process.

For $\lambda = 0$ (and $\lambda = 1$, essentially a 'negative' image of $\lambda = 0$ for $k = 2$ systems), there is no potential activity. This value of λ corresponds to all neighborhoods mapping to the 0 state (or 1 state) and describes the trivial rule sets 0 (and 255). As the value of λ is increased to 0.5, the corresponding behavior goes through fundamental changes. Low values of λ correspond to evolutions with fixed point to limit cycle attractors. With $\lambda = 0.5$, evolutions are chaotic, with no attractors. As λ is increased from low to high values of potential activity, the behavior of evolutions passes through a transition region at some critical range that corresponds to a complex regime.

This transition region and the correspondence of λ to behavior is difficult to observe in two-dimensional elementary cellular automata because of the low resolution of cells comprising a neighborhood. With only $k = 2$ states and a neighborhood of three cells, there are only eight neighborhood combinations and a λ resolution of 0.125. With larger neighborhoods and more possible states, the resolution increases (its scale decreases). In his original investigations, Langton uses 8 state, 5 neighborhood cellular automata, with a λ resolution of $\frac{1}{8^5} = \frac{1}{32768}$. It is important to note that, for $k > 2$, λ is no longer symmetric about $\lambda = 0.5$ and that the number of neighborhoods mapping to a quiescent state is $(1 - \lambda)k^{2r+1}$ with the remaining λk^{2r+1} neighborhoods mapping to any one of the non-quiescent states. With an arbitrary definition of which state is quiescent, it is therefore still possible to have multiple equivalent, 'negative' images with $k > 2$.

λ has been shown to have correlations with other complexity measures such as difference pattern spreading rates, single site temporal entropy, and mutual information [39]. All correlations indicate a critical value of λ associated with a 'phase transition' from ordered to random behavior.

While the λ parameter is useful for describing the static behavior potential in cellular automata networks, it is of limited value in other systems. It is most applicable in homogeneous systems, where all elements are identical, and the set of rules controlling system behavior is known in detail.

3.3 Hierarchical Complexity

The complexity of Hierarchical systems is based on the idea of clustering system components according to the strength of their mutual interactions. The most strongly interacting components are associated with upper parts of the hierarchy, followed by progressively weaker associated components.

Complexity in a Hierarchical system is based on a measurement of the system described by the number of interactions between and within subtrees of a hierarchy. Complexity of a hierarchy is quantified as in Equation 3.5.

$$C(T) = log_2 D(T) = log_2\{f(k_T) \prod_{j=1}^{k} D(T_j)\} \tag{3.5}$$

The diversity of tree T, denoted by $D(T)$, is found by counting all distinct interactions within clusters, or subtrees of T. If the tree branches to a single leaf or set of single leaves, the diversity of each leaf is 1, as is the diversity of the root. The diversity becomes greater than 1 when at least two subtrees from one (sub)root have different structures. The number of interactions is the product of all diversities of branches and, in the case of branches to single leaves (a set of which will always be found at the bottom tier of the hierarchy), the resulting diversity of the subtree is 1. The form factor, f_T describes the number of distinct subtrees stemming from a given (sub)root. It is calculated as the number of ways, N_k that k subtrees can interact and is given as $2^{k_T} - 1$. A tree with a constant branching ratio has only one distinct subtree at every tier and therefore has a diversity of 1 and a complexity of 0. This follows intuition as a constant branching ratio represents an ordered structure.

3.4 Simplicial Complex

A simplicial complex (here, complex is used as a noun to denote structure, not behavior) is constructed by combining elementary networks, or simplicies, which themselves define associations between elements. The dimension of a simplex is defined by the number of nodes in the simplex. An n-simplex will have $n + 1$ nodes. A 1-simplex is therefore comprised of two nodes connected via a common thread and a 0-simplex is simply a point.

As an example, consider two sets $X = (x_1, x_2, x_3, \ldots, x_n)$ and $Y = (y_1, y_2, y_3, \ldots, y_n)$. We can suppose that x_1 and x_2 are both associated to some common member of set Y, say y_1. In this example, x_1 and x_2 are nodes in a 1 dimensional simplex. If x_3 is associated with only one member of set Y that no other members of X are associated with, it forms a 0-complex.

It is quite possible for a single node to be a member of multiple distinct simplices, or for multiple nodes from one simplex to be common to multiple simplices. Because of this non-uniformity, a single edge between nodes may be shared by many simplices. This relation between simplices is characterized by the simplices' q-connectivity, which describes the minimum number of shared edges between two particular simplices. The structure of a simplicial complex (the set of all simplices) is described with the use of the first structure vector, $\vec{Q}= \{Q_D, Q_D - 1, \ldots, Q_0\}$ that describes the number of q-connected simplices for each value of q for $0 \leq q \leq D$, where D represents the largest dimension simplex of the simplicial complex.

The complexity measure, $K(C)$ of a simplicial complex, based on the first structure vector, $\vec{Q}$, is given by equation 3.6.

$$K(C) = \frac{2}{(D+1)(D+2)} \sum_{i=0}^{D} (i+1)Q_i \tag{3.6}$$

which satisfies the properties that the complexity of a complex consisting of a single simplex is 1, the complexity of a sub-complex can not be greater than the complexity of the entire complex, and that the complexity of the complex formed by two complexes is less than or equal to the sum of the individual complexes.

3.5 Number of Components

As a means for establishing rough estimates of complexity, Bar-Yam utilizes component counting as a complexity measure [3]. The rationale behind component counting is that complexity is a function of the interactions between elements and with greater numbers of interacting elements comes greater potential for complex behavior.

Key to using component counting as a complexity measure is the identification of relevant components and the scale of observation. The determination of what is relevant to complexity is often subjective and can greatly affect the estimates of complexity. Similarly, scale has an important effect. Bar-Yam asserts that microscopic scales set an upper bound on system complexity, which decreases with increasingly macroscopic views dependent on the behavior of the system at those levels.

In general, complexity is maximal on microscopic scales and decreases as the scales increase in size. The rate of the decrease and the shape of the complexity profile depends on the behavior of the system. For simple systems, such as a crystal, complexity decreases rapidly at scales above the atomic level because there is no additional information required to describe the system at larger scales. The complexity of a human is high at small and large scales however when societies are considered at a macroscopic scale.

Measuring complexity through component counting fails to address the behavior of the system and only presents the potential character of behavior that might be supported by a system. In addition, it often ignores the connectivity between elements, which is an important consideration. For low connectivity, behavior is typically simple while for complete connectivity, behavior tends towards randomness [30].

3.6 Mutual Information

Mutual information describes the correlations that exist between variables and can be used to describe how well information is communicated through a system. The mutual information between two probability distributions $\{p_i\}$ and $\{p_j\}$ is given by Equation 3.7, using the joint probability $\{p_{ij}\}$.

$$M = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{p_i p_j} \tag{3.7}$$

If the two probability distributions are statistically independent, the mutual information between them is zero and if the distributions are correlated, the mutual information is non-zero, the value of the mutual information dependent on the strength of the correlation.

Like entropy, mutual information can be determined in both spatial and temporal dimensions. In application to cellular automata, spatial mutual information is based on the state probabilities of two sites, or two blocks of sites separated by d cells. The temporal mutual information is based on the correlation of a single site at two points in its evolution history [39].

For evolutions with fixed point attractors, mutual information is zero. No correlations can exist between sites, which are divided by the deterministic 'boundaries' that form in simple evolutions. The lack of information transfer across the network also accounts for low mutual information in evolutions with finite limit cycle attractors. In disordered dynamics, mutual information is low because each site deals with conflicting inputs from multiple other sites and any coherent information is rapidly degraded. In a statistically random evolution, low correlation between two random variables is expected.

Mutual information is maximal when spatial structures are present and coherent information has the ability to move through a system. This maximum corresponds to qualitatively complex dynamics at the phase transition between ordered and chaotic dynamics (and is a requisite for the associated property of universality) [39].

Like the measure of Shannon's information, mutual information relies on statistical properties of a system evolution and is therefore an ensemble characterization. It can therefore not account for local dynamics and can not measure the complexity of a single state.

Mutual information follows intuition regarding the relationship between qualitative behavior and quantitative complexity. Highly ordered systems that are intuitively non-complex have low mutual information. Similarly, random systems also have low mutual information which again matches intuition. Maximal mutual information occurs in the complex regime, between these two regimes.

3.7 Algorithmic Complexity

The concept of algorithmic complexity was introduced by Solomonoff, Kolmogorov, and Chaitin [12, 13, 11]. The algorithmic complexity $I_C(s)$ of a binary string, s, is defined to be the shortest program p that produces the output s on a universal computer U. This is presented formally in Equation 3.8.

$$I_U(s) = min_{U(p)=s} log(p) \tag{3.8}$$

By Solomonoff, algorithmic complexity was meant to address the conciseness of a mathematical theorem. By definition, a theorem is a simpler means of describing a more complex phenomena. The smaller the theorem, the more concise and therefore the greater the understanding of the phenomena and the more capable of the theorem for making future predictions about the phenomena. Random phenomena that are not predictable can not be explained by a theorem or any other shorter form and the most concise method for representing the phenomena is a direct reiteration of the observations.

It should be noted that the value of $I_C(s)$ is dependent on the universal computer used because of different operating characteristics of the machines. However, since the computers are universal, they are by definition capable of emulating each other with only some finite sized 'translation' code. The difference between algorithmic complexities on universal computers U_1 and U_2 is therefore bounded by the addition information, $\tau_{U_1U_2}$ required for each computer to emulate the other, as in equation 3.9

$$|I_{U_1}(s) - I_{U_2}(s)| \leq |\tau_{U_1U_2}| \tag{3.9}$$

As the length of the binary string, s, increases, the difference between the two complexities becomes a decreasing fraction of the program length and is eventually considered negligible.

Application of the algorithmic complexity measure is illustrated with a comparison of two binary digit sequences shown in Figure 3.2.

(a) 10101010101010101010
(b) 10111001001010011010

Figure 3.2: Two binary digit sequences. Sequence (a) can be compressed to a short program <Print '10' ten times> and has low algorithmic complexity while the sequence in (b) is incompressible and requires a program approximately as long as the digit sequence itself to represent it, resulting in high algorithmic complexity.

In sequence (a), there is a repetitive pattern of the '10' digit sequence. This sequence lends itself to a compressed format and can be generated by a program $<$ *Print* '10' *ten times* $>$.

Because of its representation by a simpler program, sequence (a) has a low algorithmic complexity. A digit sequence of '10' repeated a billion times would therefore have approximately the same algorithmic complexity, with the difference only in the bits in the exponent of the program $<$ *Print* '10' 10^9 *times* $>$. In contrast, digit sequence (b) is in a relatively incompressible form and the shortest representation of the digit sequence is the sequence itself.

Despite its name, algorithmic complexity is more of a measure of randomness than complexity as described in the context of elementary cellular automata. In random strings, the algorithmic complexity is maximal, which runs counter to our intuitive description.

It has been demonstrated that most numbers are random (algorithmically random) and that the number of strings of a given binary length that are algorithmically simple are rare, so that for an arbitrary lower bound on complexity of $n-x$ bits in an n bit number, the fraction of simple numbers is $\frac{2^{n-x}}{2^n} = 2^{-x}$ and for a greater upper bound on complexity, there are exponentially fewer simple numbers. For a typical incompressible random binary string, s, the algorithmic complexity, $I(s)$, is given in Equation 3.10

$$I(s) \approx |s| \tag{3.10}$$

and for the decimal equivalent of s, denoted n, the algorithmic complexity is given in Equation 3.11 [68].

$$I(n) \approx log_2|n| \tag{3.11}$$

The algorithmic complexities in Equations 3.10 and 3.11 are related approximately to the numbers because of the additional bits required for the encoding of the program to represent the numbers. For large digit sequences however, this additional information is negligible.

Often critical in the complexity of the representation of a system is the scale of the system and how much detail is required or desired in the system description [3]. The algorithmic information content of a human described by the DNA 'program' is on the order of 10^{10} bits while a human described on its atomic level is on the order of 10^{30} bits. A human described on the level of the number of organs may be estimated by 10^2 bits. Complexity profiles on both time and length scales describe these relationships.

In practice, it is often difficult to establish absolute values of algorithmic complexity for a given system for a reason related to Gödel's incompleteness theorem that no formal descriptive system can encompass all true theorems. For numbers, no number can be proven random unless the complexity of the number is less than that of the formal system used to describe it. The information in a formal system is always finite and is therefore always incapable of proving whether a number with greater complexity is random. This argument originates from the paradox formed from a statement of the form, "this sentence describes a positive integer of complexity which to be specified requires greater complexity than exists

in this sentence." If this integer exists, it can be described in a more concise form by the sentence above, meaning it is not of greater complexity than the sentence allows, so that the sentence itself is a self-contradiction [12].

3.8 Computational Complexity

Computational complexity is a measure closely related to algorithmic complexity. However, instead of measuring the minimal sized program required by a universal computer, computational complexity measures the minimal time or memory required by a universal computer to solve a particular problem. In a simple instantiation, this might involve finding the total number of elementary binary mathematical operations $(+, -, \times, \div)$ required for algorithm A to evaluate a function f.

This definition is extendable by relating the time/steps required to solve a problem with the size of the problem. If $\Sigma_N^{(i)}$ is the initial state of a size-N problem with a final state solution using function f of $\Sigma_N^{(f)}$, then the computational complexity, $H_C(\Sigma_N^{(f)})$ is the time required, τ, for the program, P, running on a universal computer, U, to reach state $\Sigma_N^{(f)}$ from initial state $\Sigma_N^{(i)}$. This relation is presented in Equation 3.12.

$$H_C(\Sigma_N^{(f)}) = min_{U(P)=\Sigma_N^{(f)}} \tau_U(P) \tag{3.12}$$

Problems are classified based on how the computational complexity changes with respect to changes in the problem size. The computational complexity of class P problems increases with a polynomial relation to the problem size, as described in Equation 3.13.

$$H_C(\Sigma_N^{(f)}) \leq O(N^\alpha) \qquad | \quad \alpha < \infty \tag{3.13}$$

In class NP problems, the time required to test possible solutions is on the order of a polynomial function of the problem size, but the time required to find the solution is not necessarily related polynomially to the problem size.

The spatial measure of computational complexity is the memory required to solve a particular problem. The PSPACE class describes problems that require memory on the order of a polynomial relation to problem size. The time required to solve a problem of this class is not necessarily known.

Computational complexity treats all complexity measures in terms of functions. A drawback to this approach is that not all systems are describable in terms of functions. The complexity of a given system state or a particular object may be undefinable in this context, or of a level of abstractness that the complexity of the function no longer represents the complexity of the object.

3.9 Logical Depth

The concepts of algorithmic complexity and computational complexity are combined in the measurement of logical depth. Logical depth, D_U^L describes the time, τ required for a universal computer, U to execute a minimal program, P^* to generate some output representative of object, O, as in Equation 3.14. In Equation 3.14, $S(O)$ is the binary string representation of object O.

$$D_U^L(O) = \tau_U(P^*) \qquad where\ U(P^*) = S(O) \tag{3.14}$$

Like algorithmic complexity, the value of the logical depth is dependent on the universal machine being used to run P^*. However, as the lengths of the minimal programs increases, the difference between the logical depths on two universal machines becomes negligible.

Logical depth combines the advantages of both algorithmic complexity and computational complexity. For simple objects, logical depth is low because the program required to create the object (or bit string representation of the object) is small because of the amount of available compression. Similarly, a random object could be created by a short, stochastic based program or a $< Print >$ function. Complex objects however, can not be represented by simple programs such as $< Print >$ because they contain internal evidence of having been the result of a long computation. In addition, these types of objects can not plausibly have originated by any other method than a long computation or slow-to-simulate dynamical process [7].

Logical depth suffers from disadvantages also seen in algorithmic complexity. First, it is not directly apparent how to represent all objects as a bit string or program, particularly physical systems or processes. Second, the scope of the program affects the program size and computational effort required to represent a system. Any measure of logical depth must therefore be made in the context of the level of detail involved in simulating a system, at which point the logical depth of the program and not the system itself is measured [26].

Chapter 4

Elevator Group Control and Naval Weapons Elevators

4.1 General Vertical Transport

Elevator group control is a class of problems that has long been recognized as being non-deterministic and capable of exhibiting complicated behavior [5, 55]. However, many attempts have been made to optimize elevator group control through various methods.

When elevators first appeared in buildings, they were manually controlled by the passenger. To call an elevator, a passenger would pull a rope running the interior length of the shaft that was linked to a hydraulic control valve. The control rope was also used inside the elevator to control direction. Since passengers dictated the control of the car, service was essentially limited to one passenger (or call) at a time. Manual control was replaced by electric signaling in conjunction with attendants dedicated to a specific car. The attendant made decisions regarding which floors to service as well as the sequence of stops, based on the signals received and the destinations of passengers. The limitations of control by attendants resulted in fully automatic push button elevator systems. But with the elimination of the attendant came a requirement for a new form of car control.

The simplest form of automatic car control is single call automatic control. In this method, passengers are served on a first-come-first-served basis and passengers in the car have priority over passengers still demanding service. Upon selection of a destination, an elevator proceeds directly to the destination, bypassing any intermediate floors with landing calls, or passengers requiring service. When a car is free, it proceeds directly to the next (longest waiting) landing call. This control is reminiscent of the manually operated control valve, but eliminates the possibility of any altruistic intermediate landing calls.

Collective control systems are intended to reduce the inefficiencies resulting from bypassing

landing calls in single call systems. Collective control is categorized as either single-button or two-button. In single-button systems, the direction of the passenger is not considered and a car with a passenger will stop at all floors on its way to its destination regardless of the desired direction of travel of the passenger at that floor. A passenger may enter an elevator that is traveling in the opposite direction of the desired direction. The desired direction of travel is accounted for in two-button collective control systems. In directional collective control, landing calls describe the desired direction of travel. An elevator stops to service both car calls and landing calls in the current direction of travel and will bypass landing calls in its path if the directions do not match the current direction of the elevator. When no more calls exist ahead of the car that match the current direction of travel, the elevator reverses direction and answers all calls corresponding to its new direction.

In all of these control systems, we are only considering a system with a single elevator. But a single elevator is not always capable of handling a system's traffic. When multiple elevators are used, we encounter the problem of interconnecting the elevators to operate collectively. While multiple elevators could operate independently, the resulting performance would be inefficient, as several elevators could answer a single call. The coordinated, supervisory control of multiple elevators is known as elevator group control. In a group control system, elevators share a signaling system and serve common areas according to some set of rules.

Rules for controlling elevator groups take many forms and are often designed to accommodate specific traffic conditions. Common traffic conditions encountered in elevator systems are up-peak, down-peak, and inter-floor traffic conditions. Up-peak and down-peak conditions are common in commercial office buildings that operate according to a standard schedule. An up-peak, or incoming traffic condition occurs when passengers arrive at work and travel from a common location (the lobby) to individual destinations (upper floors). Down-peak traffic occurs when passengers leave the building from multiple floors, destined for a common location. Inter-floor traffic, or two-way traffic, involves passengers moving from multiple locations to multiple destinations. It is often assumed that an elevator system designed to handle an up-peak traffic condition will be able to accommodate down-peak traffic [5].

The approaches taken in dealing with the elevator group control problem can be divided into four main categories: reductionism, heuristics, non-linear optimization, and adaptive control. The approaches are not necessarily distinct and a control system can involve elements from two or more approaches.

Reductionism is a common approach taken in the design of linear, and sometimes non-linear systems. Reductionism involves the identification of relevant physical variables. In the case of elevator group control, examples of these variables are elevator acceleration and velocity capabilities, number of elevators involved in a system, number of floors served, and maximum car capacity. The effect of each variable is determined by varying it within a fixed frame of reference, where all other variables remain constant. Schroeder [52] presents a reductionist approach in the control of a down-peak traffic condition, using passenger boarding rate, and performance parameters such as waiting time, destination time, and load upon arrival at

the lobby as functions of the number of boarding stops during a round trip. Schlemmer *et al.* [50] consider the trade-offs between maximum power required and passenger comfort constraints related to acceleration/deceleration in the minimization of transit times. The time-optimal path is built into a control algorithm that accounts for sensed passenger loads and destinations.

Heuristic methods utilize operational rules that are assumed to provide optimal performance. Heuristics, while in operation can be effective, do not necessarily guarantee globally optimal behavior and can only be measured in the context of other control strategies. Heuristic control techniques emerged concurrently with the advent of automatic car control, and in some respects, are found in attendant controlled systems. Strakosch [55] presents heuristics for up-peak, down-peak, and inter-floor traffic conditions, including the up-peak zoning control. Barney and dos Santos [5] presents multiple supervisory group control algorithms, including directional collective control, Duplex/Triplex control, fixed sectoring priority timed, fixed sectoring common-sector, dynamic sectoring, minicomputer based system (minimization of expected car journey times), and call allocation with known destinations. These methods account for the allocation of cars to requests for elevators (landing calls), the assignment of direction to predetermined or variable zones of control, the availability of elevators for requests, and local optimization with respect to the minimization of passenger waiting time or journey time. Newell [44] further investigated the fixed sectoring heuristic and demonstrated that the number of elevators serving a sector affects the total journey time (waiting + riding time). Newell also proposed a limit on the number of floors served per trip and the effect of revisiting identical floors in successive trips for a given elevator. Pang [46] describes the use of a blackboard architecture for traffic control of elevators in a multi-story building. The method applies heuristics and basic elevator scheduling rules in a structure that accounts for large amounts of diverse and incomplete knowledge in a real time decision making process. A threshold policy, where elevators wait for a minimum number of passengers before departing, is demonstrated by Pepyne *et al.* [47] to be optimal with respect to the minimal aggregate passenger waiting time. This policy is applicable to systems involved in an up-peak traffic condition. Recently, fuzzy logic has been applied to elevator group control [33, 67, 28, 58, 22, 32].

Non-linear optimization techniques are best described as local optimization procedures that utilize some form of simulation. Yoneda *et al.* [66] use a genetic algorithm, which is itself a heuristic, as an intelligent elevator supervisory control system. Genetic algorithms were originally intended for application to adaptive systems, specifically biological evolutionary systems. Elevator group control systems must 'adapt' to changes in traffic conditions and passenger arrival rates, while accounting for their current physical configuration. This adaptive control utilizes multiple parameters in the definition of an objective function. Clark *et al.* [14] created a knowledge-based control system, similar in operation to Pang. The system uses multiple heuristics that can be compared through simulation to obtain optimal scheduling of elevators. Optima in this system were defined by passenger throughput and individual passenger waiting time.

Traffic patterns in a given building are not fixed, and may change several times during the course of a day. A given heuristic, or optimization technique may provide good performance in one traffic condition at the expense of others. The use of multiple appropriate policies in conjunction with traffic pattern recognition has been investigated [21, 41]. Neural networks learn to identify traffic patterns that are matched to pre-specified traffic patterns, resulting in the selection of an appropriate control algorithm. Identification of traffic patterns does not necessarily require matching to pre-specified traffic patterns. Kim *et al.* [34] proposed a system that utilizes cameras and call allocation with known destinations to determine traffic conditions. The traffic conditions are used to create a dispatch function through a genetic algorithm with an adaptive objective function. The dispatch function accounts for the distance between an elevator's current location and a floor where a request has been issued, the destinations of elevators relative to the request, the current number of elevator passengers, and the number of destinations between an elevator's current location and the request. Crites *et al.* [15] use reinforcement learning agents to control a group of elevators in a collective learning algorithm.

4.2 Naval Weapons Elevators

A weapons elevator system, like any elevator system, consists of four types of elements: entities (passengers or materiel), elevators, queues, and destinations. The principal function of the weapons elevator system is to transport ordnance from queues to their destinations. The flow of material is bi-directional and is associated with distinct operations, presented in Figure 4.1. In strike-down operations, ordnance is moved from the main deck to the ship's magazines for storage. In this scenario, the queue(s) is located on the main deck and the magazines are the destinations. Strike-up operations remove ordnance from the magazines (queues) and bring it to the main deck (destination). Strike-down and strike-up scenarios are similar to the up-peak and down-peak traffic conditions commonly found in commercial office buildings with the exception that retrograde of packing material is often involved in strike-down operations which may affect the operation of the system. Additionally, in an office building, the same number of passengers is typically involved in both up-peak and down-peak traffic, but in a weapons elevator, strike-down often involves more traffic volume than a strike-up operation. However, like the performance of an elevator group in up-peak and down-peak conditions, the performance of a weapons elevator during strike-up and strike-down can be significantly different. Because of the analogy with building traffic conditions, we will only consider strike-down operations for the moment. As up-peak empirically provides the greatest limitations on system performance, we will assume that its analogy, strike-down, behaves similarly.

The ultimate measure of the performance of the weapons elevator system is its ability to complete the function of transporting items, which can be equated to total cycle time. It is in the best interest of the carrier in both traffic conditions to complete transfer operations in

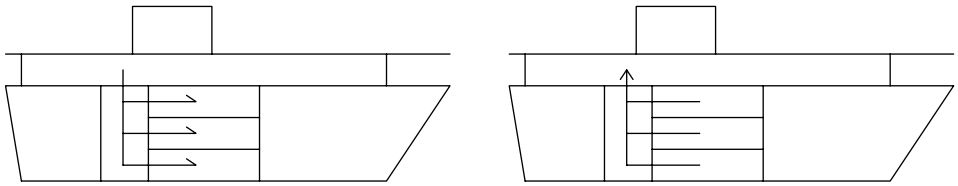


Figure 4.1: Strike-down and strike-up weapons transfer scenarios.

the shortest amount of time. During strike-down operations, it is advantageous to minimize the length of time ordnance remains on the main deck, in a relatively unsecured location. Ordnance on the main deck also inhibits flight operations, which is the primary function of the carrier. In strike-up, it is desirable to bring ordnance to the main deck at a rate that matches the demand of flight operations. If the performance of the weapons elevator were not adequate, it would again inhibit the primary function of the carrier.

4.2.1 Current Configuration

The current configuration of the weapons elevator system can be described in terms of its physical characteristics, its operational method, and the queuing of items for transport.

In an aircraft carrier, there are multiple weapons elevators, each of which serves a number of magazines. Each elevator shaft operates in a single carriage, which always remains within the confines of the shaft and has only a vertical degree of freedom. The shafts are located along transverse watertight bulkheads, which serve to isolate compartments. The elevator shafts, which are also considered watertight, typically provide access to two compartments, one of which is through a transverse watertight bulkhead, as in Figure 4.2. It is therefore possible for a shaft to access multiple compartments at the same level. This approach provides a level of redundancy with respect to the access to a given space and also increases the versatility of transfer operation scheduling.

The doors at the interfaces between the elevators shafts and magazines are also watertight, to completely isolate the shaft from the compartments when the doors are closed. Additionally, there are two hatches located within the elevator shaft to isolate regions within the shaft. The main deck and ballistic hatches are shown in Figure 4.3, along with examples of magazine doors.

The actual operation of the weapons elevator system involves multiple elevators, magazines, and queues that have significant interaction. With variability in the scheduling and order of items, the performance of the system is complicated. However, the basic operational logic of the system can be illustrated by the sequence of events in a strike-down operation in one elevator cycle in a system consisting of a single queue, carriage, and magazine. This sequence is presented in Table 4.1, where the platform is initially located one level below the main deck (2^{nd} level) and all doors and hatches are closed. A detailed description of the sequence

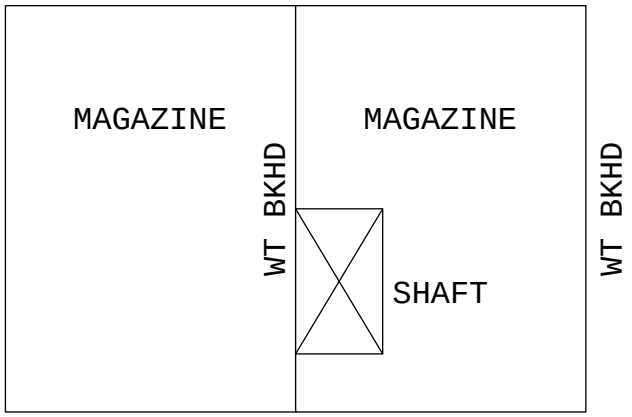


Figure 4.2: Plan view of a magazine layout. WT indicates a watertight partition.

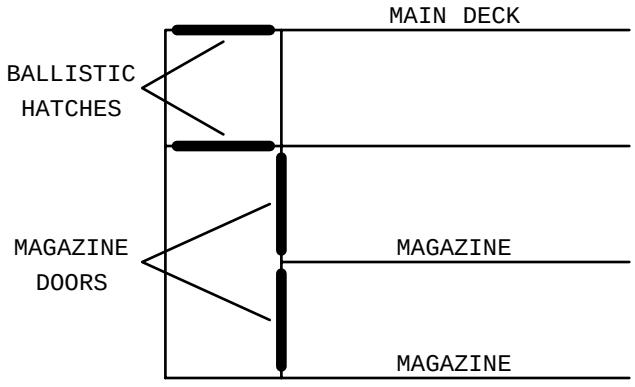


Figure 4.3: Elevation of magazine and shaft layout, showing ballistic hatches and doors.

Table 4.1: Current weapons elevator transfer operation events.

Step	Event
1	Open main deck hatch
2	Move carriage to the main deck
3	Load carriage
4	Move carriage to the 02 level
5	Close main deck hatch
6	Open ballistic hatch
7	Move carriage to the magazine (below ballistic hatch)
8	Open magazine door
9	Unload carriage
10	Close magazine door
11	Move carriage to the 02 level
12	Close ballistic hatch

of events in the model used for simulations, illustrating individual cycle times, is found in Appendix A. This description represents a modification of the baseline scenario to reflect advanced system hardware.

4.2.2 Comparison with Commercial Elevators

The most significant differences between a naval weapons elevator system and a commercial elevator system are with respect to the queues and the ballistic safety features of the weapons elevator system.

The queues in a commercial elevator system are widely recognized as stochastic [5, 55]. The rate at which items in the queue (passengers) enter the system, often described as the passenger inter-arrival time, is typically modeled as a Poisson distribution. The probability of n calls being registered in the time interval T for an average rate of arrivals λ is

$$p_r(n) = \frac{(\lambda T)^n}{n!} e^{-\lambda T} \tag{4.1}$$

Depending on the traffic condition (up-peak, down-peak, inter-floor), the average rate of arrivals will change and will vary from floor to floor. Because of the stochastic nature of the queues in a commercial building, the number of passengers carried during any given trip, depending on the method of control, is also stochastic. The utilization of commercial elevators therefore typically varies significantly during its operation. In a weapons elevator system, more information is known about the queues. Queues are often prearranged in an attempt

to minimize the time associated with bringing items to a waiting carriage. However, due to several reasons, including the number of steps required to arrange the queues, incomplete information about the state of magazines prior to a strike-down, and conflicting shipboard requirements, the queues are often not ideal and can experience variability. The operation of weapons elevators is usually threshold based, and an elevator will not leave for a magazine until a limit on volume or weight is encountered.

Unlike a commercial building, the transportation of items in a weapons elevator system requires resources such as fork trucks and personnel to conduct loading and unloading operations at the queues and in the magazines. Resources normally operate in one magazine at any given time, to keep resource utilization as high as possible and to account for a limited number of resources. A strike-down operation that involves ordnance with various destinations therefore appears like an up-peak traffic condition globally, but locally, the traffic pattern is equivalent to trips where all passengers share a common destination and a destination is visited repeatedly until no passengers remaining in a queue are destined for that destination. If items, or groups of items defining a load, bound for the same destination are not transported successively, then the system is slowed as resources move among various destinations.

A weapons elevator shaft contains physical zones, bounded by the ballistic hatches and ballistic magazine doors. Ballistic doors and hatches are a requirement for the protection of ordnance in transport and the isolation of compartments. To the elevator, the hatches appear as car calls, with deterministic waiting times and doors appear as times associated with unloading at a destination. In a round trip below the lower ballistic hatch, an elevator is guaranteed to experience four virtual car calls, two at each of the ballistic hatches in a shaft, with the current interlock system operated serially.

Chapter 5

Naval Weapons Elevator Simulations

The physical layout and mode of operation currently employed in naval weapons elevators represents one out of an extremely large but finite number of system configurations. In such a large design space corresponding to systems with non-linear dynamics, the process of reductionism, creating generalizations regarding the effects of a single variable, is not applicable. Without these generalizations, there are no predictive shortcuts to indicate the performance or behavior of a given configuration without explicit simulation of that configuration, although there exist some configurations where prediction is trivial such as a system with a single queue, shaft, and magazine. Cases like this represent a small fraction of the typically more complex design space, just as solutions to mathematical and physical problems are solvable under specific conditions, but offer no general closed form solution. As an example, consider the three body gravitational problem.

It must be remembered that the measured performance/behavior of a simulation is not the performance/behavior of the actual system, but of the simulated system only. How well the simulation results reflect reality depends on the accuracy and detail of representation of the system model, which is partially described in the system configurations. While any system configuration must be considered as a whole, it can be defined in terms of its physical characteristics, operational logic, and the system inputs.

5.1 Physical Characteristics

Physical characteristics encompass the arrangement and interactions between fixed physical spaces that correspond to various system functions. The three types of spaces found in a weapons elevator system are queues, shafts[1], and destinations. During the strike-down

[1]The term *shaft* is used to indicate a space rather than the more common term *elevator* to distinguish between a fixed and mobile entity as the term *elevator* is typically synonymous with the terms shaft and carriage. We will introduce configurations where the carriage can exit a shaft, and the distinction between

scenario, where items are transported for storage, queues are areas located on the main deck (hanger deck) and destinations are the magazines (stowage compartments). In a strike-up scenario, where ordnance is removed from storage, the roles of the main deck areas and the magazines are reversed. The simulation models used in this work consider strike-down scenarios only, which are equivalent to the worst-case, up-peak traffic condition.

The definition of the physical configuration describes the interactions between spaces, but does not include any physical metrics. If this information were included, the design space of physical configurations would be infinite and practically unsearchable. Describing the relationships between spaces is a simple, abstract means for expressing physical orientations without explicit definition of geometries or dimensions.

The relationships between spaces are described using incidence matrices. With three types of spaces, there are three distinct incidence matrices. Two of the matrices, the shaft-queue (SQ) matrix relating shafts and queues and the shaft-magazine (SM) matrix relating shafts and magazines, are defined as 0^{th} order incidence matrices, because they describe a direct physical relationship between spaces. The order of an incidence matrix is defined to be the number of indirect paths required to relate a given space to another. The 0^{th} order incidence matrices therefore indicate a direct physical relationship between two spaces. A 1^{st} order incidence matrix describes the indirect relationship between spaces, separated by one commonly shared space. A 1^{st} order incidence matrix of importance is the 1^{st} order queue-magazine matrix (QM) that shows the relationships between queues and magazines through shafts. The 1^{st} order QM matrix is related to the 0^{th} order SQ and SM matrices by the expression in Equation 5.1.

$$(QM) = (SQ)^T(SM) \tag{5.1}$$

0^{th} order incidence matrices consist of only binary values indicating the presence or lack of interactions. However, elements in higher order incidence matrices are not necessarily only binary values. The entries in the 1^{st} order QM matrix indicate not only a relationship between queues and magazines, but also the number of shafts by which the spaces are related. It is not explicitly apparent however, which shafts connect a queue and magazine. The maximum value of any entry in the QM matrix is equal to the number of shafts in the system.

As a simple example of the use of incidence matrices, consider a conventional elevator system with carriages fixed in shafts consisting of 2 shafts, 2 queues, and 3 magazines with the 0^{th} order SQ and SM incidence matrices defined in Figure 5.1. The resulting 1^{st} order QM matrix is therefore calculated in Figure 5.2.

The QM matrix indicates that all magazines can receive items from the 1^{st} queue. For

the fixed space, *shaft*, and the mobile *platform* becomes important. To avoid confusion, the term *elevator* is never used except as a description of a vertical material handling system in general, the term *shaft* denotes a physical location, and the term *carriage* denotes the transport vehicle.

$$(SQ) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \qquad (SM) = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 5.1: In the example 0^{th} order incidence matrices, the first shaft is connected only to the first queue while the second shaft is connected to both queues. The first shaft is connected to all three magazines but the second shaft is connected only to the third magazine.

$$(QM) = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 5.2: The QM matrix is determined from the SQ and SM matrices. The connectivity of the first shaft results in many of the connections between the first queue and all magazines. Since the first shaft is not connected to the second queue and the second shaft is only connected to the third magazine, there is no connection between the second queue and the first and second magazines.

the third magazine, there are two paths from the first queue, the first and second shafts. Furthermore, the matrix shows that the second queue only serves the 3^{rd} magazine and that the 1^{st} and 2^{nd} magazines are only served by the 1^{st} shaft.

A conventional elevator system uses carriages that are fixed within shafts, requiring the bi-directionality of shafts. Bi-directionality implies that incidence matrices are undirected - that is, if space A communicates with space B, it is possible to reach A from B as well as B from A. Less conventional systems might involve uni-directional shafts, which result in directed incidence matrices, where it may be possible to reach A from B, but not B from A. In descriptions of these systems, information regarding the directionality of shafts is included as a shaft direction vector (SDV) and the 0^{th} order directed incidence matrices are determined by row by row scalar multiplication of each element in the shaft direction vector (or 1 minus the element) with the SQ and SM matrices, as shown in Equations 5.2 to 5.5.

$$\overrightarrow{SQ}_{UP_i} = SDV_i \quad \overrightarrow{SQ}_i \qquad \{i | i = 1 \ldots s\} \tag{5.2}$$

$$\overrightarrow{SQ}_{DOWN_i} = (1 - SDV_i) \quad \overrightarrow{SQ}_i \qquad \{i | i = 1 \ldots s\} \tag{5.3}$$

$$\overrightarrow{SM}_{UP_i} = SDV_i \quad \overrightarrow{SM}_i \qquad \{i | i = 1 \ldots s\} \tag{5.4}$$

$$\overrightarrow{SM}_{DOWN_i} = (1 - SDV_i) \quad \overrightarrow{SM}_i \qquad \{i | i = 1 \ldots s\} \tag{5.5}$$

Because of their importance and the fact that no additional 0^{th} order incidence matrices besides the SQ and SM matrices and no additional 1^{st} order incidence matrices other than the QM matrix are used in this work to describe relationships between spaces, the description of the order is dropped in all subsequent references to these matrices and are referred to simply as the SQ, SM, and QM matrices.

The SQ and SM matrices are synthesized and are the only two matrices that are required to define a configuration. Physical configurations are generated systematically based on the number of each type of physical space in a system. The quantities of physical space types are used as a sort of "base" system which is used in the encoding/decoding of configurations, described in further detail later in this chapter. For a system with q queues, s shafts, and m magazines (base q-s-m), there are theoretically 2^{sq} SQ incidence matrices and 2^{sm} SM incidence matrices, for a total of $2^{sq} \cdot 2^{sm} = 2^{sm+sq} = 2^{s(m+q)}$ possible physical configurations. Included in these possible configurations however, are invalid configurations. For instance, every possible system size contains SQ and SM incidence matrices with all 0 entries, which represents a null system. Valid configurations are based on the following rules:

1. All shafts must be associated with at least one queue and magazine (no all zero rows in the SQ or SM incidence matrices)
2. All queues and magazines must communicate with at least one shaft (no all zero columns in the SQ or SM incidence matrices)
3. SM matrices must be in the lowest energy state with respect to shafts and magazines
4. SQ matrices must be in the lowest energy state with respect to queues
5. Any rows in the SQ matrix corresponding to repeated rows in SM must be in the lowest energy state

If a shaft does not communicate with any queue or magazine, it is a 'dead-end' shaft with no source and/or destination for items and is therefore not a meaningful part of the system. Similarly, if a queue or magazine has no access to any shaft, it is an isolated space and therefore not a part of the system.

A corollary to the second rule for valid configurations exists for systems with uni-directional shafts:

2A. All queues and magazines must be associated with at least one up-shaft and one down-shaft

With at least one input and output, no space is isolated in a uni-directional shaft system. This rule implies that, in systems with uni-directional shafts, the total number of shafts must be greater than 1 and that there must be at least one up-shaft and one down-shaft for a valid configuration.

The arbitrary numeration of physical spaces results in the repetition of logically equivalent configurations, assuming that all spaces of each type are identical. As an example, consider the two SM matrices in Equation 5.6.

$$SM_1 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \qquad SM_2 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \tag{5.6}$$

For simplicity, if the corresponding SQ matrices are assumed to have all 1 entries, then SM_1 is equivalent to SM_2 with only an interchange of the two rows. Instead of naming the shaft corresponding to the first row 'shaft 1', it is just as arbitrary to name it 'shaft 2' and perform a row interchange as long as the shafts are identical (and as long as the corresponding rows are interchanged in the SQ matrix). Similarly, an interchange of the first and second columns in SM_1 yields the logical equivalent of SM_2. To avoid the repetition of physical configurations due to arbitrary nomenclature, the SM matrices are required to be in their 'lowest energy state' with respect to shafts and magazines (rows and columns)[2].

The lowest energy state with respect to row (columns) is defined to be the minimum value of the decimal representation of the flattened matrix with the most significant bits corresponding to the first row (column) and least significant bits corresponding to the last row (column) over all permutations of row (column) interchanges. Another way to think of the lowest energy state is through the decimal representations of the bytes formed by the rows of the matrix and its transpose. If the decimal representation of bits comprising each row $\{i|1 \leq i \leq s\}$ of the matrix is denoted by σ_i and the decimal representation of the bit sequences of each row $\{j|1 \leq j \leq m\}$ of the transpose of the matrix is denoted by μ_j, then the elements of the vectors $\vec{\sigma}^* \in \{\sigma_i|1 \leq i \leq s\}$ and $\vec{\mu}^* \in \{\mu_j|1 \leq j \leq m\}$ must be ordered to satisfy the conditions in Equations 5.7 and 5.8 to be in the lowest energy state.

$$\sigma_{i-1} \leq \sigma_i \quad \{i|1 \leq i \leq s\} \tag{5.7}$$

$$\mu_{j-1} \leq \mu_j \quad \{j|1 \leq j \leq m\} \tag{5.8}$$

In the example SM matrices in Equation 5.6, the 'energy' vectors are presented in Equation 5.9.

$$\begin{aligned} \vec{\sigma}_1^* &= \{5, 3\} & \vec{\sigma}_2^* &= \{3, 5\} \\ \vec{\mu}_1^* &= \{2, 1, 3\} & \vec{\mu}_2^* &= \{1, 2, 3\} \end{aligned} \tag{5.9}$$

For a given valid SM matrix, the names of shafts are no longer arbitrary as the positions are fixed in their lowest energy states. The SQ matrix does not therefore have to be in a lowest energy state with respect to shafts (except for identical shafts). However, the identities of queues remains arbitrary and the SQ matrix must be in its lowest energy state only with respect to queues to be a valid matrix. A valid SQ matrix satisfies the requirement in

[2]The SM matrix is used by convention. It would be equivalent to reverse the roles of the SQ and SM matrices for configuration identification

Equation 5.10, where ν_k is the decimal equivalent of the byte formed by each row, $\{k|1 \leq k \leq q\}$ in the transpose of SQ.

$$\nu_{k-1} \leq \nu_k \qquad \{k|1 \leq k \leq q\} \tag{5.10}$$

When rows in the SM matrix are identical (two or more shafts serve identical magazines), the shaft numbering is arbitrary for the identical shafts within the SM matrix, but not necessarily in the SQ matrix (unless the shafts that serve identical magazines also serve identical queues). To distinguish between valid and repeated configurations, the sub-matrix consisting of the rows in SQ corresponding to identical rows in SM must be in their lowest energy state with respect to shafts. In the example configuration in Figure 5.3, the second and third shafts are identical with respect to magazines and the SQ matrices are equivalent with respect to a single row interchange. The sub-matrices sq_1 and sq_2 are formed from the second and third rows in SQ_1 and SQ_2 respectively and are shown in Figure 5.4. The sq_1 sub-matrix is in its lowest energy form and therefore indicates that SQ_1 represents the valid configuration of the logically equivalent SQ_1 and SQ_2 matrices.

$$SM_1 = SM_2 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

$$SQ_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad SQ_2 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

Figure 5.3: Example configurations 1 and 2 are equivalent with respect to connectivity when location names are arbitrary. Since the two rows corresponding to the 2^{nd} and 3^{rd} shafts are identical, only a row interchange distinguishes the SQ matrices of the configurations. To avoid repetitions, incidence matrices are required to exist in their lowest energy state.

$$sq_1 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad sq_2 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

Figure 5.4: The lowest energy states of the sub-matrices of SQ matrices consisting of rows corresponding to shafts with identical connectivity with respect to magazines determine which configuration is a repeat.

The number of valid configurations with respect to system base size is shown in Table 5.1. The number of valid configurations is typically a small fraction of the total number of possible configurations. However, this fraction does not represent the number of configurations that are logically correct, but are simply repeats of valid configurations.

While all valid configurations are logically valid, they are not necessarily physically realizable. For example, in a conventional shaft, a carriage may theoretically serve four magazines (one

Table 5.1: The number of valid configurations and percentages of total possible configurations for various system sizes. The last three system sizes are not used in simulations and are only intended to indicate the change in the potential configuration with increases in the number of components.

System Base (q-s-m)	Possible Configs	Valid Configs	Valid Fraction
1-1-1	4	1	0.25
1-1-2	8	1	0.125
1-1-3	16	1	0.0625
1-2-1	16	1	0.0625
1-2-2	64	3	0.0469
1-2-3	256	6	0.0234
1-3-1	64	1	0.0156
1-3-2	512	6	0.0117
1-3-3	4096	24	0.0059
2-1-1	8	1	0.125
2-1-2	16	1	0.0625
2-1-3	32	1	0.0313
2-2-1	64	3	0.0469
2-2-2	256	11	0.0430
2-2-3	1024	23	0.0225
2-3-1	512	6	0.0117
2-3-2	4096	57	0.0139
2-3-3	32768	270	0.0082
3-1-1	16	1	0.0625
3-1-2	32	1	0.0313
3-1-3	64	1	0.0156
3-2-1	256	6	0.0234
3-2-2	1024	22	0.0215
3-2-3	4096	46	0.0112
3-3-1	4096	24	0.0059
3-3-2	32768	241	0.0074
3-3-3	262144	1165	0.0044
4-4-4	$\approx 4.3 \cdot 10^9$	?	?
5-5-5	$\approx 1.13 \cdot 10^{15}$	?	?
6-6-6	$\approx 4.72 \cdot 10^{21}$	?	?

magazine per shaft face). A configuration with a shaft connected to five magazines on one level is therefore unrealizable. These systems remain valid because the physical assumptions are ignored as the constraints are self-imposed and physically unrealizable systems might be constructed when common assumptions are disregarded and unconventional configurations are employed. In the previous example, the constraint regarding one door per shaft face assumes a rectangular shaft. We might equally consider a circular shaft that is capable of serving as many magazines per level as doors can be physically constructed around the shaft.

5.2 Operational Logic

The operational logic of the elevator system is the set of 'rules' that are recursively applied to evolve a system from its initial state to a final state. In terms of discrete event simulation, the operational logic determines the evolution of states.

Cellular automata provide a simple example for illustrating the application of rule sets in discrete event evolutions. Elevator system evolutions can be thought of as inhomogeneous cellular automata, where cells are not identical and have different transition functions and connectivities. A given cell in the elevator system is dependent only on systems states (often including itself) that are relevant to it.

In the homogeneous cellular automata presented earlier, a complete set of all possible transition functions is created by computing all combinations of inputs and outputs, resulting in k^{k^n} rule sets. The rules for elevator systems can also be created in this manner, as long as the size of the system, *i.e.* the number of states required to accurately define the system, is known. In an inhomogeneous system in which system states reflect physical realities, searching through all possible rule sets with various connectivities, irrespective of it being a practical impossibility, would result in a large number of meaningless rules. For example, the state of a door is represented explicitly by four bits describing whether it is opening, open, closing, or closed. There is only one rule set using this representation that reflects reality[3], shown in Figure 5.5.

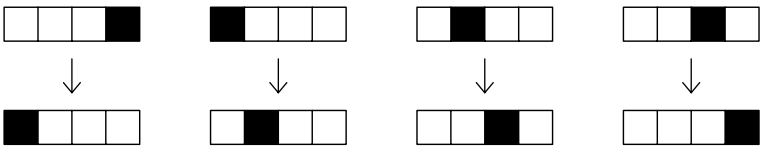


Figure 5.5: The only meaningful states and evolutions of the four bits representing door states. The bits indicate whether the door is opening, open, closing, or closed.

To say a door evolves to an opening state from an opening state or that a door is closing and opening simultaneously is an invalid physical representation. Out of the $(2^{16})^2 \approx 4.3$ *billion*

[3]This rule set excludes the case of a door closing/opening midway through the opening/closing state

possible rule sets, only one, or $2.3 \cdot 10^{-8}\%$ of these provide a meaningful representation of reality. The use of unconstrained rule sets would require a search for the meaningful results, but with the difficulty of defining what constitutes a meaningful result, the search for valid rule sets becomes impossible. Rather than using unconstrained rule sets, specific rule sets are built up rather than decomposed such that the evolution of states reflects meaningful physical changes.

This approach is describable in terms of cellular automata evolutions. Cellular automata use local rules, where a given state is connected to a typically small fraction of the total number of states. If the evolutions of entire system states are specified, k^N rules would be required to define a *single* rule set. To specify all rule sets would require $(k^N)^{k^N}$ evolution definitions, which equates to a large amount of information for even a moderately number of cells, N. This approach is the programming equivalent of specifying an 'if-then' statement involving all system states for all combinations of system states: "if bit 1=0 and bit 2=1 and bit 3=1 ... and bit N=1, then bit 1=1 and bit 2=1 and bit 3=0 ... and bit N=1". Of course, this is not the technique applied in typical conditional programming, which is reducible to local rules where variables are dependent on a typically small fraction of system states. This is analogous to homogeneous cellular automata, which use local rules with local connectivity. Inhomogeneous cellular automata like the one used in the discrete evolutions of the elevator system only differ in that their connectivity is distributed and varies for different system states.

There are two categories of decision logic that are used in the elevator system simulations. General logic is common to all elevator systems and represents the actual state changes in an evolution. An example of this type of logic is requiring that a carriage go up if its destination is above its current position and it is free to do so without collision. Another example is requiring a hatch to go from an opening to open state when its cycle is complete.

Specific operational logic is not explicitly visible in a system evolution. It controls the timing of general logic or, in other words, permits the application of certain sets of general logic. Specific logic is analogous to the control policies of commercial elevator systems. Specific operational logic is not common to all configurations and therefore defines the operational behavior of the system. The two primary operational logic parameters that relate to the operation of carriages and hatches are the serial/parallel operation of carriages and doors/hatches, and interlock logic.

Both the general and variations in specific operational logic originate from the baseline strike-down operation cycle of the existing naval weapons elevator system, which is presented in Appendix A. Although the baseline case represents an over-simplification of the actual operation of weapons elevator system, it not only serves as a basis for operational logic, but also in part as a means for validation of the simulation code, which is presented in Appendix B for this simple case consisting of a single queue, shaft, and magazine with serial carriage and hatch/door operations and the use of interlocks. Appendix B also presents validation of the model using more complex variations.

In order to bring a weapons elevator carriage from one level to another, several steps are required involving the movement of the carriage in concert with the logical sequencing of hatches and doors. In a serial system, these events are performed sequentially, with the completion of one event serving as the trigger for the next. As a result, no carriage can move in a shaft while a door or hatch in its 'line of sight' is opening or closing. Conversely, no door or hatch associated with a shaft can open or close while the carriage is moving in that shaft. The expression line of sight as used here describes the region within a shaft bounded by all closed doors and hatches. If the lower ballistic hatch is open, opening, or closing, the line of sight includes both upper and lower shafts, regardless of the position of a carriage. If the lower ballistic hatch is closed, the line of sight is the upper shaft if the carriage is in the upper shaft and is the lower shaft if the carriage is in the lower shaft. It is possible to define line of sight to include magazines in addition to shafts, which has the effect of additional safety, but more restricted operations. In a parallel system, carriage movement and door/hatch cycling can occur simultaneously, but the timing of the operations for both serial and parallel logic is dependent on the use of interlocks.

Interlocks are intended to isolate sections of the elevator system for damage control. Isolation is accomplished through the proper sequencing of hatches and doors. Without interlocks, the extent of isolation changes, depending on the timing of events and the cycling rate of doors/hatches.

The combinations of operational logic variables, if considered binary, create distinctly different modes of operation and can result in different performance and behavior for a system with identical architecture. The sequencing of doors/hatches and the use of interlocks are two of many specific operational logic parameters that define a configuration and affect performance and behavior. Examples of other parameters include what determines end conditions of the transfer operation, how items are rejected from the system if they do not belong, how resources are allocated, if carriages are fixed within a shaft or are mobile, and if local optimization in decision making is employed. In this work, many of these logical variables are fixed and we concentrate on the effects of serial/parallel carriages/hatches and interlocks. Resources are modeled in this work as a limit on the number of carriages served at any space at any given time, but the problem of allocating mobile resources to targeted spaces is not addressed. All spaces are assumed to have non-zero resources capable of performing the functions associated with the spaces and resource allocation is not a factor in determining carriage destinations and item selection[4].

The issue of fixed versus mobile carriages is closely related to, but not synonymous with bi-directional and uni-directional shafts as there can be mobile carriages in a system with bi-directional shafts, a case not considered in this work. The logic required for uni-directional shaft systems is more complicated than for bi-directional shafts mainly because the decisions required for item and path selection are not relevant in systems with bi-directional shafts.

The specific control logic combinations that are considered in this work account for shaft

[4]These assumptions are more realistic when automation is considered

directionality, serial/parallel shaft operations, and the use of interlocks, resulting in eight logic combinations, listed below.

1. Bi-directional, serial shafts, interlocks
2. Bi-directional, serial shafts, no interlocks
3. Bi-directional, parallel shafts, interlocks
4. Bi-directional, parallel shafts, no interlocks
5. Uni-directional, serial shafts, interlocks
6. Uni-directional, serial shafts, no interlocks
7. Uni-directional, parallel shafts, interlocks
8. Uni-directional, parallel shafts, no interlocks

5.3 Input Streams

The input stream in a weapons elevator system is defined to be all of the elements to be transported by the system. These elements can be considered as individual items, or a load of items with a common destination. The four major characteristics defining an input stream are the number, type, and order of items in the queues and the arrival rate of items throughout an evolution. In this work, queues are assumed to contain all items at the start of an evolution and the inter-arrival rate of items is zero.

The number of items involved in a typical input stream in an actual strike-down scenario is fairly large. As a result, the possible permutations of items within a single queue is also quite large, following $n!$ for n distinct items. Even for a small number of items, the number of permutations of distinct items can be too large to consider practically. With a dozen distinct items, there are already nearly 500 million possible input streams for one queue. The addition of queues increases the number of possible input streams as the items are distributed between queues and the orders of these distributions are permuted.

When two queues are considered, the first queue holds v_1 items, ranging from 0 to all n items, which can be arranged in $v_1!$ ways. For v_1 distinct items, there are $P^n_{v_1}$ distinct sets, so that the number of distinct item sets and arrangements, R_1, for v_1 items in the first queue is described by Equation 5.11

$$R_1 = P^n_{v_1} v_1! \tag{5.11}$$

The second queue holds v_2 items, which is the difference between the total number of items and those held in the first queue ($v_2 = n - v_1$). There is only one subset of these items, $P_{v_2}^{n-v_1} = P_{v_2}^{v_2} = 1$, and the total number of possible orders for the second queue, denoted by R_2, is given in Equation 5.12.

$$R_2 = v_2! \tag{5.12}$$

The total number of possible input streams, R_T, is the combination of all variations of the first and second queues, described by the product of Equations 5.11 and 5.12 as in Equation 5.13.

$$R_T = P_{v_1}^{n} v_1! v_2! = P_{v_1}^{n} v_1!(n - v_1)! \tag{5.13}$$

For more than two queues, a given queue k with the exception of the 'last' queue, holds from 0 to $n - \sum_{j=1}^{k-1} v_j$ items, of which there are $P_{v_j}^{n-\sum_{j=1}^{k-1} v_j}$ distinct subsets. The number of distinct arrangements for $q > 2$ is presented in Equation 5.14.

$$\begin{aligned} R_T = \sum_{v_1=0}^{n} P_{v_1}^{n} v_1! \Bigg(\sum_{v_1=0}^{n-v_1} P_{v_2}^{n-v_1} v_2! \Bigg(\sum_{v_3=0}^{n-v_1-v_2} P_{v_3=0}^{n-v_1-v_2} v_3! \cdots \\ \Bigg(\sum_{v_k}^{n-\sum_{j=1}^{k-1} v_k} P_{v_k}^{n-\sum_{j=1}^{k-1} v_k} v_k! \cdots \\ \Bigg(\sum_{v_{q-2}}^{n-\sum_{j=1}^{q-3} v_j} P_{v_{q-2}}^{n-\sum_{j=1}^{q-3} v_j} v_{q-2}! \\ \Bigg(\sum_{v_{q-1}}^{n-\sum_{j=1}^{q-2} v_j} P_{v_{q-1}}^{n-\sum_{j=1}^{q-2} v_j} v_{q-1}! \left(n - \sum_{k=1}^{q-1} v_k \right)! \Bigg) \Bigg) \cdots \Bigg) \cdots \Bigg) \Bigg) \end{aligned} \tag{5.14}$$

The apparent size of the possible arrangements of input streams is somewhat misleading. While there exist a large variety of types of ordnance stored on an aircraft carrier, each item is ordinarily not distinct as multiple items of the same type are typically found. The order of these identical items is therefore not important, resulting in a significant reduction of the total number of possible input streams. Additionally, ordnance of the same type is commonly stored together and is often placed in the same queue. Compatibility between certain types of ordnance is another factor that results in grouping of identical items in queues. These constraints result in the ability to make generalized statements about the contents or arrangements of queues, reducing the total number of possible arrangements of items in queues.

The arrangement of items within the queues during a strike-down operation is often influenced by the initial state of the magazines, which are rarely empty or loaded identically. While the state of the magazines can impose constraints on the initial contents of the queues, mainly because of compatability constraints, it also increases the total number of initial conditions of the system, depending on the number and type of items stored, the compatability rules, and the number and types of items in the queues.

The ordering of items assumes only one item is available at a time and items are treated serially. This assumption significantly increases the design space of the input streams and is associated with the factorial terms in Equation 5.14. The factorial terms disappear when the queues are assumed to be parallel - that a carriage has access to all items in a queue. The assumption of parallel queues is not unrealistic if a queue is composed of a limited number of item types. A carriage may not be able to access all items simultaneously, but as long as it can access one item of each type, the queue is effectively parallel. In the parallel queue, items/destinations are selected by the carriage based on some decision logic intended, but not guaranteed, to minimize total operation time. The decision logic assumed in this work is based on the availability of destinations and the current state of the queues and magazines. For configurations with mobile carriages, shaft availability is also considered.

The order that items are selected in an evolution using a parallel queue represents only one case out of the large input stream design space. However, it does define a near upper bound on performance for the given configuration. How near the resulting performance is to the actual upper bound is dependent on the decision logic (heuristic) used, but is arguably better than establishing a bound based on a small random selection of input streams with a serial queue and is significantly lest costly than fully characterizing performance with one set of items in one configuration based on testing all possible input streams, which is a practical impossibility in most cases.

The validation of a heuristic applied in a parallel queue corresponding to near optimal performance for a set of items is based on determining the effects of perturbations of the ideal queue in systems using serial queues. The ideal queue is defined as the order in which items are selected in a parallel queue. If items are ordered as in the ideal queue in a system using serial queues, the resulting system performance is identical to the performance of a system with a parallel queue with the same set of items. The concept of the ideal queue therefore makes it possible to emulate a parallel queue with its associated heuristic in a serial queue for a given set of items. This concept also yields performance corresponding to the heuristic which is arguably better than random selection of one of the $n!$ item arrangements. By perturbing the ideal queue at various percentages of random re-orderings, the sensitivity of performance to item order can be determined in simulations using serial queues with identical item sets. This analysis also provides an indication of the robustness of the heuristic.

The parallel queue concept that eliminates the factorial term in Equation 5.14 does not affect the combination terms and their product, which alone result in a large set of input streams. The combination terms arise from having distinct queues which can hold all varieties

of numbers and types of items from a common set. If all queues are assumed identical with respect to the number and type of items, the design space of input streams decreases significantly. Using this assumption, input streams are generated based on the relative ratios of item types. The values of the ratios are dependent on the number of item types, which is equal to the number of destinations. Ratios are systematically varied in 20% increments with the constraint that the percentages of all item types must sum to 100%. The total number of items in any queue for all configurations is assumed constant and the actual number of items of any type is the nearest integer value to the product of the corresponding percentage and the total number of items per queue.

For a discontinuous performance landscape where a single change in the make-up of the input stream has the potential to significantly affect performance, it would be necessary to test all item ratios to fully characterize a configuration's performance with a given set of items and identical queues. It is assumed however, that the limited set of ratios created with 20% increments is capable of defining the general bounds of performance/behavior for a given configuration with a practical consideration to computability. Additionally, evidence in cellular automata systems suggests that behavior is often inherent for a rule set and can be characterized by a limited number of evolutions [2]. In systems with greater numbers of magazines, the set of possible distributions of item types increases as does the set of valid physical configurations. It is the product of these two sets that determines the total configuration design space and the limitations of the problem exploration.

With a fixed number of items allowed in a queue, it is possible that different queues in the same configuration initially contain different numbers of items and that different configurations have a different total number of items in their input streams. These differences are attributed to the incomplete connectivity between queues and magazines in some systems and the item rejection logic. If an item bound for a certain destination is located in a queue with no access to that destination, it is considered non-system inventory and is rejected. This rejection occurs in any physical configuration with any non-zero QM incidence matrix entry. It is possible to consider logic to relocate these items to appropriate queues, keeping the total number of items in all configurations with the same number of magazines identical. However, redistribution ignores the physical limits of queue capacity and the rejection of items indicates a lower potential performance.

5.4 Summary of Assumptions

The common motives for the number of assumptions used in the simulation model are to simplify the search space while offering complete characterization of the ranges of behavior and performance. These assumptions occur with respect to physical attributes, operational logic, and input streams.

In the description of physical arrangements, only the connectivity between spaces is consid-

ered. Actual distances between spaces are not specified and the vertical distance between each queue and magazine is identical, reducing the design space to a tractable size. While this simplification means the model may not accurately reflect the performance and behavior of the actual naval weapons elevator system, we must recall that our goal is not to evaluate the performance of the actual system towards its optimization. Our objective is the identification of the causes of the relationships between behavior and performance and robustness, which is facilitated by stripping configurations to their essential architectures. Cycle times are fixed and deterministic for the same reasons. While making the model more accurate increases the accuracy of its predictions of the real system, variability may veil the fundamental causes of useful correlations.

The operational logic used in the simulation model represents a small set out of a large design space intended to reflect the basic operation of the actual current weapons elevator system, recognizing that carriages, not a central controller (the ordnance handling officer), dictate the operation and behavior of the system. The differences that result from automatic, distributed control may result in faults in system operation - we will see later that evolutions can "freeze" and that carriages can be "tricked" into entering empty queues. However, our objective is not to design or search for the optimal rules or configuration. We might therefore consider our approach as not using the naval weapons elevator for characterization of relationships, but only as the basis for creating a system we can use for searching for correlations between behavior and performance or robustness. That is, we need not use the weapons elevator system specifically for this task. It is simply a means to an end and any similar system could also serve as the basis for our analysis.

In another attempt to limit the design space, in this case from an infinite space, we assume a set of queues based on predetermined ratios of item types. Additionally, all queues contain identical item distributions at the start of a given simulation. In an actual system, we would not expect identical item distributions in queues, but rather distributions suited to the specific connectivities of individual queues. However, we are interested not in finding rules for distributing items that result in efficient system operation, but complete characterization of how the elevator systems described respond in variable environments, such as a range of item distributions, in order to relate behavior and performance to robustness or adaptability. By varying item ratios systematically however, each configuration is bound to experience queue distributions that both are and are not suited to its specific connectivity, which could be thought of as varying the level of perturbation in input streams from ideal sets. Identical queues of systematically varied item distributions are therefore intended to offer complete characterizations of the range of behavior and performance while limiting the set of required simulations to tractable levels.

5.5 Encoding

With a large configuration design space, an encoding process is used to systematically define and identify alternatives. Encoding enables the expression of a configuration in the compact form of a decimal number rather than a cumbersome, explicit description. This encoding process is analogous to the encoding of evolution rules in cellular automata, described previously.

A configuration code is based on the system size, or base value, described in Section 5.1. The values in the base description represent, in order, the number of queues, number of shafts, and number of magazines. A 3-1-2 base system for example, therefore consists of 3 queues, 1 shaft, and 2 magazines. The total number of bits, b, required to describe the physical configuration in a particular base is equal to the sum of the product of the number of queues and shafts and the product of the number of magazines and shafts, which is equal to the total number of entries in the SQ and SM incidence matrices. This relationship is expressed in Equation 5.15. The maximum decimal value for b bits is $2^b - 1$, the maximum number of distinct configurations corresponding to a given base.

$$b = s \cdot q + s \cdot m = s(q + m) \tag{5.15}$$

Valid configurations are determined by systematically creating and testing all configurations corresponding to codes 0 to $2^b - 1$. Configurations are constructed from a decimal value by first converting the decimal value to a binary digit sequence, buffered if necessary with zeros in higher significant bit locations to create a sequence b digits long. As an example, consider code 119 for base 3-2-1. The total number of bits for this base is $2(3 + 1) = 8$. The binary equivalent of code 119 with 8 bits is 01110111. To derive the SQ and SM matrices, the digit sequence is partitioned into two bytes, with lengths equal to the number of entries in the SQ and SM matrices. The first byte is $s \cdot q$ bits long and is comprised of the most significant bits of the total binary sequence. The second byte, $s \cdot m$ bits long, consists of the least significant bits. These bytes are themselves partitioned into s equal length bits. This equates to s, q-length bytes from the first byte and s, m-length bytes from the second byte. In the example system, the partitioning process is illustrated in Figure 5.6.

$$\underset{\text{(a)}}{01110111 \rightarrow (011101)(11)} \quad \underset{\text{(b)}}{011101 \rightarrow (011)(101)} \quad \underset{\text{(c)}}{11 \rightarrow (1)(1)}$$

Figure 5.6: The partitioning of the binary digit sequence representing code 119 into SQ and SM bytes (a). These bytes are in turn partitioned to form bytes corresponding to rows in the SQ matrix, (b), and SM matrix, (c).

The SQ and SM matrices are built up from these partitioned bytes. In each case, the first byte corresponds to the first row in the matrix with subsequent bytes corresponding to following

rows. The assembly of the SQ and SM matrices for the example system is illustrated in Figure 5.7.

$$(011)(101) \rightarrow \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} = SQ$$

$$(1)(1) \rightarrow \begin{pmatrix} 1 \\ 1 \end{pmatrix} = SM$$

Figure 5.7: The assembly of the SQ and SM matrices from the partitioned bytes.

The encoding process is simply the reverse of the decoding process. In the encoding process however, the system base need not be explicitly stated since the system size is implicitly known from the dimensions of the incidence matrices. For all coded configuration descriptions, the code number and the base system are provided along with a description of the operational logic used in a binary format, letting serial shaft operations equal 0, parallel shaft operations equal 1, the lack of interlocks equal 0, the use of interlocks equal 1. The distribution of item types may also be included if a description of the input stream is required. A full description of the example system might therefore appear as in Figure 5.8, where the 0-1 indicates parallel carriage and door/hatch operations and the use of interlocks.

119 3-2-1 0-1 (100)

Figure 5.8: The coded description of system 119, consisting of 3 queues, 2 shafts, and a single magazine. The system has parallel shaft operations and uses interlocks. 100% of the items in the queue are bound for the single magazine.

5.6 Visualization Techniques

The evolution of a weapons elevator configuration with a given input stream reveals both the performance (in terms of metrics such as total cycle time, throughput, and elevator utilization) and the behavior of the system. For our temporally-based model, throughput is defined as the ratio of the number of items transported at the end of an evolution to the total length (time) of the evolution, where cycle times are defined in Appendix A. Utilization of a single carriage is defined as the fraction of the evolution length that the carriage is moving while transporting in a loaded condition. While performance measures are easily expressed as numbers, a numerical representation does not necessarily fully characterize the behavior of the evolution and provide a means for involving subjective human perceptive abilities. Since many of the evolutions are not simply repetitive, they have relatively low compression and conceptually high algorithmic complexity so that a full representation of the evolution requires explicit description of all system states. For evolutions with sufficiently

long evolutions, explicit state representations, especially in terms of verbal descriptions, is a practical impossibility, requiring the use of more compact visual representations.

There are several visualization techniques that can be used to present evolutions. We primarily will use evolution histories, with respect to both temporal and logical evolution lengths, of both system states and the states of individual carriages. However, for completeness, we also discuss cellular evolutions and state evolution trajectories. These techniques are not used extensively, although they have unique characteristics that may be advantageous for particular conditions.

5.6.1 Cellular Representations

Cellular evolution representations are analogous to the cellular automata evolutions presented earlier. Like cellular automata evolutions, elevator system attributes are represented by colored cells. Unlike cellular automata, the elevator system is inhomogeneous, with cells representing distinct system attributes. A row in a cellular representation represents the system state at a given instant and a column of cells presents the history of a single system attribute, with time advancing down the page. The width of the evolution is proportional to the number of attributes required to completely describe the system and increases with larger system sizes. For any single evolution, the width can also vary, depending on the level of detail desired in the system description. As stated in Section 5.2, if all system attributes are included, no two system states will ever be identical and no strictly defined pattern will ever emerge. For consistency, all cellular representations are composed of only the attributes listed in Table 5.2 along with a description of their possible states.

Many of the system attributes are not mutually exclusive and therefore require declaration of all states. For example, if a hatch is not open, it does not imply that the hatch is necessarily closed. The hatch could be opening, closing, or closed. This explains why several apparently 'opposite' states are listed in Table 5.2.

All of the system states are binary variables and are represented by a colored cell (black=true, white=false). For carriage direction, a black cell indicates up and a black cell indicates down. Destination types are coded so that a queue equals a white cell and a magazine equals a black cell. Carriage locations are described by one of h states, where h indicates the maximum number of levels in the carrier that encompass all queues and magazines. Colors indicating levels are scaled so that a white cell represents a carriage located at the main deck and a black cell indicates a carriage at the lowest possible level. Without fully describing the positions of attributes, the configuration, or the input stream, the cellular representation provides an indication of the behavior of the evolution. Repeated patterns in the cellular representation provide evidence of simplicity while the inability to shorten the description of the evolution indicates complex or chaotic behavior.

While the cellular representations represent a powerful means for visualizing an evolution

Table 5.2: System attributes included in cellular representation visualization techniques.

Attribute	Possible States
Carriage location	integer level
Carriage movement	true/false
Destination type	queue/magazine
Direction	up/down
Carriage loading	true/false
Carriage unloading	true/false
Upper ballistic hatch opening	true/false
Upper ballistic hatch open	true/false
Upper ballistic hatch closing	true/false
Upper ballistic hatch closed	true/false
Lower ballistic hatch opening	true/false
Lower ballistic hatch open	true/false
Lower ballistic hatch closing	true/false
Lower ballistic hatch closed	true/false
Magazine door opening	true/false
Magazine door open	true/false
Magazine door closing	true/false
Magazine door hatch closed	true/false

and its behavior, it is not particularly compact and is physically cumbersome for longer evolutions, even for compressed cellular representations. Evolution trajectories address this space issue while still illustrating behavior patterns (or lack thereof).

5.6.2 Evolution Trajectories

With b binary bits defining the state of an elevator system evolution, there are 2^b decimal values that define all possible system states. Conversion of the evolution of binary states to encoded decimal form yields a one-dimensional (temporal) vector describing the trajectory of system states. This trajectory is visualized by constructing a list of coordinates of the form (s_{t-1}, s_t) for $(t|1 \leq t \leq t_f)$ where s_t is the decimal representation of the state of the system at time step t and $s_{t=0}$ is the initial state of the system. An example trajectory plot is presented in Figure 5.9(a) for an evolution corresponding to a 1-1-1 base conventional system for 100 time steps. Since there is only one item type, one source of items, and one carriage, the evolution is repetitive and will follow the same evolution trajectory regardless of the number of items or length of the evolution.

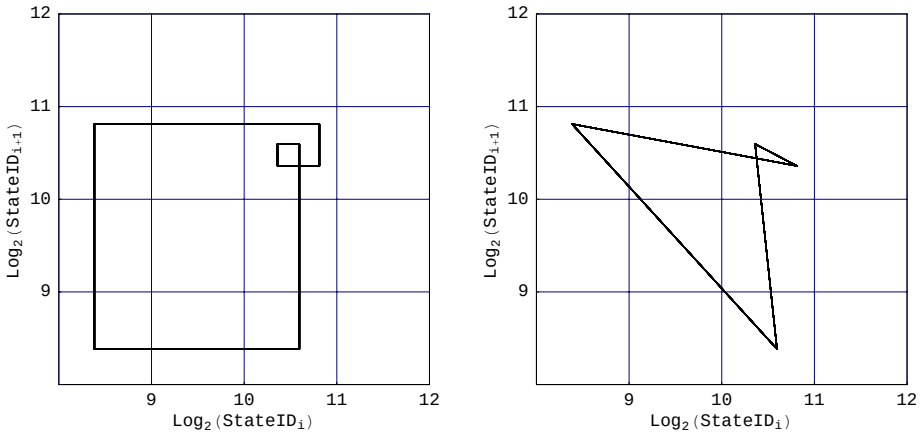


Figure 5.9: Example evolution trajectories for the simple 1-1-1 size system. There are only four states entered by the single carriage. For the evolution trajectory using the temporal evolution in (a), repeated states are encountered, so all edges are parallel. Each step in the logical trajectory in (b) corresponds to a state change.

The edges of the trajectory are all parallel in this evolution, (except from the initial state) because it corresponds to the states of the full evolution which may contain many state repetitions ($s_{t-1} = s_t$). These repetitions are attributable to the duration of timers. The removal of these additional nodes corresponds to the trajectory of the compressed evolution, which shows the logical rather than temporal progression of states and requires that at least one system attribute changes with every evolution step such that $s_{t-1} \neq s_t$. The compressed evolution trajectory of the same system evolved in Figure 5.9(a) is shown in Figure 5.9(b).

5.6.3 Evolution Histories

Evolution trajectories represent an extremely compact graphical representation. All information and patterns are presented simultaneously in one space, which is particularly advantageous for long evolutions with many patterns. However, the price of compactness is the loss of the temporal or logical sequence of states. To retain the information associated with the sequencing of states, evolution histories are used. These histories are similar to cellular evolutions, except states are represented as encoded values like those used for trajectories rather than graphic equivalents of binary values. Because the evolutions considered in this work are sufficiently short and the number of states is sufficiently small, we are able to use evolution histories and retain the sequence and timing of state changes, which is useful for determining the reasons for pattern changes and the complexity of patterns.

As with evolution trajectories, evolution histories can be presented with respect to both the temporal evolution length and the logical evolution length. The compressed evolution history, which uses the logical evolution length, removes repeated runs of identical states, illustrating only the logical sequence of states. The full and compressed evolution histories are both used, based on which form is suitable to the complexity measure being discussed. For example, the logical evolution history is typically used in analyses of evolutions with respect to logical complexity, which is a normalization of the number of unique states by the logical evolution length.

Evolution histories can be presented in terms of both system states and individual carriage states. When using system states, global, system level patterns are apparent. However, because of the representation of a state with an encoded value, carriages that correspond to bits with more significance in the binary state description tend to dominate a pattern and pattern changes in less significant carriages are difficult to visualize. A solution to this inequality is to present the individual carriage histories, either independently or superposed, so that each corresponding bit in each bit stream describing each carriage has the same significance. While global patterns may be more difficult to identify, the patterns of individual carriages are apparent.

The number of possible states that define valid history or trajectory points is dependent on the system size as well as the arbitrary level of detail used to describe the system. The finest detail considers all system states, including timers and item locations and no states are ever identical. As the detail becomes coarser, not all information is present and repeated states are possible. The differences in detail affect the qualitative complexity of the evolution trajectories and the level of detail must be selected to correspond to the scale of interest.

In this work, the level of detail is fairly coarse and only attributes relevant to defining carriage states are considered. In conventional systems, the carriage attributes used to define a carriage state are presented in Table 5.3 along with the number of bits required to describe each attribute.

These attributes are used to describe a carriage in five possible carriage states, listed in

Table 5.3: The carriage attributes used to define evolution states in conventional elevator systems

Attribute	Number of Bits
Global destination type	1
Global destination number	$Max(Ceiling(log_2 q), Ceiling(log_2 m))$
Current location type	2 (0=queue, 1=magazine, 2=shaft)
Current location number	$Max(Ceiling(log_2 q), Ceiling(log_2 m))$
Loading	1
Unloading	1
Up direction	1
Down direction	1
Magazine reservation number	$Ceiling(log_2 m)$
Queue reservation number	$Ceiling(log_2 q)$
Carriage unsure of queue	1

Table 5.4.

Table 5.4: The five carriage states used in conventional elevator systems.

State Number	State
1	Carriage loading
2	Moving down to destination magazine
3	Carriage unloading
4	Moving up with a known destination
5	Moving up without a known destination

In conventional elevator systems, shaft selection by carriages is not a factor in the item selection or destination selection process as carriages are fixed in shafts and carriage and shaft designations are synonymous. In the selection and reservation of queues that occurs after the unloading process, carriages may have multiple options. If all candidate queues are reserved or busy, a carriage remains undecided regarding a destination queue. Rather than remain idle, the carriage moves to the main deck while continually searching for an available queue as the same shaft is always used regardless of the queue selection. This state is regarded as sufficiently distinct from moving up with a queue reserved as it affects the validity of the combinations of carriage states. State 5 is therefore required in addition to state 4.

Each carriage state is defined by a unique combination of some carriage attributes in Table 5.4. Combinations of the possible values of carriage attributes determines the possible

variations of each state. Combinations of these state variations in turn define system states, the validity of which is based upon the physical limitations and constraints imposed on the system. The attribute combinations defining each of the five states are presented in Figure 5.10.

	CurrentLocationType	CurrentLocationNo	GlobalDestType	GlobalDestNo	Loading	Unloading	UpDirection	DownDirection	MagReservationNo	QueueReservationNo	CarriageUnsureOfQueue
State 1	1	xx	0	0	xx	1	0	1	0	0	0
State 2	1	xx	1	0	0	0	0	0	1	xx	0
State 3	1	xx	0	1	xx	0	1	0	0	0	0
State 4	0	xx	1	0	0	0	0	0	1	0	0
State 5	0	xx	1	0	0	0	0	0	0	0	1

Figure 5.10: The five possible state descriptors and bit locations for a conventional system. The x's indicate variable bits that can be filled with binary values corresponding to possible attribute values. The length of x's are not fixed and depend on the system size. Possible state variations are found by finding valid combinations of possible bytes indicates by x's.

The x's indicate variable bits that specify the global destination numbers, current location numbers, magazine reservation numbers, and queue reservation numbers and provide more detail to the state of the carriage and determine the validity of carriage state combinations. The values of the variable bytes depend on the system size.

The combinations of variable bits within a single carriage state and the combinations of several carriage states defining a system state are tested to verify their physical and logical validity. The number of valid rules is dependent on the system size and physical limitations of the system, such as the maximum occupancy limits and resource availabilities.

The technique for visualizing the evolution histories in systems with mobile carriages and uni-directional shafts (virtual conveyor systems) is identical to the technique used for conventional systems. However, the differences between the systems result in differences in the attributes used to define carriage states and in the number of carriage states.

In virtual conveyor systems, a carriage must wait to select an available queue before moving up from a magazine since the selection of a queue depends on the selection of an appropriate shaft. The 'CarriageUnsureOfQueue' bit in conventional systems seen in Table 5.3 is therefore not required in virtual conveyor systems.

Rather than using explicit definition of directions, direction is implied based on the type of shaft (up/down) the carriage is in, which is associated with the load. The virtual conveyor

systems assume that a carriage travels in a down-shaft while loaded and up in a shaft when unloaded. Virtual conveyors therefore use a bit describing if the carriage is loaded and eliminate the direction bits used in conventional system state descriptions. The resulting attributes used to describe carriage states in virtual conveyors are listed in Table 5.5 along with the number of bits required to describe each attribute.

Table 5.5: The carriage attributes used to define evolution states in virtual conveyor elevator systems.

Attribute	Number of Bits
Global destination type	1
Global destination number	$Max(Ceiling(log_2 q), Ceiling(log_2 m))$
Current location type	2 (0=queue, 1=magazine, 2=upper shaft, 3=lower shaft)
Current location number	$Max(Ceiling(log_2 q), Ceiling(log_2 m))$
Loading	1
Unloading	1
Magazine reservation number	$Ceiling(log_2 m)$
Queue reservation number	$Ceiling(log_2 q)$
Carriage loaded	1

With mobile carriages, it is possible to have multiple carriages in a single shaft, magazine, or queue. To account for the physical constraints associated with having multiple carriages in the same location, virtual conveyors are divided into zones, each with a maximum occupancy limit. The zones in a virtual conveyor are listed along with a description of their boundaries or the state of the carriages within them in Table 5.6.

The use of zones results in an increase in the level of detail of state representations and the number of possible carriage states. In a virtual conveyor system, there are eleven carriage states, described in Table 5.7.

The qualitative nature of an evolution's behavior is captured from an evolution history plot based on the system patterns, or the combined patterns of indivdual carriages. In the simplest evolutions, the same states are visited and a single repetitive pattern is present. As the complexity of the evolutions increases, the number of distinct patterns increases and the system shifts between multiple 'steady-states'. Highly complex or chaotic trajectories lose the cyclic patterns and most points correspond to distinct system states. The number of states visited is larger and the ratio of states visited to the total number of possible states is larger than a simple evolution in a system with an equivalent number of valid states.

The qualitative descriptions of evolution histories are loosely related to the concept of algorithmic complexity. In a simple evolution, it is possible to describe the patterns that emerge in a compressed form, while for the most complex trajectories, a description of the pattern is

Table 5.6: Zone definitions for virtual conveyors. Zones are defined by physical boundaries or the state of carriages within them.

Queue	Contains carriage loading or waiting to load.
Queue to Shaft	All carriages are loaded. Bounded by the upper ballistic hatch of the destination shaft.
Upper Shaft	From the upper to lower ballistic hatch.
Lower Shaft	From the lower ballistic hatch to all magazine doors in that lower shaft.
Shaft to Magazine	Loaded carriages moving from the magazine door connected to the down shaft to the magazine unloading area.
Magazine	Carriage are unloading or waiting to unload.
Magazine to Shaft	From the magazine unloading area to the selected up shaft door. All carriages are unloaded and have selected their queue and shaft.

Table 5.7: The eleven possible carriage states used in a virtual conveyor elevator system.

State Number	State
1	Carriage loading
2	Moving from queue to shaft
3	Moving down in upper shaft
4	Moving down in lower shaft
5	Moving to magazine (includes entering magazine from shaft)
6	Unloading
7	Finished unloading and unsure of destination
8	moving from magazine to shaft with known destination queue
9	Moving up in lower shaft
10	Moving up in upper shaft
11	Moving to queue (includes entering queue from shaft)

incompressible and explicit representation is required to describe the pattern without losing information. Of course, the evolution history plots are intended as a visualization technique only and rely on human perceptive abilities. Their interpretation is therefore subjective and provide no quantitative measure of complexity by themselves. However, the same tools used in their construction can be used to create objective, quantitative measures of the complexity of the evolutions.

Chapter 6

Measures of Complexity for Weapons Elevator Simulations

The problems with complexity measures discussed in Chapter 3 are that they are often abstract and difficult, if not impossible, to implement. For instance, the concept of hierarchical complexity has no direct analog in elevator systems, with distributed control.

Algorithmic complexity has a great potential for application towards quantifying behavior in elevator systems. The algorithm governing the operation of the elevator system, the simulation code, is not the measure of the complexity as there is no means for proving the simulation code is the shortest form and, more importantly, the complexities of all simulations are equivalent if the same code is used as the basis for measure. However, algorithmic complexity is applicable to the evolutions, and provides the basis for several measures used to describe the compressibility of evolutions and algorithmic complexity is used throughout analyses of these measures as an indication of their validity.

Like the complexity definitions in Chapter 3, both static and dynamic measures are used for the elevator systems. Static measures are made *a priori* to any evolution and, lacking information on the actual evolution, describe the potential complexity of the system. The measures used to quantify the complexity of elevator systems that are potential measures are those describing the total number of possible valid states the system can enter, the physical connectivity, and the logical connectivity of the system. Dynamic measures are based on explicit evolutions and describe not potential, but "as-is" complexity. Because they are based on actual evolutions, dynamic measures are intrinsically more accurate, but the accuracy comes with a cost. In this case, the cost is the time/computational cost associated with evolving candidate systems. As described in Chapter 5, the complete design space of elevator systems is quite large and is cumbersome even for the limited space considered in this work. The use of static versus dynamic measures and relative accuracy is therefore of great significance. The static and dynamic measures used in this work are presented in Table 6.1.

6.1 Total Number of Possible States

The counting of the total number of possible states as a measure of complexity is based on the counting of states as an estimate of complexity used by Bar-Yam [3] and discussed in Chapter 3. When used as a static measure, all potential system states are counted, rather than the number of states a system exists in during an evolution.

Valid states are determined and enumerated following the same approach described in Section 5.6.2 for establishing the coordinates of evolution trajectories for visualization of qualitative evolution behavior. As with evolution trajectories, the total number of possible states is dependent on the scale of the system representation and the desired level of detail. As mentioned in Section 5.6.2, the level of detail used is relatively coarse, with only 5 distinct carriage states for conventional systems and 11 distinct carriage states for virtual conveyors. To describe each carriage state, approximately one dozen carriage attributes are required per carriage.

Although the resolution of states affects the absolute number of valid states, it must be remembered that the measures of complexity are being used in the context of their relationships with system performance. Therefore, the level of detail used is not overly critical as the relative complexity, rather than the absolute complexity is of importance and, as long as the level of detail is constant, comparison is possible.

The use of the total number of possible states follows an intuitive notion of complexity. With more physical spaces (queues, shafts, and magazines), the number of possible states increases. With a greater number of valid states, the evolution has a greater potential for exploration, affecting the corresponding dynamic measures of complexity. The addition of carriages has a more significant impact on the number of states than the number of physical spaces, as the number of possible systems states is related to the possible individual carriage states as in Equation 6.1, where I is the number of possible states for an individual carriage and n is the number of carriages. The actual number of system states is always less than or equal to the number of combinations of carriage states because of the invalidity of some combinations.

Table 6.1: Static and dynamic measures of complexity specific to elevator systems

Static	Dynamic
Total Number of Possible States	Number of Unique States Used
Average Physical Connectivity	Fraction of States Used
Fraction of Potential Connectivity	Logical Complexity
Average Logical Connectivity	State Complexity
	Compressed State Complexity

$$(SQ) = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \qquad (SQ) = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

Figure 6.1: An example of two completely different architectures with equivalent average physical connectivity. Since the numbers of connections are identical, both systems have the same average connectivity, but the distribution of those connections are significantly different.

$$Total\ Valid\ States \leq I^n \tag{6.1}$$

6.2 Average Physical Connectivity

The average physical connectivity is loosely associated with the λ parameter, described in another format with connectivity by Kauffman [30]. Kauffman used the λ parameter as a measure analogous to the thermodynamic properties used to describe the state of a gas, like temperature or pressure. Kauffman concludes that, for systems with low connectivity with one or two connections, behavior can only be trivial or simple. For systems with complete connectivity, behavior is almost always chaotic except for the simplest of initial conditions. In between, complex behavior is found with the proper tuning of the λ parameter and the system connectivity.

Average physical connectivity is also related to the complexity of simplices, described in Section 3.4. The connectivity does not describe the existence and interrelations between sub-networks, but does provide an indication of system architecture.

Physical connectivity is calculated from the SM and SQ incidence matrices and measures the accessibility between spaces. Because different spaces typically access different numbers of spaces, the most descriptive single measure of the system connectivity is the average connectivity of all spaces. The average does not give a complete description of the accessibility of the entire system, and completely different architectures may have identical average physical connectivities, as for the example systems in Figure 6.1. However, it does provide an indication of how spaces are connected and the potential behavior of a system.

The expression used to calculate average physical connectivity is presented in Equation 6.2. The total number of physical connections considers queues, magazines and shafts. The connection between a queue and shaft, for instance, indicates a connection for the shaft and a connection for the queue. This apparent double counting of connections requires the total number of incidences in SQ and SM be increased by a factor of two. However, the total number of connections is averaged over the total number of spaces, $s + q + m$.

$$Average\ Physical\ Connectivity = \frac{2\left(\sum_{i=1}^{s}\sum_{j=1}^{q} SQ_{ij} + \sum_{i=1}^{s}\sum_{k=1}^{m} SM_{ik}\right)}{s+q+m} = \frac{2\sum_{i=1}^{s}\left(\sum_{j=1}^{q} SQ_{ij} + \sum_{k=1}^{m} SM_{ik}\right)}{s+q+m} \tag{6.2}$$

Average physical connectivity is an absolute measure and is not normalized to account for system size. Increases in system size therefore enable greater potential average connectivity, which follows intuition. Access to a greater number of spaces results in a greater number of destination options, which results in a greater number of valid system states. However, because the physical connectivity is a potential measure, larger connectivity and valid state spaces do not necessarily equate to greater complexity and the actual states visited in an evolution are ultimately dependent on the dynamics of the system.

6.3 Fraction of Potential Physical Connectivity

The fraction of potential connectivity represents the normalization of the average physical connectivity with respect to the maximum average physical connectivity for a system with the same number of physical spaces. The expression for the maximum average physical connectivity is given in Equation 6.3. The expression for the fraction of physical connectivity is the ratio of the average physical connectivity to the maximum average physical connectivity and is shown in Equation 6.4. Since both the actual and maximum average physical connectivities are averaged over the same number of queues, shafts, and magazines, the fraction of potential connectivity is equivalent to the ratio of the total number of connections in the system to the maximum number of connections in the system.

$$Maximum\ Average\ Physical\ Connectivity = \frac{2\left(sq+sm\right)}{s+q+m} \tag{6.3}$$

$$Fraction\ of\ Potential\ Connectivity = \frac{\sum_{i=1}^{s}\left(\sum_{j=1}^{q} SQ_{ij} + \sum_{k=1}^{m} SM_{ik}\right)}{sq+sm} \tag{6.4}$$

The normalization of the average physical connectivity runs counter to intuition regarding larger systems' capability for supporting greater complexity. However, normalization is useful for comparisons of potential complexity in identically sized systems. If a system size is fixed, as it might be in an aircraft carrier with little to no flexibility for physical rearrangements, the fraction of potential physical connectivity indicates how associations may affect behavior.

6.4 Average Logical Connectivity

In a cellular automata network, physical connectivity is evident in the structure of the network, where each cell is associated with $2r + 1$ neighbors. The connectivity can also be thought of in terms of the evolution rules, which describe the logical connectivity of the system - how many bits are required to evolve a given bit representing a system attribute. In homogeneous cellular automata, where system "attributes" are identical and follow the same evolution rules, the physical and logical connectivities are synonymous and are defined by the system neighborhoods. However, in inhomogeneous systems like the weapons elevator, logical connectivity is distinct from physical connectivity.

Logical connectivity describes the amount of information comprising an evolution rule associated with a system attribute. A simple example of a description of logical connectivity is the neighborhood definition in cellular automata. In the elementary cellular automata described in Section 2.2, each cell bases its evolution on a neighborhood consisting of $2r + 1$ cells. Each cell requires $2r + 1$ bits of information describing the states of the cells in its neighborhood to fully determine its evolved state. Elementary cellular automata represent a somewhat trivial example of logical connectivity, as all cells are identical and share the same logical connectivity. In an inhomogeneous system, like the elevator system, where the system attributes can be thought of as analogous to the cells of a cellular automata network, the amount of information required to evolve system attributes is variable and connections to a single attribute may be distributed over a wide range of the attribute "network", not restricted to a local neighborhood.

Logical connectivity is determined by searching through the simulation code and identifying the number of variables that are involved in the decision logic for each attribute. For example, consider the *Mathematica* code given in Figure 6.2 from a conventional system that checks whether upper ballistic hatches should open.

Three conditions must be met for an upper ballistic hatch to start opening when the operational logic specifies serial shaft operations with the use of interlocks (OLV[[SERIAL]]=1 AND OLV[[INTERLOCKS]]=1). A hatch will start opening if the carriage within the shaft has requested an open hatch, the carriage is stopped, and the magazine doors above the lower ballistic hatch as well as the lower ballistic hatch itself are closed. To determine if the carriage is stopped, one bit each in the 'CARRIAGEUPMOVEMENTVECTOR' and 'CARRIAGEDOWNMOVEMENTVECTOR' are queried. One bit corresponding to the shaft in question is required to determine if the lower ballistic hatch is closed and the states of m magazine doors are examined to identify whether the doors of those magazines that access the upper shaft are closed. Another single bit from the 'CARRIAGEWANTINGUPPERBALLISTICHATCHOPENVECTOR' is required to indicate whether an request to open the hatch has been made. The required information is therefore $4 + m$ bits/attributes to determine whether an upper ballistic hatch in a single shaft begins to open. When the upper ballistic hatch begins to open, four attributes are affected: three bits describing the state of the upper ballistic hatch and

```
CheckForUpperBallisticHatchToStartOpening:=
    Scan[
        Module[{carriageID},carriageID=#
            If[(IsUpperBallisticHatchClosed[carriageID]∪
              IsUpperBallisticHatchClosing[carriageID])∩
              Switch[OLV[[serial]],
                  0,True,
                  1,¬IsCarriageMoving[carriageID]]∩
              Switch[OLV[[interlocks]],
                  0,True,
                  1,(IsEveryMagazineInUpperShaftClosed[carriageID]∩
                    IsLowerBallisticHatchClosed[carriageID])],
              UpperBallisticHatchOpeningVector[[#]]=1;
              UpperBallisticHatchClosedVector[[#]]=0;
              UpperBallisticHatchClosingVector[[#]]=0;
              UpperBallisticHatchOpeningTimer[[#]]=
                  TimeToOpenUpperBallisticHatch-
                  UpperBallisticHatchClosingTimer[[#]];]]&,
        Union[Complement[
            CarriageWantingUpperBallisticHatchOpenVector,{{}}]]]
```

Figure 6.2: Example code for checking when to open an upper ballistic hatch to illustrate logical connectivity. The attributes that are changed in a section of code are logically connected to the attributes in the conditional statements.

the timer indicating the cycle time of the upper ballistic hatch. Each of these attributes is therefore connected to $4 + m$ other system attributes through this piece of evolution logic. The total connectivity of any attribute is the sum of all individual connectivities across evolutionary logic algorithms, so the $4 + m$ connectivity for each of the affected attributes in the example code represents a component of the total logical connectivity.

Since different evolution logic may be used depending on the control logic, the values of logical connectivity may vary. In the example code, affected attributes are logically connected to a single attribute when the control logic specifies parallel shaft operation without interlocks. Serial shaft operations and interlocks require additional logic and increase the logical connectivity of the code. The expressions for the logical connectivities for conventional and virtual conveyors for the various control logics are presented in Table 6.2, where l is the number of items carried per carriage load.

In addition to the operational logic, the values of the logical connectivities are dependent on the system size. This dependence illustrates the fact that more entities affect the evolution of an attribute in larger systems, increasing the complexity of the system towards chaos.

Table 6.2: The average logical connectivity for tested operational logics. The logical connectivity is dependent on system size as well as the operational logic employed.

Operational Logic	Average Logical Connectivity
Conventional, Serial, Interlocks	$\frac{335+59n+l+68m+10q}{42}$
Conventional, Serial, No Interlocks	$\frac{329+59n+l+38m+10q}{42}$
Conventional, Parallel, Interlocks	$\frac{296+59n+l+51m+10q}{42}$
Conventional, Parallel, No Interlocks	$\frac{290+59n+l+22m+10q}{42}$
VirtualConveyor, Serial, Interlocks	$\frac{507+346n+53m+6q+50s}{58}$
VirtualConveyor, Serial, No Interlocks	$\frac{491+331n+20m+6q+44s}{58}$
VirtualConveyor, Parallel, Interlocks	$\frac{502+228n+41m+6q+50s}{58}$
VirtualConveyor, Parallel, No Interlocks	$\frac{476+211n+8m+6q+44s}{58}$

It also illustrates that the information comprising the inputs to evolution logic is typically distributed across a network and is not restricted to a local neighborhood as with homogeneous cellular automata. The expressions in Table 6.2 indicate that the logical connectivity is constant for a fixed system size and control logic, regardless of any physical or structural differences among the systems of that size. In this respect, the average logical connectivity is analogous to the measure of the maximum average physical connectivity of a system, measuring the potential complexity of a system size. Unlike physical complexity, it is difficult to define configuration-specific logical connectivity as all referenced values are used, regardless of their value. A dynamic measure of logical connectivity is possible that would vary for systems of the same size with different architectures. This measure would require the counting of all states referenced per changed attribute, averaged over all evolution steps (similar to identifying the number of actual neurons firing to trigger a memory in models of the brain), which is costly in large simulations and is not considered in this work.

The expressions and values of average logical connectivity are also dependent on the evolution logic and number of attributes considered. The evolution logic is not absolute - there are many ways to code the same simulation. Included in this variability are the attributes selected to define a system. For example, if resources are considered as relevant to accurately describing a system, the logical connectivity may be considerably different than in an equivalent model where resources are ignored. Logical connectivity remains applicable despite the differences that might arise from model scope and instantiation because the code for all conventional configurations and the code for all virtual conveyor configurations remain constant, allowing comparison between configurations using the same code.

As for average physical connectivity, it is intuitive that greater values of logical connectivity correspond to greater potential complexity. Work with connectivity has demonstrated that sparse connectivity often corresponds to simple behavior only [30]. Beyond a low threshold of connectivity, it is possible to obtain complex behaviors. The elementary cellular automata

from Section 2.2 demonstrate this phenomena. However, at high values of connectivity, near and including complete connectivity, chaos is often encountered. Above this transition region of connectivity, the use of connectivity as a measure of complexity is counterintuitive as greater values often correspond to the low complexity associated with chaos. For this reason, connectivity is used as a potential measure only.

6.5 Number of Unique States Used

The number of states visited in an evolution is the dynamic equivalent to the static measure of the total number of possible states comprising an evolution state space. The number of visited states is also based on the complexity estimation technique of Bar-Yam, but, since it is based on the actual evolution, is a more accurate measure of the complexity than a potential measure [3].

Like the total number of possible valid states, the measure of the number of visited states is dependent on the scale of the definition of the attributes used to define system states. For a given evolution, a coarser resolution results in fewer system states visited as similar states are indistinguishable without additional information. As the resolution becomes finer, more possible states become distinct and a greater number of unique states are found in an evolution. For the finest resolution, where all relevant system attributes are included, all evolution states are distinct and the number of visited states is equal to the number of evolution steps. Due to the subjectivity associated with the definition of system states, the number of visited states is not an absolute measure. Its applicability as a relative measure is possible within a fixed scale of state representation.

A greater number of visited states is an indication of greater complexity. In the simplest of evolutions, the system cycles through a limited number of states in sequence. Simple repetition is not limited to smaller systems with one carriage but can also occur in systems with multiple carriages that essentially act as independent subsystems with synchronized simple cycles. Complexity increases in repetitive cycles as synchronization breaks down. If each carriage effectively remains an independent subsystem, but the state cycles of different carriages have different repetition periods, the system as a whole visits unique states until the time step corresponding to the least common multiple of all carriage repetition periods. As an example, consider a system with two carriages. The first carriage cycles through three states (A, B, and C) and the second carriage cycles through three different states (D, E, and F). The system states, formed by joining carriage states together then follows the progression in Figure 6.3(a), where a state change is assumed to occur at each evolution step for each carriage.

If the number of states in the repetition cycle of the second carriage is changed to four (D, E, F, and G), the collective period of repetition changes as illustrated in Figure 6.3(b) and the evolution visits twelve distinct states rather than four. Despite the insignificant

Time step	Carriage States
1	AD
2	BE
3	CF
4	AD
5	BE
6	CF
7	AD
8	BE
9	CF
10	AD
11	BE
12	CF
13	AD
14	⋮

(a)

Time step	Carriage States
1	AD
2	BE
3	CF
4	AG
5	BD
6	CE
7	AF
8	BG
9	CD
10	AE
11	BF
12	CG
13	AD
14	⋮

(b)

Figure 6.3: Two scenarios for simple cycles involving essentially independent carriages are possible. In (a), the repetition periods for both carriages are identical and the carriages are synchronized. In (b), each carriage cycles through a sequence of states, but the carriage sequences are not synchronized, resulting in a repetition period equal to the least common multiple of the carriage repetition periods. Despite the simplicity of independent repetitions, unsynchronized cycles result in greater measured complexity when counting the number of states used

difference between synchronized and unsynchronized independent carriage evolutions, the complexity as measured by the number of distinct systems states visited is significantly higher for an unsynchronized evolution. However, this increase in measured complexity does not necessarily make the number of states an invalid measure. By an interpretation of algorithmic complexity, the amount of information required to describe the unsynchronized cycles is greater than that for the synchronized cycles. This extra information takes the form of a description of the phase shift, an explicit description of the two cycles, or a description of the complete, higher order repetition pattern of the entire system. Whether the difference between the number of unique states for synchronized and unsynchronized simple evolutions is equal to the additional information required to describe an unsynchronized evolution is subject to interpretation. If an unsynchronized combination of independent carriage cycles is expressed as a description of the entire higher order repetition pattern formed using all system attributes, the measure of complexity in terms of the number of states is indistinguishable from an evolution involving an equivalent number of unique states, but resulting from the interactions between carriages. The additional information required to express an unsynchronized combination of independent carriage states lies between the difference between the complexities of an evolution resulting from carriage interactions and an evolution with synchronized, independent carriage cycles, implying that the number of states can be an overestimation of complexity in some cases.

6.6 Fraction of States Used

The measure of the fraction of states used is truly a mixture of dynamic and static complexity measures. However, since explicit simulation is required, it is defined as a dynamic measure. The expression for the fraction of states used is presented in Equation 6.5.

$$Fraction\ of\ States\ Used = \frac{Number\ of\ Unique\ States\ Used}{Total\ Number\ of\ Possible\ States} \tag{6.5}$$

The number of valid system states is a measure of the potential complexity - the greater the number of valid states, the greater number of evolution trajectories and the greater the potential complexity. A large potential space does not however guarantee higher complexity of the evolution, only the possibility of higher complexity. A comparison of the actual space explored to the potential space indicates how the actual evolution "lives up" to its potential. The comparison also validates the use of the total number of possible states as a potential measure of complexity.

Normalizing the number of unique states visited in an evolution with respect to potential states (which is essentially a normalization with respect to system size because of the relationship between system size and the number of valid states) allows comparison of complexities between systems of different sizes. A smaller system that explores a larger fraction of its

possible space may be arguably considered more complex than a larger system that visits a larger absolute number of states but a lower fraction of its potential space. Similarly, a small and large system could both visit the same number of states, having the same complexity in the context of the number of unique states visited. However, with a greater potential for complexity, the larger system does not explore a proportionate amount of its potential space.

The validity of this argument and this measure depends on the scalability of actual states visited. If the system definitions result in different expansion rates for unique states visited and potential system states, comparison across different size systems is invalid, although for systems of the same size, comparisons are valid, being nothing more than a scaled comparison of the number of unique states visited.

The validity and usefulness of this measure is also dependent on the subjective definitions of validity of states and the composition of the attributes used to define a state, much like the quantification of the absolute number of states and potential states. If the number of unique states visited for an evolution remains constant when the validity conditions are changed to result in more potential states, the fraction of unique states visited implies the same evolution has different complexity in the context of different definitions of valid states. But, as for the static and dynamic complexity measures involving states, consistency is essential for comparing evolutions and the rules for determining the validity of states are constant for all configurations.

6.7 Logical Complexity/Logical Compression

The logical complexity is a normalized complexity measure enabling comparison of evolutions of systems of different sizes. Unlike the measure of the fraction of states used or the fraction of physical connectivity, normalization in this case does not necessarily violate intuition regarding complexity. The logical complexity, Cx_L, describes the amount of information required to express the logical sequence of states and is defined in Equation 6.6.

$$Cx_L = \frac{Logical\ Evolution\ Length}{Temporal\ Evolution\ Length} \tag{6.6}$$

Related to the logical complexity is the logical compression, which describes the amount of redundant information in an evolution with respect to the logical sequence of states. The logical compression, Cm_L, is defined in Equation 6.7.

$$Cm_L = 1 - \frac{Logical\ Evolution\ Length}{Temporal\ Evolution\ Length} = 1 - Cx_L \tag{6.7}$$

The logical complexity is similar to the number of unique states used. However, the sequencing of states, not just their presence in an evolution, is relevant. The logical evolution length

is defined as the length of an evolution logically equivalent to an evolution history but with at least one system attribute change per evolution step. The logical evolution therefore does not change the sequencing of states, but removes the temporal dependence. A logical evolution by definition is always less than or equal to the equivalent temporal evolution length and the logic complexity and compression therefore range from 0 to 1.

Comparison of logical evolution lengths as absolute measures is not valid, as evolutions may have different numbers of items to transport and the steps required are no indication of complexity. A simple system with a long, repetitive evolution is not more complex than a complex system with a short, non-repetitive evolution. Similarly, a comparison of the evolution created by a system operating on a queue with n identical items and the evolution of the same system with $n + 1$ identical items shows that no significant difference exists between the evolutions, despite the additional logical steps resulting from the larger queue. Normalization with respect to the evolution length addresses this discrepancy and provides a relative means to apply the evolution length to the quantification of complexity.

The amount of logical compression is dependent on the cycle times of elevator operations. For instance, an increase in the hatch cycle times in a simple system will increase the total number of evolution steps, but the number of logical state changes remains constant, effectively increasing the amount of logical compression and decreasing the logical complexity, despite no change in actual complexity. In more complex systems however, the changes in cycle times do not affect overall evolution length or logical sequences linearly. That is, changes in cycle times may lead the evolution on a completely different trajectory. In addition, different systems may respond differently to changes in cycle times. To avoid the non-linear effects of changing cycle times, they are assumed fixed for all configurations. It should be noted that, when the cycle times are reduced to equate to a single time step, the logical evolution is equal to the temporal evolution and no compression is possible as a state change occurs at every step in the temporal evolution. For this reason, the time steps are set to a low value (1 second) and cycle times for elevator operations are all greater than 1 second, approximating their values in practice. The amount of compression is also dependent on the attributes required for state definitions and the scale of system analysis. A fine level of detail results in state changes at all temporal evolution steps with no redundancies, resulting in minimal compression.

Logical complexity/compression is based fundamentally on the measure of algorithmic complexity. Algorithmic complexity is greatest when the 'algorithm' is incompressible - there is no shorter program for representing an output. An evolution with repeated states can be compressed by substituting the explicit state representations with a description of the number of repetitions, resulting in significantly less required information. The information reduction is greatest as the information required to describe a state and the number of repetitions increases. The information compression for a single repeated state follows the expression in Equation 6.8, for a state composed of b bits with p repetitions.

$$1 - \frac{b+p}{bp} \tag{6.8}$$

For states composed of a large number of attributes, the additional information required to describe the number of repetitions in the compressed form becomes negligible, and Equation 6.8 reduces to Equation 6.9, which essentially states that any information additional to the declaration of the first state is redundant.

$$1 - \frac{1}{b} \tag{6.9}$$

An evolution with no state change at every temporal evolution time step is incompressible and no shortcuts exist to describe the evolution - the shortest description of the evolution is the evolution itself.

The use of measures based on algorithmic complexity suffer from the same limitations of algorithmic complexity - primarily that the maximum algorithmic complexity corresponds to chaotic behavior, not necessarily complex behavior. In this respect, the logical complexity is applicable as a filter of simpler behaviors, but can not necessarily distinguish between complex and chaotic behaviors.

Another inherent drawback of using logical complexity/compression as a complexity measure is related to the sequential nature of the evolutions. The logically compressed form of the temporal evolution filters out runs of repeated states, but does not capture runs of repeated *sets* of states. That is, the compression identifies local, "first-order" repetitions, but does not identify global, higher order patterns.

(a) 111222333111222333
(b) 456456456456456456

Figure 6.4: Despite the identical form of their logically compressed states, indicating a higher order repeating pattern, sequences (a) and (b) have different logical complexities. The first order pattern in sequence (a) is identified, resulting in a logical complexity of $1 - \frac{6}{18} = \frac{2}{3}$, but does not account for the second order pattern in (b), giving a complexity of 1.

The compression of the sequences in Figure 6.4 are therefore not equivalent, despite the obvious similarity between their global patterns. To account for the compression possible from identification of global patterns, it is necessary to ignore sequential patterns and utilize the number of unique states visited. This technique is done in the state complexity and compressed state complexity measures.

6.8 State Complexity/State Compression

State complexity is another application of algorithmic complexity. However, unlike logical complexity, the state complexity accounts for global patterns as well as local patterns. Rather than identify the number of partitions of identical states, the state complexity identifies the number of unique states in an evolution, indicating the information necessary to represent both runs of identical states and possible global sequences. The number of unique states is normalized with respect to the temporal evolution length, as in Equation 6.10, where the state complexity is denoted by Cx_S.

$$Cx_S = \frac{Number\ of\ Unique\ States\ Used}{Temporal\ Evolution\ Length} \tag{6.10}$$

The state compression, Cm_S, describes the amount of redundant information in a temporal evolution when the local and global patterns are taken into account, and is related to the state complexity as in Equation 6.11. Following the definition of algorithmic complexity, maximum complexity occurs when there is no redundant information and each state in the temporal evolution is unique. Compression is minimal ($Cx_S = 1,\ Cm_S = 0$) and the evolution is, by definition, random. Minimal complexity occurs in the trivial case of an evolution composed of a single repeated state ($Cx_S \approx 0,\ Cm_S \approx 1$).

$$Cm_S = 1 - \frac{Number\ of\ Unique\ States\ Used}{Temporal\ Evolution\ Length} = 1 - Cx_S \tag{6.11}$$

In strictly repetitive evolutions, the global repetitions normally missed by logical compression are accounted for by identifying the *sets* of repeated states. The information required to express the evolution completely is reduced to the bits required to express the set of repeated states and the number of repetitions, which is assumed negligible when compared to the product of unique states comprising a set and the number of bits defining each state. For a given evolution length, as the number of states comprising a set defining a sequence grows, or multiple repetitive sets emerge, the amount of compression decreases, corresponding to greater complexity.

In a truly deterministic evolution of states, state complexity will account for global patterns. In a repetitive cycle of a set of states, any one of the states leads to a unique state within the set. Following this logical progression, only "second-order" repetitions are possible - that is, no higher order global patterns, like nesting, are possible. Evolutions of strictly deterministic systems will therefore only consist of a single repetitive sequence or no repetition at all[1]. While the elevator rules are deterministic, the representations of the evolutions are not

[1]This includes the possibility that the evolution involves a repeating sequence, but the sequence period is greater than the length of the evolution. If the evolution were given sufficient time, the sequence would repeat.

necessarily deterministic, because not all system information is included in the composition of system states. For example, all of the information which affects a carriage decision to select a queue is not included in the attributes defining a system state. The inventories of candidate queues and the number of carriages that can be accommodated in a given queue are two examples of excluded information. It is therefore possible for a single system state to evolve to multiple states, depending on the logical connectivities of the evolvable attributes. In these "non-deterministic" evolutions, patterns are not limited to second-order repetitions, and more complex patterns may emerge, including multiple repetition sequences. A single state may therefore be a member of multiple sets of repeating sequences. The extent of the non-determinism is dependent on the subjective scale of the state definitions, which remains constant for all configurations. The effectiveness of the state complexity in identifying global patterns towards quantifying algorithmic complexity is therefore diminished as it underestimates the amount of information required to represent an evolution. For instance, if the same state occurs in two distinct sets of repeating sequences, additional information is required to indicate that the state occurs in both sequences, which is neglected in a counting of unique states.

The number of unique states visited is normalized by the length of the temporal evolution to obtain a complexity measure corresponding to the *actual* evolution. For a simple repeating sequence consisting of a set of states, additional repetitions increase the temporal evolution length, but do not increase the number of unique states visited, since the same set of states is always used in each additional repetition. An evolution with a greater number of repetitions of the same sequence will therefore have lower state complexity, since it is more compressible, despite having nearly equivalent algorithmic complexity. The difference in algorithmic complexity between two evolutions with different numbers of repetitions of the same sequence of states is only the number of bits required to describe the additional number of repetitions, which is negligible when compared to the bits required to express the set of unique states.

The discrepancy is most apparent when the number of unique states comprising a repetitive cycle is on the same order of magnitude of the evolution length, as the ratio of the number of unique states to the evolution length is closer to unity. The rate of change of the state complexity with respect to increases in the evolution length, keeping the set of unique states visited constant, is described in Equation 6.12, where the evolution length is denoted by T and the number of unique states is denoted by U. The magnitude of the rate of change of the complexity is greatest for low values of T relative to U. Since, by definition, $U \leq T$, the maximum rate of change occurs when U is on the same order of magnitude as T.

$$\left| \frac{\partial \, Cx_S}{\partial T} \right| = \frac{U}{T^2} \tag{6.12}$$

In practice, the number of unique states in a repetitive set is low in comparison to the evolution length, attributable to the length of elevator operation cycle times and the amount of items in the queues. Elevator operation cycle times are typically much greater than one second. For a given repetitive sequence, greater cycle times effectively increase the temporal

evolution length with respect to the number of unique states. The number of repetitions is dependent on the number of items in the queues of a given type. For sufficient numbers of items, the number of states defining a repetitive sequence is low compared to the number of sequence repetitions - a ratio on the order of the inverse of the number of items of a type. The combined effects of realistic operation cycle times and numbers of items in the queue result in negligible effects on the complexity due to additional repetitions as the change in the state complexity is proportional to a constant number of unique states and the inverse of the *square* of the length of the temporal evolution.

The dependence of the state complexity on the elevator operation cycle times also implies that the state complexity is affected by changes in the cycle times. As described in Section 6.7, changes in the cycle times may result in non-linear changes in the evolution and therefore the measure of complexity. As for logical complexity, the effects of the dependency are minimized by using constant values of operation cycle times for all configurations, permitting comparison between evolutions. To eliminate the effects of constant ratio changes of timers on the state complexity, the compressed state complexity is used.

6.9 Compressed State Complexity/Compressed State Compression

The compressed state complexity is identical to the state complexity, except that the number of unique states visited in an evolution is normalized by the length of the logical evolution, rather than the length of the temporal evolution. The resulting expression for the compressed state complexity, Cx_C, is given in Equation 6.13.

$$Cx_C = \frac{Number\ of\ Unique\ States\ Visited}{Logical\ Evolution\ Length} \tag{6.13}$$

The amount of redundant information in the logical evolution is described by the compressed state compression, Cm_C, which is related to the compressed state complexity by the relationship in Equation 6.14.

$$Cm_C = 1 - \frac{Number\ of\ Unique\ States\ Visited}{Logical\ Evolution\ Length} = 1 - Cx_C \tag{6.14}$$

The compressed state complexity is related to the logical complexity and the state complexity by the expression in Equation 6.15. This relation states that the compressed state complexity is the ratio of the information required to account for local and global patterns to the information required to account for local patterns only. The amounts of compression are not related in this manner, and the compressed state compression does not represent the excess information describing repetitions of global patterns.

$$Cx_C = \frac{Cx_S}{Cx_L} \tag{6.15}$$

As with state complexity, the number of unique states visited in an evolution is never greater than the length of the evolution (in this case the logical evolution). The values of compressed state complexity therefore range from a minimum of approximately 0, corresponding to a trivial evolution consisting of a single repeated state, to a maximum of 1, corresponding to an evolution composed completely of unique states.

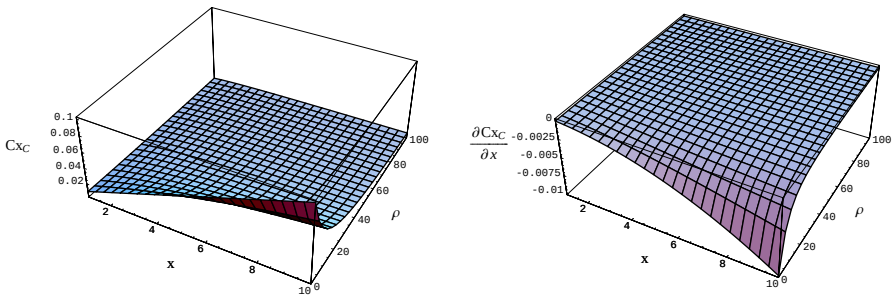
As an application of the definition of algorithmic complexity, compressed state compression ignores the additional information associated with sequentially repeated states attributed to operation cycle times. Compressed state complexity therefore only accounts for the information required to express higher order, more global patterns. While the complexity of the "actual" evolution is lost, it must be recalled that operation cycle times are variable and scaling them by constant ratios changes temporal evolution length, but does not alter the algorithmic complexity. The first order, sequential state repetitions can be assumed to be less critical as the higher order patterns in affecting the algorithmic complexity. The compressed state complexity, because it is based on the logical evolution, might therefore represent a truer application of algorithmic complexity than the state complexity, which includes the first order patterns. For an equivalent evolution, the complexity associated with the logical evolution will be greater than that for a temporal evolution, indicating the differences between the information required to describe higher and first order patterns, assuming first order patterns exist.

Compressed state complexity shares the advantages of state complexity over logical complexity by accounting for higher order patterns as well as first order patterns, though compressed state complexity accounts for higher order patterns by means of an evolution that has already accounted for first order repetitions. However, compressed state complexity also shares the disadvantage of underestimating complexity through the over-compression associated with non-deterministic evolutions and the discrepancies between state complexities of equivalently complex evolutions due to additional sequence repetitions.

In the case of longer evolutions resulting from a greater number of repetitions of a single sequence of states, the relative values of the fraction of unique visited states that a set of repeating states comprises, the number of states in the set of repeated states, the number of repetitions of the set, and the original logical evolution length collectively affect the complexity. The compressed state complexity is generalized in Equation 6.16, where u is the number of unique states visited in the evolution excluding the states comprising the set of repeating states, x. The original number of logical evolution steps, T_L, increases with each additional repetition, ρ, of the x states.

$$Cx_C = \frac{u + x}{T_L + \rho x} \qquad where \;\; u + x \leq T_L \tag{6.16}$$

In the simplest repetitive pattern, where all unique states are involved in the repeating sequence, $(u = 0)$, the plot of complexity values, shown in Figure 6.5(a) indicates that larger repetition periods have a greater effect on complexity. For all repetition periods however, the effect of the number of additional repetitions on the complexity diminishes as the number of repetitions increases. This trend is validated by determining the derivative of the complexity with respect to the number of additional repetitions, presented in Equation 6.17 and plotted in Figure 6.5(b) for a range of repetition periods and sizes.

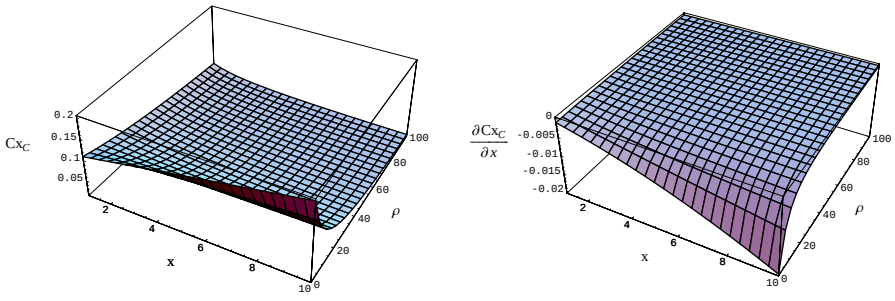


(a) (b)

Figure 6.5: The effect of changes in the number of additional repetitions and the lengths of their periods with $T_L = 100$ and $u = 0$ on (a) the compressed state complexity and (b) the rate of change of the compressed state complexity with respect to the number of additional repetitions. Lower repetition periods diminish the effects of additional repetitions. As the number of repetitions increases, the effect on the complexity decreases.

$$\frac{\partial\, Cx_C}{\partial \rho} = -\frac{x\,(u+x)}{(T_L+\rho x)^2} = \left.\frac{-x^2}{(T_L+\rho x)^2}\right|_{u=0} \tag{6.17}$$

As the number of unique states visited not contained in the set of states in additional repetitions increases, the trend remains the same - more repetitions have a diminishing effect on complexity and larger repetition periods have a greater impact on complexity. However, for greater values of u, the magnitude of the change of complexity increases, evident in a comparison of Figures 6.5 and 6.6. The magnitudes of the effects of these variables are all dependent on the length of the original evolution, T_L. For greater values of T_L, the effects of the changes of any other variables are diminished.


Cx_C
0.2
0.15
0.1
0.05
2
4
6
8
10
x
0
20
40
60
80
100
ρ
$\frac{\partial Cx_C}{\partial x}$
0
-0.005
-0.01
-0.015
-0.02
2
4
6
8
10
x
0
20
40
60
80
100
ρ