Security Applications of the ϵ-Machine

By

SEAN HARRISON WHALEN
B.S. (University of Nebraska at Omaha) 2002

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Matt Bishop, Chair

Jim Crutchfield

Felix Wu

Committee in Charge

2010

Dedicated to Linda Anderson, Paul Clarke, and Nancy Velardi, who set me on my way.

Abstract

The field of computational mechanics applies ideas from statistical mechanics, information theory, automata theory, and machine learning to create minimally-sized, optimal predictors of stochastic processes. These predictors, called ϵ-machines, are a subset of a well known statistical model class called the Hidden Markov Model (HMM). Despite being a subset, ϵ-machines have several important advantages over traditional HMMs. This dissertation illustrates these advantages by applying ϵ-machines to several problems in computer security: anomaly-based intrusion detection in High Performance Computing (HPC) environments, automated protocol reverse engineering, and structural drift.

Intrusion detection systems (IDSs) detect attacks on computer systems at the host or network level. IDS research is largely ad hoc, and often produces systems that cannot generalize to new attacks or raise prohibitive amounts of alerts. Our first application attempts to address these shortcomings for HPC environments. We construct ϵ-machine classifiers from the communication patterns of cluster nodes, as well as hardware counters including floating point and integer operation counts. We find these features are sufficient for accurate classification of parallel computation as well as detection of anomalous behavior.

Next, consider computers on a network exchanging data using some protocol whose specification is unknown—for example, a botnet command and control channel. Our work in automated protocol reverse engineering constructs a protocol ϵ-machine using only observed network traffic. The ϵ-machine captures both the topological and probabilistic structure of the protocol and is used for anomaly detection, traffic generation, and fuzzing without requiring access to binaries or source code.

Finally, we introduce a model of sequential inference to study the propagation of errors in chains of ϵ-machine learners. This model, called structural drift, is a generalization of memoryless drift models found in the field of population dynamics. We examine the drift of mem-

oryful models in process space and discuss the impact of model structure on the propagation of errors through time. This propagation has implications for all finite-data applications of the ϵ-machine.

Acknowledgments

For their endless patience and encouragement, I'd like to thank my past and present advisors Blaine Burnham, Matt Bishop, and Jim Crutchfield. Thanks also to my colleagues at the Complexity Sciences Center, in particular John Mahoney, Chris Ellison, and Benny Brown, for their friendship and collaboration.

I am eternally grateful for the love and support of family, especially my parents Mary and Tom and my sister Wendy. To my partner Sophie, you mean the world to me—you made this journey possible. To my extended family the Engles, you are and will always be dear to my heart.

To those I have lost, Fred, Ray, Violet, and Dea, you continue to inspire. And finally, to my darling nieces Brenna and Kirra, you continue to give me hope.

Table of Contents

List of Figures

CHAPTER 1
Introduction

Our national infrastructure is vulnerable to both real and simulated cyber attacks, such as the recent "Cyber Shockwave".[1] This simulation proved what security professionals suspected—our infrastructure is susceptible to decades-old attack vectors such as buffer overflows and malware. Our increasingly connected society presents an ever-growing attack surface [HOW05], as the binary distinction between insider and outsider is eroded by a plethora of always-on mobile computing devices [BIS09A, BIS09B].

Intrusion detection systems (IDSs) alert us when events happen on a computer or a network that violate security policy. Systems based on *misuse detection* look for signatures of disallowed behavior, and often miss new or evolving attacks that have previously unobserved signatures. *Anomaly detection* takes the opposite approach by defining a model of allowed behavior, and flagging behavior that strays too far from the norm. This approach suffers from high false positive rates, to the extent that deployed systems are sometimes disabled due to the volume of alerts [PAT07].

This situation can be partially attributed to the often ad hoc nature of systems-level research. The selection of a model or algorithm and its associated parameters are rarely explained or placed in a broad context. Custom algorithms and highly parametric models are used while simpler and more general solutions from other fields go unnoticed. For example, a survey of 63 published data mining algorithms, ranging from 5 to 10 parameters each, found that nearly all were outperformed by a simple non-parametric method using 12 lines of code [KEO04]. These same over-fit models are typically trained on data sets with well documented flaws [MCH00, MAH03, SAB04].

[1] http://bit.ly/dtg5cM

Thus, IDS research suffers from two primary problems: a preference for ad hoc models over mathematical models, and ad hoc parameter selection when mathematical models are used. Our research addresses these issues by turning to the field of *computational mechanics* [CRU89, SHA01A]—an interdisciplinary field combining models and techniques from statistical mechanics, information theory, automata theory, and machine learning. A primary goal of computational mechanics is the creation of minimally-sized, optimal predictors of stochastic processes. These predictors, called ϵ-machines, are a subset of a well known class of statistical model called the Hidden Markov Model (HMM). Despite being a subset, ϵ-machines have several important advantages over traditional HMMs including reconstruction of the minimal model architecture directly from data and closed-form calculation of information theoretic quantities.

Information theory and HMMs have been applied separately to several problems in security. Previous work by Lee et al [LEE01] used information theory to detect anomalies in network traces, demonstrating the utility of non-parametric approaches for understanding data without the benefit of domain knowledge. Kayacik et al [KAY05A] continued applying information theory, using information gain to identify the most relevant features for detecting intrusions in the KDD99 dataset. Extending previous work, Warrender et al [WAR99] compared HMMs with other techniques for detecting anomalies in sequences of system calls. HMMs were applied by Florez et al [FL05A, FL05B] to detect anomalies in parallel computation as well as the DARPA99 dataset, while Koo et al [KOO03] used the Viterbi algorithm [RAB89] with audit trail data. Most recently, Fava [FAV08] used Markov models of variable order for identifying correlated attacks in a custom dataset.

The goal of this thesis is to detail the practical application of computational mechanics, most specifically the ϵ-machine, to anomaly detection and the related area of protocol inference. The techniques described are part of a general framework that should enable reconstruction from complex datasets across many disciplines, including those outside computer security. This framework places emphasis on non-parametric methods for constructing models in the absence of domain knowledge, and a systematic approach to reducing the complexity of data to enable ϵ-machine reconstruction.

We proceed in 4 independent parts, beginning with a summary of the field of computational mechanics. This requires a discussion of stochastic processes and how their complexity

is defined and measured using information theory. We also contrast the ϵ-machine with more traditional Markov and Hidden Markov Models. ϵ-Machines are first used to classify parallel computations using processor communication patterns, with the ultimate goal of constructing a focused anomaly detection system for High Performance Computing environments. We next show how ϵ-machines can learn the structure of network protocols in order to mimic them for traffic generation and botnet infiltration, as well as anomaly detection and intelligent protocol fuzzing. Finally, we introduce a new inter-disciplinary paradigm for sequential learning using chains of re-inferred ϵ-machines.

The following chapters contribute a general machine learning framework for the application of ϵ-machines to problems with finite, complex data. This framework uses information theory for the construction of quantifiable models in the presence or absence of domain knowledge. We confront the time complexity of current reconstruction algorithms by reducing the number of symbols in a dataset using filtering, quantization, and higher order symbolic representations. By enabling inference from complex datasets, the advantages of the ϵ-machine can be brought to bear on otherwise prohibitive problems. Using these techniques we contribute novel results to two such problems in security including anomaly detection and protocol inference. We also generalize two long standing results in the field of population dynamics, as well as give a new interpretation of punctuated equilibria with potential applications to computational biology.

CHAPTER 2
Computational Mechanics

Computational mechanics [CRU89, SHA01A] provides a framework for modeling the intrinsic information processing of a system by using the statistics of external observations that indirectly reveal the internal state of the system:

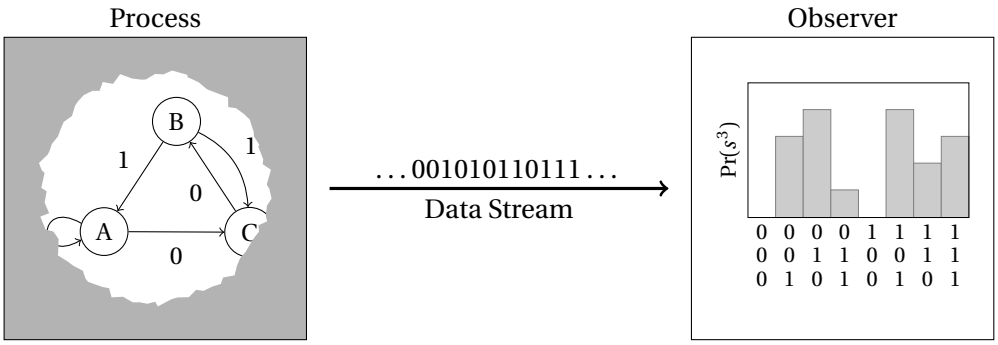

Figure 2.1: The indirect modeling of a hidden process via observed measurements (adapted from Crutchfield [CRU03A])

This division between internal state and external observation is useful for a wide range of statistical modeling tasks. The most common of these is the Hidden Markov Model (HMM), an extension of standard Markov models and also the simplest type of dynamic Bayesian network.

The construction of HMMs is often ad hoc due to insufficient domain knowledge, resulting in non-minimal models that serve as sub-optimal predictors. Some parameters of the model can be trained from observed data, but this training is slow and prone to get stuck in local optima. Computational mechanics offers an alternative statistical hidden-state model called the ϵ-machine that addresses these problems for stationary stochastic processes.

In fact, ϵ-machines are a subset of HMMs. However, such a brief description undercuts the significant differences between ϵ-machines and their HMM brethren. Both as an abstract mathematical construct and an instantiated model using finite data, the ideas behind the ϵ-machine

are deep and touch a wide range of inter-disciplinary research topics [VAR07, HAN93, CRU06]. To motivate, we first give a brief background on stochastic processes and information theory while introducing our notation. We then build up to the ϵ-machine by way of traditional Markov and Hidden Markov Models.

§2.1 Stochastic Processes

Let a discrete random variable be denoted by an upper case letter, i.e. X. Then let a particular value of X be denoted by a lower case letter, i.e. x. We define the alphabet $\mathcal{A}$ as the finite set of all $x \in X$, while the probability that X takes a particular value is given by $\Pr(X = x)$ or just $\Pr(x)$. The joint probability of $X = x$ and $Y = y$ is $\Pr(x, y)$ and the conditional probability of $Y = y$ given $X = x$ is $\Pr(y|x) = \frac{\Pr(x,y)}{\Pr(x)}$.

A discrete stochastic process is a sequence $\ldots X_1, X_2, X_3 \ldots$ of random variables X_n indexed by time. Realizations $\ldots x_1, x_2, x_3 \ldots$ are called *time series*. Examples of time series include stock prices or rainfall amounts—the topic of time series analysis touches many fields such as statistics and economics [GER98]. We will refer to time series as *strings* as they are more commonly called in computer science.

It is often mathematically convenient to consider a bi-infinite string $\overleftrightarrow{X}$ and divide it into a semi-infinite past $\overleftarrow{X}$ and future $\overrightarrow{X}$ at some time t. Given a process:

$$\overleftrightarrow{X} = \ldots X_{t-2}, X_{t-1}, X_t, X_{t+1}, X_{t+2} \ldots$$

the division into past and future, respectively, is:

$$\overleftarrow{X} = \ldots X_{t-2}, X_{t-1}$$

$$\overrightarrow{X} = X_t, X_{t+1}, X_{t+2} \ldots$$

Good models make accurate predictions about the future given knowledge of the past.[1] Towards this end, we are often interested in computing $\Pr(\overrightarrow{X}|\overleftarrow{X})$, but cannot use semi-infinite sequences when working outside the realm of mathematical abstraction and using finite amounts of data. Instead, we approximate semi-infinite sequences using finite blocks of L consecutive symbols, denoted:

[1] It is often necessary to make measured tradeoffs between model size and accuracy to work within processor and memory limitations. See Optimal Causal Inference [STI07] for details on one such tradeoff using ϵ-machines.

$$X_t, X_{t+1}, \ldots, X_{t+L-1} = X_t^L$$

To reflect the stationary assumption that the probability distribution of X is independent of time, we drop the time index when working with pasts and futures with the understanding that they divide the string at some implied time t. The next-symbol probability given the previous L symbols is then:

$$\Pr(\overrightarrow{X}^1 \,|\, \overleftarrow{X}^L)$$

Prediction may be easy or difficult, depending on the complexity of the process being modeled. However, "complexity" is a loaded term—there are at least 400 definitions [SHA01B]—so let's detour to present our take on complexity before continuing.

§2.2 Complexity Measures

§2.2.1 Algorithmic Information Content

We focus on distinguishing two notions of string complexity. First, the Algorithmic Information Content (AIC) introduced by Kolmogorov, is the length of the shortest computer program for a Universal Turing Machine that exactly reproduces the string [CHA77, LI97]. This definition ties complexity to randomness in the sense that a uniformly random string has maximal AIC. Since a truly random string does not have a compact representation, the length of the program to print the string will have to store the string itself, and thus the AIC grows with the length of the string. For the opposing case, any periodic string that repeats a block of numbers will have a small AIC, since the minimal program will simply print the block inside a loop.

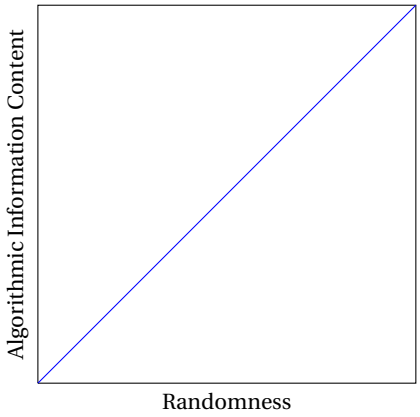

Figure 2.2: Algorithmic Information Content as a function of randomness

For example, consider a string formed by the outcome of a million flips of a fair coin. The AIC of this string will be maximal as each flip of the coin is independent and identically distributed (IID). Since there are no correlations between flips, the generating program must store the string explicitly to reproduce it.

§2.2.2 Statistical Complexity

If instead we look at a statistical description of a uniformly random string, trading exact reproducibility of the string for a smaller model, we get a vastly different result: the string has zero complexity. A finite state machine with a random number generator can produce strings with the same distribution of heads and tails as a random string. Moreover, it can do this using only a single state. Because a single state machine needs $\log_2(1) = 0$ bits of internal state, we say the fair coin has a *statistical complexity* of zero. This measure, given symbol C_μ, is the amount of internal memory in a hidden process.

Figure 2.3: Statistical Complexity as a function of randomness

Thus we have two complementary notions of complexity as demonstrated by the simple case of a biased coin. Any periodic process, such as a completely biased coin, has both zero AIC and statistical complexity. Where they differ is the classification of purely random processes such as a fair coin: AIC associates randomness with complexity, while statistical complexity C_μ associates structure with complexity. AIC says a fair coin produces strings with maximal complexity, while C_μ says such strings are not complex.

It is worth noting that AIC is not a computable number due to the halting problem [Cov91], though the Lempel-Ziv compression algorithm is sometimes used to place an upper bound on the AIC [Eva02]. Statistical complexity, however, is computable from the ϵ-machine of a process. Another measure of complexity, Shannon's entropy rate h_μ (see section 2.3.2), is also calculable in closed form from the ϵ-machine and is asymptotically related to the AIC [Li97]. We will show later that C_μ and h_μ are but two of several quantities useful for characterizing hidden processes of varying complexity, but first we ground the discussion in the basics of information theory.

§2.3 Information Theory

The complexity of a model is tied both to its size and the diversity of patterns it generates or predicts. A period-L process repeats a block of L symbols and so its finite state presentation will have L states. It turns out that for any period-L process, $C_\mu = \log_2(L)$. This means that a period-2 process and a period-4 process will have different C_μ. In some sense, however, they are similar: neither process surprises us when we observe the next symbol, once we *synchronize* ourselves to the current position in the block. From the point of synchronization onward, we gain no additional information by observing the next symbol and the process is completely predictable.

A different measure of complexity is this average per-symbol information generated by a process. We'll move towards defining this measure by covering several concepts from information theory.

§2.3.1 Entropy

We first introduce the ubiquitous Shannon entropy [Cov91], the expected value (in bits) of the information content of a discrete random variable X:

$$H[X] = -\sum_{x \in X} \Pr(x) \log_2 \Pr(x)$$

The use of brackets indicates H is not a function of X but rather its distribution. Entropy is maximized by the uniform distribution, shown here for a binary random variable:

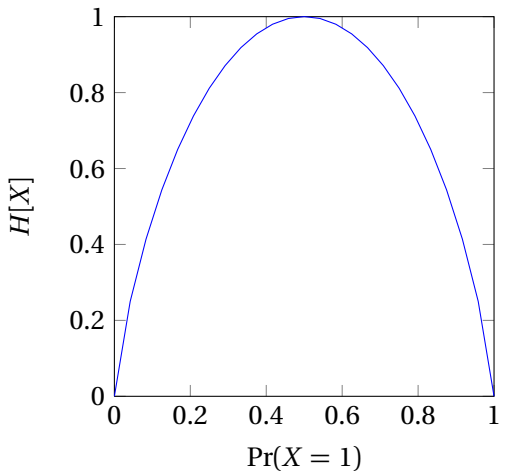

Figure 2.4: Entropy of a binary random variable

The words information, surprise, and uncertainty are often used interchangeably with entropy. We are more surprised by the outcome of an uncertain event than a certain one and thus it conveys to us more information. Entropy quantifies this information.

§2.3.2 Entropy Rate

We estimate the entropy using finite blocks of L consecutive symbols:

$$H(L) = -\sum_{x^L \in X^L} \Pr(x^L) \log_2 \Pr(x^L)$$

Having defined H over finite blocks of symbols, we can now consider the per-symbol information termed the *entropy rate*:[2]

$$h_\mu = \lim_{L \to \infty} \frac{H(L)}{L}$$

An alternate way of viewing this convergence is the difference between the entropy of consecutive block lengths:

$$h_\mu = \lim_{L \to \infty} [H(L) - H(L-1)]$$

It is often convenient give the block-L estimate of the entropy rate by dropping this limit:

[2] Entropy rate goes by several names, including metric entropy [Kol58].

$$h_\mu(L) = \frac{H(L)}{L} = H(L) - H(L-1)$$

Of course, machines with finite memory can only approximate these limits. However, the entropy rate is computable in closed form directly from the ϵ-machineẆe later give this equation after introducing specifics of the model.

Consider the Golden Mean process that generates binary strings where consecutive zeros cannot occur. This property gives the process a finite dependence on past symbols, unlike the IID fair coin example. We examine the rate of convergence towards the true entropy rate by computing the entropy over block length $L = 1 \ldots 16$ symbols:

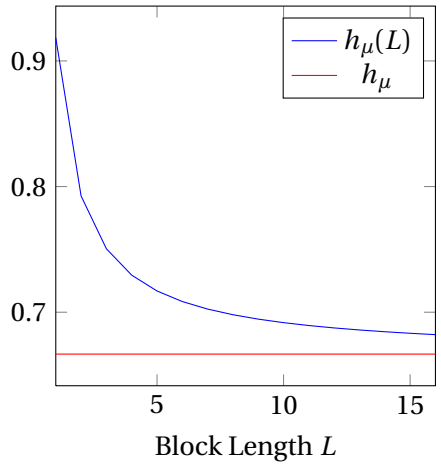

Figure 2.5: Numerical convergence of block estimate to actual entropy rate

Note the excess randomness as our estimate converges. An abstract view of this convergence is given below:

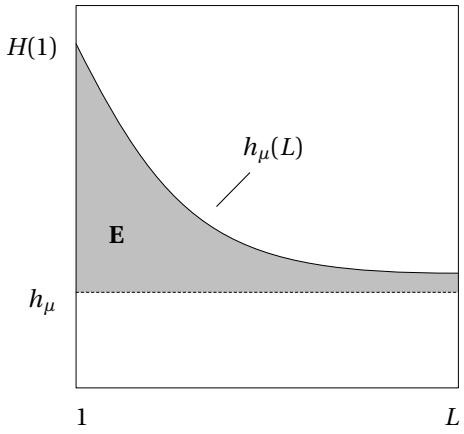

Figure 2.6: Abstract convergence of block estimate to actual entropy rate

As it turns out, the area between the true entropy rate h_μ and the estimate $h_\mu(L)$ is another measure of complexity, discussed next.

§2.3.3 Excess Entropy

As block length increases, longer range correlations between symbols are discovered and the apparent randomness of the blocks is reduced. If we take the sum of these over-estimates of randomness over all block lengths, we obtain another measure of complexity termed the *excess entropy*:[3]

$$\mathbf{E} = \sum_{L=1}^{\infty} [h_\mu(L) - h_\mu]$$

Excess entropy says how much information an observer needs to collect in order to measure the true entropy rate of a process. A larger excess entropy indicates the process has longer-range correlations between symbols and thus contains more internal structure.

One interpretation of excess entropy is the cost of amnesia, as a string will appear **E** bits more random to an observer that forgets the past. The observer will subsequently need to collect **E** bits to optimally predict the future. Prediction accuracy is limited by the irreducible per-symbol uncertainty given by the entropy rate, and **E** measures how much information is needed to predict with an error rate that approximates h_μ.

§2.3.4 Mutual Information

It is often useful to compute the *conditional entropy* of X given knowledge of Y:

$$H[X|Y] = -\sum_{x \in X} \sum_{y \in Y} \Pr(x, y) \log_2 \Pr(x|y)$$

We can use entropy and conditional entropy to measure the dependency between X and Y by computing their *mutual information* (MI):

$$I[X; Y] = H[X] - H[X|Y]$$

This equation interprets mutual information as the uncertainty remaining in X when Y is known. The following Venn diagram illustrates this idea:

[3] Excess entropy has many alternate names, the most well known being Grassberger's *effective measure complexity* [GRA86].

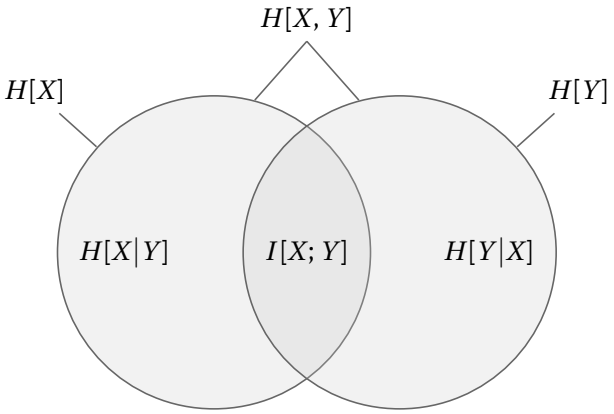

Figure 2.7: Information diagram showing the joint, marginal, and conditional entropies of two random variables X and Y as well as their mutual information

Mutual information differs from the well-known Pearson correlation coefficient in that it can measure non-linear dependencies, while Pearson's cannot [Li90]. This comes at a cost, as the mutual information is symmetric and cannot distinguish between negative and positive correlations. In addition, mutual information is sensitive to the size of a variable's event space [Ste02]. This sensitivity is seen by comparing I between two uniform random strings and increasing the size of the event space.[4] As we increase the number of values a random variable can take but keep data length fixed, mutual information is increasingly over-estimated.

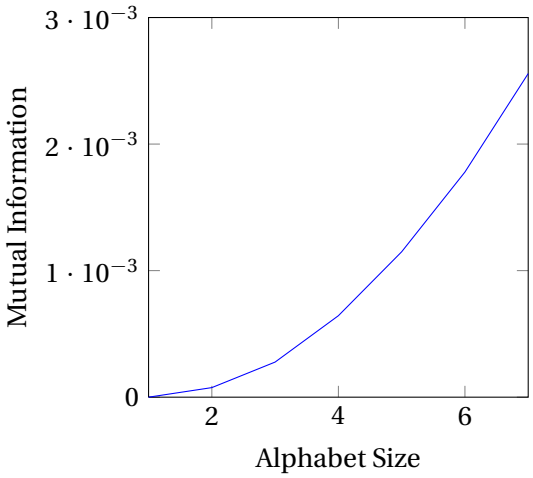

Figure 2.8: Finite data effects of alphabet size on mutual information

If we keep the event space fixed but vary data length, we see I is similarly over-estimated for smaller amounts of data:

[4] For example, the first two strings can take binary values, the second two from the set {0, 1, 2}, and so on.

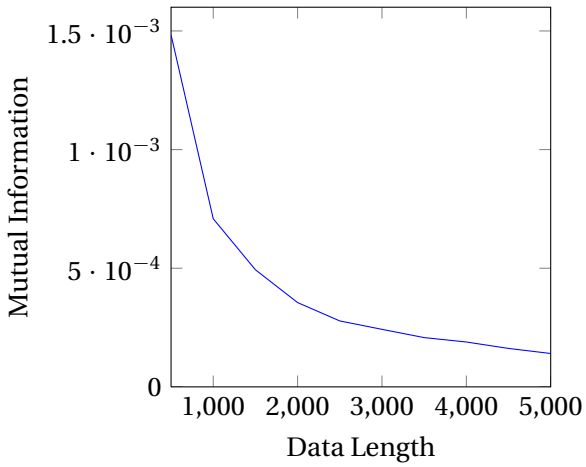

Figure 2.9: Finite data effects of data length on mutual information

Another interpretation of the mutual information is the distance between the joint and marginal distributions of X and Y:

$$I[X;Y] = \sum_{x \in X} \sum_{y \in Y} \Pr(x,y) \log_2 \frac{\Pr(x,y)}{\Pr(x)\Pr(y)}$$

This allows us to restate **E** as the mutual information between the past and the future:

$$\begin{aligned} \mathbf{E} &= I[\overleftarrow{X}; \overrightarrow{X}] \\ &= \sum_{\{\overleftrightarrow{x}\}} \Pr(\overleftrightarrow{x}) \log_2 \frac{\Pr(\overleftrightarrow{x})}{\Pr(\overleftarrow{x})\Pr(\overrightarrow{x})} \end{aligned}$$

This "distance" between distributions is a common quantity used in information theory, which we introduce next.

§2.3.5 Kullback-Liebler Divergence

The *Kullback-Liebler (KL) divergence*, also called the KL distance or relative entropy, gives the distance between two discrete probability distributions X and Y:

$$D_{\mathrm{KL}}\left(X||Y\right) = \sum_{x \in X} \Pr(X=x) \log_2 \frac{\Pr(X=x)}{\Pr(Y=x)}$$

This quantity is always positive ($D_{\mathrm{KL}} >= 0$) and is zero only when the distributions are equal ($D_{\mathrm{KL}}\left(X||X\right) = 0$). Relative entropy also gives an alternate interpretation of mutual information as the distance between joint and marginal distributions:

$$I[X;Y] = D_{\mathrm{KL}}\left(\Pr(x,y)||\Pr(x)\Pr(y)\right)$$

It is important to note several shortcomings with KL divergence. Firstly, it is not a true distance metric since it does not obey the triangle inequality. Secondly, it is not symmetric ($D_{\mathrm{KL}}\left(X||Y\right) \neq D_{\mathrm{KL}}\left(Y||X\right)$), though it can be made so by summing or averaging both directions [KUL59]. Lastly, it requires $\Pr(X = x) > 0$ and $\Pr(Y = x) > 0$ for all $x \in X$. This is rarely the case for the empirical distributions we encounter in this dissertation. We employ Laplace Smoothing as a workaround, assigning pseudo-counts to non-existent words and subtracting those counts from existing word counts to preserve probability mass [JUR00].

So far we have talked about stochastic processes, discussed our notion of complexity, and defined several information-theoretic quantities including entropy, entropy rate, excess entropy, mutual information, and KL divergence. We now describe two finite state machine models of stochastic processes—Markov and Hidden Markov Models—and conclude the chapter by introducing and contrasting these models with the ϵ-machine.

§2.4 Markov Models

A Markov Chain is a statistical model of a discrete stochastic process defined by a set of states and transition probabilities between states, as well as a vector of initial state probabilities. It assumes the Markov property holds for the underlying process being modeled, where the probability of the next state X_{t+1} is determined by the current state and not the entire past sequence of states [NOR98]. This property is expressed as:

$$\Pr(X_{t+1}|X_t, X_{t-1}, \ldots, X_1) = \Pr(X_{t+1}|X_t)$$

The Markov property is often assumed to hold even when the underlying process is not Markovian. This approximation affects the accuracy of the model, but greatly reduces the number of model parameters.

The future state of an order-m Markov Chain is conditionally dependent on the last m states of the system. An order-0 Markov process is IID, where the future state is completely independent of the past. When the order is not made explicit, it is typically assumed the model is of order 1. Using an order greater than 1 allows the model to predict the future using more of the recent past.

Model parameters are often estimated from observed data. The number of parameters grows exponentially fast with model order, and using an insufficient amount of data quickly results in an overfit model. For example, an order-9 Markov Chain of English literature requires an entire Library of Congress worth of data [LUC91].

Let S denote the set of states and T a stochastic matrix[5] of state transition probabilities. The probability of transitioning from state i to state j is located in row i, column j and is denoted T_{ij}. A simple two-state Markov Chain is specified by:

$$T = \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix}$$

where $\alpha, \beta \in [0,1]$ are parameters. This corresponds graphically to the following state machine diagram:

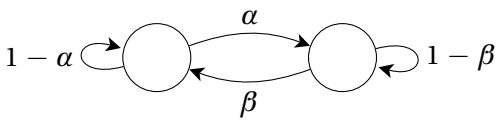

Figure 2.10: A two state Markov Chain

The *stationary distribution* π gives the asymptotic probability of being in each state after an infinite number of time steps. This can be approximated by exponentiating T to a large power and examining any row, or obtained analytically by computing the left eigenvector for eigenvalue 1. The Perron-Frobenius theorem states this eigenvector always exists for a stochastic matrix [MEY01].

Markov Chains represent only a subset of stochastic processes. Consider the Even Process, generating binary strings where all blocks of consecutive ones are even in length (for example, 0110 and 011110 but not 010). It turns out that this process is not equivalent to a Markov Chain of any finite order [CRU03A]. In such cases, one must employ a more sophisticated model class such as a Hidden Markov Model.

§2.5 Hidden Markov Models

Hidden Markov Models (HMMs) extend traditional Markov Chains by allowing each state to gen-

[5] Each row in a stochastic matrix sums to one, meaning the transitions from every state form a valid probability distribution.

erate a symbol from some distribution.[6] This allows modeling hidden processes where the internal state of the system is revealed to an observer only indirectly through these generated symbols. Since a string of symbols can often be generated by multiple internal state sequences, observations do not uniquely identify the state of the system.

More formally, an HMM is defined by a set of hidden states S, a state transition matrix A, a symbol emission matrix B, and an initial state distribution π.[7] We can define the Even Process from the previous section as follows:

$$A = \begin{pmatrix} 0.5 & 0.5 & 0 \\ 0 & 0 & 1 \\ 0.5 & 0.5 & 0 \end{pmatrix}, \; B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}, \; \pi = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

This is equivalent to the following state machine diagram:

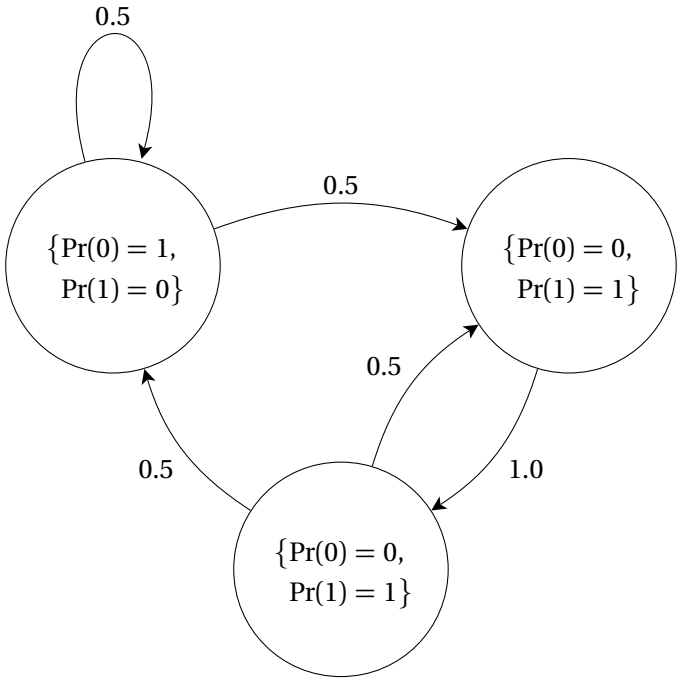

Figure 2.11: Three state Moore HMM of the Even Process

Each edge is labeled with a state transition probability, and each node with the distribution of symbols it emits. This type of node-emitting model is called a *Moore machine*, in contrast with an edge-emitting *Mealy machine*. The latter representation is used by ϵ-machines, though the

[6] Since this distribution is a function of state, HMMs are sometimes called probabilistic functions of Markov Chains.

[7] Note the reuse of the symbol π, here specifying initial instead of asymptotic state probabilities.

two are mathematically equivalent.

The seminal paper by Rabiner [RAB89] describes three quantities an HMM can compute for any given observation sequence: 1) the probability of the model generating the observations, 2) the most likely internal state sequence that generates the observations, and 3) the model parameters that maximize the first quantity. These are computed by the forward-backward algorithm, the Viterbi algorithm, and the Baum-Welch algorithm, respectively. The time complexity of the forward-backward and Viterbi algorithms is $O(N^2 L)$ for N states and L observations. Baum-Welch commonly trains on M strings, giving a time complexity of $O(MN^2 L)$.

Though successfully applied to problems across many domains, the specification of HMM parameters is often ad hoc. The underlying HMM architecture, including the number of states and their connectivity, must be specified by the modeler. For systems without available domain knowledge, this can be a difficult decision and greatly impacts the accuracy of the model. In addition, though transition and emission probabilities may be estimated from data using Baum-Welch, the algorithm may not converge and is susceptible to local optima.

Architecture specification and training convergence may not be issues when domain knowledge is available. When it is not available, however, these issues can be addressed by a special type of HMM called an ϵ-machine.

§2.6 ϵ-Machines

An *epsilon machine* (ϵ-machine) is the minimally sized, optimal predictor of a stochastic process [CRU89]. They are also unique in that any predictor having these properties is the ϵ-machine, and so stochastic processes and ϵ-machines are in one-to-one correspondence. Though a subset of HMMs, ϵ-machines have several important differences: 1) states are equivalence classes of histories called *causal states*, 2) transitions are *unifilar*, 3) transient states of the process are represented, and 4) they are edge-generating Mealy machines. We'll discuss each of these in turn.

Causal states are equivalence classes of histories $\overleftarrow{X}$. The *causal state partition* is a unique mapping of histories to causal states, and is created by grouping all histories with the same conditional distribution over futures $\Pr(\overrightarrow{X}|\overleftarrow{X})$. That is, if the future "looks" the same after observing some set of histories, those histories are considered causally equivalent. This equivalence re-

lation is a *sufficient statistic* [FIS22], meaning the complete dataset can be discarded and the causal states themselves used for optimal prediction. This establishes a close connection between causal equivalency and universal data compression as first defined by Rissanen [RIS83].

Causal states are divided into those that are *transient* and those that are *recurrent*. Transient states represent an observer's incomplete knowledge of the current causal state. As more symbols are seen, the observer will eventually synchronize to the process and enter the recurrent portion of the model. From this point forward the transient states are no longer reachable[8] and the future can be optimally predicted from the recurrent causal states. Before this, prediction is sub-optimal due to the incomplete information of the observer.

Transitions between causal states are unifilar, meaning a causal state and a symbol uniquely determine the next causal state. Such transitions are referred to as *deterministic* in automata theory.[9] Though stochastic processes have unifilar and non-unifilar *presentations*, ϵ-machines are unifilar. This unifilarity is required to compute information-theoretic quantities such as h_μ or $\mathbf{E}$.

We now discuss these differences from traditional HMMs in the context of the Even Process, presented as an edge-generating ϵ-machine where a transition emitting symbol s with probability p is labeled $s \mid p$:

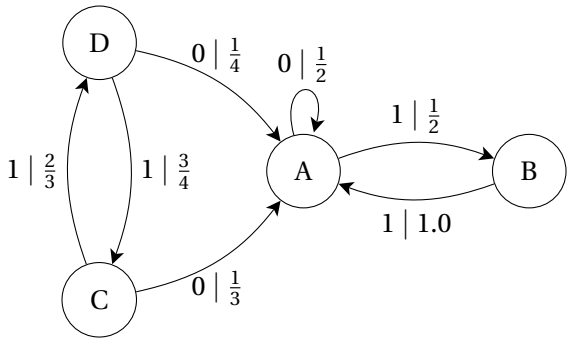

Figure 2.12: The ϵ-machine of the Even Process

States $\{C, D\}$ are transient and $\{A, B\}$ are recurrent. We start in transient state C having observed no symbols. For as long as we observe 1s we cannot be sure if the 1 was generated by recurrent state A or B, so we alternate between states C and D with a sub-optimal forecast of future symbols. However, we synchronize as soon as a 0 is observed since the process must then

[8] Thus, the asymptotic probability of all transient states is zero.

[9] "Unifilar" is used to avoid the conceptual baggage that "deterministic" carries in some fields.

be in state A. Once synchronized there are no transitions back to C and D, reflected by the stationary distribution $\{A = \frac{2}{3}, B = \frac{1}{3}, C = 0, D = 0\}$. Causal state B enforces the constraint that symbol 1 is always followed by another 1, resulting in even length blocks of 1s.

Formally, an ϵ-machine is defined by a set of causal states $\boldsymbol{\mathcal{S}}$, an emission alphabet $\mathcal{A}$, and a set of transition matrices $\{T^{(s)} : s \in \mathcal{A}\}$. For the recurrent portion of the Even Process, these are:

$$\boldsymbol{\mathcal{S}} = \{A, B\},\ \mathcal{A} = \{0, 1\},\ T^{(0)} = \begin{pmatrix} 0.5 & 0 \\ 0 & 0 \end{pmatrix},\ T^{(1)} = \begin{pmatrix} 0 & 0.5 \\ 1 & 0 \end{pmatrix}$$

The per-symbol transition matrices allow simple calculation of the likelihood of a string (the *likelihood function*) by taking the inner product of the transition matrices corresponding to each symbol in the string:

$$\Pr(x_1, x_2, \ldots, x_n) = \left(\pi \prod_{i=1}^{n} T^{(x_i)}\right) \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

The column vector of ones sums the probability of all causal state paths, and the π term weights each path by the stationary probability of its start state. For example:

$$\begin{aligned} \Pr(0110) &= \pi T^{(0)} T^{(1)} T^{(1)} T^{(0)} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} \frac{2}{3} & \frac{1}{3} \end{pmatrix} \begin{pmatrix} 0.5 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0.5 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0.5 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0.5 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ &= \frac{1}{12} \end{aligned}$$

This likelihood function serves the same purpose as the forward-backward algorithm for HMMs: to compute the probability of an observation sequence given a particular model. There are also analogs of the Viterbi and Baum-Welch algorithms, and thus we can use ϵ-machines to answer, at the very least, the same set of questions asked of HMMs.

In addition, the ϵ-machine provides closed form expressions of the information-theoretic quantities introduced earlier such as the entropy rate:

$$h_\mu = -\sum_{\sigma\in\mathcal{S}} \Pr(\sigma) \sum_{\sigma'\in\mathcal{S},a\in\mathcal{A}} T^{(a)}_{\sigma\sigma'} \log_2 T^{(a)}_{\sigma\sigma'}$$

This is the weighted sum over the entropy of each casual state's transition probabilities. We can also compute the statistical complexity as the entropy of the stationary distribution:

$$C_\mu = -\sum_{\sigma\in\mathcal{S}} \Pr(\sigma) \log_2 \Pr(\sigma)$$

Historically, **E** could not be calculated analytically for arbitrary processes. However, Ellison et al recently introduced the *crypticity* χ of a process as the difference between its stored and expressed information [ELL09], and manipulating this quantity gives an expression for excess entropy:

$$\mathbf{E} = C_\mu - \chi$$

These complementary measures, along with those inherited from HMMs, allow ϵ-machines to characterize the information processing of hidden processes. So far, however, we have dealt with ϵ-machines primarily as a mathematical object inferred from infinite data. The final introductory section on computational mechanics discusses techniques for creating ϵ-machines from finite strings of data and the difficulties that ensue.

§2.7 ϵ-Machine Reconstruction

An ϵ-machine is inferred or *reconstructed* from finite strings of data using a *reconstruction algorithm*. Multiple such algorithms exist, and each makes different assumptions that affects their ability to effectively reconstruct certain types of processes. Time series reconstruction algorithms include subtree merging [CRU89], state splitting [SHA04], and optimal causal inference [STI07]. Algorithms for other data sources such as power spectra also exist [VAR07].[10] This section details reconstruction of the Golden Mean process, generating binary strings with no consecutive zeros, using subtree merging.

First, we construct a finite approximation to the full joint distribution of past and future sequences by scanning consecutive length-D blocks of the time series with a sliding window. Each block is inserted into a *parse tree*[11] one symbol at a time, and each successive symbol becomes a

[10] We implemented the subtree merging and state splitting algorithms in Java, C++, and Python, and the latter is available in the open source Computational Mechanics in Python (CMPy) package.

[11] Also known as a prefix tree or trie.

child node of the previous symbol. The tree initially contains a root node corresponding to the empty string λ, and we return to this root after inserting each block. A node at depth $d \leq D$ stores the frequency count of the length-d block it represents, and this count is incremented each time the node is visited. The counts of adjacent nodes at depths d and $d-1$ are used to compute the conditional probability of the symbol that connects them.

Below is a parse tree of depth 3 progressively constructed from a sliding window (shown as an underline) over the string 01101. Nodes are labeled with their frequency counts, though we later use the corresponding block instead. The node with count 2 results from the overlap of the first symbol of the second window 110 and third window 101.

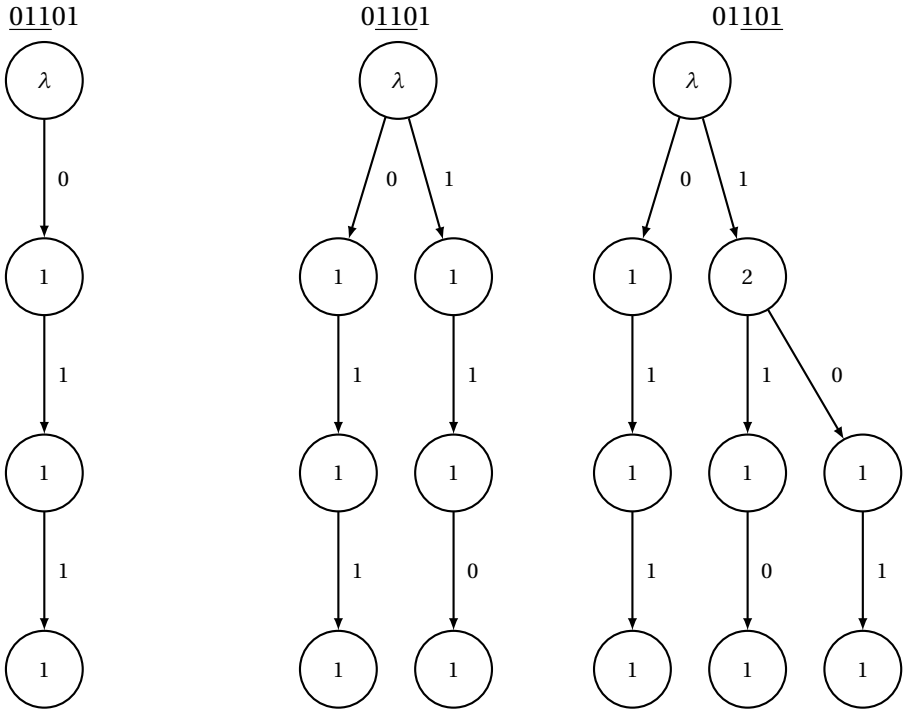

Figure 2.13: Incremental parse trees for the string 01101 with a length-3 sliding window

Next is the depth-4 parse tree, given infinite data:

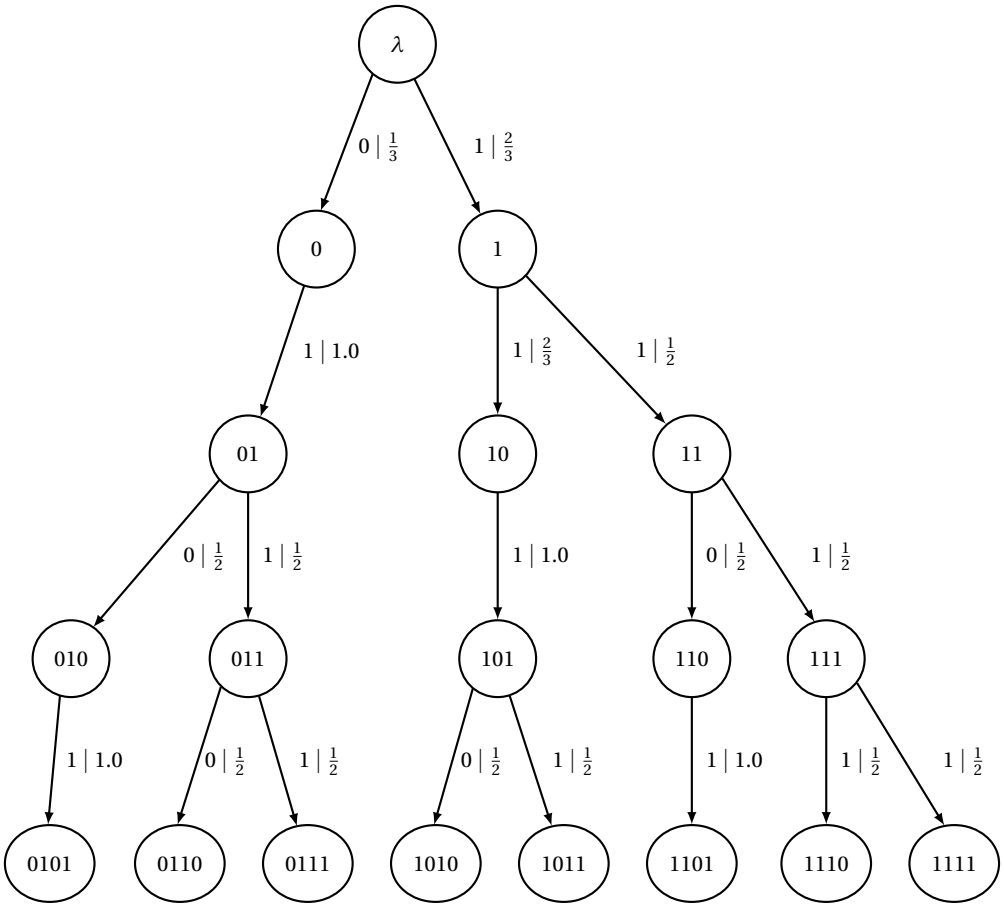

Figure 2.14: Complete parse tree of the Golden Mean Process

Here we label nodes with blocks instead of counts, and edges with both a symbol and conditional probability. For block b and symbol s, this is computed as $\Pr(s|b) = \frac{\Pr(b,s)}{\Pr(b)}$.

Given the depth-D parse tree, we find the set of depth-L subtrees that represent unique conditional distributions over future symbols. These subtrees are called *morphs*, and each unique morph becomes a labeled causal state. We create the causal state set $\boldsymbol{\mathcal{S}}$ by traversing the parse tree and comparing the morph rooted at each node to all $\sigma \in \boldsymbol{\mathcal{S}}$. If the morph is not equivalent to any causal state,[12] we assign the morph a label and add it to $\boldsymbol{\mathcal{S}}$.

From the parse tree above we find $\boldsymbol{\mathcal{S}} = \{A, B, C\}$:

[12]There are multiple ways to test for morph equivalence. A simple approach tests that all edge symbols are the same, and all edge probabilities are the same within a fixed tolerance. Instead, we use the Kolmogorov-Smirnov test with a significance level α of 0.01.

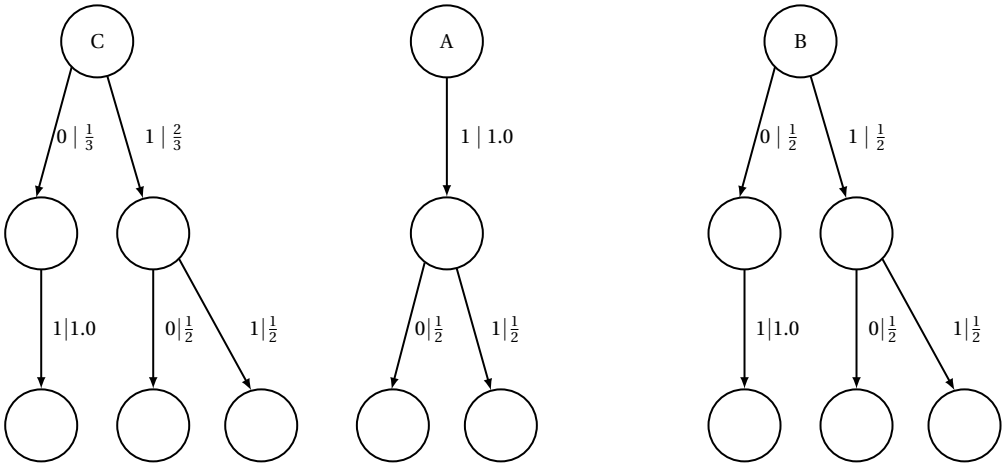

Figure 2.15: Causal state subtrees of the Golden Mean Process

Each node in the parse tree is given a label corresponding to its causal state. Nodes without a subtree of at least depth L are pruned:

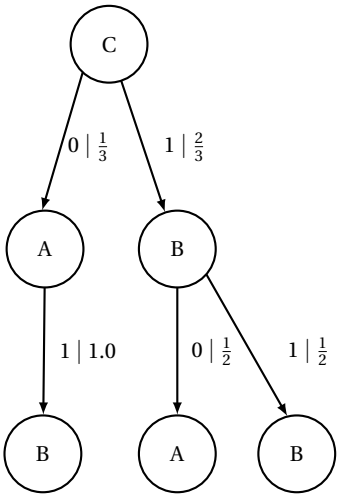

Figure 2.16: Pruned parse tree for the Golden Mean Process with causal state labels

The final ϵ-machine is produced by merging nodes with identical causal state labels:

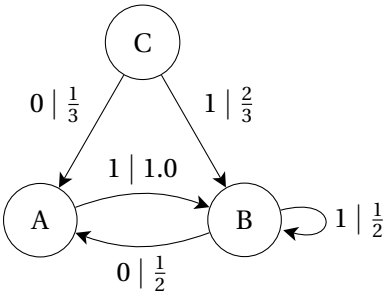

Figure 2.17: The ϵ-machine of the Golden Mean Process

As with any inference from finite data, the amount of available data impacts the accuracy of the model. For example, consider the number of inferred causal states as a function of data length, averaged over 100 realizations of the Golden Mean:

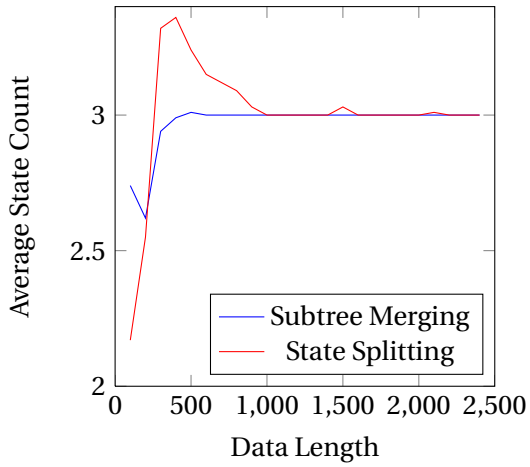

Figure 2.18: Effect of data length on number of inferred causal states

for state splitting and subtree merging reconstruction algorithms

The state splitting and subtree merging algorithms both converge to the correct number of causal states at different rates—here, subtree merging needs roughly 3 times less data.

Inference is also sensitive to the choice of model parameters. For subtree merging, these include the parse tree depth D and morph length L. As detailed by Hanson [HAN93], these parameters interact with the data length N and can result in non-unifilar models or "dangling states" with no outgoing transitions. When L is too small, morphs that are distinct at length $L' > L$ may appear equivalent, resulting in a model that generates sequences forbidden by the hidden process. If $L' < L$ there may be insufficient data in the parse tree, and morphs that should be equivalent will not be. A search over parameter space that results in local stability of the machine for fixed N can often identify suitable values of L and D.

The time complexity of current reconstruction algorithms is exponential in the alphabet size $|\mathcal{A}|$ and is affected differently by algorithm-specific parameters. For example, the state splitting reconstruction algorithm has a worst-case time complexity of $O(|\mathcal{A}|^{2L_{\max}+1})+O(N)$ for unifilar and $O(L_{\max}|\mathcal{A}|^{L_{\max}})+O(N)$ for non-unifilar presentations [SHA04]. Here, $L_{\max}$ is the maximum *history length*, the number of past symbols to condition on when creating the causal partition. The runtime growth of both presentations is shown below:

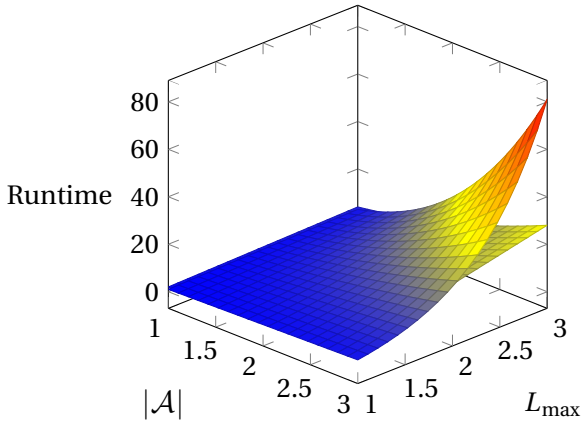

Figure 2.19: Reconstruction runtime as a function of alphabet size and maximum history length for unifilar (top surface) and non-unifilar (bottom surface) presentations

This illustrates that for all but the most simple processes, unifilar presentations require vastly more computation than their non-unifilar counterparts. We explore the tradeoffs between prediction accuracy, alphabet size, and unifilarity that are necessary for reconstruction of complex processes in chapter 3.

CHAPTER 3

Anomaly Detection in High Performance Computing

Intrusion Detection Systems (IDSs) look for policy violations, commonly in the form of cyber attacks, on individual computers or networks of computers. There are two primary paradigms: 1) *misuse detection*, looking for well defined signatures of disallowed behavior, and 2) *anomaly detection*, looking for deviations from statistical models of expected behavior. IDSs must balance *false positive* rates (raising alerts for benign behavior) and *false negative* rates (failing to raise alerts for disallowed behavior). Hybrid approaches combine misuse and anomaly detection in an effort to better address these different requirements.

Misuse detection looks for well-defined signatures derived from patterns of behavior. If the behavior changes even slightly it may no longer match its original signature, and thus many systems will fail to detect the changed behavior. Such systems cannot adapt to new, unseen vulnerabilities without signatures, and thus have a high false negative rate.

Anomaly detection suffers from the complementary problem—while very good at detecting new attacks, benign behavior that strays too far from the model is also labeled anomalous. Such false positives occur so often that administrators routinely ignore or disable many of the alerts generated by anomaly detection systems [PAT07].

This chapter details our work developing an anomaly detection framework specifically for High Performance Computing (HPC), a field concerned with computational solutions to large scale scientific research problems. The massive computational power of these resources makes them potential targets for attackers as evidenced by, for example, the growing black market for encryption cracking services.[1] Other potential abuses include password cracking and computa-

[1] The website wpacracker.com will crack WPA-PSK or ZIP passwords in 20 minutes for $17 using a cloud computing infrastructure.

tion of nuclear simulations.

At the very least, such abuses impede the progress of legitimate research, and at the very worst threaten national security—government labs operate 6 of the 10 most powerful supercomputers in the world. Given the vulnerabilities demonstrated by recent events such as the previously mentioned “Cyber Shockwave”, current mechanisms for protecting HPC resources may be insufficient to prevent their abuse. Our work aims to detect such abuse in commonly used HPC environments, and is being developed and tested at Lawrence Berkeley National Laboratory.

Developing a system without false positives or false negatives that also generalizes to new attacks is the ultimate goal of IDS research, and is an open problem in computer security. By using environmental assumptions and constraints to focus the statistical accuracy of our models, we hope to show that targeted anomaly detection can strike an effective balance between these ideals of generality and specificity. In agreement with recent observations by Sommer et al [SOM10], we claim previous approaches have difficulty precisely because such constraints are missing. And, by taking a more mathematical approach, we can be quantitative both in our design and evaluation of the system.

§3.1 Background

§3.1.1 Message Passing Interface

Message Passing Interface (MPI) is a communications protocol standard used by parallel programs to exchange data. There are many implementations such as OpenMPI and MPICH, but all are based on the idea of logical processors with unique labels called *ranks* communicating in groups called *communicators*. MPI programs have an initialization phase where each processor joins a communicator, is assigned a rank, and learns the size of the communicator, as well as a finalization phase to gracefully terminate.

Consider the following C program for computing π to some number of digits specified on the command line:[2]

```
1  int main(int argc, char **argv)
2  {
3      MPI_Init(&argc, &argv);
```

[2]Code adapted from David Letscher at St. Louis University.

```
4       long terms_total = atol(argv[1]);
5
6       int rank, size;
7       MPI_Comm_rank(MPI_COMM_WORLD, &rank);
8       MPI_Comm_size(MPI_COMM_WORLD, &size);
9
10      double pi;
11      calculate_pi(terms_total * rank / size, terms_total * (rank + 1) / size, pi);
12
13      if(rank == 0)
14      {
15          double data;
16          MPI_Status status;
17
18          for(int i = 1; i < size; i++)
19          {
20              MPI_Recv(&data, 1, MPI_DOUBLE, i,
21                      MPI_ANY_TAG, MPI_COMM_WORLD, &status);
22              pi += data;
23          }
24
25          printf("%.18f", pi);
26      }
27      else
28      {
29          MPI_Send(&pi, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
30      }
31
32      MPI_Finalize();
33  }
```

Lines 3, 7, and 8 perform MPI initialization. The desired number of terms in the decimal expansion is read from the command line into TERMS_TOTAL. Each processor calls CALCULATE_PI and tells the function which one of SIZE chunks to compute based on its rank. After computing its chunk, every rank sends its result to rank 0 on line 29. Rank 0 collects these results on line 20 and adds them to the final output, displayed to the desired precision on line 25.

This example demonstrates a basic pattern of communication: all ranks call MPI_SEND with destination rank 0. Sophisticated applications have more complex communications, and we are interested characterizing their pattern and structure. However, not all attributes[3] of communication may be helpful in classification—some may be redundant, or even detrimental. An information-theoretic approach to evaluating these attributes is presented later. For now, we describe how these attributes are captured from active MPI applications.

[3]From here on, we use the machine learning term *features* to refer to properties or attributes of the objects we wish to classify.

§3.1.2 Communication Logging

The *Integrated Performance Monitoring* (IPM) package is "a portable profiling infrastructure for parallel codes... (providing) a low-overhead performance profile of the performance aspects and resource utilization in a parallel program". It logs features of MPI function calls to an XML file, such as the call name, the ranks involved, and the number of bytes sent, as well as hardware counters such as the number of integer and floating point operations. The library is enabled with a compile-time flag, usually provided to the MPICC compiler wrapper.

Consider the following IPM log entry:

<hent call="MPI_Isend" bytes="599136" orank="1" region="0" count="26" />

These entries become rows in a two dimensional feature matrix, with each column representing a different feature. Each function name is assigned a unique integer so the contents of the feature matrix are purely numerical.

$$\begin{pmatrix} \text{int(MPI\_Isend)} & 599136 & 1 & 0 & 26 \end{pmatrix}$$

The result is a matrix of features for each parallel program we wish to classify, and from this we obtain the full joint feature distribution. The task at hand, then, is how best to extract patterns from this distribution.

§3.1.3 Computational Dwarfs

A computational dwarf[4] is "a pattern of communication and computation common across a set of applications" [Asanovich06]. Each dwarf is an equivalence class of computation, and is invariant to the programming language or numerical methods used for implementation. The common use of shared libraries such as BLAS and LAPACK provides some evidence of these equivalence classes, though dwarfs imply more than simple code reuse.

Colella et al identified seven dwarfs in HPC applications [Colella04], and Asanovich et al asked if these seven also captured patterns from areas outside of HPC [Asanovich06]. They found six additional dwarfs were needed to capture the distinct patterns of computation found in areas such as machine learning, computer graphics, and databases.

[4] Named in reference to the Seven Dwarfs in the story of Snow White.

Dwarfs are quite distinct when visualized as an adjacency matrix. Consider a three node cluster where rank 1 sends messages to ranks 2 and 3, and these send messages back to 1. This leads to the following matrix representation:

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Adjacency matrices are plotted as a grid where the axes are rank numbers, and black pixels denote communicating ranks. Consider the adjacency matrix for the general relativity simulator CACTUS:

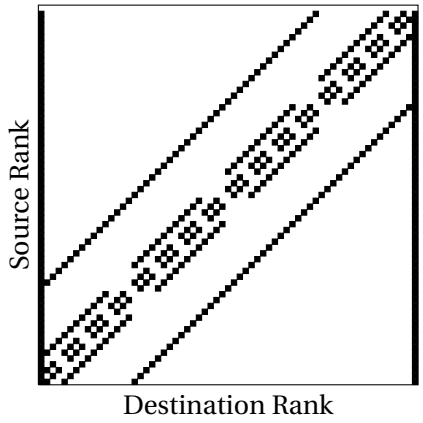

Figure 3.1: Adjacency matrix of a general relativity simulator

as well as the atmospheric dynamics simulator FVCAM:

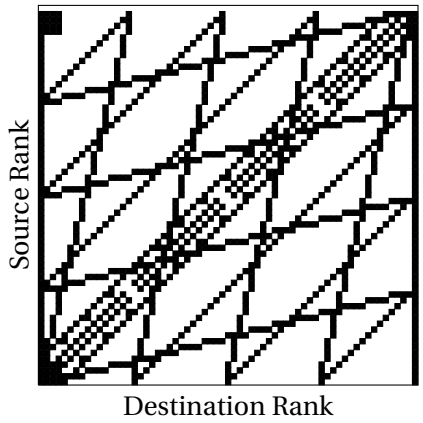

Figure 3.2: Adjacency matrix of an atmospheric dynamics simulator

and the linear equation solver SUPERLU:

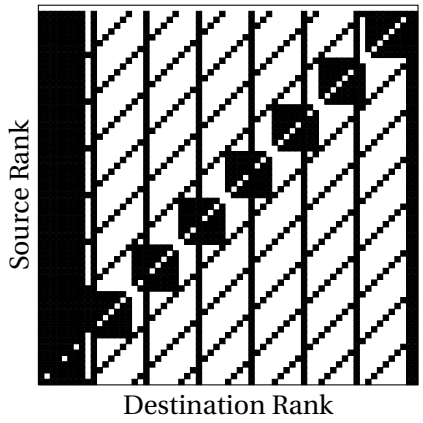

Figure 3.3: Adjacency matrix of a linear equation solver

Communication patterns are strongly tied to memory access within a parallel program. To see this, examine the diagonal of CACTUS and note the communication between a rank and its immediate neighbors. Such a pattern is a signature of finite difference equations and is found across many HPC applications. A different type of equation will have a different signature, unless its pattern of memory access is similar to a finite difference equation.

An additional layer of structure is seen by extending the adjacency matrix into a third dimension. Here we see the MPI call number assigned to the z-axis for CACTUS:

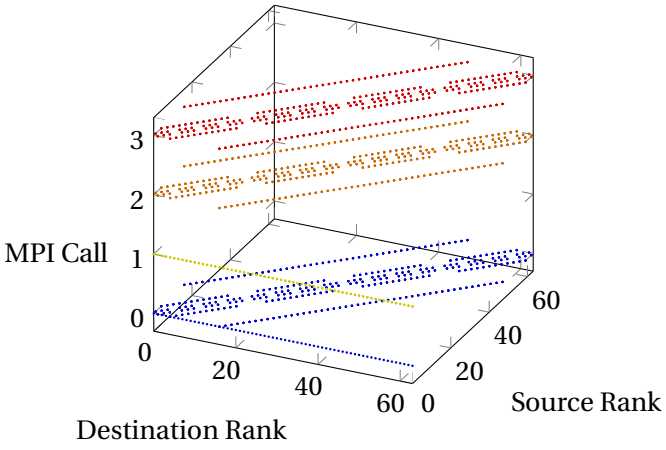

Figure 3.4: Adjacency matrix of a general relativity simulator, augmented by MPI call

as well as the message size:

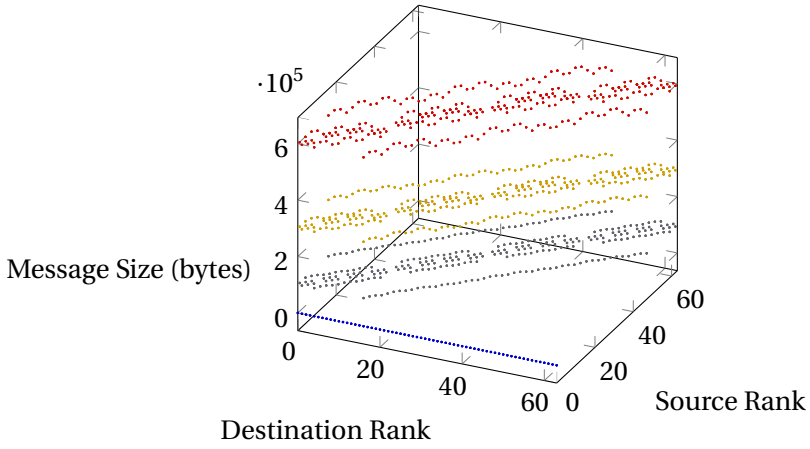

Figure 3.5: Adjacency matrix of a general relativity simulator, augmented by message size

and finally, the number of times a message is repeated:

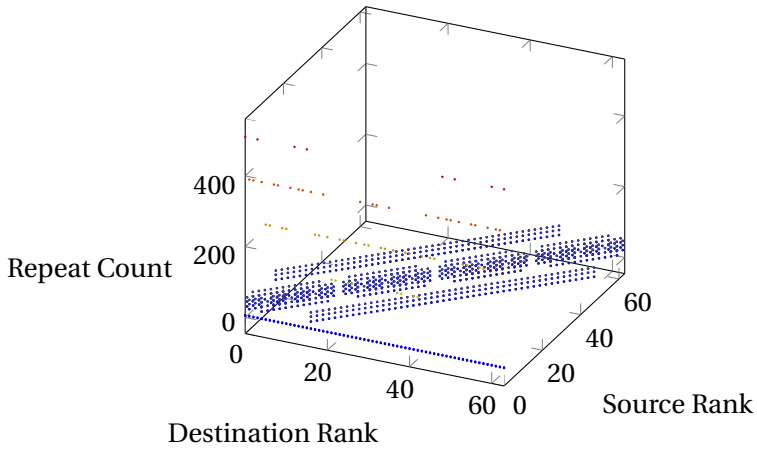

Figure 3.6: Adjacency matrix of a general relativity simulator, augmented by number of repeated messages

These distinct visual patterns suggest that classification of different applications should be possible. By the same argument, however, classification of applications within a particular dwarf class may be difficult due to their similarity. As a result, we must examine these patterns beyond their matrix representations, and for this we first turn to graph theory.

§3.2 Exploratory Data Analysis

§3.2.1 Graph Theory

Treating a communication pattern as a graph with ranks as nodes and messages as edges, we can measure various statistical properties of the graph. The first of these, the node degree distribution, maps a number of edges (the *degree*) to the total number of nodes with that degree. In our

previous 3-node example the node degree distribution is $\{1:2, 2:1\}$, meaning two nodes have one edge and one node has two edges. A distribution for a single run of FVCAM is:

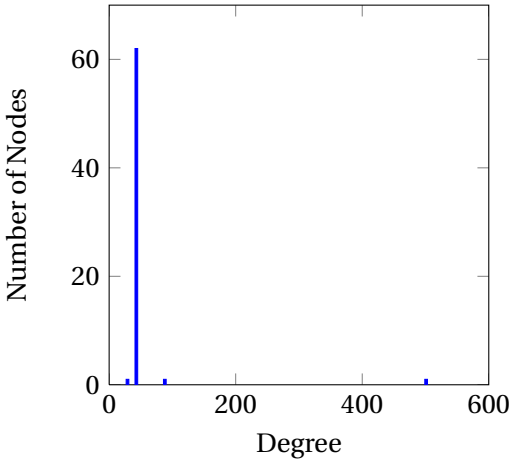

Figure 3.7: Node degree distribution of an atmospheric dynamics simulator

while for the magnetic fusion simulator GTC we have:

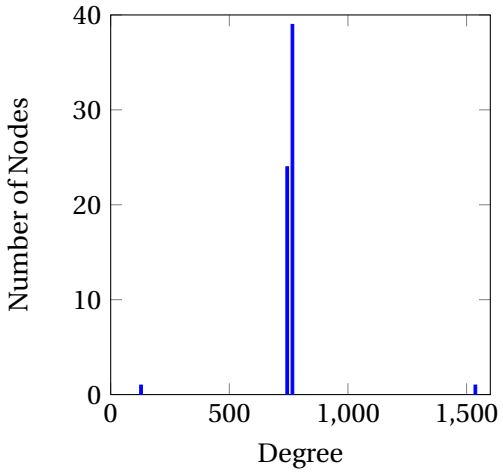

Figure 3.8: Node degree distribution of a magnetic fusion simulator

This difference is notable because the adjacency matrices of FVCAM and GTC are nearly identical, and therefore are insufficient as a basis for classification.

In addition to which ranks communicate, the degree distribution captures the types of messages exchanged. In the example above, the ranks of GTC send many different types of MPI calls to their neighbors, resulting in a higher average node degree than FVCAM. Having the degree distribution, then, is akin to summing over a set of per-call adjacency matrices as in figure 3.4.

Though more informative than adjacencies alone, node degree distributions summarize a

single aspect of the graph that may fail to distinguish different patterns. In these cases, calculating the *centrality* of the nodes can be helpful. Centrality measures the importance of a node in the graph, and this importance can be defined in several ways. We look at the *betweenness centrality* (C_B), measuring the percent of shortest paths passing through a node v in an undirected graph [FRE77]]:

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where $\sigma_{st}(v)$ is the number of shortest paths between nodes s and t that pass through v. This number is normalized by the total number of shortest paths between s and t. The C_B of v, then, is the sum of these normalized shortest path counts over all node pairs not containing v.

Intuitively, nodes acting as coordinators of computation should have high C_B. There will be at least two such nodes in most MPI applications—rank 0, and a pseudo-rank assigned to broadcast messages. The distribution of C_B for two runs of the MADBENCH performance benchmark exhibits these two highly central nodes, and demonstrates how graphs with similar degree distributions can have very different centrality distributions:

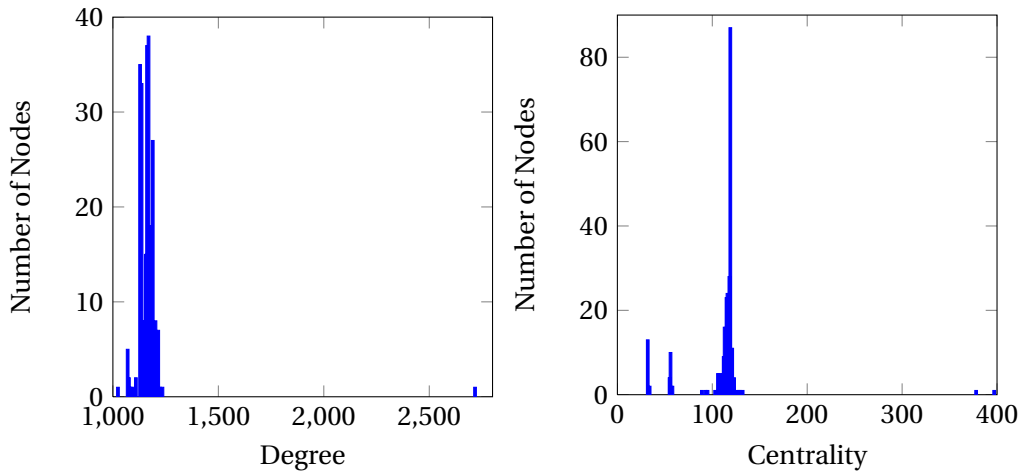

Figure 3.9: Betweenness centrality distribution of a performance benchmark

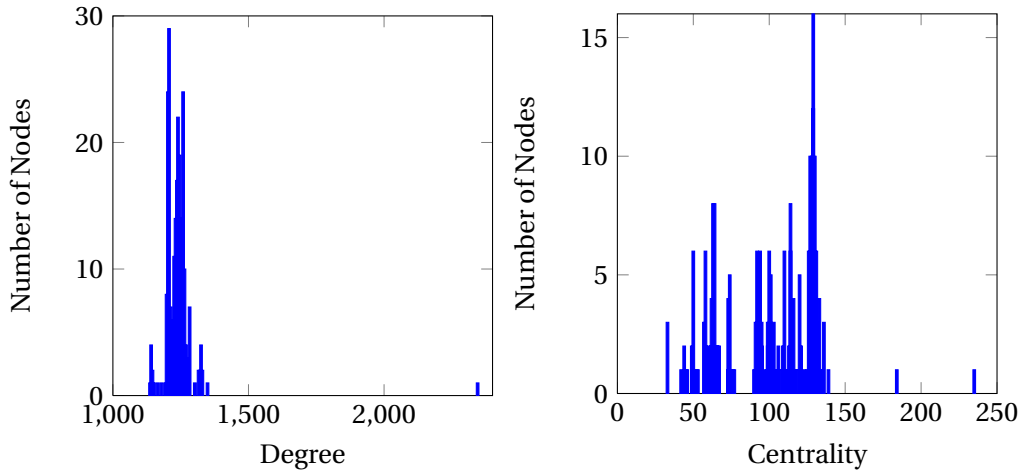

Figure 3.10: Alternate betweenness centrality distribution of a performance benchmark

Graph theory provides multiple ways of distinguishing parallel computation based on a limited subset of communication features. Though they may eventually be used for classification, we use them now primarily for intuition and instead focus on methods that incorporate the complete set of communication features.

§3.2.2 Self-Organizing Maps

A common first step towards understanding the structure of a high dimensional dataset is assigning each element to some subset of related elements, a process called *clustering*. One such clustering algorithm is k-means, which creates a random set of k initial clusters and iteratively assigns each element to the cluster whose center is deemed "closest" by some distance metric [MAC02]. The center of a cluster is the average of all its elements, and is re-computed any time a new element is added. A generalization of this approach is the fuzzy c-means algorithm that allows each element to belong to multiple clusters, reflecting the uncertain membership of elements near cluster boundaries.

The Iris dataset collected by R.A. Fisher [FIS36] is a classic in the machine learning literature. It consists of 150 four-dimensional feature vectors containing measurements of the sepal length, sepal width, petal length, and petal width for three species of Iris flower. One possible k-means clustering[5] for $k = 3$ is below:

[5]Since the initial clusters are chosen randomly, different clusterings can result.

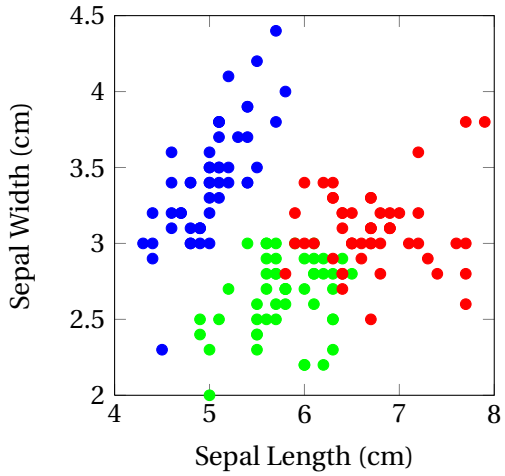

Figure 3.11: Clustering of the Iris dataset using k-means with $k = 3$

Here, the goal of clustering is gaining intuition about the separability of communication patterns. This separability is revealed by the proximity and shape of each cluster. In the Iris data, we can see the blue cluster is mostly well-defined. However, if we were to draw lines separating the elements of each cluster, we could not use straight lines to divide the red and green elements. Such data is not *linearly separable*, and requires more complex techniques for accurate classification.

Self-Organizing Maps (SOMs) perform clustering and dimensionality reduction [Koh82], mapping elements of a high-dimensional input space onto a two dimensional grid. This mapping preserves the topological properties of the input space, meaning "close" elements in the input space are also close on the grid. It has been shown that SOMs are a nonlinear generalization of Principal Component Analysis and that extremely small maps behave similarly to k-means [Koh00].

The map is initially comprised of random vectors, each having the same dimension as the input space. Training is an iterative process of selecting a random input vector, finding the closest vector in the map, and moving this closest vector and its neighbors towards the input vector.[6] The number of neighbors, and the amount they are adjusted, decreases each training iteration until the map converges or a maximum number of iterations is reached.

Algorithms such as k-means must be given the number of clusters. In contrast, SOMs infer this number from the data. These inferred clusters, in addition to feature correlations, are easily

[6]This paradigm is called *competitive learning*, as each map vector "competes" to be closest to the input vector.

visualized using the two dimensional map. This makes SOMs well-suited for exploratory data analysis.

SOMs are a highly effective visualization tool, and as such, we want to see the inferred data clustering. If the input space is three dimensional, the map can be visualized as a grid of pixels where each trained vector represents red, green, and blue components. For input spaces beyond three dimensions, as is normally the case, there is no such convenient mapping. Here we compute the *U-matrix* [KOH82], storing the average distance between each map vector and its nearest neighbors. These distances are then displayed as intensities in a color-mapped grid where cluster boundaries appear as high intensity curves. Looking at the U-matrix for the Iris dataset:

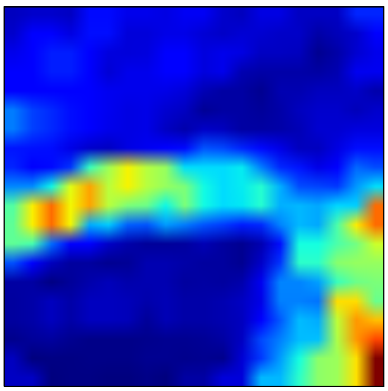

Figure 3.12: U-matrix of the Iris dataset

two clusters are identified, separated by a high intensity curve in the center. Since the data is not linearly separable, the map cannot distinguish the third cluster.

For further intuition, the U-matrix of a SOM trained with random data is given below:

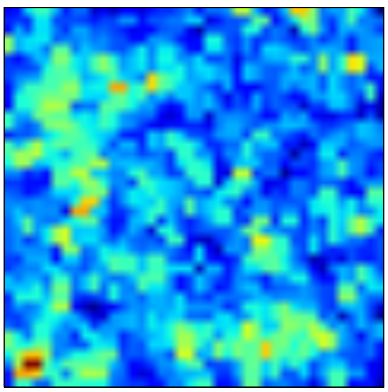

Figure 3.13: U-matrix of a uniformly random dataset

No clear clusters emerge and the distance between most vectors and their neighbors is relatively

large. For structured data of sufficient complexity, one might expect a U-matrix somewhere between randomness and order. Indeed, this is seen by the U-matrix of CACTUS:

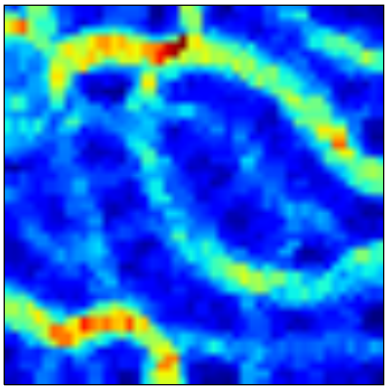

Figure 3.14: U-matrix of a Cactus dataset

Though the cluster boundaries are not rigorously defined, we can see the communication pattern has approximately four clusters. Thus, the U-matrix gives a qualitative view of how high dimensional data is clustered, without knowing the number of clusters a priori.

Correlations between features are visualized by slicing the map along a particular dimension. For example, if the number of bytes sent is the third element of our 5-dimensional input vectors, then viewing only the third dimension of the map shows how the inputs cluster according to byte count. Again for CACTUS, these per-feature layers are shown below:

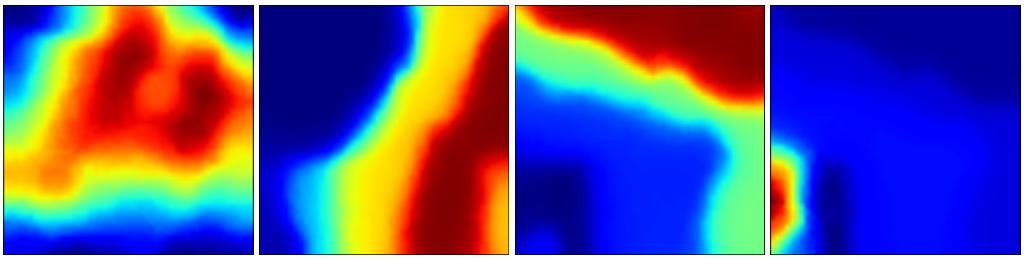

Figure 3.15: Feature layers of a Cactus dataset including source rank, MPI call, bytes sent, and repeat count

From left to right these are “source rank”, “MPI call”, “bytes sent”, and “repeat count”. Examining the southwest corner of the “repeat count” layer reveals a high intensity region corresponding to often-repeated messages. Since the map preserves topology, looking at the intensity of the same region in the “bytes sent” layer tells us that repeated messages have small byte counts.

In concert with the U-matrix, these feature layers make SOMs a powerful tool for visualizing the clustering structure and feature correlations of high dimensional datasets. For our purposes they are a qualitative tool, useful for further demonstrating the level of structure in our commu-

nication datasets. Having now established this structure using several methods, we next discuss a more quantitative analysis, and finally present our preliminary classification results.

§3.2.3 Pairwise Distances

The KL divergence, introduced in section 2.3.5, measures the distance in bits between two discrete probability distributions. Computing pairwise distances between MPI message distributions allows us to quantify the similarity of their generating programs. Ideally, the distance between two runs of the same program with different datasets or number of processors should be very low to reduce false positives due to normal statistical fluctuations.

Though these pairwise distances can be summarized in a table, they are quick to visualize as concentric circles. The radius of each circle is equal to the distance between the distribution of messages for two programs. Circles are semi-transparent and become more opaque where there is overlap, giving a visual estimate of the distance distribution.

Given below is a distance diagram for the molecular dynamics simulator PMEMD using 64 processors:

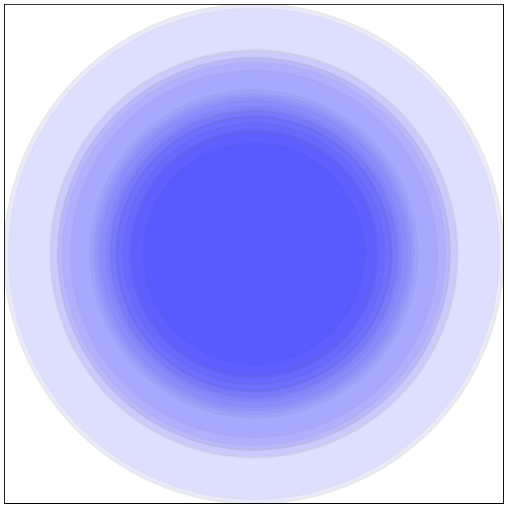

Figure 3.16: Pairwise relative entropies between a molecular dynamics simulator and all other datasets

Again, each circle's radius represents the D_{KL} between the message distribution of PMEMD and some other parallel application from our datasets. The exact D_{KL} values for the identity, the closest program, and the furthest program:

$$D_{KL}(\text{pmemd} \,||\, \text{pmemd}) = 0$$

$$D_{KL}(\text{pmemd} \,||\, \text{paratec}) = 5.79$$

$$D_{KL}(\text{pmemd} \,||\, \text{gtc}) = 13.13$$

This suggests PMEMD is distinguishable from other applications by using the joint distribution of communication features. This is not always the case, as with the quantum mechanics simulator PARATEC:

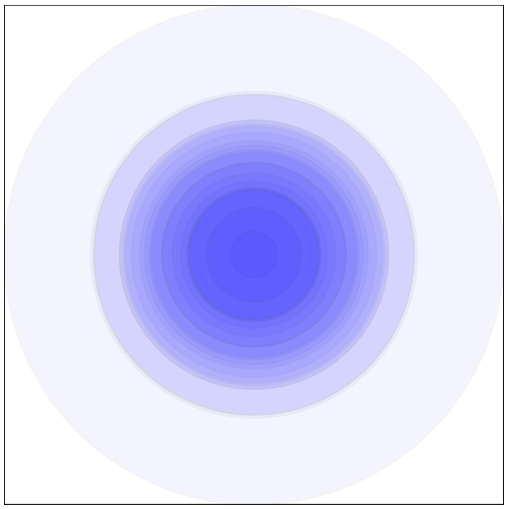

Figure 3.17: Pairwise relative entropies between a quantum mechanics simulator and all other datasets

Here the smallest D_{KL} is 1.8 bits to SUPERLU, which may be within the normal fluctuations of PARATEC. Thus, we will need features such as hardware counters in addition to purely message-based data to accurately classify some patterns.

§3.3 Classification

Classification is a machine learning task that assigns labels to members of a dataset, where each label corresponds to a subset of members with similar features. An algorithm performing this task is a *classifier*. A dataset is typically partitioned into a *training set* of labeled examples to train the classifier, and a *test set* for evaluating how well the classifier predicts the labels of new data.

A classifier's accuracy with the test set gives some indication of how well it *generalizes* to new data, as typically a small percent of all possible data is available for training. One simple evaluation of accuracy is to divide the number of correctly predicted labels in the test set by the total size of the dataset. Perfect accuracy is often a sign of *over-fitting* and results from highly

dimensional data, insufficient amounts of data, numerous classification parameters, or some combination thereof. Thus, over-fitting and generalization are opposing forces, and the former should be avoided when constructing a classifier.

A method of evaluating generalization is *k-fold cross validation*. Instead of a single partition, the data is split into k chunks and the classifier is trained and evaluated k times.[7] Each iteration, a different chunk is left out of training and instead becomes the test set. The average prediction accuracy over these different test sets gives a better measure of generalization than any single partition.

This section presents our preliminary results for three progressively more complex classifiers: Naive Bayes, Hidden Markov Models, and ϵ-machines. We also discuss the identification of redundant features using information theory, which leads to lower dimensional data and thereby reduces the risk of over-fitting.

§3.3.1 Naive Bayes

Bayes' Theorem gives the probability of some random event A given the occurrence of event B as follows:

$$\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B)}$$

This simple theorem is the cornerstone for Bayesian inference, a framework for reasoning about evidence in support of a hypothesis. Updating the equation to reflect this interpretation for evidence E and hypothesis H gives:

$$\Pr(H|E) = \frac{\Pr(E|H)\Pr(H)}{\Pr(E)}$$

Each of these terms has a name, and these will be used in place of the mathematical notation:

$$\text{posterior} = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}$$

In the context of classification, we compute the posterior probability of different models given some sequence of input data. We then select the model with the highest posterior probability as the label, and evaluate our prediction against the actual label from the test set.

[7] A common value for k is 10.

Computing the posterior requires multiplying the likelihood of the data given the model by the prior probability of the model itself. The evidence is often ignored since it remains constant. For our joint set of communication pattern features, the calculation becomes:

$$\Pr(\text{feature}_1, \text{feature}_2, \ldots, \text{feature}_n | \text{pattern}) \Pr(\text{pattern})$$

Features are "naively" assumed to be conditionally independent, so the above factors into:

$$\Pr(\text{pattern}) \prod_{i=1}^{n} \Pr(\text{feature}_i | \text{pattern})$$

Previous analysis showed that our features are not conditionally independent. In fact, this simplifying independence assumption is rarely true in practice, resulting in the classifier's pejorative name. Therefore, our primary use of the Naive Bayes classifier is a null hypothesis: classifiers with more complete knowledge should perform at least as well, and if not, some explanation is in order.

Computed over 17 datasets, including some collected from the same program using different numbers of processors, our Naive Bayes classifier has 82.2% accuracy. Considering the approach completely ignores feature correlations,[8] this sets a rather high bar for our next classifier using Hidden Markov Models.

§3.3.2 Hidden Markov Models

Hidden Markov Models are probabilistic state machine models of stochastic processes. By decoupling the internal state of the model from the symbols it generates, HMMs are useful for a variety of tasks including classification. The details of the model can be found in section 2.5.

Of primary concern when utilizing HMMs is selecting the model architecture, including the number of states and their connectivity. State transition and symbol emission probabilities can be trained to maximize the likelihood of generating some set of input data, or estimated directly if the hidden state sequence of each training sequence is known. We use the latter approach since this information is available to create a left-right HMM for each MPI communication pattern.

Emission probabilities are maximum likelihood estimates of conditional feature distributions. By extending this conditional dependence to L past observations, the model leverages

[8] Despite the weakness of its independence assumption, Naive Bayes is often surprisingly good in practice. Among other reasons, estimating marginal instead of joint probabilities increases the accuracy of the estimate.

knowledge of feature correlations for label prediction. This requires the joint distribution over length $L+1$ blocks, and exponentially more data for larger L. The maximum model order is somewhat limited as a result, and we compromise with $L=6$ in order to span one complete set of communication features.

Once the HMMs are constructed, the model with the highest posterior probability for an observation sequence is selected as that sequence's label. This requires computing the likelihood term of an observation sequence using the forward algorithm [RAB89]. A vector of *forward probabilities* is created where each element $\alpha_t(i)$ is the probability of being in state i after generating t observations. The initial value of $\alpha_1(i)$ is the stationary probability π_i multiplied by the emission probability B_i for the first observation x_1:

$$\alpha_1(i) = \pi_i B_i(x_1)$$

These forward probabilities are updated for each time-indexed observation, where $|S|$ is the number of states and A is the transition matrix:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{|S|} \alpha_t(i) A_{ij}\right] B_j(x_{t+1})$$

This multiplies the current forward probability of state i with the probability of transitioning to state j and emitting the next observation. The per-state forward probabilities are summed at the end, giving the likelihood that the observations were generated by the model:

$$\Pr(x_1 \ldots x_T | \text{model}) = \sum_{i=1}^{|S|} \alpha_T(i)$$

To obtain the posterior, this likelihood is multiplied by the model's prior probability given by the percentage of inputs in the training data mapped to that model.

Using the set of posteriors, the most likely HMM is selected as the sequence's label. This yields an average accuracy of 98.7% over all MPI communication classifiers, a standard deviation of 0.01%, and a sizable improvement over Naive Bayes (see figure 3.18).

ϵ-machine Classification		
Dataset	Size (kb)	Accuracy (%)
cactus-64	57	100
fvcam_1d	33	100
gtc-64	537	98.9
lbmhd-64	20	1.0
mdh2d	160	99.1
paratec	1289	97.2
slu-64	2646	96.0

Figure 3.18: Accuracy of Hidden Markov Model classifiers.

§3.3.3 ϵ-Machines

Introduced in section 2.6 is the ϵ-machine, a subset of traditional HMMs with several unique properties. As a type of HMM, the same approach is used to classify communication patterns using ϵ-machines: construct a model for each pattern, compute the posterior probability of each model for some new input, and select the model with the highest posterior as the input's label. The likelihood function for ϵ-machines is given in equation 2.6, and is essentially the forward algorithm with per-symbol transition matrices.

There are two primary advantages of ϵ-machines over traditional HMMs: 1) the number of states and their transitions are inferred from the data instead of specified by the modeler, and 2) in the limit of infinite data, an ϵ-machine is the optimal predictor of a process with the fewest number of states. In practice we have only finite amounts of data, and so this claim of optimality is approximated to varying degrees.

To model with ϵ-machines, the underlying process must be stationary and have some temporal or spatial structure. As our data is unordered, we cannot use the temporal structure of MPI calls, and instead the ϵ-machine captures the non-temporal structure of individual call features. Thus, we are using ϵ-machines in an atypical way due to data-imposed limitations, and the advantages of the model class will not fully emerge until this is remedied.

Due to the time complexity of the algorithm, large alphabets are an obstacle to ϵ-machine reconstruction. As a result, we employ equal-frequency binning [WIT05, GUY03] to reduce the alphabet size. This takes the range between the minimum and maximum values of a feature and divides it into a number of variable sized intervals (bins) such that each bin contains an equal

number of values. Each value is then replaced by its bin number. We used 32 bins as it was the largest divisor of our minimum cluster size that resulted in reasonable reconstruction times.

Using binning and non-deterministic presentations with a history length of 3, classification with ϵ-machines had 93% accuracy with a 5% standard deviation (see figure 3.19. While lower than the previous result, it should be noted that the alphabet size and model order is half that of our HMMs. In fact, when using the same preprocessing techniques, HMM accuracy fell to 87% with a standard deviation of 7%—on equal footing, ϵ-machines performed better. Accuracy could be further increased by using deterministic presentations and longer history lengths, though this was not rigorously evaluated due to speed limitations of our Python implementation.

ϵ-machine Classification

Dataset	Size (kb)	Accuracy (%)
cactus-64	57	97.7
fvcam_1d	33	82.7
gtc-64	537	99.2
lbmhd-64	20	87.1
mdh2d	160	95.1
paratec	1289	95.2
slu-64	2646	93.8

Figure 3.19: Accuracy of ϵ-machine classifiers.

§3.3.4 Feature Selection

All features are not created equal; some may improve accuracy more than others, or even cause a reduction. In addition, each feature increases the dimensionality of the data, compounding the "curse of dimensionality" [DON00]. Therefore, a systematic approach to selecting the most informative features is important both for classification speed and generality.

There are two primary approaches to feature selection: variable ranking and variable subset search [GUY03]. The former, called a *filter*, ranks features by some correlation criteria. The latter, called a *wrapper*, searches through feature space and evaluates the predictor's accuracy on different subsets. Filters avoid predictor bias and are generally considered faster, while wrappers offer simplicity and independence from the biases of correlation criteria.

Information gain is another name for mutual information, and is a correlation criterion often used in machine learning. It has also found some application in security research [LEE01,

KAY05A]. This quantity, introduced in section 2.3.4, measures the reduction of uncertainty in variable X given knowledge of Y. For example, if Bob and Alice eat lunch together except when Alice is out of town, then knowing that Alice is in town removes all uncertainty about who Bob ate lunch with. If Bob's lunch date and Alice's travel status were variables X and Y, we only need the outcome of Y to completely determine X. Thus, we say X is redundant. The mutual information I is a measure of this redundancy, and is depicted by the overlapping region in the information diagram of figure 2.7. The more information gained about X by knowing Y, the larger this region.

As noted earlier, mutual information varies with the size of a random variable's event space. Instead, we use the *symmetric uncertainty* to obtain a normalized measure of redundancy between features [WIT05]:

$$D = 1 - \frac{I[X;Y]}{\max(H[X], H[Y])}$$

Symmetric uncertainty was calculated over pairwise features for each program. The "region code" feature was found to be completely redundant, and was later identified as an IPM logging attribute not used in our datasets. Other redundancies varied by program: "byte count" and "repeat count" were almost completely redundant for GTC but nearly independent for FVCAM, and such conflicting results were common. As additional features such as hardware counters become available, we will perform a more rigorous feature selection using both the filtering and wrapping techniques, in hopes of identifying optimal subsets despite such conflicts.

§3.4 Related Work

Florez et al [FL05B] built neural network and HMM classifiers using both MPI calls and system calls. They used a Linux cluster with 4 processors and a training set of 2 parallel applications to classify sequences as anomalous or normal. Anomalous behavior was created by modifying normal datasets. In contrast, we use clusters of 64 and 256 processors running roughly 20 parallel programs under various workloads, and more formally select model parameters and features using information theory.

Kamil et al [KAM10] characterized the communication patterns of 8 parallel programs using IPM logs. They evaluated the utilization of hardware interconnects in fully connected networks, and concluded these interconnects are underutilized by typical parallel workloads. Utilization

was measured by the distribution of buffer sizes for collective and point-to-point communication, as well as the distribution of MPI calls and the communication topology. The latter involved visualization of the rank adjacency matrix.

Ma et al [MA09] introduce a communication correlation coefficient to characterize the similarity of parallel programs using several metrics. The first metric involves the average transmission rate, message size, and unique neighbor count for each logical processor. The second computes the maximum common subgraph using an NP-hard graph isomorphism algorithm; an approach designed for invariance to the underlying architecture. Their evaluation was limited to 4 programs in the NAS parallel benchmark, and the choice of features for the correlation coefficient was not discussed.

§3.5 Conclusion and Future Work

The preceding sections explored various structural properties of MPI communication patterns. The ultimate goal of this exploration was to inform the construction of machine learning classifiers that correctly map communication patterns to their originating parallel programs.

Graph-theoretic properties such as the adjacency matrix, node degree distribution, and betweenness centrality distribution revealed patterns of rank connectivity. SOMs gave an approximate number of clusters and showed how different features were correlated. Finally, we looked at a quantitative measure of structure using relative entropy, and progressively constructed three machine learning classifiers.

Our classifiers, while accurate for the current datasets, will not generalize due to the unordered nature of the data. Without ordering, classifiers can only label individual MPI inputs since consecutive calls in the IPM logs are not consecutive in time. We expect the same approach with ordered data will produce generalized classifiers with low cross-validation error.

In addition to building and evaluating classifiers, we also examined the usefulness of each communication feature by computing pairwise redundancies using information gain. It was found that one feature, the region code, was entirely redundant and thus was eliminated to reduce the dimensionality of the data. We will further explore the use of feature selection to identify optimally predictive features as hardware counter information becomes available. This is

expected to arrive alongside ordered datasets in the next phase of research at Lawrence Berkeley National Lab.

CHAPTER 4
Automated Protocol Reverse Engineering

Hidden Markov Models (HMMs) have applications in several areas of computer security. One drawback of HMMs is the selection of appropriate model parameters, which is often ad hoc or requires domain-specific knowledge. While algorithms exist to find local optima for some parameters, the number of states must always be specified and directly impacts the accuracy and generality of the model. In addition, domain knowledge is not always available or may be based on assumptions that prove incorrect or sub-optimal.

We apply the ϵ-machine—a special type of HMM—to the task of constructing network protocol models solely from network traffic. Unlike previous approaches, ϵ-machine reconstruction infers the minimal HMM architecture directly from data and is well suited to applications such as anomaly detection. We draw distinctions between our approach and previous research, and discuss the benefits and challenges of ϵ-machines for protocol model inference.

§4.1 Introduction

Understanding the structure of a network protocol allows us to "speak" its language and converse with other systems on the network that use it. The structure of commonly used protocols, such as HTTP and FTP, are provided by their specification. In addition, these protocols use fragments of English as well as other ASCII text such as domain names. As a result, the presence of HTTP or FTP traffic can be identified by visual inspection of a network trace, assuming the channel is not encrypted. One can then use its specification, or one of many free or commercial tools, to understand the traffic present in the trace.

The task becomes more difficult when the protocol in question uses non-ASCII representations of state to establish connections and exchange data. Still, there are many approaches to

identify the protocol such as using port numbers, unique payload signatures, or machine learning techniques [ERM06]. Once the protocol is identified, the traffic can again be understood by using the specification.

In contrast to protocol identification, consider the scenario where we do not have access to the protocol's specification—it is either proprietary, undocumented, or otherwise obfuscated. We can treat the protocol as a black box, where a hidden state machine governs the transmission of packets on the network. To understand the structure of the packets, the task of protocol inference is to approximate this hidden state machine with only the observed packets as a guide.

Hidden Markov Models (HMMs) [RAB89] are a common statistical model for systems with hidden internal states that can be measured only indirectly by observation. These models have numerous applications in computer science, including several in computer security. An HMM is specified by a state transition matrix and a symbol emission matrix. This means that, for an N-state HMM with a discrete alphabet of size M, there are $N(N-1)+N(M-1)$ free parameters to be specified. These parameters can be trained using the Baum-Welch algorithm [RAB89], but training is often slow and gets stuck in local optima. In addition, the number of states must still be specified. A model with too many states tends to over-fit the data, while too few states may not fit the data at all. Worse, when dealing with an unknown protocol, there is little if any knowledge available for selecting appropriate model parameters.

Here, we treat a network protocol as a stochastic process and the traffic it generates as input strings for the reconstruction algorithm. Our approach has several advantages over previous work: 1) probabilistic modeling enables anomaly detection and traffic generation, and 2) ϵ-machine reconstruction avoids ad hoc specification of model parameters. Protocol ϵ-machines also capture previous work involving protocol mimicry and intelligent fuzzing. For an overview of HMMs and ϵ-machines, see sections 2.5 and 2.6.

We next discuss recent work done on protocol inference and the benefits of our approach. We follow this with background on HMMs and ϵ-machines, and demonstrate our reconstruction technique using several simple protocols. Finally, we discuss future applications and the limitations of our approach.

§4.2 Related Work

Current approaches to protocol inference can be divided into two primary groups: those that infer partial or complete protocol formats [BED05, CUI06, CUI07, LIN08, WON08, CAB09], and those that infer a state machine model [LEI05, COM09]. Both groups can be further divided into those that examine network traces [BED05, LEI05, CUI06, CUI07], and those that additionally examine how a protocol implementation processes those traces [LIN08, WON08, CAB09, COM09]. Each approach has different strengths and weaknesses, but both must identify the location and size of protocol headers.

Much of the recent work can be traced back to Protocol Informatics, which attempted to "determine the location and length of fields within protocol packets" using sequence alignment algorithms typically found in bioinformatics [BED05]. This approach was extended by RolePlayer, which used heuristics to identify the locations of IP addresses and domain names in a packet, in order to "successfully replay one side of a [protocol] session" [CUI06].

This work led to Discoverer, which focused on "reverse engineering the [complete] message format specification" [CUI07]. In this work, Cui et al. found that selecting robust parameters for sequence alignment was difficult, and that alignment has trouble identifying variable length fields in messages of the same format. In response, they developed a type-based sequence alignment algorithm that infers the semantics of different fields, and used these semantics to cluster messages of the same format. Inference of the state machine, which is the focus of our approach, was left to future work.

Prospex addressed this issue by inferring non-probabilistic state machines from execution traces of a protocol implementation [COM09]. Their state machine "reflects the sequences in which messages may be exchanged". They converted their machines into input specifications for the fuzzing tool Peach, and found several known and unknown flaws in open source software.

In contrast, our ϵ-machine approach infers the minimal HMM from passively observed network data without using execution traces. This strikes a middle ground between Discoverer and Prospex, with several unique contributions. These include using a probabilistic model that enables anomaly detection via model comparison techniques, and avoiding ad hoc specification of model parameters by inferring them from the data. For an overview of HMMs and ϵ-machines,

see sections 2.5 and 2.6.

§4.3 Protocol Inference

We first define a network protocol as a set of *message types*. Each message type consists of a sequence of bits, and related bits are often grouped into *headers*. A particular message type may contain a set of headers, as well as a data *payload*. This payload may contain data provided by the user, or may *encapsulate* messages of a higher level protocol. Thus, we can think of a protocol message at several levels of abstraction: as a sequence of bits, bytes, or headers and payloads.

As discussed in chapter 3, ϵ-machines have numerous advantages over Markov and Hidden Markov Model representations of a stochastic process, but their practical application is limited primarily by the alphabet size $|\mathcal{A}|$ of the process. By changing the level of abstraction, we can adjust the order of the underlying Markov Chain as well as its alphabet size. In addition, we are interested in the structure of the protocol and not highly entropic user data such as images or movies, so we attempt to detect and ignore payloads. This further reduces the alphabet size, and is essential to ϵ-machine reconstruction.

Consider a minimal protocol having a single message type, consisting of a 2-byte length header and a payload. The length header specifies the number of bytes in the payload as an unsigned 16-bit integer. A four byte message sending the ASCII characters for “no” could then be viewed as a sequence of bits:

00000000 00000010 01101110 01101111

or of bytes:

0 2 110 111

or of headers and payloads:

2 “no”

This binary sequence has $|\mathcal{A}| = 2$, but requires a prohibitive order-16 model to capture the first header. At the byte level this becomes order-2, but $|\mathcal{A}|$ increases to 4. Finally, if we know where the separation between header and payload is, we can use an order-1 model with $|\mathcal{A}| = 2$. If more messages are observed, the alphabet size of the byte representation could increase to 256, so operating at the header level is desirable. Of course, knowing the location and size of message headers requires either heuristics or access to the protocol specification.

Protocols such as HTTP and FTP use ASCII tokens, so header boundaries are easily identified by inspection—typically tabs, spaces, newlines, and carriage returns. For more difficult binary protocols, Beddoe aligns bytes across different messages using bioinformatics algorithms and then uses simple statistics as a boundary detection heuristic [BED05]. Cui et al discuss difficulties with sequence alignment and devise a significantly improved type-based alignment algorithm [CUI07]. For simplicity, we use minimum entropy clustering [LI04] to group messages of the same type and apply the approach of Beddoe to identify likely header boundaries and payloads.

Thus, our inference approach can be separated into three primary tasks: 1) grouping bytes into headers, 2) filtering highly entropic data, and 3) reconstructing the ϵ-machine. Tasks 1 and 2 exist primarily to reduce the alphabet size, and can be changed independently of task 3. For example, type-based alignment could be exchanged with minimum entropy clustering to transparently improve the accuracy of the inferred model. Our heuristics are adequate for the protocols discussed here, but complex protocols may require additional sophistication.

§4.3.1 ICMP

We introduce protocol ϵ-machines and their accompanying notation using two simple binary protocols, followed by two more complex protocols. The first of these, the Internet Control Messaging Protocol (ICMP) [POS81] defines several message types used for network diagnostics. One of these types, the echo request/reply, finds common use in the ping command line utility bundled with most operating systems. For this discussion we focus on echo requests, consisting of a 1-byte type set to `0x80`, 1-byte code set to `0x00`, 2-byte checksum whose contents are a function of the message, 2-byte identifier, 2-byte sequence number, and zero or more bytes of payload.

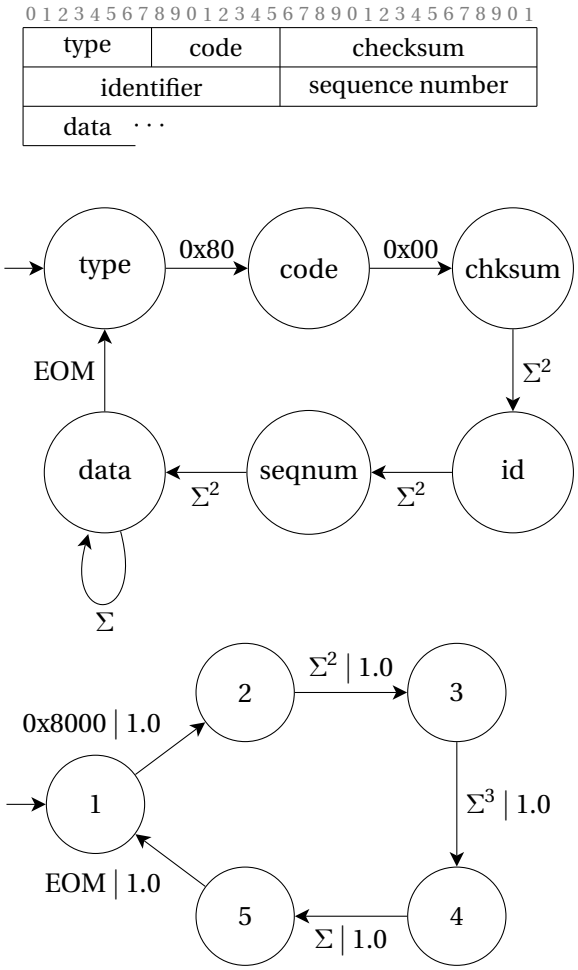

Figure 4.1: *Top:* Specification of an ICMP echo request [Pos81], *Middle:* HMM representation, *Bottom:* ϵ-Machine representation. The symbol Σ represents a random byte of data, with Σ^n denoting n consecutive random bytes.

The message specification, a corresponding 6-state HMM, and the ϵ-machine are shown in figure 4.1. A state is created in the HMM for each header, with transitions between states whose headers are adjacent in the specification. Transitions are labeled with the symbols to be generated. The symbol Σ^n denotes n consecutive random bytes, and the symbol `EOM` signals the message is complete and ready for transmission.

The ϵ-machine inferred from captured echo requests is shown below the HMM. Transitions are labeled with the symbol s generated by the transition and the probability p of the transition being taken, denoted $s \mid p$. This intentionally simple example has no branching between states, resulting in transition probabilities of 1. The type and code headers are constant values, causing their separate HMM states to be merged in the ϵ-machine.

Many protocols contain a sequence number header represented as a 16-bit integer. However, the first byte of this header changes very rarely compared to the second byte that is incremented with each message. In the requests captured for this example, the identifier header and the first byte of the sequence number remained constant, resulting in their grouping into a single value by the boundary detection heuristic (see the $\mathcal{A}^3$ transition between state 3 and 4). While this does not match the specification, it is a reasonable grouping to make based solely on the statistics of the observed messages. Given enough data, the bytes will be grouped into the correct headers..

§4.3.2 Modbus

We next examine Modbus, a protocol commonly used in supervisory control and data acquisition (SCADA) systems for managing industrial and infrastructure processes such as power generation and waste management. Designed in the late 70s to operate on simple programmable logic controllers, Modbus has gained recent notoriety due to a lack of security in the Modbus/TCP variant that connects these systems to standard TCP/IP networks [MOD06].

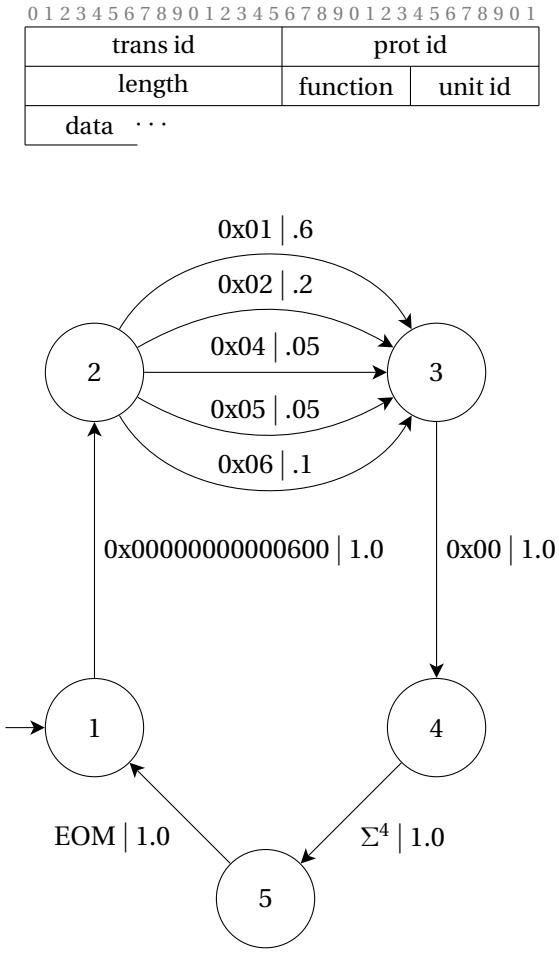

Figure 4.2: *Top:* Specification of a Modbus/TCP request [MOD06], *Bottom:* ϵ-Machine representation.

The specification of Modbus/TCP requests and an inferred ϵ-machine are shown in figure 4.2. A request consists of a 2-byte transaction id, 2-byte protocol id set to `0x00`, 2-byte length, 1-byte unit id, 1-byte function code, and variable length payload.

The captured traffic, generated by a protocol simulator, consists of 150 requests where all transaction ids and unit ids are set to `0x00`. Traffic generated by the simulator is valid Modbus traffic, and not a statistical approximation. Observed function codes include `0x01` for reading coils, `0x02` for reading discrete inputs, `0x04` for reading input registers, `0x05` for writing single coils, and `0x06` for writing single registers. Each payload contains 4 bytes specifying the range of coils or registers to read or write. Branching occurs between state 2 and 3, with each transition probability representing the maximum likelihood estimate of a different function code.

§4.3.3 FTP and HTTP

Reconstructing more complex protocols such as FTP and HTTP is also possible. Model size prevents including the full ϵ-machines here, so instead we present summaries of their reconstruction in figure 4.3. Shown is the scaling of inferred states and reconstruction time as a function of data length, performed on a single core of an Intel Core 2 Duo 2.4GHz CPU under OS X 10.6.3. A Python implementation of the state splitting reconstruction algorithm [SHA04] was used, and will soon be available in the open source Computational Mechanics in Python (CMPy) library. Captured traffic was obtained from the UCDavis Honeynet Project.

FTP

Symbols	Inferred States	Time (Seconds)
300	6	0.18
600	9	0.20
900	10	0.23
1200	10	0.11
1500	11	0.32

HTTP

Symbols	Inferred States	Time (Seconds)
14337	12	0.18
28674	14	0.35
43011	18	0.84
57348	20	1.16
71685	22	1.86

Figure 4.3: Scaling of inferred states and inference time as a function of preprocessed data length, for FTP (top) and HTTP (bottom). Time is not necessarily monotonically increasing due to finite sample effects. State counts are given for non-deterministic presentations of the ϵ-machines.

Notably, the 18-state ϵ-machine was inferred from 60,000 symbols in less than a second despite using an interpreted language. Random walks on these machines generate new packets that are validated by remote protocol implementations, indicating the structure of the protocol is correctly captured. These preliminary results demonstrate that probabilistic reconstruction of both binary and text-based protocols is possible when alphabet size is managed by selecting an appropriate level of symbol abstraction and removing high-entropy payloads. Given this, we next discuss future applications of probabilistic models to the task of protocol inference.

§4.4 Future Work

A distinct advantage of the ϵ-machine for protocol inference over other models such as minimized prefix tree acceptors [BUG05] is its probabilistic nature. This enables applications such as anomaly detection where a statistical model of behavior is inferred from “normal" traffic traces, and deviations are flagged using various model comparison techniques [BUR02]. Towards this end, we have investigated the use of relative entropy (see section 2.3.5). A large relative entropy between models may indicate anomalous behavior, where “large” is some threshold chosen for an acceptable false positive rate. We leave the selection of this threshold, as well as a comparison of distance metrics for ϵ-machines, to future work.

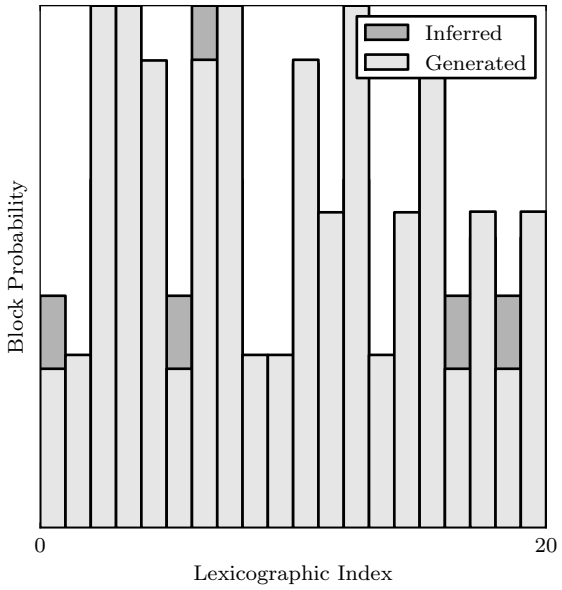

Figure 4.4: Overlayed distributions of block length 4 symbols using an ϵ-machine inferred from Modbus traffic (dark gray) to generate new traffic (light gray). Relative entropy between the distributions is 0.09 bits, indicating the distributions are close.

An additional application of the ϵ-machine is probabilistic traffic generation, as the distribution of strings generated by the model should be close to that of its data source. Random walks on the model generate ready-to-transmit packets both for traffic generation and protocol mimicry as introduced by Cui et al [CUI06]. The fit of 500 generated Modbus packets is shown in figure 4.4, resulting in a small D_{KL} of 0.09 bits.

Random walks on the ϵ-machine can also be used for testing the robustness of a program by sending it random inputs or mutating valid inputs. This process is called *fuzzing*. If the program does not correctly handle invalid input, fuzzing it may crash or leave the system vulnerable to attack. While generally considered effective for finding bugs, a substantial drawback to this approach is code coverage. For example, if the code's execution path depends on the value of a 32-bit integer, a random input has a 1 in 2^{32} chance of evaluating that code path [GOD07]. Working with mutated valid inputs enables more targeted testing, but requires some knowledge of their specification. The inferred ϵ-machine enables such targeted fuzzing when no specification is available.

Consider a previously known flaw in Golden FTP Server 2.70 for Windows.[1] A "change directory" command sent from the client with certain large arguments crashes the server and enables remote code execution. Using an ϵ-machine inferred from FTP traffic and tuned to produce longer runs of random data, this flaw was reproduced by a random walk on the machine. A subgraph of the ϵ-machine that causes the crash is given in figure 4.5. We plan to investigate if probabilistic models confer additional benefits to targeted fuzzing in future work.

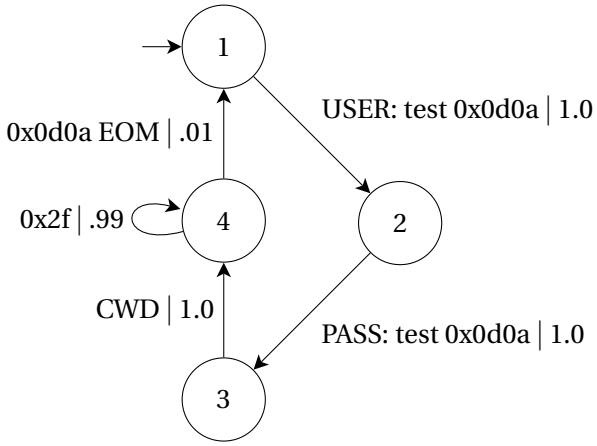

Figure 4.5: Subgraph of the ϵ-machine used for fuzzing Golden FTP Server 2.70, crashing the server when a "change directory" command is followed by more than 150 bytes. Symbol probabilities in the inferred ϵ-machine were tuned to produce longer sequences of random data for guided fuzzing.

[1] http://bit.ly/bO22hx

§4.5 Conclusion

We presented a novel HMM-based approach for inferring the state machine of network protocols using only their traffic. While generally applicable to any non-encrypted protocol stream, our emphasis is on protocols without a publicly available specification.

Our approach uses ϵ-machine reconstruction [CRU89, SHA04] to infer the minimal deterministic HMM of a protocol. The parameters of the HMM are inferred directly from the data, which avoids the typical pitfalls involved in parameter selection. We demonstrated our approach by inferring ϵ-machines from ICMP, Modbus, FTP, and HTTP traffic, and discussed preliminary applications of these models to protocol mimicry, fuzzing, and traffic generation. We plan to use the probabilistic nature of our models for anomaly detection, and have early success doing so in high performance computing environments.

Due to the limitations of traffic-based approaches, as well as sensitivity to alphabet size, more work remains to adapt reconstruction to high complexity protocols. In some cases where domain knowledge is available, traditional HMMs may scale better than ϵ-machines. However, our approach is well suited to protocol inference when there is insufficient domain knowledge for the manual construction of state machine models.

CHAPTER 5
Structural Drift

We introduce a theory of sequential causal inference in which learners in a chain estimate a structural model from their upstream "teacher" and then pass samples from the model to their downstream "student". It extends the population dynamics of genetic drift, recasting Kimura's selectively neutral theory as a special case of a generalized drift process using structured populations with memory. We examine the diffusion and fixation properties of several drift processes and propose applications to learning, inference, and evolution. We also demonstrate how the organization of drift process space controls fidelity, facilitates innovations, and leads to information loss in sequential learning with and without memory.

§5.1 "Send Three- and Four-Pence, We're Going to a Dance"

This phrase was heard, it is claimed, over the radio during WWI instead of the transmitted tactical phrase "Send reinforcements we're going to advance" [SMI88]. As illustrative as it is apocryphal, this garbled yet comprehensible transmission sets the tone for our investigations here. Namely, what happens to knowledge when it is communicated sequentially along a chain, from one individual to the next? What fidelity can one expect? How is information lost? How do innovations occur?

To answer these questions we introduce a theory of sequential causal inference in which learners in a communication chain estimate a structural model from their upstream "teacher" and then, using that model, pass along samples to their downstream "student". This reminds one of the familiar children's game *Telephone*. By way of quickly motivating our sequential learning problem, let's briefly recall how the game works.

To begin, one player invents a phrase and whispers it to another player. This player, believing they have understood the phrase, then repeats it to a third and so on until the last player is

reached. The last player announces the phrase, winning the game if it matches the original. Typically it does not, and that's the fun. Amusement and interest in the game derive directly from how the initial phrase evolves in odd and surprising ways.

The game is often used in education to teach the lesson that human communication is fraught with error. The final phrase, though, is not merely accreted error but the product of a series of attempts to parse, make sense, and intelligibly communicate the phrase. The phrase's evolution is a trade off between comprehensibility and accumulated distortion, as well as the source of the game's entertainment. We employ a much more tractable setting to make analytical progress on sequential learning,[1]intentionally selecting a simpler language system and learning paradigm than likely operates with children.

Specifically, we develop our theory of sequential learning as an extension of the evolutionary population dynamics of genetic drift, recasting Kimura's selectively neutral theory [KIM69] as a special case of a generalized drift process of structured populations with memory. Notably, this requires a new and more general information-theoretic notion of fixation. We examine the diffusion and fixation properties of several drift processes, demonstrating that the space of drift processes is highly organized. This organization controls fidelity, facilitates innovations, and leads to information loss in sequential learning and evolutionary processes with and without memory. We close by describing applications to learning, inference, and evolutionary processes.

To get started, we briefly review genetic drift and fixation. This will seem like a distraction, but it is a necessary one since available mathematical results are key. Then we introduce in detail our structured variants of these concepts—defining the *generalized drift process* and introducing a generalized definition of fixation appropriate to it. With the background laid out, we begin to examine the complexity of structural drift behavior. We demonstrate that the diffusion takes place in space that can be decomposed into a connected network of structured subspaces. We show how to quantify the degree of structure within these subspaces. Building on this decomposition, we explain how and when processes jump between these subspaces—innovating new structural information or forgetting it—thereby controlling the long-time fidelity of the commu-

[1]There are alternative, but distinct notions of *sequential learning*. Our usage should not be confused with notions in education and psychology, sometimes also referred to as *analytic* or step-by-step learning [FEL88]. Our notion also differs in motivation from those developed in machine learning, such as with statistical estimation for sequential data [DIE02], though some of the inference methods may be seen as related. Perhaps the notion here is closer to that implicated in mimicked behavior which drives financial markets [LOW02].

nication chain. We then close by outlining future research and listing several potential applications for structural drift, drawing out consequences for evolutionary processes that learn.

Those familiar with neutral evolution theory are urged to skip to Sec. 5.5, after skimming the next sections to pick up our notation and extensions.

§5.2 From Genetic to Structural Drift

Genetic drift refers to the change over time in genotype frequencies in a population due to random sampling. It is a central and well studied phenomenon in population dynamics, genetics, and evolution. A population of genotypes evolves randomly due to drift, but typically changes are neither manifested as new phenotypes nor detected by selection—they are *selectively neutral*. Drift plays an important role in the spontaneous emergence of mutational robustness[NIM99, BLO06], modern techniques for calibrating molecular evolutionary clocks [RAV07], and nonadaptive (neutral) evolution [CRU03B, KOE06], to mention only a few examples.

Selectively neutral drift is typically modeled as a stochastic process: A random walk in genotype space that tracks finite populations of individuals in terms of their possessing (or not) a variant of a gene. In the simplest models, the random walk occurs in a space that is a function of genotypes in the population. For example, a drift process can be considered to be a random walk of the *fraction* of individuals with a given variant. In the simplest cases there, the model reduces to the dynamics of repeated binomial sampling of a biased coin, in which the empirical estimate of bias becomes the bias in the next round of sampling. In the sense we will use the term, the sampling process is *memoryless*. The biased coin, as the population being sampled, has no memory: the past is independent of the future. The current state of the drift process is simply the bias, a number between zero and one that summarizes the state of the population.

The theory of genetic drift predicts a number of measurable properties. For example, one can calculate the expected time until all or no members of a population possess a particular gene variant. These final states are referred to as *fixation* and *deletion*, respectively. Variation due to sampling vanishes once these states are reached and, for all practical purposes, drift stops. From then on, the population is homogeneous. These states are fixed points—in fact, absorbing states—of the drift stochastic process.

The analytical predictions for the time to fixation and time to deletion were developed by Kimura and Ohta [KIM69, KIM83] in the 1960s and are based on the memoryless models and simplifying assumptions introduced by Wright [WRI31] and Fisher [FIS00] in the early 1930s. The theory has advanced substantially since then to handle more complicated and realistic models and to predict the additional effects due to selection and mutation. One example is the analysis of the drift-like effect of pseudohitchhiking ("genetic draft") recently given [GIL00].

The following explores what happens when we relax the memoryless assumption. The original random walk model of genetic drift forces the statistical structure at each sampling step to be an independent, identically distributed (IID) stochastic process. This precludes any memory in the sampling. Here, we extend the IID theory to use time-varying probabilistic state machines to describe memoryful population sampling.

In the larger setting of sequential learning, we will show that memoryful sequential sampling exhibits structurally complex, drift-like behavior. We call the resulting phenomenon *structural drift*. Our extension presents a number of new questions regarding the organization of the space of drift processes and how they balance structure and randomness. To examine these questions, we require a more precise description of the original drift theory [GIL04].

§5.3 Genetic Drift

We begin with the definition of an *allele*, which is one of several alternate forms of a gene. The textbook example is given by Mendel's early experiments on heredity [MEN25], in which he observed that the flowers of a pea plant were colored either white or violet, this being determined by the combination of alleles inherited from its parents. A new, *mutant* allele is introduced into a population by the mutation of a *wild-type* allele. A mutant allele can be passed on to an individual's offspring who, in turn, may pass it on to their offspring. Each inheritance occurs with some probability.

Genetic drift, then, is the change of allele frequencies in a population over time: It is the process by which the number of individuals with an allele varies generation after generation. The Fisher-Wright theory [WRI31, FIS00] models drift as a stochastic evolutionary process with neither selection nor mutation. It assumes random mating between individuals and that the

population is held at a finite, constant size. Moreover, successive populations do not overlap in time.

Under these assumptions the Fisher-Wright theory reduces drift to a binomial or multinomial sampling process—a more complicated version of familiar random walks such as Gambler's Ruin or Prisoner's Escape [FEL68]. Offspring receive either the wild-type allele A_1 or the mutant allele A_2 of a particular gene $\mathcal{A}$ from a random parent in the previous generation with replacement. A population of N diploid[2] individuals will have $2N$ total copies of these alleles. Given i initial copies of A_2 in the population, an individual has either A_2 with probability $i/2N$ or A_1 with probability $1 - i/2N$. The probability that j copies of A_2 exist in the offspring's generation given i copies in the parent's generation is:

$$p_{ij} = \binom{2N}{j} \left(\frac{i}{2N}\right)^j \left(1 - \frac{i}{2N}\right)^{2N-j}.$$

This specifies the transition dynamic of the drift stochastic process over the discrete state space $\{0, 1/N, 2/N, \ldots, N-1/N, 1\}$.

This model of genetic drift is a discrete-time random walk, driven by samples of a biased coin, over the space of biases. The population is a set of coin flips, where the probability of HEADS or TAILS is determined by the coin's current bias. After each generation of flips, the coin's bias is updated to reflect the number of HEADS or TAILS realized in the new generation. The walk's absorbing states—all HEADS or all TAILS—capture the notion of fixation and deletion.

§5.4 Genetic Fixation

Fixation occurs with respect to an allele when all individuals in the population carry that specific allele and none of its variants. Restated, a mutant allele A_2 reaches fixation when all $2N$ alleles in the population are copies of A_2 and, consequently, A_1 has been *deleted* from the population. This halts the random fluctuations in the frequency of A_2, assuming A_1 is not reintroduced.

Let X be a binomially distributed random variable with bias probability p that represents the fraction of copies of A_2 in the population. The expected number of copies of A_2 is $\mathrm{E}[X] = 2Np$. That is, the expected number of copies of A_2 remains constant over time and depends only on its

[2] Though haploid populations can be used, we focus on diploid populations (two alleles per individual) for direct comparison to Kimura's simulations. This gives a sample length of $2N$.

initial probability p and the total number $(2N)$ of alleles in the population. However, A_2 eventually reaches fixation or is deleted due to the variance introduced by random finite sampling and the presence of absorbing states.

Prior to fixation, the mean and variance of the change Δp in allele frequency are:

$$\begin{aligned} \mathrm{E}[\Delta p] = &\quad 0 \text{and} \\ \mathrm{Var}[\Delta p] = &\quad \frac{p\,(1-p)}{2N}, \end{aligned}$$

respectively. On average there is no change in frequency. However, sampling variance causes the process to drift towards the absorbing states at $p = 0$ and $p = 1$. The drift rate is determined by the current generation's allele frequency and the total number of alleles. For the neutrally selective case, the average number of generations until A_1's fixation (t_1) or deletion (t_0) is given by[KIM69]:

$$\begin{aligned} t_1(p) = &\quad -\frac{1}{p}\left[4N_e(1-p)\log(1-p)\right] \text{and} \\ t_0(p) = &\quad -4N_e\left(\frac{p}{1-p}\right)\log p, \end{aligned}$$

where N_e denotes effective population size. For simplicity we take $N_e = N$, meaning all individuals in the population are candidates for reproduction. As $p \to 0$, the boundary condition is given by:

$$t_1(0) = 4N_e.$$

That is, excluding cases of deletion, an initially rare mutant allele spreads to the entire population in $4N_e$ generations.

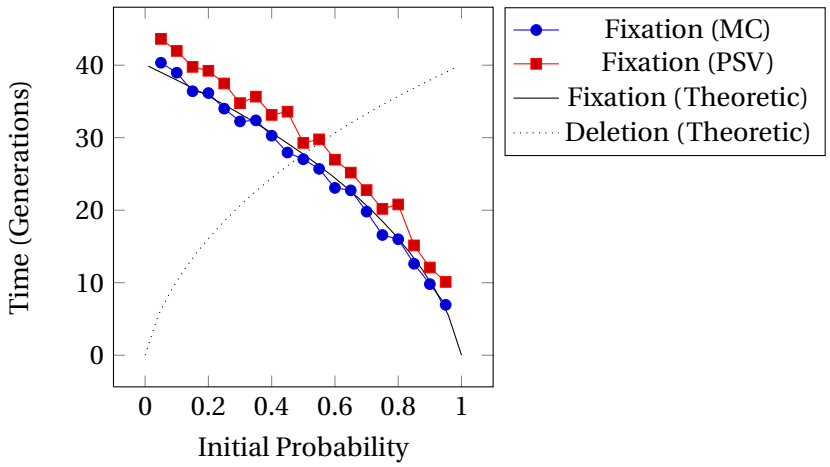

Figure 5.1: Number of generations to fixation for a population of $N = 10$ individuals (sample size $2N = 20$), plotted as a function of initial allele frequency p under different sampling regimes: Monte Carlo (MC), Monte Carlo with pseudo-sampling variable (MC w/ PSV), and theoretical prediction (solid line, Theory). The time to deletion is also shown (dashed line, Theory).

One important observation that immediately falls out from the theory is that when fixation ($p = 1$) or deletion ($p = 0$) are reached, variation in the population vanishes: $\mathrm{Var}[\Delta p] = 0$. With no variation there is a homogeneous population, and sampling from this population produces the same homogeneous population. In other words, this establishes fixation and deletion as absorbing states of the stochastic sampling process. Once there, drift stops.

Figure 5.1 illustrates this, showing both the simulated and theoretically predicted number of generations until fixation occurs for $N = 10$, as well as the predicted time to deletion, for reference. Each simulation was performed for a different initial value of p and averaged over 400 realizations. Using the same methodology as[KIM69], we include only those realizations whose allele reaches fixation.

Different kinds of sampling dynamic have been proposed to simulate the behavior of genetic drift. Simulations for two are shown in the figure. The first uses binomial sampling, producing an initial population of $2N$ uniform random numbers between 0 and 1. An initial probability $1 - p$ is assigned to allele A_1 and probability p to allele A_2. The count i of A_2 in the initial population is incremented for each random number less than p. This represents an individual having the allele A_2 instead of A_1. The maximum likelihood estimate of allele frequency in the initial sample is simply the number of A_2 alleles over the sample length:$p = i/2N$. This estimate of p is then used

to generate a new population of offspring, after which we re-estimate the value of p. These steps are repeated each generation until fixation at $p = 1$ or deletion at $p = 0$ occurs.

The second sampling dynamic uses a *pseudo-sampling variable* ξ in lieu of direct sampling [KIM80]. The allele frequency p_{n+1} in the next generation is calculated by adding ξ_n to the current frequency p_n:

$$\begin{aligned} p_{n+1} &= p_n + \xi_n, \\ \xi_n &= \sqrt{3\,\sigma_n^2}(2r_n - 1), \end{aligned}$$

where σ_n^2 is the current variance given by Eq. (5.4) and r_n is a uniform random number between 0 and 1. This method avoids the binomial sampling process, sacrificing some accuracy for faster simulations and larger populations. As Fig. 5.1 shows for fixation, this method (MC w/ PSV, there) overestimates the time to fixation and deletion.

Kimura's theory and simulations predict the time to fixation or deletion of a mutant allele in a finite population by the process of genetic drift. The Fisher-Wright model and Kimura's theory assume a memoryless population in which each offspring inherits allele A_1 or A_2 via an IID binomial sampling process. We now generalize this to memoryful stochastic processes, giving a new definition of fixation and exploring examples of structural drift behavior.

§5.5 Sequential Learning

How can genetic drift be a memoryful stochastic process? Consider a population of N individuals. Each generation consists of $2N$ alleles and so is represented by a string of $2N$ symbols, e.g. $A_1A_2 \ldots A_1A_1$, where each symbol corresponds to an individual with a particular allele. In the original drift models, a generation of offspring is produced by a memoryless binomial sampling process, selecting an offspring's allele from a parent with replacement. In contrast, the structural drift model produces a generation of individuals in which the sample order is tracked. The population is now a string of alleles, giving the potential for memory and structure in sampling—temporal interdependencies between individuals within a sample or some other aspect of a population's organization. (Later, we return to give several examples of alternative ordered-sampling processes.)

The model class we select to describe memoryful sampling consists of ϵ-machines, since each is a unique, minimal, and optimal representation of a stochastic process [SHA01A]. More to the point, ϵ-machines give a systematic representation of all the stochastic processes we consider here. As will become clear, these properties give an important advantage when analyzing structural drift, since they allow one to monitor the amount of structure innovated or lost during drift, as we will show. We next give a brief overview of ϵ-machines and refer the reader to the previous references for details.

ϵ-Machine representations of the finite-memory discrete-valued stochastic processes we consider here form a class of deterministic probabilistic finite-state machine. An ϵ-machine consists of a set of *causal states* $\boldsymbol{\mathcal{S}} = \{0, 1, \ldots, k-1\}$ and a set of transition matrices:

$$\{T_{ij}^{(a)} : a \in \mathcal{A}\},$$

where $\mathcal{A} = \{A_1, \ldots, A_m\}$ is the set of alleles and where the transition probability $T_{ij}^{(a)}$ gives the probability of transitioning from causal state $\mathcal{S}_i$ to causal state $\mathcal{S}_j$ and emitting allele a. Maintaining our connection to diploid theory, we think of an ϵ-machine as a generator of populations or length-$2N$ strings: $\alpha^{2N} = a_1 a_2 \ldots a_i \ldots a_{2N}, a_i \in \mathcal{A}$. As a model of a sampling process, an ϵ-machine gives the most compact representation of the distribution of strings produced by sampling.

We are now ready to describe *sequential learning*. We begin by selecting an initial population generator M_0—an ϵ-machine. A random walk on M_0, guided by its transition probabilities, generates a length-$2N$ string $\alpha_0^{2N} = \alpha_1 \ldots \alpha_{2N}$ that represents the first generation of N individuals possessing alleles $a_i \in \mathcal{A}$. We then infer an ϵ-machine M_1 from the population α_0^{2N}. M_1 is then used to produce a new population α_1^{2N}, from which a new ϵ-machine M_2 is estimated. (We describe alternative inference procedures shortly.) This new population has the same allele distribution as the previous, plus some amount of variance. The cycle of inference and reinference is repeated while allele frequencies drift between generations until fixation or deletion is reached. At that point, the populations (and so ϵ-machines) cannot vary further. The net result is a stochastically varying time series of ϵ-machines—$M_0, M_1, M_2, \ldots$—that terminates when the populations α_t^N generated stop changing.

Thus, at each step a new representation or model is estimated from the previous step's sam-

ple. The inference step highlights that this is learning: a model of the generator is estimated from the given data. The repetition of this step creates a sequential communication chain. Said simply, sequential learning is closely related to genetic drift, except that sample order is tracked and this order is used in estimating the next model.

The procedure is analogous to flipping a biased coin a number of times, estimating the bias from the results, and re-flipping the newly biased coin. Eventually, the coin will be completely biased towards HEADS or TAILS. In our drift model the coin is replaced by an ϵ-machine, which removes the IID constraint and allows for the sampling process to take on structure and memory. Not only do the transition probabilities $T_{ij}^{(a)}$ change, but the structure of the model itself—number of states and transitions—drifts over time.

Before we can explore this dynamic, we first need to examine how an ϵ-machine reaches fixation or deletion.

§5.6 Structural Stasis

Consider the *Alternating Process*—a binary process that alternately generates 0s and 1s. The ϵ-machine for this process, shown in Fig. 5.2, generates the strings $0101\ldots$ and $1010\ldots$ depending on the start state.

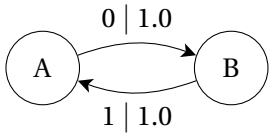

Figure 5.2: ϵ-Machine for the Alternating Process, consisting of two causal states $\boldsymbol{\mathcal{S}} = \{A, B\}$ and two transitions. Each transition is labeled $p \mid a$ to indicate the probability $p = T_{ij}^{(a)}$ of taking that transition and emitting allele $a \in \mathcal{A}$. State A generates allele 0 with probability one and transitions to state B, while B generates allele 1 with probability one and transitions to A.

Regardless of the start state, the ϵ-machine is re-inferred from any sufficiently long string it generates. In the context of sequential learning, this means the population at each generation is the same. However, if we consider allele A_1 to be represented by symbol 0 and A_2 by symbol 1, neither allele reaches fixation or deletion according to current definitions. Nonetheless, the Alternating Process prevents any variance between generations and so, despite the population not being all 0s or all 1s, the population does reach an equilibrium: half 0s and half 1s.

For these reasons, one cannot use the original definitions of fixation and deletion. This leads us to introduce *structural stasis* to combine the notions of fixation, deletion, and the inability to vary caused by periodicity. However, we need a method to detect the occurrence of structural stasis in a drift process.

A state machine representing a periodic sampling process enforces the constraint of periodicity via its internal memory. One measure of this memory is the *population diversity* $H(N)$ [PIE67]:

$$\begin{aligned} H(N) &= H[\mathcal{A}_1 \ldots \mathcal{A}_{2N}], \\ &= -\sum_{a^{2N} \in \mathcal{A}^{2N}} \Pr(a^{2N}) \log_2 \Pr(a^{2N}), \end{aligned}$$

where the units are [bits]. (For background on information theory, as used here, the reader is referred to Ref. [CRU03A].)

The population diversity of the Alternating Process is $H(N) = 1$ bit at any size $N \geq 1$. This single bit of information corresponds to the process's current phase or state. The population diversity does not change with N, meaning the Alternating Process is always in structural stasis. However, using population diversity as a condition for detecting stasis fails for an arbitrary sampling process.

Said more directly, structural stasis corresponds to a process becoming nonstochastic, since it ceases to introduce variance between generations and so prevents further drift. The condition for stasis could be given as the vanishing (with size N) of the growth rate, $H(N) - H(N-1)$, of population diversity. However, this can be difficult to estimate accurately for finite population sizes.

A related, alternate condition for stasis that avoids these problems uses the entropy rate of the sampling process. We call this *allelic entropy*:

$$h_\mu = \lim_{N \to \infty} \frac{H(N)}{2N},$$

where the units are [bits per allele]. Allelic entropy gives the average information per allele in bits, and structural stasis occurs when $h_\mu = 0$.

This quantity is also difficult to estimate from population samples since it relies on an asymptotic estimate of the population diversity. However, in structural drift we have the ϵ-machine representation of the sampling process. Due to the ϵ-machine's unifilarity, the allelic entropy

can be calculated in closed-form over the causal states and their transitions (see section 2.6). When $h_\mu = 0$, the sampling process has become periodic and lost all randomness generated via its branching transitions. This new criterion simultaneously captures the notions of fixation and deletion, as well as periodicity. An ϵ-machine has zero allelic entropy if any of these conditions occur. More formally, we have the following statement.

Definition. Structural stasis *occurs when the sampling process's allelic entropy vanishes:* $h_\mu = 0$.

Proposition 1. *Structural stasis is a fixed point of finite-memory structural drift.*

Proof. *Finite-memory means that the* ϵ*-machine representing the population sampling process has a finite number of states. Given this, if* $h_\mu = 0$*, then the* ϵ*-machine has no branching in its recurrent states:* $T^{(a)}_{ij} = 0$ or 1*, where* $\mathcal{S}_i$ *and* $\mathcal{S}_j$ *are asymptotically recurrent states. This results in no variation in the inferred* ϵ*-machine when sampling sufficiently large populations. Lack of variation, in turn, means that* $\Delta p = 0$ *and so the drift process stops. Since no mutations are allowed, a further consequence is that, if allelic entropy vanishes at time* t*, then it is zero for all* $t' > t$*. Thus, structural stasis is an absorbing state of the drift stochastic process.*

§5.7 Examples

While more can be said analytically about structural drift, our present purpose is to introduce the main concepts. We will show that structural drift leads to interesting and nontrivial behavior. First, we calibrate the new class of drift processes against the original genetic drift theory.

§5.7.1 Memoryless Drift

The Biased Coin Process is represented by a single-state ϵ-machine with a self loop for both HEADS and TAILS symbols. It is an IID sampling process that generates populations with a binomial distribution. Unlike the Alternating Process, the coin's bias p is free to drift during sequential inference. These properties make the Biased Coin Process an ideal candidate for exploring memoryless drift.

Two measures of the structural drift of a single realization of the Biased Coin Process are shown in Fig. 5.3, with initial $p = \Pr[\text{HEADS}] = \Pr[\text{TAILS}] = 0.5$. Structural stasis ($h_\mu = 0$) is reached after 115 generations. Note that the drift of allelic entropy h_μ and $p = \Pr[\text{TAILS}]$ are

inversely related, with allelic entropy converging quickly to zero as stasis is approached. This reflects the rapid drop in population diversity. The initial Fair Coin ϵ-machine occurs at the left of Fig. 5.3, and the final completely biased ϵ-machine occurs at the right. After stasis occurs, all randomness has been eliminated from the transitions at state A, resulting in a single transition that always produces TAILS. Anticipating later discussion, we note that during the run one only sees Biased Coin Processes.

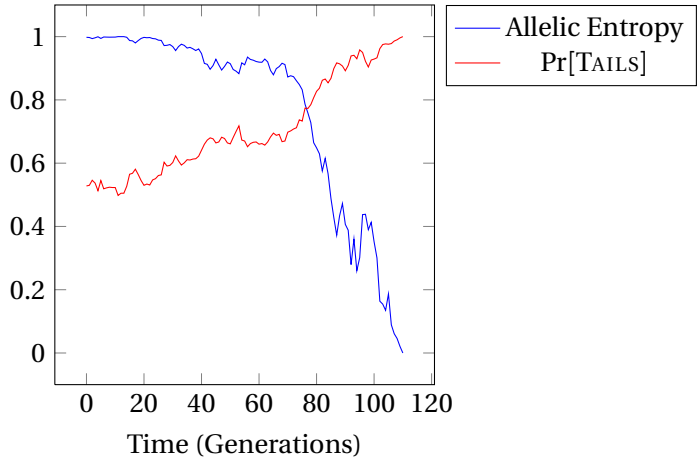

Figure 5.3: Drift of allelic entropy h_μ and Pr[TAILS] for a single realization of the Biased Coin Process with sample length $2N = 100$ and state splitting reconstruction ($\alpha = 0.01$).

The time to stasis of the Biased Coin Process as a function of initial $p = \Pr[\text{HEADS}]$ is shown in Fig. 5.4. Also shown is Kimura's previous Monte Carlo drift simulation modified to terminate when either fixation or deletion occurs. This experiment, with a 100 times larger population than Fig. 5.1, illustrates the definition of structural stasis and allows direct comparison of structural drift with genetic drift in the memoryless case.

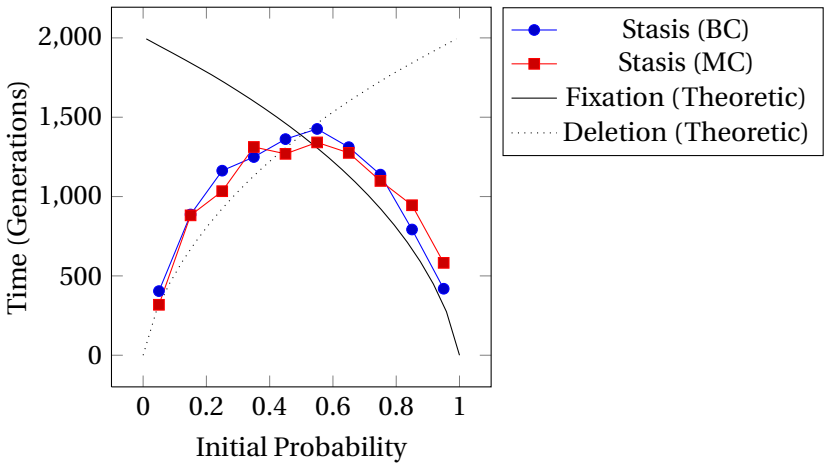

Figure 5.4: Average time to stasis as a function of initial Pr[HEADS] for the Biased Coin Process: Structural drift of the Biased Coin and Monte Carlo simulation of Kimura's equations. Both employ vanishing allelic entropy at stasis. Kimura's predicted times to fixation and deletion are shown for reference. Each estimated time is averaged over 100 drift experiments with sample length $2N = 1000$ and state splitting reconstruction ($\alpha = 0.01$).

Not surprisingly, we can interpret genetic drift as a special case of the structural drift process for the Biased Coin.[3] Both simulations follow Kimura's theoretically predicted curves, combining the lower half of the deletion curve with the upper half of the fixation curve to reflect the initial probability's proximity to the absorbing states. A high or low initial bias leads to a shorter time to stasis as the absorbing states are closer to the initial state. Similarly, a Fair Coin is the furthest from absorption and thus takes the longest average time to reach stasis.

§5.7.2 Structural Drift

The Biased Coin Process is an IID sampling process with no memory of previous flips, reaching stasis when Pr[HEADS] $= 1.0$ or 0.0 and, correspondingly, when $h_\mu(M_t) = 0.0$. We now introduce memory by starting drift with M_0 as the *Golden Mean Process*, which produces binary populations with no consecutive 0s. Its ϵ-machine is shown in Fig. 5.5.

[3] Simulations used both the causal-state splitting [SHA04] and subtree merging [CRU89] ϵ-machine reconstruction algorithms, with approximately equivalent results. Unless otherwise noted, state splitting used a history length of 3, and tree merging used a morph length of 3 and tree depth of 7. Both algorithms used a significance-test value of $\alpha = 0.01$ in the Kolmogorov-Smirnov hypothesis tests for state equivalence. (Other tests, such as χ^2, may be substituted with little change in results.) For the Biased Coin Process, a history length of 1 was used for direct comparison to binomial sampling.

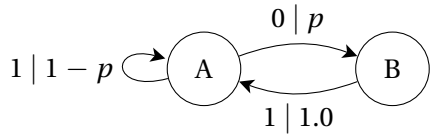

Figure 5.5: The ϵ-machine for the Golden Mean Process, which generates a population with no consecutive 0s. In state A the probabilities of generating a 0 or 1 are p and $1 - p$, respectively.

Like the Alternating Process, the Golden Mean Process has two causal states. However, the transitions from state A have nonzero entropy, allowing their probabilities to drift as new ϵ-machines are inferred from generation to generation. If the $A \to B$ transition parameter p (Fig. 5.5) drifts towards zero probability and is eventually removed, the Golden Mean reaches stasis by transforming into the Fixed Coin Process identical to that shown at the bottom right of Fig. 5.3. Instead, if the same transition drifts towards probability $p = 1$, the $A \to A$ transition is removed. In this case, the Golden Mean Process reaches stasis by transforming into the Alternating Process (Fig. 5.2).

Naturally, one can start drift from any one of a number of processes. Let's also consider the *Even Process* and then compare drift behaviors. Similar in form to the Golden Mean Process, the Even Process produces populations in which blocks of consecutive 1s must be even in length when bounded by 0s.

Figure 5.6 (left) compares the drift of Pr[HEADS] for single runs starting with the Biased Coin, Golden Mean, and Even Processes. One observes that the Biased Coin and Even Processes reach stasis via the Biased Coin fixed point, while the Golden Mean Process reaches stasis via the Alternating Process fixed point.

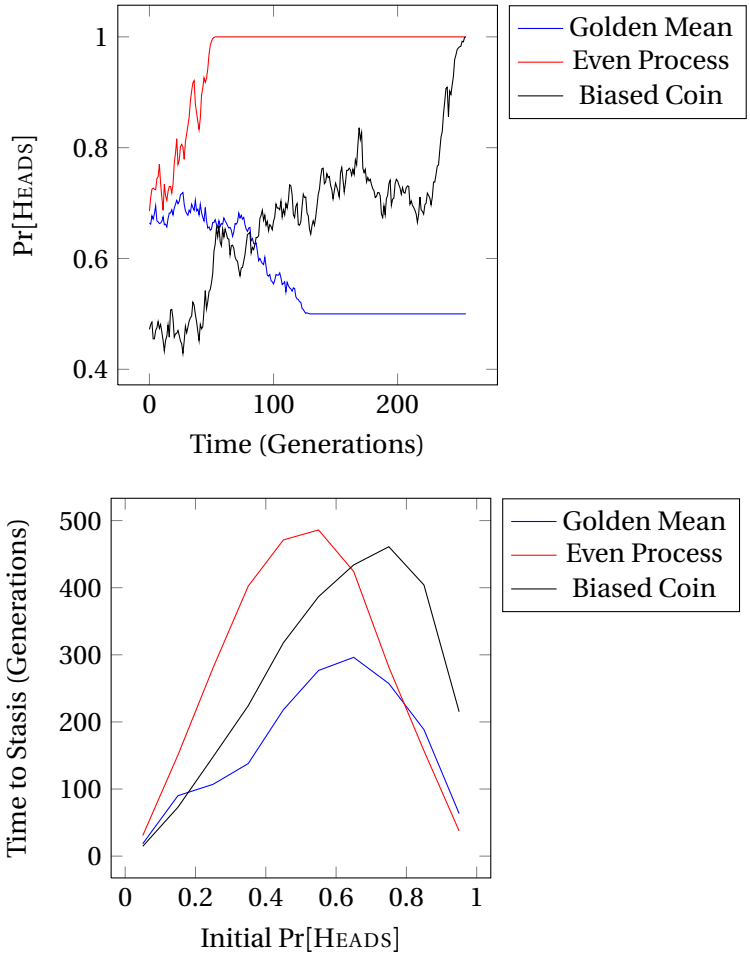

Figure 5.6: Comparison of structural drift processes. *Top*: Pr[HEADS] for the Biased Coin, Golden Mean, and Even Processes as a function of generation. The Even and Biased Coin Processes become completely biased coins at stasis, while the Golden Mean becomes the Alternating Process. Note that the definition of structural stasis recognizes the lack of variance in that periodic-process subspace, even though the allele probability is neither 0 nor 1. *Bottom*: Time to stasis as a function of initial bias parameter for each process. Each estimated time is averaged over 100 drift experiments with sample length $2N = 1000$ and state-splitting reconstruction ($\alpha = 0.01$).

It should be noted that the memoryful Golden Mean and Even Processes reach stasis markedly faster than the memoryless Biased Coin. While the top panel of Fig. 5.6 shows only a single realization of each sampling process type, the bottom panel shows that the large disparity in stasis times holds across all settings of each process's initial bias. This is one of our first general observations about memoryful processes. The structure of memoryful processes substantially impacts the average time to stasis by increasing variance between generations.

§5.8 Isostructural Subspaces

To illustrate the richness of structural drift and to understand how it affects average time to stasis, we examine the complexity-entropy (CE) diagram [FEL08] of the ϵ-machines produced over several realizations of an arbitrary sampling process. The CE diagram displays how the allelic entropy h_μ of an ϵ-machine varies with its allelic complexity C_μ:

$$C_\mu = -\sum_{\sigma \in \mathcal{S}} \Pr(\sigma) \log_2 \Pr(\sigma),$$

where the units are [bits]. The allelic complexity is the Shannon entropy over an ϵ-machine's stationary state distribution $\Pr(\mathcal{S})$. It measures the memory needed to maintain the internal state while producing stochastic outputs. ϵ-Machine minimality guarantees that C_μ is the smallest amount of memory required to do so. Since there is a one-to-one correspondence between processes and their ϵ-machines, a CE diagram is a projection of process space onto the two coordinates (h_μ, C_μ). Used in tandem, these two properties differentiate many types of sampling process, capturing both their intrinsic memory (C_μ) and the diversity (h_μ) of populations they generate.

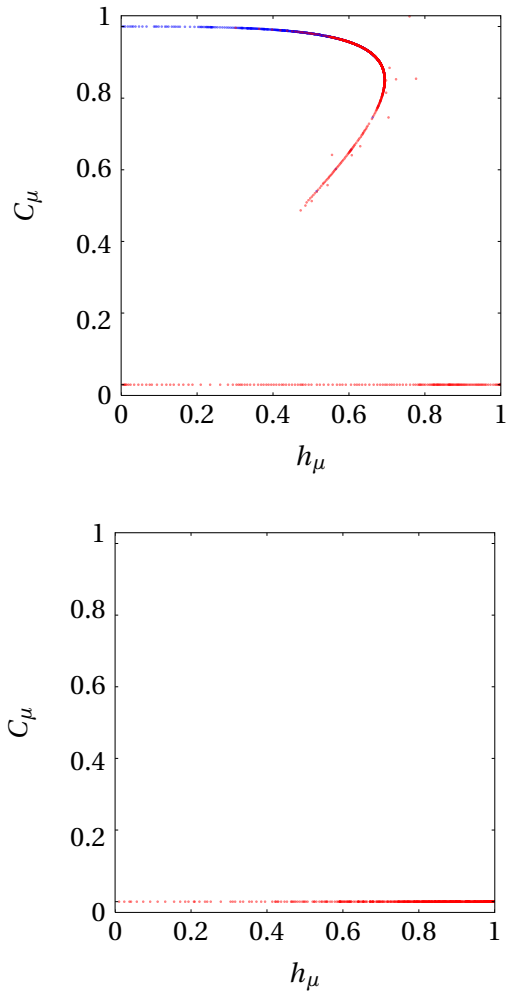

Figure 5.7: *Top:* Allelic complexity C_μ versus allelic entropy h_μ for 100 realizations starting with the Golden Mean Process at $p_0 = \frac{1}{2}$ and $(h_\mu, C_\mu) \approx (\frac{2}{3}, 0.918)$, showing time spent in the Alternating and Biased Coin subspaces. *Bottom:* Drift starting with the Biased Coin Process with initially fair transition probabilities: $p_0 = \frac{1}{2}$ and $(h_\mu, C_\mu) = (1, 0)$. Point density in the higher-h_μ region indicates longer times to stasis compared to drift starting with the Golden Mean Process, which enters the Biased Coin subspace nearer the fixed point $(0, 0)$, jumping around $(h_\mu, C_\mu) \approx (0.5, 0.5)$. Red-colored dots correspond to M_t that go to stasis as the Fixed Coin Process; blue correspond to those which end up in stasis as the Alternating Process. Each of the 100 runs used sample length $2N = 1000$ and state-splitting reconstruction ($\alpha = 0.01$).

Let's examine the structure of drift-process space further. The CE diagram for 100 realizations of the Golden Mean Process is shown in the left panel of Fig. 5.7. The M_t reach stasis by transforming into either the Fixed Coin Process or the Alternating Process, depending on how the transition parameter p (Fig. 5.5) drifts. The Fixed Coin Process, all 1s or all 0s, exists at

$(h_\mu, C_\mu) = (0, 0)$ since a process in stasis has no allelic entropy and no memory is required to track a single state. The Golden Mean Process transforms into the Fixed Coin at $p = 0$. The Alternating Process exists at point $(h_\mu, C_\mu) = (0, 1)$ as it also has no allelic entropy, but requires 1 bit of information storage to track the phase of its 2 states. The Golden Mean Process becomes the Alternating Process when $p = 1$. The two stasis points are connected by the isostructural curve $(h_\mu(M(p)), C_\mu(M(p))$, where $M(p)$ is the Golden Mean ϵ-machine of Fig. 5.5 with $p \in [0, 1]$.

What emerges from examining these overlapping realizations is a broad view of how the structure of M_t drifts in process space. Roughly, they diffuse locally in the parameter space specified by the current, fixed architecture of states and transitions. During this, transition probability estimates vary stochastically due to sampling variance. Since C_μ and h_μ are continuous functions of the transition probabilities, this variance causes the M_t to fall on well defined curves or regions corresponding to a particular process subspace. (See Figs. 4 and 5 in [FEL08] and the theory for these curves and regions there.) We refer to the associated sets of ϵ-machines as *isostructural subspaces*. They are metastable subspaces of sampling processes that are quasi-invariant under the structural drift dynamic. That invariance is broken by jumps between the subspaces in which one or more ϵ-machine parameters diffuse sufficiently that inference is forced to shift ϵ-machine topology—that is, states or transitions are gained or lost.

Such a shift occurs in the lower part of the Golden Mean isostructural curve, where states A and B merge into a single state as transition probability $p \to 0$, corresponding to $(h_\mu, C_\mu) \approx (0.5, 0.5)$. The exact location on the curve where this discontinuity occurs is controlled by the significance level (α), described earlier, at which two causal states are determined to be equivalent by a statistical hypothesis test. Specifically, states A and B are merged closer to point $(h_\mu, C_\mu) = (0, 0)$ along the Golden Mean Process isostructural curve when the hypothesis test requires more evidence.

When causal-state merging occurs, the ϵ-machine leaves the Golden Mean subspace and enters the Biased Coin subspace. In the CE diagram, the latter is the one-dimensional interval $C_\mu = 0$ and $h_\mu \in [0, 1]$. In the new subspace, the time to stasis depends only on the entry value of p. The right panel of Fig. 5.7 shows a CE diagram of 100 realizations starting with the fair Biased Coin Process. The process diffuses along the line $C_\mu = 0$, never innovating a new state or jumping to a new subspace. This demonstrates that movement between subspaces is often

not bidirectional—innovations from a previous topology may be lost either temporarily (when the innovation can be restored by returning to the subspace) or permanently. For example, the Golden Mean Process commonly jumps to the Biased Coin, but the opposite is improbable.

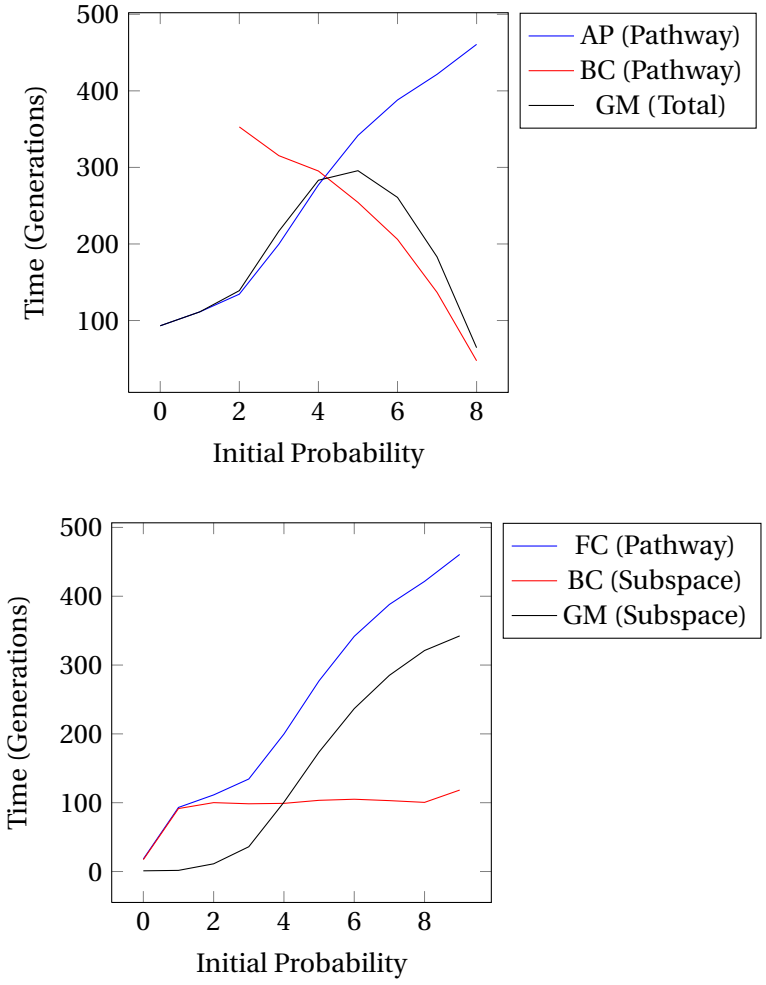

Figure 5.8: *Top*: Mean time to stasis for the Golden Mean Process as a function of initial transition probability p. The total time to stasis is the sum of stasis times for the Fixed Coin pathway and the Alternating Process pathway, weighted by their probability of occurrence. For initial p less than $\approx$ 0.3, the Alternating Process pathway was not observed during simulation due to its rarity. As initial p increases, the Alternating pathway is weighted more heavily while the Fixed Coin pathway occurs less frequently. *Bottom*: Mean time to stasis for the Fixed Coin pathway as function of initial transition probability p. The total time to stasis is the sum of the pathway's subspace stasis times. The Fixed Coin pathway visits the Golden Mean subspace before jumping to the Biased Coin subspace, on its way to stasis as the Fixed Coin. Each estimated time is averaged over 100 drift experiments with sample length $2N = 1000$ and state-splitting reconstruction ($\alpha = 0.01$).

All realizations eventually find their way to structural stasis on the CE diagram's left boundary at absorbing states $(h_\mu, C_\mu) = (0, \log_2 P)$ with periods P. Nonetheless, it is clear that the Golden Mean process, which starts at $(h_\mu, C_\mu) = (\frac{1}{2}, 1)$, leads the drift to innovate M_ts with substantially more allelic entropy and complexity. New structures—states and transitions—are spontaneously discovered.

In addition to the CE diagram helping to locate the fixed points and subspaces, the density of points on the isostructural curves gives an alternate view of the time to stasis plots (Fig. 5.4). Dense regions on the curve correspond to initial p values "furthest away" from the fixed points. ϵ-Machines typically spend longer diffusing on these portions of the curve, resulting in longer stasis times and a higher density of neighboring ϵ-machines. The difference in densities between the post-jump Biased Coin subspace of the Golden Mean (left) and the initially fair Biased Coin subspace (right) highlights that the majority of time spent drifting in the latter is near high-entropy, initially fair values of p. Golden Mean M_t jump into the Biased Coin subspace only after a state merging has occurred due to highly biased, low-entropy transition probabilities. This causes the M_t to arrive in the subspace nearer an absorbing state, resulting in a shorter average time to stasis. This is a consequence of the Golden Mean's structured process subspace. However, once the Biased Coin subspace is reached, the time to stasis from that point forward is independent of the time spent in the previous isostructural subspace. It is determined by the effective transition parameters of the M_t at time of entry.

Figure 5.8 demonstrates how the total stasis times decompose, accounting for the total time as weighted sum of the average stasis times of its pathways. A pathway is a set of subspaces passed through by all realizations reaching a particular fixed point. The left panel shows time to stasis $T_s(GMP(p_0))$ for starting the Golden Mean Process with initial transition probability $p = p_0$ as the weighted sum of the time spent diffusing in the Alternating and Fixed Coin pathways:

$$T_s(GMP(p_0)) = \sum_{\gamma \in \{FC, AP\}} k_\gamma T_s(\gamma | GMP(p_0)),$$

where $k_\gamma \in [0, 1]$ are the weights coefficients and $T_s(\gamma|\cdot)$ is the time to reach each terminal (stasis) subspace γ, having started in the Golden Mean Process with $p = p_0$.

For low p_0, the transition from state A to state B is unlikely, so zeros are rare and the Alter-

nating Process pathway is taken infrequently. Thus, the total stasis time is initially dominated by the Fixed Coin pathway. As $p_0 \to 0.3$ and above, the Alternating Process pathway becomes more frequent and this stasis time begins to contribute to the total. Around $p_0 = 0.6$ the Fixed Coin pathway becomes less likely and the total time becomes dominated by the Alternating Process pathway.

Time to stasis for a particular pathway is simply the sum of the times spent in the subspaces it connects. Figure 5.8's right panel examines the left panel's Fixed Coin pathway time $T_s(FC|GMP(p_0))$ in more detail. There are two contributions. These include the time diffusing on the portions of the Golden Mean subspace before the subspace jump, as well as the time attributed to the Biased Coin subspace *after* the subspace jump. Note that the Biased Coin subspace stasis time is independent of p_0, because the subspace is entered when the states A and B are merged. This merging occurs at the same p regardless of p_0. There is a dip for values of p_0 that are lower than the state-merging threshold for p, placing the initial subspace bias even closer to its absorbing state.

Thus, the time to reach each stasis from a subspace β consists of the times taken on pathways $c \in \beta$ to structural stasis that can be reached from β. As a result, the total time to stasis starting in β is the sum of each pathway c's stasis time $T_s(c|\beta)$ weighted by the pathway's likelihood $\Pr(c|\beta)$ starting from β:

$$T_s(\beta) = \sum_{c \in \beta} \Pr(c|\beta) T_s(c|\beta),$$

where the probabilities and times depend implicitly on the initial process's transition parameter(s).

§5.9 Discussion

§5.9.1 Summary

The Fisher-Wright model of genetic drift can be viewed as a random walk of coin biases, a stochastic process that describes generational change in allele frequencies based on a strong statistical assumption: the sampling process is memoryless. Here, we developed a generalized structural

drift model that adds memory to the process and examined the consequences of such population sampling memory.

The representation selected for the population sampling mechanism was the class of probabilistic finite-state machines called ϵ-machines. We discussed how a sequential chain of inferring and re-inferring ϵ-machines from the finite data they generate parallels the drift of alleles in a finite population, using otherwise the same assumptions made by the Fisher-Wright model.

We revisited Kimura's early results measuring the time to fixation of drifting alleles and showed that the generalized structural drift process reproduces these well known results, when staying within the memoryless sampling process subspace. Starting with populations outside of that subspace led the sampling processes to exhibit memory effects, including greatly reduced times to stasis, structurally complex transients, structural innovation, and structural decay. We introduced structural stasis to combine the concepts of deletion, fixation, and periodicity for drift processes. Generally, structural stasis occurs when the population's allelic entropy vanishes—a quantity one can calculate in closed form due to the use of the ϵ-machine representation for sampling processes.

Simulations demonstrated how an ϵ-machine diffuses through isostructural process subspaces during sequential learning. The result was a very complex time-to-stasis dependence on the initial probability parameter—much more complicated than Kimura's theory describes. We showed, however, that a process's time to stasis can be decomposed into sums over these independent subspaces. Moreover, the time spent in an isostructural subspace depends on the value of the ϵ-machine probability parameters at the time of entry. This suggests an extension to Kimura's theory for predicting the time to stasis for each isostructural component independently. Much of the phenomenological analysis was facilitated by the global view of drift process space given by the complexity-entropy diagram.

Drift processes with memory generally describe the evolution of structured populations without mutation or selection. Nonetheless, we showed that structure leads to substantially shorter stasis times. This was seen in drifts starting with the Biased Coin and Golden Mean Processes, where the Golden Mean jumps into the Biased Coin subspace close to an absorbing state. This suggests that even without selection, population structure and sampling memory matter in evolutionary dynamics. It also suggests that memoryless models severely restrict sequential

learning, leading to overestimates of the time to stasis.

Finally, we should stress that none of these phenomena occur in the limit of infinite populations or sample size. The variance due to finite sampling drives sequential learning, the diffusion through process space, and the jumps between isostructural subspaces.

§5.9.2 Applications

Structural drift gives an alternative view of drift processes in population genetics. In light of new kinds of evolutionary behavior, it reframes the original questions about underlying mechanisms and extends their scope to phenomena that exhibit memory in the sampling process. Examples of the latter include environmental toxins [MED07], changes in predation [TRE08], and socio-political factors [KAY05B] where memory lies in the spatial distribution of populations. In addition to these, several applications to areas beyond population genetics proper suggest themselves.

Epochal Evolution

An intriguing parallel exists between structural drift and the longstanding question about the origins of *punctuated equilibrium* [GOU77] and the dynamics of *epochal evolution* [MIT94]. The possibility of evolution's intermittent progress—long periods of stasis punctuated by rapid change—dates back to Fisher's demonstration of metastability in drift processes with multiple alleles [FIS00].

Epochal evolution presents an alternative to the view of metastability posed by adaptive landscapes [WRI32]. Within epochal evolutionary theory, equivalence classes of genotype fitness, called *subbasins*, are connected by fitness-changing *portals* to other subbasins. A genotype is free to diffuse within its subbasin via selectively neutral mutations, until an advantageous mutation drives genotypes through a portal to a higher-fitness subbasin. An increasing number of genotypes derive from this founder and diffuse in the new subbasin until another portal to higher fitness is discovered. Thus, the structure of the subbasin-portal architecture dictates the punctuated dynamics of evolution.

Given an adaptive system which learns structure by sampling its past organization, structural drift theory implies is that its evolutionary dynamics are inevitably described by punctu-

ated equilibria. Diffusion in an isostructural subspace corresponds to a period of structured equilibrium and subspace shifts correspond to rapid innovation or loss of organization.

Thus, structural drift establishes a connection between evolutionary innovation and structural change and identifies the conditions for creation or loss of innovation. This suggests that there is a need to bring these two theories together by adding mutation and selection to structural drift.

Graph Evolution

The evolutionary dynamics of structured populations have been studied using undirected graphs to represent correlation between individuals. Edge weights w_{ij} between individuals i and j give the probability that i will replace j with its offspring when selected to reproduce.

By studying fixation and selection behavior on different types of graphs, Lieberman et al found, for example, that graph structures can sometimes amplify or suppress the effects of selection, even guaranteeing the fixation of advantageous mutations [LIE05].

Jain and Krishna [JAI02] investigated the evolution of directed graphs and the emergence of self-reinforcing autocatalytic networks of interaction. They identified the attractors in these networks and demonstrated a diverse range of behavior from the creation of structural complexity to its collapse and permanent loss.

Graph evolution is a complementary framework to structural drift. In the latter, graph structure evolves over time with nodes representing equivalence classes of the distribution of selectively neutral alleles. Additionally, unlike ϵ-machines, the multinomial sampling of individuals in graph evolution is a memoryless process. A combined approach would allow one to examine how amplification and suppression are affected by external influences on the population structure; for example, including how a population might use temporal memory to maintain desirable properties in anticipation of structural shifts in the environment.

Molecular Clocks

The notion that evolutionary changes occur at regular time intervals was introduced more than 40 years ago by Zuckerkandl et al [ZUC62]. Such regularity, when it holds, allows one to estimate the date of common ancestry for two divergent species. Kimura's theory of neutral evolution lent support to the idea of molecular clocks by stating that most selectively neutral single-nucleotide

mutations accumulate at the same rate across species due to fixed error rates in DNA replication [KIM68].

Molecular clocks have been controversial due to uncertainties about the molecular mechanisms on which they rely and due to the discovery of varying mutation rates. Several modifications to the theory were proposed, though none is considered universally satisfactory [HER03]. Schwartz et al proposed that regular changes in germ cells are "not, in our present understanding of cell biology, tenable" [SCH06] and, instead, they suggested evolutionary changes happen suddenly and without clock-like regularity.

The phenomena of epochal evolution and structural drift offer alternative views of the evolutionary dynamics underlying molecular clocks, modeling periods of stable diffusion punctuated by rapid change. Specifically, the architecture of generalized drift process space could dictate how and where structured populations could have diverged in the past. They also challenge us, however, to design experiments for testing if their mechanisms operate during evolutionary change. In vitro evolutionary experiments with bacteria [ELE96] seem particularly amenable to this kind of investigation.

Sequential Learning in Communication Chains

Let's briefly return to our motivating problem of learning in chains of sequential communication channels. In the drift behaviors explored above, the M_T went to stasis ($h_\mu = 0$) corresponding to periodic formal languages. Clearly, such a long-term condition falls short as a model of human communication chains. In the latter, the resulting messages, though distant from those at the beginning of the chain, are not periodic. In addition, human language is constantly varying, and stasis points are transient. To more closely capture drift in the context of sequential language distortion, mutation and selection must be added to prevent permanent stasis and give preference to intelligible phrases.

However, the current framework does capture the language-centric notion of dynamically changing semantics. The symbols and words in the strings generated have a semantics given by the structure of the ϵ-machine[CRU92]. Briefly, causal states provide dynamic contexts for interpretation of individual symbols and words. Moreover, the allelic complexity is the total amount of semantic content that can be generated by an M_t. In this way, changes in the architecture of

the M_t during drift correspond to semantic innovation and loss.

§5.10 Final Remarks

Structural drift is amenable to extension and application to a range of biological problems, as was the original Fisher-Wright drift model. Here, we focused on motivating the model, drawing comparisons to Kimura's theory, and explaining the basic mechanisms underlying the resulting phenomena. We will report elsewhere on more technical aspects including a predictive theory and extensions that include mutation and selection. We close by indicating some of the challenging open technical problems.

ϵ-Machine minimality allowed us to monitor information processing, information storage, and causal architecture during the drift process. However, due to the influence of inference parameters on ϵ-machine reconstruction, it will be more realistic to use non-minimal representations in the drift process as populations and sampling processes need not be minimal. One would transform these representations to ϵ-machines so that informational properties still can be properly measured.

There are various kinds of memory in structural drift with mutation that one must distinguish. On the one hand, a high mutation rate destroys a population's ability to remember its past. A high mutation rate leads to more rapid exploration and discovery of beneficial genotypes, but past innovations are rapidly forgotten. On the other hand, a vanishingly small mutation rate leads to an arbitrarily slow evolution process: There are no innovations worth remembering. This kind of population memory is not necessarily the same as the sampling memory implicated in basic structural drift. Nonetheless, these types of memory interact and this interaction must eventually be understood.

We demonstrated how structural drift—diffusion and structural innovation and loss—are controlled by the architecture of connected isostructural subspaces. Many questions remain about these subspaces. What is the degree of subspace-jump irreversibility? Can we predict the likelihood of these jumps? What does the phase portrait of a drift process look like? Thus, to better understand structural drift, we need to analyze the high-level organization of generalized drift process space.

Fortunately, ϵ-machines are in one-to-one correspondence with structured processes. Thus, the preceding question reduces to understanding the space of ϵ-machines and how they can be connected by diffusion processes. Is the diffusion within each process subspace predicted by Kimura's theory or some simple variant? We have given preliminary evidence that it does. And so, there are reasons to be optimistic that in face of the original complexity of structural drift, a good deal can be predicted analytically. And this, in turn, will lead to quantitative applications.

CHAPTER 6
Looking Forward to Looking Back

Surveying the current research landscape in computer security, I have a feeling of frustration. A feeling, shared by some with far more experience than myself, that the field seems stuck in limit cycle where decades-old bugs and design flaws continually reappear in both new and existing technologies. We know that security must be integrated into design, yet never is, and so the timeless struggle between exploit and patch marches on.

In some sense this dilemma is very predictable, though its components may not be. Technology continues to radically transform global society and does so in often unexpected ways. Few could have predicted the near-triumph of the Green Revolution in the 2008 Iranian elections via Twitter and other social media. In response, a pro-regime group launched a DNS poisoning attack against Twitter, redirecting most Twitter users to the group's homepage. Suddenly a tool for broadcasting daily minutiae became a fulcrum of freedom, and yet this global medium was subverted by a handful of miscreants wielding a flaw in the domain name system first documented in 1993 [SCH93]. While the rise of Twitter may have been surprising, its takedown was not.

There are many less dramatic examples, but this one touches on many of the causes of this cycle of insecurity. First is the problem of *composability*: to manage complexity we engineer independent layers of functionality, and understanding the emergent[1] interaction of these composable layers is an effort that spans nearly every science. In computer security, this is compounded by the asymmetric relation between attacker and defender [AT01, RUI09]: while the attacker needs only one opening in an attack surface [HOW05], the defender must plug every conceivable hole within the constraints of budget, usability, and performance. The defender's task is compounded by the insider problem and varying degrees of "insiderness": an employee that sets

[1] Ah, emergence—a term as maligned and nebulous as it is ubiquitous.

up their own wireless access point[2], or trojan USB sticks[3], or posts a socially-engineerable factoid to their Facebook page increases the attack surface. Last but not least, how can we develop secure systems when computational substrates are constantly evolving? When the free market demands our kitchen appliances can connect to Twitter? In contrast, how often does a physicist have to adapt to changes in classical mechanics?

In my studies, I examined how other fields—robotics, neuroscience, genetics, complex networks, nonlinear dynamics, statistical mechanics—engage the astounding complexity of both natural and artificial systems. One anecdote that crystallized this battle with complexity was given by Professor Raissa D'Souza[4] who explained how Tom Knight at MIT would sometimes point to a diagram of a biological network with thousands of interactions. "See this single link here? That was a Ph.D dissertation." Surely the natural sciences have some recommendations for proceeding in the face of overwhelming complexity.

Others have picked up on this idea. A quick perusal of many security conference proceedings will show there is some outward-looking research, and a particular influx of work motivated by biological metaphors. Though valuable to explore, these can be murky waters to navigate: see the debate on connectionist versus classical architectures in the cognitive sciences [FOD88] for one of many historical fallouts over tenuous metaphors. Are we learning the right lessons from history, or do we simply want to take an algorithm here and a buzzword there [SHA56, ELI58]? Instead of copying metaphors, should we be adapting methodologies?

Look no further than the study of complex networks for a recent example of a field rejuvenated by going deeper than the metaphor. Lead by researchers such as Barabási and Albert in the late 1990s and early 2000s, the techniques of statistical physics were applied to problems in graph theory to create the new, hybridized field of complex networks [ALB02]. Tools such as percolation theory and nonlinear dynamics have brought new understanding to models of network growth and robustness [BAR99, ALB00], optimal resource distribution [GAS06], community structure and clustering [NEW06, CLA08], and epidemic models [KEE05].

There are faint echoes of such connections with computer science, including the application of symbolic dynamics [MAR95] and dynamical systems [MYT09]. In particular, the study of

[2] http://bit.ly/baVZb6
[3] http://bit.ly/bww45j
[4] Personal correspondence.

phase transitions in the solution space of K-SAT has lead to a new algorithm for finding satisfying assignments of boolean variables [Mól02], and could contribute to our understanding of other problems in NP. As more mathematically trained scientists become interested in our field, might one day a security researcher require both a CISSP certification and the ability to write down the Hamiltonian of a system?[5]

Driven by such ideals, I combined the non-parametric methods of information theory with the techniques of computational mechanics to model systems where we don't have the domain knowledge to make an informed selection of model parameters. Of course, these ideals break down in practice—finite data affects the optimality of our models, non-stationarity creeps in, and alphabet size explodes runtime. That is, assuming we have data at all. Legitimate concerns for privacy made my initial research interest, the extension of Beddoe's work on protocol inference, nearly impossible. In the time that passed between having the ideas and having the data, all but two novel contributions had been made by other papers. In the meantime, I explored an ϵ-machine approach to several ideas from population genetics, resulting in my work on structural drift. This required solving several rare corner cases for finite data ϵ-machine reconstruction, as well as general improvements to implementation robustness and efficiency, that proved essential to my later work. Along the way, we generalized a long standing result in population dynamics. Finally, with the emergence of a new project to detect anomalies on parallel computers given gigabytes of log files, I was able to more fully develop an inter-disciplinary computational mechanics framework that integrates machine learning, information theory, and ϵ-machine reconstruction. This framework enables reconstruction of previously prohibitive processes. Most simply, this is my contribution.

While insights from all three projects contributed to the framework, there is much room left for experimental validation and new ideas. I aim to explore the connection between ϵ-machine reconstruction and grammatical inference [DE 05], as well as the broader class of graphical models that ϵ-machines belong to [JOR98]. Along these lines, the effectiveness of ϵ-machines will be evaluated against other graphical models using established model selection and validation frameworks [BUR02] including Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC). The anomaly detection work is an ideal staging ground for this evaluation as we are

[5] No—the CISSP will be long dead by then.

constructing multiple types of classifiers that should be evaluated not only in terms of their predictive power, but in terms of their number of parameters. Initial work applying AIC and BIC to selection of traditional HMMs and ϵ-machines suggests we can leverage the advantages of both models. This selection will be extended to other probabilistic models such as those inferred by the ALERGIA algorithm [DE 05].

Returning to the protocol inference work, I plan to investigate directed information transfer [MAR73, MAS90], independent component analysis [BEL95], and co-information [BEL03] for more robust identification of protocol header boundaries. Here, the idea is to treat protocol messages as signals to be separated by their statistical properties. This again depends on access to data that was previously the limiting factor in developing a robust preprocessing step for protocol ϵ-machine reconstruction.

Finally, we plan to apply the ideas of structural drift to empirically testable domains. Our initial direction is the evolution of human language: how the structural properties of a language evolve over time, how words and phrases are innovated or lost, and how different language subspaces may be connected. In the presence of massive language corpi and increasingly accurate translation tools, we can ask a new set of questions. Can we measure the time to stasis of language? What are the rates of loss incurred by these tools? How is stasis time affected by language innovation? How is sequential inference influenced by interacting language dynamics? Many questions present themselves.

I certainly don't claim that I have or will solve the grandiose problems outlined in this chapter. I am merely motivated by the desire to question:

- Why model? [EPS08]
- Why *that* model? [BUR02]
- Why those parameters? [KEO04] (Most famously, "Why six?" [TAN02])
- Why parameters at all? [SHA48, LEE01]

In doing so, I hope this dissertation is the first step in a long journey towards understanding the complexity of security: less of a random walk, and more of a directed run.

Bibliography

ALB00 Albert, Réka, Hawoong Jeong, and Albert-László Barabási. "Error and attack tolerance of complex networks." *Nature, volume 406* (6794), 2000: pages 378–382.

ALB02 Albert, Réka and Albert-László Barabási. "Statistical mechanics of complex networks." *Reviews of Modern Physics, volume 74* (1), 2002: pages 47–97.

AT01 Arreguín-Toft, Ivan. "How the Weak Win Wars: A Theory of Asymmetric Conflict." *International Security, volume 26* (1), 2001: pages 93–128.

BAR99 Barabási, Albert-László. "Emergence of Scaling in Random Networks." *Science, volume 286* (5439), October 1999: pages 509–512.

BED05 Beddoe, M. "Network Protocol Analysis using Bioinformatics Algorithms." Technical report, McAfee Inc., 2005.

BEL95 Bell, Anthony J. and Terrence J. Sejnowski. "An information-maximization approach to blind separation and blind deconvolution." *Neural Computation, volume 7* (6), 1995: pages 1129–1159.

BEL03 Bell, Anthony J. "The co-information lattice." In *Proceedings of the International Workshop on Independent Component Analysis and Blind Signal Separation*, 2003, pages 921–926.

BIS09A Bishop, Matt, Sophie Engle, Sean Peisert, Sean Whalen, and Carrie Gates. "Case Studies of an Insider Framework." In *Proceedings of the 42nd Hawaii International Conference on System Sciences*, 2009, pages 1–10.

BIS09B Bishop, Matt, Sophie Engle, Sean Peisert, Sean Whalen, and Carrie Gates. "We have met the enemy and he is us." In *Proceedings of the 2008 Workshop on New Security Paradigms*, 2009, pages 1–12.

BLO06 Bloom, Jesse D., Sy T. Labthavikul, Christopher R. Otey, and Frances H. Arnold. "Protein stability promotes evolvability." *Proceedings of the National Academy of Sciences of the United States of America, volume 103* (15), 2006: pages 5869–5874.

BUG05 Bugalho, Miguel and Arlindo L. Oliveira. "Inference of regular languages using state merging algorithms with search." *Pattern Recognition, volume 38* (9), 2005: pages 1457–1467.

BUR02 Burnham, Kenneth P. and David Anderson. *Model Selection and Multi-Model Inference*. Springer, 2002.

CAB09 Caballero, Juan, Pongsin Poosankam, Christian Kreibich, and Dawn Song. "Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering." In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009, pages 621–634.

CHA77 Chaitin, G. J. "Algorithmic information theory." *IBM Journal of Research and Development, volume 21* (4), 1977: pages 350–359.

CLA08 Clauset, Aaron, Cristopher Moore, and M. E. J. Newman. "Hierarchical structure and the prediction of missing links in networks." *Nature, volume 453* (7191), 2008: pages 98–101.

COM09 Comparetti, Paolo Milani, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. "Prospex: Protocol Specification Extraction." In *Proceedings of the 2009 IEEE Symposium on Security and Privacy*, 2009, pages 110–125.

COV91 Cover, Thomas M. and Joy A. Thomas. "Elements of information theory." Wiley-Interscience, 1991.

CRU89 Crutchfield, James P. and Karl Young. "Inferring statistical complexity." *Physical Review Letters, volume 63* (2), 1989: pages 105–108.

CRU92 Crutchfield, James P. "Semantics and Thermodynamics." In M. Casdagli and S. Eubank (Editors) *Nonlinear Modeling and Forecasting*. Addison-Wesley, 1992, pages 317–359.

CRU03A Crutchfield, James P. and David P. Feldman. "Regularities unseen, randomness observed: Levels of entropy convergence." *Chaos, volume 13* (1), 2003.

CRU03B Crutchfield, James P. and Peter Schuster. *Evolutionary Dynamics: Exploring the Interplay of Selection, Accident, Neutrality, and Function*. Oxford University Press, 2003.

CRU06 Crutchfield, James P. and Olof Görnerup. "Objects That Make Objects: The Population Dynamics of Structural Complexity." *Journal of the Royal Society Interface, volume 3* (7), 2006: pages 345–349.

CUI06 Cui, Weidong, Vern Paxson, Nicholas C. Weaver, and Y H. Katz. "Protocol-Independent Adaptive Replay of Application Dialog." In *Proceedings of the 13th Annual Symposium on Network and Distributed System Security*, 2006.

CUI07 Cui, Weidong, Jayanthkumar Kannan, and Helen J. Wang. "Discoverer: automatic protocol reverse engineering from network traces." In *Proceedings of 16th USENIX Security Symposium*, 2007.

DE 05 de La Higuera, Colin. "A bibliographical study of grammatical inference." *Pattern Recognition, volume 38* (9), 2005: pages 1332–1348.

DIE02 Dietterich, Thomas G. "Machine Learning for Sequential Data: A Review." In *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, 2002, pages 15–30.

DON00 Donoho, David L. "High-Dimensional Data Analysis: The Curses and Blessings of Dimensionality." *American Math Society on Math Challenges of the 21st Century*, 2000: pages 1–32.

ELE96 Elena, S. F., V. S. Cooper, and Richard E. Lenski. "Punctuated Evolution Caused by Selection of Rare Beneficial Mutations." *Science, volume 272* (5269), 1996: pages 1802–1804.

ELI58 Elias, Peter. "Two famous papers." *IEEE Transactions on Information Theory, volume 4* (3), 1958: pages 99–99.

ELL09 Ellison, Christopher J., John Mahoney, and James P. Crutchfield. "Prediction, Retrodiction, and the Amount of Information Stored in the Present." *Journal of Statistical Physics, volume 136* (6), 2009: pages 1005–1034.

EPS08 Epstein, Joshua M. "Why Model?" *Journal of Artificial Societies and Social Simulation, volume 11* (4), 2008.

ERM06 Erman, Jeffrey, Anirban Mahanti, and Martin Arlitt. "Internet Traffic Identification using Machine Learning." In *Proceedings of the 49th IEEE Global Telecommunications Conference*, 2006, pages 1–6.

EVA02 Evans, S. C., J. E. Hershey, and G. Saulnier. "Kolmogorov Complexity Estimation and Analysis." Technical Report 2002GRC177, General Electric Global Research, 2002.

FAV08 Fava, Daniel S., Stephen R. Byers, and Shanchieh Jay Yang. "Projecting Cyberattacks Through Variable-Length Markov Models." *IEEE Transactions on Information Forensics and Security, volume 3* (3), 2008: pages 359–369.

FEL68 Feller, William. *An Introduction to Probability Theory and Its Applications, Vol. 1, 3rd Edition*. Wiley, 1968.

FEL88 Felder, Richard M. and Linda K. Silverman. "Learning and Teaching Styles in Engineering Education." *Engineering Education, volume 78* (7), 1988: pages 674–681.

FEL08 Feldman, David P., Carl S. McTague, and James P. Crutchfield. "The organization of intrinsic computation: complexity-entropy diagrams and the diversity of natural information processing." *Chaos, volume 18* (4), 2008.

FIS22 Fisher, R. A. "On the Mathematical Foundations of Theoretical Statistics." *Philosophical Transactions of the Royal Society of London A, volume 222*, 1922: pages 309–368.

FIS36 Fisher, R. A. "The use of multiple measurements in taxonomic problems." *Annals of Eugenics, volume 7*, 1936: pages 179–188.

FIS00 Fisher, R. A. *The Genetical Theory of Natural Selection*. Oxford University Press, 2000.

FL05A Florez-Larrahondo, German, Susan M. Bridges, and Rayford B. Vaughn. "Efficient Modeling of Discrete Events for Anomaly Detection Using Hidden Markov Models." In Jianying Zhou, Javier Lopez, Robert H Deng, and Feng Bao (Editors) *Information Security*, volume 3650 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005, pages 506–514.

FL05B Florez-Larrahondo, German, Zhen Liu, Susan M. Bridges, Anthony Skjellum, and Rayford B. Vaughn. "Lightweight monitoring of MPI programs in real time." *Concurrency and Computation: Practice & Experience, volume 17* (13), 2005: pages 1547–1578.

FOD88 Fodor, Jerry A. and Zenon W. Pylyshyn. "Connectionism and cognitive architecture: a critical analysis." In Steven Pinker and Jacques Mehler (Editors) *Connections and symbols*. MIT Press, 1988, pages 3–71.

FRE77 Freeman, Linton. "A Set of Measures of Centrality Based on Betweenness." *Sociometry, volume 40* (1), 1977: pages 35–41.

GAS06 Gastner, Michael T. and M. E. J. Newman. "Optimal design of spatial distribution networks." *Physical Review E, volume 74* (1), 2006.

GER98 Gershenfeld, Neil. *The Nature of Mathematical Modeling*. Cambridge University Press, 1998.

GIL00 Gillespie, John H. "Genetic Drift in an Infinite Population: The Pseudohitchhiking Model." *Genetics, volume 155* (2), 2000: pages 909–919.

GIL04 Gillespie, John H. *Population Genetics: A Concise Guide*. Johns Hopkins University Press, 2004.

GOD07 Godefroid, Patrice. "Random testing for security: blackbox vs. whitebox fuzzing." In *Proceedings of the 2nd International Workshop on Random Testing*, 2007.

GOU77 Gould, S. J. and Niles Eldredge. "Punctuated equilibria; the tempo and mode of evolution reconsidered." *Paleobiology, volume 3* (2), 1977: pages 115–151.

GRA86 Grassberger, Peter. "How to measure self-generated complexity." *Physica A, volume 140* (1-2), 1986: pages 319–325.

GUY03 Guyon, Isabelle and André Elisseeff. "An introduction to variable and feature selection." *The Journal of Machine Learning Research, volume 3*, 2003: pages 1157–1182.

HAN93 Hanson, James E. *Computational Mechanics of Cellular Automata*. Ph.D. Dissertation, University of California, Berkeley, 1993.

HER03 Hermann, Gilbert. "Current Status of the Molecular Clock Hypothesis." *The American Biology Teacher, volume 65* (9), 2003: pages 661–663.

HOW05 Howard, Michael, Jon Pincus, and Jeannette Wing. "Measuring Relative Attack Surfaces." In D. T. Lee, S. P. Shieh, and J .D. Tygar (Editors) *Computer Security in the 21st Century*. Springer, 2005, pages 109–137.

JAI02 Jain, Sanjay and Sandeep Krishna. "Graph theory and the evolution of autocatalytic networks." In Stefan Bornholdt and Hans Georg Schuster (Editors) *Handbook of Graphs and Networks*. Wiley-VCH Verlag GmbH & Co. KGaA, Weinheim, FRG, 2002.

JOR98 Jordan, Michael I. *Learning in Graphical Models (Adaptive Computation and Machine Learning)*. MIT Press, 1998.

JUR00 Jurafsky, Daniel and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Prentice Hall, 2000.

KAM10 Kamil, Shoaib, Leonid Oliker, Ali Pinar, and John Shalf. "Communication Requirements and Interconnect Optimization for High-End Scientific Applications." *IEEE Transactions on Parallel and Distributed Systems, volume 21* (2), 2010: pages 188–202.

KAY05A Kayacik, H. G., A. N. Zincir-Heywood, and M. I. Heywood. "Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets." In *Proceedings of the 3rd Annual Conference on Privacy, Security and Trust*, 2005.

KAY05B Kayser, Manfred, Oscar Lao, Katja Anslinger, Christa Augustin, Grazyna Bargel, Jeanett Edelmann, Sahar Elias, Marielle Heinrich, Jürgen Henke, Lotte Henke, Carsten Hohoff, Anett Illing, Anna Jonkisz, Piotr Kuzniar, Arleta Lebioda, Rüdiger Lessig, Slawomir Lewicki, Agnieszka Maciejewska, Dorota Marta Monies, Ryszard Pawlowski, Micaela Poetsch, Dagmar Schmid, Ulrike Schmidt, Peter M. Schneider, Beate Stradmann-Bellinghausen, Reinhard Szibor, Rudolf Wegener, Marcin Wozniak, Magdalena Zoledziewska, Lutz Roewer, Tadeusz Dobosz, and Rafal Ploski. "Significant genetic differentiation between Poland and Germany follows present-day political borders, as revealed by Y-chromosome analysis." *Human Genetics, volume 117* (5), 2005: pages 428–43.

KEE05 Keeling, Matt J. and Ken T. D. Eames. "Networks and epidemic models." *Journal of the Royal Society Interface, volume 2* (4), 2005: pages 295–307.

KEO04 Keogh, Eamonn, Stefano Lonardi, and Chotirat Ann Ratanamahatana. "Towards parameter-free data mining." In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pages 206–215.

KIM68 Kimura, Motoo. "Evolutionary Rate at the Molecular Level." *Nature, volume 217* (5129), 1968: pages 624–626.

KIM69 Kimura, Motoo and Tomoko Ohta. "The Average Number of Generations until Fixation of a Mutant Gene in a Finite Population." *Genetics, volume 61* (3), 1969: pages 763–771.

KIM80 Kimura, Motoo. "Average Time until Fixation of a Mutant Allele in a Finite Population under Continued Mutation Pressure: Studies by Analytical, Numerical, and Pseudo-Sampling Methods." *Proceedings of the National Academy of Sciences of the United States of America, volume 77* (1), 1980: pages 522–526.

KIM83 Kimura, Motoo. *The Neutral Theory of Molecular Evolution*. Cambridge University Press, 1983.

KOE06 Koelle, Katia, Sarah Cobey, Bryan Grenfell, and Mercedes Pascual. "Epochal evolution shapes the phylodynamics of interpandemic influenza A (H3N2) in humans." *Science, volume 314* (5807), 2006: pages 1898–1903.

KOH82 Kohonen, Teuvo. "Self-organized formation of topologically correct feature maps." *Biological Cybernetics, volume 43* (1), 1982: pages 59–69.

KOH00 Kohonen, Teuvo. *Self-Organizing Maps*. Springer, 2000.

KOL58 Kolmogorov, Andrey Nikolaevich. "New Metric Invariant of Transitive Dynamical Systems and Endomorphisms of Lebesgue Spaces." *Doklady of Russian Academy of Sciences, volume 119* (5), 1958: pages 861–864.

KOO03 Koo, Ja-Min and Sung-Bae Cho. "Viterbi Algorithm for Intrusion Type Identification in Anomaly Detection System." In Kijoon Chae and Moti Yung (Editors) *Information Security Applications*, volume 2908 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003, pages 2031–2047.

KUL59 Kullback, Solomon. *Information Theory and Statistics*. John Wiley and Sons, 1959.

LEE01 Lee, Wenke and Dong Xiang. "Information-Theoretic Measures for Anomaly Detection." In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 2001.

LEI05 Leita, Corrado, Ken Mermoud, and Marc Dacier. "ScriptGen: an automated script generation tool for honeyd." In *Proceedings of the 21st Annual Computer Security Applications Conference*, 2005, pages 203–214.

LI90 Li, Wentian. "Mutual information functions versus correlation functions." *Journal of Statistical Physics, volume 60* (5-6), 1990: pages 823–837.

LI97 Li, Ming and Paul Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 1997.

LI04 Li, Haifeng, Keshu Zhang, and Tao Jiang. "Minimum Entropy Clustering and Applications to Gene Expression Analysis." In *Proceedings of the 2004 IEEE Computational Systems Bioinformatics Conference*, 2004, pages 142–151.

LIE05 Lieberman, Erez, Christoph Hauert, and Martin A. Nowak. "Evolutionary dynamics on graphs." *Nature, volume 433* (7023), 2005: pages 312–316.

LIN08 Lin, Zhiqiang, Xuxian Jiang, Dongyan Xu, and Xiangyu Zhang. "Automatic protocol format reverse engineering through context-aware monitored execution." In *Proceedings of the 15th Annual Network and Distributed System Security Symposium*, 2008.

LOW02 Lowry, Michelle and G. William Schwert. "IPO Market Cycles: Bubbles or Sequential Learning?" *The Journal of Finance, volume 57* (3), 2002: pages 1171–1200.

LUC91 Lucky, Robert W. *Silicon Dreams: Information, Man, and Machine*. St. Martin's Press, 1991.

MÓ2 Mézard, M., G. Parisi, and R. Zecchina. "Analytic and algorithmic solution of random satisfiability problems." *Science, volume 297* (5582), 2002: pages 812–815.

MA09 Ma, Chao, Yong Meng Teo, Verdi March, Naixue Xiong, Ioana Romelia Pop, Yan Xiang He, and Simon See. "An approach for matching communication patterns in parallel applications." In *Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing*, 2009, pages 1–12.

MAC02 MacKay, David J. C. *Information Theory, Inference & Learning Algorithms*, volume 50. Cambridge University Press, 2002.

MAH03 Mahoney, Matthew V. and Philip K. Chan. "An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection." In *Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection*, 2003, pages 220–237.

MAR73 Marko, H. "The Bidirectional Communication Theory–A Generalization of Information Theory." *IEEE Transactions on Communications, volume 21* (12), 1973: pages 1345–1351.

MAR95 Marcus, Brian. "Symbolic Dynamics and Connections to Coding Theory, Automata Theory and System Theory." In *Proceedings of the AMS Symposium on Applied Math*, 1995, pages 95–108.

MAS90 Massey, J. "Causality, feedback and directed information." In *Proceedings of the 1990 International Symposium on Information Theory and Its Applications*, 1990, pages 303–305.

MCH00 McHugh, John. "Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory." *ACM Transactions on Information and System Security, volume 3* (4), 2000: pages 262–294.

MED07 Medina, Matías H., Juan A. Correa, and Carlos Barata. "Micro-evolution due to pollution: possible consequences for ecosystem responses to toxic stress." *Chemosphere, volume 67* (11), 2007: pages 2105–2114.

MEN25 Mendel, Gregor. *Experiments in Plant Hybridisation.* Harvard University Press, 1925.

MEY01 Meyer, Carl D. *Matrix Analysis and Applied Linear Algebra Book and Solutions Manual.* SIAM, 2001.

MIT94 Mitchell, Melanie, James P. Crutchfield, and Peter T. Hraber. "Evolving cellular automata to perform computations: mechanisms and impediments." *Physica D, volume 75* (1), 1994: pages 361–391.

MOD06 "Modbus Messaging on TCP/IP Implementation Guide 1.0b.", 2006.

MYT09 Mytkowicz, Todd, Amer Diwan, and Elizabeth Bradley. "Computer systems are dynamical systems." *Chaos, volume 19* (3), 2009.

NEW06 Newman, M. E. J. "Modularity and community structure in networks." *Proceedings of the National Academy of Sciences of the United States of America, volume 103* (23), 2006: pages 8577–8582.

NIM99 van Nimwegen, E. "Neutral evolution of mutational robustness." *Proceedings of the National Academy of Sciences of the United States of America, volume 96* (17), 1999: pages 9716–9720.

NOR98 Norris, J. R. *Markov Chains.* Cambridge University Press, 1998.

PAT07 Patcha, A. and Jung-Min Park. "An overview of anomaly detection techniques: Existing solutions and latest technological trends." *Computer Networks, volume 51* (12), 2007: pages 3448–3470.

PIE67 Pielou, E. C. "The use of information theory in the study of the diversity of biological populations." In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pages 163–177.

POS81 Postel, J. "Internet Control Message Protocol.", 1981.

RAB89 Rabiner, Lawrence R. "A tutorial on hidden Markov models and selected applications in speech recognition." *Proceedings of the IEEE, volume 77* (2), 1989: pages 257–286.

RAV07 Raval, Alpan. "Molecular Clock on a Neutral Network." *Physical Review Letters, volume 99* (13), 2007.

RIS83 Rissanen, Jorma. "A universal data compression system." *IEEE Transactions on Information Theory, volume 29* (5), 1983: pages 656–664.

RUI09 Ruighaver, Tobias, Matthew Warren, and Atif Ahmad. "Ascent of Asymmetric Risk in Information Security: An Initial Evaluation." In *Proceedings of the 10th Australian Information Warfare and Security Conference*, 2009.

SAB04 Sabhnani, Maheshkumar and Gursel Serpen. "Why machine learning algorithms fail in misuse detection on KDD intrusion detection data set." *Intelligent Data Analysis, volume 8* (4), 2004: pages 403–415.

SCH93 Schuba, Christoph. *Addressing Weaknesses in the Domain Name System Protocol.* Ph.D. Dissertation, Purdue Universitry, 1993.

SCH06 Schwartz, Jeffrey H. and Bruno Maresca. "Do Molecular Clocks Run at All? A Critique of Molecular Systematics." *Biological Theory, volume 1* (4), 2006: pages 357–371.

SHA48 Shannon, Claude E. "A mathematical theory of communication." *The Bell System Technical Journal, volume 27* (1), 1948: pages 379–423.

SHA56 Shannon, Claude E. "The Bandwagon." *IRE Transactions on Information Theory, volume 2* (1), 1956: pages 3–3.

SHA01A Shalizi, Cosma Rohilla and James P. Crutchfield. "Computational Mechanics: Pattern and Prediction, Structure and Simplicity." *Journal of Statistical Physics, volume 104* (3), 2001: pages 817–879.

SHA01B Shalizi, Cosma Rohilla and Martin Olsson. *Causal architecture, complexity and self-organization in time series and cellular automata.* Ph.D. Dissertation, University of Michigan, 2001.

SHA04 Shalizi, Cosma Rohilla and Kristina Lisa Shalizi. "Blind construction of optimal nonlinear recursive predictors for discrete sequences." In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, 2004, pages 504–511.

SMI88 Smith, Chris. "Send reinforcements we're going to advance." *Biology and Philosophy, volume 3* (2), 1988: pages 214–217.

SOM10 Sommer, Robin and Vern Paxson. "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection." In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, 2010, pages 305–316.

STE02 Steuer, R., J. Kurths, C. O. Daub, J. Weise, and J. Selbig. "The mutual information: Detecting and evaluating dependencies between variables." *Bioinformatics, volume 18* (2), 2002: pages 231–240.

STI07 Still, Susanne, James P. Crutchfield, and Christopher J. Ellison. "Optimal Causal Inference.", 2007.

TAN02 Tan, Kymie M. C. and Roy A. Maxion. “"Why 6?" Defining the Operational Limits of Stide, an Anomaly-Based Intrusion Detector.” In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 2002.

TRE08 Tremblay, Ashley, David Lesbarreres, Thomas Merritt, Chris Wilson, and John Gunn. “Genetic Structure and Phenotypic Plasticity of Yellow Perch (Perca Flavescens) Populations Influenced by Habitat, Predation, and Contamination Gradients.” *Integrated Environmental Assessment and Management, volume 4* (2), 2008: page 264.

VAR07 Varn, Dowman P., Geoffrey S. Canright, and James P. Crutchfield. “Inferring planar disorder in close-packed structures via e-machine spectral reconstruction theory: structure and intrinsic computation in zinc sulfide.” *Acta Crystallographica Section B, volume 63* (2), 2007: pages 169–182.

WAR99 Warrender, Christina, Stephanie Forrest, and Barak Pearlmutter. “Detecting Intrusions Using System Calls: Alternative Data Models.” In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, 1999, pages 133–145.

WIT05 Witten, Ian H. and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. Morgan Kaufmann, 2005.

WON08 Wondracek, Gilbert, Paolo Milani Comparetti, Christopher Kruegel, and Engin Kirda. “Automatic Network Protocol Analysis.” In *Proceedings of the 15th Annual Symposium on Network and Distributed System Security*, 2008.

WRI31 Wright, Sewall. “Evolution in Mendelian Populations.” *Genetics, volume 16*, 1931: pages 97–126.

WRI32 Wright, Sewall. “The roles of mutation, inbreeding, crossbreeding, and selection in evolution.” In *Proceedings of the 6th International Congress on Genetics*, 1932, pages 355–366.

ZUC62 Zuckerkandl, E. and L. B. Pauling. “Molecular disease, evolution, and genetic heterogeneity.” In M. Kasha and B. Pullman (Editors) *Horizons in Biochemistry*. Academic Press, 1962, pages 189–225.