Tree Stochastic Processes

Kevin Tian
University of Pennsylvania, ktian@cis.upenn.edu

Tree Stochastic Processes

Abstract

Stochastic processes play a vital role in understanding the development of many natural and computational systems over time. In this thesis, we will study two settings where stochastic processes on trees play a significant role. The first setting is in the reconstruction of evolutionary trees from biological sequence data. Most previous work done in this area has assumed that different positions in a sequence evolve independently. This independence however is a strong assumption that has been shown to possibly cause inaccuracies in the reconstructed trees \cite{schoniger1994stochastic,tillier1995neighbor}. In our work, we provide a first step toward realizing the effects of dependency in such situations by creating a model in which two positions may evolve dependently. For two characters with transition matrices M_1 and M_2, their joint transition matrix is the tensor product $M_1 \otimes M_2$. Our dependence model modifies the joint transition matrix by adding an `error matrix,' a matrix with rows summing to 0. We show when such dependence can be detected.

The second setting concerns computing in the presence of faults. In pushing the limits of computing hardware, there is tradeoff between the reliability of components and their cost (e.g. \cite{kadric2014energy}). We first examine a method of identifying faulty gates in a read-once formula when our access is limited to providing an input and reading its output. We show that determining \emph{whether} a fault exists can always be done, and that locating these faults can be done efficiently as long as the read-once formula satisfies a certain balance condition. Finally for a fixed topology, we provide a dynamic program which allows us to optimize how to allocate resources to individual gates so as to optimize the reliability of the whole system under a known input product distribution.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Computer and Information Science

First Advisor
Sampath Kannan

Subject Categories
Computer Sciences

STOCHASTIC TREE PROCESSES

Kevin Tian

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial

Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2016

ABSTRACT

STOCHASTIC TREE PROCESSES

Kevin Tian

Sampath Kannan

Stochastic processes play a vital role in understanding the development of many natural and computational systems over time. In this thesis, we will study two settings where stochastic processes on trees play a significant role. The first setting is in the reconstruction of evolutionary trees from biological sequence data. Most previous work done in this area has assumed that different positions in a sequence evolve independently. This independence however is a strong assumption that has been shown to possibly cause inaccuracies in the reconstructed trees [76, 83]. In our work, we provide a first step toward realizing the effects of dependency in such situations by creating a model in which two positions may evolve dependently. For two characters with transition matrices M_1 and M_2, their joint transition matrix is the tensor product $M_1 \otimes M_2$. Our dependence model modifies the joint transition matrix by adding an 'error matrix,' a matrix with rows summing to 0. We show when such dependence can be detected.

The second setting concerns computing in the presence of faults. In pushing the limits of computing hardware, there is tradeoff between the reliability of components and their cost (e.g. [46]). We first examine a method of identifying faulty gates in a read-once formula when our access is limited to providing an input and reading its output. We show that determining *whether* a fault exists can always be done, and that locating these faults can be done efficiently as long as the read-once formula satisfies a certain balance condition. Finally for a fixed topology, we provide a dynamic program which allows us to optimize how to allocate resources to individual gates so as to optimize the reliability of the whole system under a known input product distribution.

Contents

Chapter 1

Introduction

The study of natural and computational processes requires us to understand the influence of random events that are often a part of them. In biology, a central question is inferring the details of the evolutionary process that has given rise to all the extant species. The biomolecular approach to the study of evolution is to analyze the variation of biological sequences over time and across species. For example, genes are encoded in DNA by sequences of nucleotides, and over time, mutations affect these sequences by substituting certain nucleotides for others, or adding or deleting nucleotides, or more complex operations. This is modeled by having a random variable associated with each index of the sequence, with states corresponding to nucleotides. We can then model the evolution of these random variables over time.

In computing, a question of renewed interest is how reliable circuits can be built from unreliable components. Applications for modern computing devices often push on hardware boundaries, whether due to power constraints for devices operating in hostile environments, or continued miniaturization even in mainstream computers. These applications are sometimes forced to use components which function in the given environments, but give up reliability in exchange. This unreliability manifests as errors that occur during computation, and errors in even just parts of a circuit can cause the output of the entire circuit to be wrong. We model this process by representing the values transmitted by wires between components as random variables. These variables capture the random faults occurring in faulty components, and we study the interactions of the variables and the faults over the course of a computation. This process as well as the biomolecular process for evolution are examples of *stochastic processes*.

A stochastic process is a set of time-dependent random variables. In evolution, the sequence is the set of random variables, and in computing with unreliable components, the values transmitted along wires which are subjected to random failures at gates are the random variables. Stochastic processes are studied in a variety of contexts, such as machine learning and artificial intelligence. They are also studied outside computer science, for example physics, where stochastic processes model thermodynamic and dynamical systems [44, 70], and finance, where stochastic processes are used as predictors for markets.

One classical example of a stochastic process is a branching process, which models a population where each individual in a generation randomly yields a number of offspring for the next generation. The number of offspring generated by each individual is represented by a random variable. The main question is the probability of extinction – is there a point in time when the population reaches zero?

A related process is the random walk; a random walk describes the path taken by a randomly-moving object. The process describes how the initial position x_0 of the object is (randomly) chosen, and then how the object moves, choosing a location x_{i+1} based on its previous location x_i. A branching process in which individuals are considered to be identical is a random walk, which tracks the movement of the size of the population over successive generations. There is a significant body of work studying properties of random walks, such as the average time taken to reach a specific state: the minimum i for which x_i is a specific state s (for example, when the population size reaches 0 in a branching process), how often the walk reaches a certain state: the limit over time of the fraction of times i for which x_i is a specific state s, or for finite settings, the time taken to have visited every state at least once.

A specific case is the random walk on a graph. At each time step, we are at some vertex v of the graph, and we choose one of the neighbors of v uniformly at random, and move to that vertex. This is also an example of a finite Markov chain, a stochastic process which transitions between a set of states. The distinguishing feature of a Markov chain is that the distribution of the next state, x_{i+1}, depends only on the current state x_i and not any preceding states $x_{i-1}, x_{i-2}, \ldots, x_0$. Observe in the random walk on a graph that this is true: the next vertex we choose depends only on the current vertex v, in that it is a neighbor of v, but the history of where the random walk has already been does not factor in at all. This property is called the Markov property, or sometimes 'memorylessness' since once we reach a state we can forget the previous states. Markov chains in the general literature can be continuous-time or discrete-time. In this dissertation, we will focus only on discrete-time Markov chains. Evolution in particular is generally viewed as a continuous-time process, but we will examine the process from an angle that makes it discrete-time.

A Markov chain can be viewed as a stochastic process with the Markov property on a path, where each vertex represents a discrete point in time. The vertices in one direction are the points in time occurring earlier, and the vertices in the other direction occur later. We can generalize from paths to trees, where multiple distinct paths from the past can contribute to a current state, and where a current state influences multiple distinct future paths. These represent two operations occurring on trees: the *merging* of past paths, and *branching* into future paths. Abstractly, the Markov property now says that for any vertex v, its removal separates the tree into components whose values are independent given the value of v (observe how this lines up with the path version when the tree does not branch or merge). This combines the Markov chain intuition of the present state capturing all the relevant information about the future evolution of the system, along with simply enforcing that the distinct past processes or distinct future processes are actually distinct.

Evolution is an example of a branching tree stochastic process. An ancestor species yields multiple modern species. In addition, the model of evolution we use will exhibit memorylessness – the sequence at a node is generated from its immediate parent. Its ancestors further up the tree do not affect its sequence, except for how they have already affected the sequence of the parent. We will consider another stochastic process, of computation in the presence of faulty components on read-once formulas. Read-once formulas are formulas for which each input appears only once, leading to a tree-like structure for the formula. This is an example of a tree stochastic process with merging paths. In a read-once formula, we see another manifestation of memorylessness: the output of a node is the totality of the relevant information to its parent. How the node came to reach this output is irrelevant to the parent.

1.1 Evolutionary Trees

Reconstructing the phylogeny or evolutionary tree of a set of organisms is a very important problem in bioinformatics ([30], [77]). The general formulation of the problem is the following: data corresponding to the species alive today is observed at the leaves of an (unknown) tree that models the evolutionary history of these species. The goal is to find the best tree 'fitting the data' under some specified objective function. Nowadays, the most common type of data we observe is biomolecular sequences, such as DNA or protein sequences.

Let σ_i be the sequence obtained from the i-th species. These input sequences are taken to be *aligned*, that is if each σ_i are written down in a table, one in each row, then each column represents the same location in a sequence. Formally, for all i, i' and for any position j, $\sigma_i[j]$ and $\sigma_{i'}[j]$ come from a common process[1], where $\sigma_i[j]$ represents the j-th symbol in the i-th sequence σ_i. The most principled method of finding a phylogeny is to view the evolution of each position of the aligned sequences as a stochastic process, more specifically, a tree Markov random field, whose parameters are chosen from a rich family of possible parameters.

Let T be a rooted tree underlying an evolutionary process. The internal vertices of the tree represent speciation events, and are points where the process branches into separate and distinct future paths. A *character* on such a tree is a stochastic process that takes on a value at each point of the tree from a set of finitely many *states* (for example, the 4 nucleotides or 20 amino acids). At the root of T, the value is assumed to be selected from some initial distribution over the states. Each parent then attempts to pass its state to its children. However, the state is mutated along each edge with probabilities given by a Markov transition matrix corresponding to the edge. Every position in a set of aligned molecular sequences is regarded as a character evolving in this manner, and we observe the states of the leaves for each character.

[1]This might sound circular, since alignment seems to require knowledge of the evolutionary process which we have stated as our goal. However, biologists have implemented this process so successfully that is it a standard technique now in building phylogenies.

All commonly-studied families of stochastic models of evolution are tree stochastic processes. Among the simplest are two-state symmetric models, called the Cavender-Farris-Neyman (CFN) ([9], [23], [64]) models, where on any edge e, all characters have a symmetric 2×2 transition matrix M_e. The Jukes-Cantor model is a 4-state model where for any transition matrix there is a parameter ϵ that is the probability of any change of state ([45]). The Kimura model is another 4-state model, where the 4 states are paired (like the base pairs in DNA), and the probability of change to the paired state is lower than the probability of change to a state in the other pair ([49]).

In this field, *the tree itself is considered to be fixed, but unknown.* There are three main approaches to computational phylogenetics: distance-matrix methods, maximum parsimony, and maximum likelihood. We will give a brief overview of these methods; for a more complete discussion, we refer the reader to [32]. Distance-matrix methods, as the name suggests, are methods that attempt a reconstruction using a matrix of distances between species, which are often defined as the fraction of mismatches in the aligned sequences. The general idea of distance-matrix methods is to create clusters of species, which would ideally correspond to the species in a subtree of the true evolutionary tree. Variations occur in details like how distances involving clusters are treated. The main such methods are neighbor-joining, introduced by Saitou and Nei [74], Unweighted Pair Group Method with Arithmetic Mean (UPGMA) and Weighted Pair Group Method with Arithmetic Mean (WPGMA), attributed to Sokal and Michener [79], and the Fitch-Margoliash method [33]. The advantage of distance-matrix methods was in the computational efficiency, but they are not resilient to the sorts of errors which occur in distance measurements.

Authors such as Edwards, Cavalli-Sforza, Camin, and Hendy and Penny considered maximum parsimony methods [8, 16, 41, 78]. Parsimony methods are those which attempt to minimize the number of mutations which occur in the tree. Felsenstein also considered parsimony in his 1973 paper [25], but found in 1978 that parsimony could perform very poorly and ultimately converge to the wrong tree [26]. As parsimony minimized the number of mutations that occurred, it suffered when the number of mutations was not small, as is the case in many biologically-realistic scenarios [28].

Instead, Felsenstein proposed the use of a Maximum Likelihood objective function in

1981 [28] to find the most likely values of the parameters given the observed data at the leaves. Chang continues this line in his 1995 paper, and shows that maximum likelihood, unlike parsimony is consistent for reconstructing evolutionary trees [11]. In addition, he considers the question of reconstructing the remaining parameters of the evolutionary tree, in addition to its topology. He shows that maximum likelihood is in fact consistent when it is able to also recover the transition matrices on the edges. The shortcoming of these works leveraging Maximum Likelihood Estimations was computational intractability. Farach and Kannan address this in their 1998 paper [22]. In this paper, they provide the first algorithm which provably converges to the correct tree while also running in polynomial time.

A second question often considered is how many sample species are required for reconstructing the tree. Erdos showed that a number of characters which is polylog in the number of leaves is sufficient in 'almost' all trees [18, 19]. The question of how many samples are required is particularly well-studied in the CFN model. Steel, in 2001, conjectured that $O(\log n)$ characters would suffice if the probability of transition ϵ on each edge of the tree were upper bounded by $\epsilon^* = (\sqrt{2} - 1)/2^{3/2}$ [81]. Mossel, in 2004, proved that this is true for balanced trees, and also showed that if the transition probability exceeded this critical value, that some trees required at least a polynomial number of characters [60]. The full conjecture was then proved by Daskalakis, Mossel, and Roch in a 2006 paper [13]. The result is of particular interest because the number of characters required exhibits a 'phase transition' at the critical error value ϵ^*, where below this value only $O(\log n)$ characters are required, but above this value, we require $n^{\Omega}(1)$ characters.

In all of these works however, there is an underlying assumption that the stochastic processes governing each character are independent and identically distributed. We take aim at the first portion of this assumption, the independence. This assumption is too strong. Dependence between characters arises because changes at one position of a DNA sequence or amino acid sequence are likely to be correlated with changes at other positions because of such constraints as size, charge, and hydrophobicity on the molecules involved ([59], [58]). For example, the double-helical structure of the DNA and the fact that each turn of the double helix corresponds to roughly 7 nucleotides means that there are likely to be dependencies between a character and another that is 7 positions away.

There is some previous work concerning dependence in characters, in both detection and in studying its effect on the methods we have presented above. Schoniger (1994) showed through experiments that due to independence assumptions, methods for reconstructing evolutionary trees tended to underestimate edge lengths. He also provides a procedure for accounting for these underestimates [76]. Tillier (1995) also studied how dependencies would affect algorithms that assumed independence of characters. His computer simulations suggested that while Maximum Likelihood Estimates did not suffer significantly, neighbor-joining methods did not fare well in the presence of dependence [83]. As dependence can have a real effect on algorithms for reconstructing evolutionary trees which assume independence of characters, it is important to understand how to detect dependence, so we are at least aware of when our algorithms may be performing suboptimally.

Previous work also exists directly for the question of detecting independence. In 1994, Pagel used a likelihood ratio test to provide a statistical method for detecting independence [66]. His methods have the property that they do not rely on any prior knowledge of ancestral states or the tree topology, but his methods were limited to rate-based evolutionary models. Felsenstein (1996) used methods on Hidden Markov models and a likelihood test to detect a different type of dependence in rate-based evolutionary models, one where the rates themselves were correlated [31]. However, once rates were fixed, the changes were taken to occur in uncorrelated fashion.

In our work, we consider specifically the question of detecting dependence in the changes of states in pairs of characters. Like Pagel, we will separate the detection process from the reconstruction question completely, and aim solely to detect dependence. Our result looks at a particular dependence model that modifies a baseline stochastic model of evolution.

We consider two characters, each evolving following a baseline stochastic model of evolution (for example, CFN, though we are certainly not restricted to rate-based models of evolution). The dependence model modifies their joint probability distribution. In the case that the two characters evolve independently, their joint transition matrix is simply the tensor product of the marginal transition matrices. If they are dependent, then the joint transition matrix is modified by adding an ‘error’ matrix. In the simplest model, the error matrix is a rank-1 matrix, where each row is some vector $\vec{d}^\top$ whose entries sum to 0. The matrix is the same across all edges of the tree. We call this the 'uniform rank-1' model of dependence. We also consider a more complex model in which the error matrix consists of rows which are vector that generally point in the same direction. More precisely, there is a unit vector $(\vec{v}^*)^\top$ for which each row $\vec{v}^\top$ of each error matrix satisfies that the dot product of $\vec{v}^\top$ with $(\vec{v}^*)^\top$ is at least some fixed constant δ. This model also allows the error matrix to vary from edge to edge. We call this a 'directional-drift' model of dependence.

Since we are looking to determine the nature of the joint transition matrices of a pair of characters, this question is one of determining hidden variables in a stochastic process. As in previous work, both the tree and the transition matrices are hidden from us, and all we observe are the states of characters for various modern species. We are able to show that under many classical stochastic models of evolution, along with some more general theoretical models, one can distinguish between a joint process which evolves as the tensor product of the marginal distributions, and a joint process in which the tensor product has been modified by an error matrix.

1.2 Computation in the presence of faults

The problem of reliably computing functions using circuits with unreliable components was first studied by von Neumann [86]. He considered a probabilistic failure model where the failure was per gate. Each failing gate would produce an output that is the complement of the correct output, relative to the inputs it is given. Other failure models also exist, for example a gate can be 'stuck at' a value, and always output that value [37, 52].

In von Neumann's original paper [86] in 1956, he showed that if the probability ϵ of failure satisfies $\epsilon < .0073$, then it is possible to construct a formula where the output is correct with probability strictly greater than $\frac{1}{2}$. His techniques could be extended slightly to allow for errors with $\epsilon < .09471$. Von Neumann's result suggested that careful use of majority gates could improve this bound. An easy check shows that a formula consisting only of majorities copying a single node will have a correctness probability bounded above $\frac{1}{2}$ as long as $\epsilon < \frac{1}{6}$. It was not until 1991 however, that Hajek and Weller [39] showed that if each gate fails with probability $\epsilon < \frac{1}{6}$, then it is possible to construct a formula to compute a boolean function with correctness probability strictly more than $\frac{1}{2}$. In addition, they show that if $\epsilon \geq \frac{1}{6}$ that computation can not be done reliably. In 2003, Evans and Schulman extended this result (with different bounds) to formulas with gates of any fixed odd arity [21].

In a different direction, Dobrushin and Ortyukov showed in 1977 both that for a formula of size L, reliable computation could be done with a formula of size at most $O(L \log L)$ [15]. They also attempted to show that if a formula has sensitivity s that a formula of size $s \log s$ is required for reliable computation [14]. However, their proof contained a flaw, which was fixed by Gács and Gál in their 1994 paper [1]. In spite of these, Pippenger showed in 1985 that almost all boolean functions can be computed with only constant overhead – that is $O(L)$, rather than $O(L \log L)$ [67, 68].

We can view a faulty gate as one whose output is fed through a binary symmetric channel, with noise ϵ. The second eigenvalue of the transition matrix representing this channel is $1 - 2\epsilon$. Evans and Schulman (1993) were able to use this eigenvalue to bound the mutual information between the inputs and the output in formulas. This allowed them to improve lower bounds in all arities except for the arity 3 case (which Hajek and Weller had already solved at this point) [20].

Assaf and Upfal took a different direction for faulty computation [4]. A comparator is a gate which takes 2 inputs, and then places the smaller of the two on the first output and the larger on the second. They showed that in a very general error model, where failures could be that the outputs were switched, or the same input is copied on both outputs, that they could still use these faulty 2-input comparators to construct circuits which would sort their n inputs.

While earlier results focused on general formulas, our work involves only read-once formulas. A read-once formula is a formula where each variable appears exactly once. This makes the topology of the formula a tree, unlike general formulas, where there can be many nodes connected to each input. Due to this property, read-once formulas are often studied in place of general formulas, where repeated availability of inputs causes complex interconnectivity. Read-once formulas are necessarily the smallest formulas that depend on each of their variables.

For the personal computers we use today, faults do not occur nearly as often as the rates von Neumann originally showed feasible in 1956. However, with continued miniaturization for consumer products to embedded systems designed to run on minimal power, there is a significant amount of computing we would like to be able to do in environments that are not as lenient as the environments of our personal computers. In such contexts, everything comes with a cost: miniaturization requires an increase in voltage to maintain reliability, but this comes with a cost in energy usage [65]. Of course, power use itself is something we aim to minimize. The results above concerning the history of fault-tolerant computing suggest that we can construct our formula to be slightly larger, and this would improve reliability. However, increasing the number of gates also costs us in energy. Kadric et al. consider this regime and show a result that allows an improvement of reliability, which costs less energy than having a larger circuit or improving the reliability of each gate [46]. Their result works with 'components' which are generally multiple gates. However, we will abstract this to be single gates.

We will deal with circuits where errors occur on a per-gate basis. We approach this problem under the significant constraint that we are not looking to redesign circuits. We consider two questions in this regime. Under the topology restriction, we solve the following problems: 1. diagnosis of faults in a given formula, and 2. determining an optimal allocation of resources to a formula where the topology is fixed, but we can improve or sacrifice gate quality for a cost. In the von Neumann line of literature, the read-once requirement does not appear, but we can show for example in the diagnosis problem that diagnosing a read-once formula both exactly and efficiently is information-theoretically impossible. On the other hand, for the allocation problem, the dependencies that arise in different parts of the formula when inputs occur in multiple places complicates the problem greatly. Thus, we will focus on read-once formulas.

Faulty computation on read-once formulas is an example of a stochastic process where distinct past paths are *merged* into a current state. The distinctness of the paths comes from the fact that we consider only read-once formulas, otherwise multiple occurrences of inputs would cause multiple paths to be influence by the same inputs. The computation of the value at a node v depends on the computation of the values of children of v. Once the value of v is known, however, the values of the children are not needed for determining the value of the parent of v. We consider two problems in this regime.

In the diagnosis problem, we are given an explicit formula, which we call the *blueprint*, and a real implementation of the formula. This implementation, which we will call the 'real formula' may have any number of faulty gates. In this problem, however, we will have deterministic faults. A faulty gate will *always* output the complement of the correct answer [2]. We have input-output access to the real formula: we are allowed to provide it with any n-bit boolean input (called a probe) and observe the 1-bit output it produces. We seek necessary and sufficient conditions on the formula that allow us to determine if the blueprint and real formulas are the same using polynomially many queries. In addition, when they are not the same, we aim to identify exactly which gates are faulty, also within polynomially many probes.

We will show that determining whether there is a difference between the blueprint and the real formula is always possible. We also show that we can pinpoint exactly which gates are at fault in the diagnosis problem using a randomized algorithm under a 'balance' condition we will properly define later. On the other hand, we can also show that if the real formula is *not* balanced, then diagnosis cannot be done in polynomially many probes. In addition, we show that our result can be viewed as a sort of learning result in a PAC-like world. If our queries are instead drawn from a product distribution, we can learn the locations of significant errors with high probability. Our work on this problem can also be seen as a special case of the general theory of fault diagnosis defined by Reiter in 1987 [72]. As with phylogenies, this problem focuses on inferring the values of hidden variables from visible data.

In the optimization problem, we again are confronted with a read-once formula of some fixed topology. We are now in charge of building the formula. Each gate can be one of two qualities, which we call 'good' and 'bad'. We can consider good gates to be perfect for now, while bad gates fail in the von Neumann model with some fixed probability ϵ. In addition to the blueprint for this read-once formula, we are given a product distribution over its

[2]We found that deterministic or probabilistic made little difference in the result that we present. However, there may be options available to an algorithm with a probabilistically failing real formula that are not available to us.

inputs. Our goal is to allocate good and bad gates across the formula so as to maximize the probability that the output of the constructed formula is correct relative to its inputs. We exhibit a dynamic program which is able to solve this problem approximately.

The spirit of this problem is that in an energy-constrained world, not all gates are of equal value. The question then is which gates are most important to the output of the formula, which we attempt to answer by seeing which gates are assigned to be good. Additionally, unlike the previous two problems, this problem considers the question of designing a stochastic process under constraints, rather than learning hidden variables.

1.3 Outline

In Chapter 2, we will first introduce the relevant mathematical background.

In Chapter 3, we will discuss our work on dependency in evolutionary trees. The result essentially states conditions under which a census performed at the leaves is sufficient for determining whether two characters are dependent. This requires two pieces: we first prove a concentration bound by bounding the variance of these censuses. We follow up by proving that the gap between the expected values of these censuses is linear in the number of leaves, where the coefficient will depend on the 'size' of the deviation at each step due to dependence. This latter result spans a varying set of conditions on the base evolutionary model as well as the precise error model used. This work will appear in the Journal of Computational Biology[10].

Then in Chapter 4, we provide a method of diagnosing read-once formulas where the gates may be individually faulty. We will show an abstract result where a black box, which provides an input that causes the formula to output 0 and an input that causes the formula to output 1, can be used to solve this problem. We then use a notion of the 'balance' of a formula and show that if a formula is balanced, then random sampling can produce the desired black box, yielding a randomized algorithm for this problem under a balance condition. In addition, we show that there are formulas that lack this balance condition which cannot be learned in any polynomial number of queries. Finally, we re-frame the problem as a learning problem, and briefly adapt our results to this new framework. This manuscript is in preparation [48].

Then in Chapter 5, we will consider the optimization angle of faulty read-once formulas with some in-progress work. We provide a dynamic program which can find a configuration which is approximately optimal (within $1 + \epsilon$ factor for any $\epsilon > 0$). These ideas have been discussed in private correspondence[47].

In each of these chapters, we will also state some directions that future research related to each of these problems might take.

Chapter 2

Preliminaries

In this dissertation, we will assume familiarity with basic concepts of probability theory, such as random variables, state spaces, and the expectations and variances of random variables. We now state, without proof, three standard inequalities in probability theory[1].

Theorem 1 (Union bound). *Let $A_1, \ldots, A_k$ be a set of events, then*

$$\Pr[A_1 \cup \cdots \cup A_2] \leq \sum_{i=1}^{k} \Pr[A_i]$$

Theorem 2 (Chebyshev's Inequality). *Let X be a random variable with expected value $\mathbb{E}[X]$ and variance $Var[X]$. Then for any real number $k > 0$,*

$$\Pr[(X - \mathbb{E}[X])^2 \geq k^2 \, Var[X]] \leq \frac{1}{k^2}$$

Theorem 3 (Chernoff bound). *Suppose $X_1, \ldots, X_n$ are independent random variables. Let $X = \sum_i X_i$, and let $\mu = \mathbb{E}[X]$. Then*

- $\Pr[X \geq (1+\delta)\mu] \leq e^{-\delta^2\mu/3}$, *for* $0 < \delta < q$
- $\Pr[X \geq (1+\delta)\mu] \leq e^{-\delta\mu/3}$, *for* $1 < \delta$
- $\Pr[X \leq (1-\delta)\mu] \leq e^{-\delta^2\mu/2}$, *for* $0 < \delta < 1$

The main use of these inequalities in our work will be to show that repeated experiments allow us to estimate the expectation of random variables sufficiently closely to the true expectation.

[1]Proofs can be found in most textbooks, such as [63]

2.1 Stochastic Processes

We will generally represent a distribution over states as a row vector $\vec{q}^\top$ whose entries are non-negative and sum to 1. We emphasize that we denote row vectors with the transpose $\top$, as standard vectors are column vectors.

We discuss briefly some concepts in stochastic processes, but a more thorough treatment can be found in [54].

Discrete-time Markov chains consist of an initial state drawn from an initial distribution $\vec{q}_0^\top$, and subsequent states drawn from subsequent distributions $\vec{q}_1^\top, \vec{q}^2\top, \ldots$. The distribution at time $i+1$, given by $\vec{q}_{i+1}^\top$ is determined from the distribution at time i by a transition matrix P_i: $\vec{q}_{i+1}^\top = \vec{q}_i^\top P_i$. A *homogeneous* Markov chain is one where the transition matrices P_i are all equal – that is $P_i = P = (p_{ij})$ for all i. Note that in this case, $\vec{q}_{i+k}^\top = \vec{q}_i^\top P^k = \vec{q}_0^\top P^{i+k}$.

We are interested first in the long-term behavior of homogeneous Markov chains. A *stationary distribution* of a homogeneous Markov chain is a distribution $\pi^\top$ such that $\pi^\top P = \pi^\top$. We are interested in the conditions under which a Markov chain has a unique stationary distribution, and the rate of convergence of a Markov process to its stationary distribution.

A subset S of the state space is *closed* if there are no transitions from the state space out of the state space: for all $i \in S$, $\sum_{j \in S} p_{ij} = 1$. If there is no proper subset of the state space of a Markov chain, then the Markov chain is called *irreducible.*

The *period* of a state i of a Markov chain is defined as the greater common divisor of $\{n > 0 \,|\, (P^n)_{ii} > 0\}$, provided the set is non-empty. A state is *aperiodic* if its period is 1, and a Markov chain is *aperiodic* if every state is aperiodic.

A state i is called *positive recurrent* if the expected number of visits to the state is infinite: $\sum_{n=1}^{\infty} (P^n)_{ii} = \infty$ ($(P^n)_{ii}$ is the probability of transition from i to i in exactly n steps). A Markov chain is *positive recurrent* if every state is positive recurrent.

Theorem 4 (Fundamental Theorem of Markov Chains)**.** *For any irreducible, aperiodic, positive recurrent Markov chain, there exists a unique stationary distribution $\pi^\top$ satisfying $\pi^\top P = \pi^\top$.*

Note that a Markov chain on finitely many states which is irreducible and aperiodic is necessarily positive recurrent. We will not deal with Markov chains on infinitely many states, and the finite-state Markov chains we do consider will be irreducible and aperiodic. As a result, they will have unique stationary distributions.

The next question is how quickly a Markov chain converges to its stationary distribution. In the literature, this is measured by the *mixing time*: the time t after which for any initial distribution $\vec{q}_0^\top$, we have that $|\vec{q}_t^\top 0 \pi^\top| \leq \frac{1}{4}$.

Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of the transition matrix P arranged such that $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$. Note that the existence of a unique stationary distribution from Theorem 4 implies both that $\lambda_1 = 1$ and that $|\lambda_2| < 1$. The rate of convergence is then controlled by λ_2 in the following sense. Let $\vec{v}_1^\top, \ldots, \vec{v}_n^\top$ be the eigenvectors corresponding the the eigenvalues $\lambda_1, \ldots, \lambda_n$. If we write $\vec{q}_0^\top = \sum a_i \vec{v}_i^\top$, then we have

$$\vec{q}_t^\top = \sum a_i \lambda_i^t \vec{v}_i^\top$$

As we are interested in the distance of $\vec{q}_t^\top$ from the stationary distribution $\pi^\top$, we consider

$$\|\vec{q}_t^\top - \pi^\top\|_2 = \|(\sum_{i=1}^{n} a_i \lambda_i^t \vec{v}_i^\top) - \pi^\top\|_2 = \|\sum_{i=2}^{n} a_i \lambda_i^t \vec{v}_i^\top\|_2 \leq \lambda_2^t \|\sum_{i=2}^{n} a_i \vec{v}_i^\top\|_2 = \lambda_2^t (\sum_{i=2}^{n} \|a_i \vec{v}_i^\top\|_2)$$

So we see that the smaller λ_2 is, the more quickly the Markov chain converges to its stationary distribution in ℓ_2-distance[2]. In our work in phylogenies, we will also use the second eigenvalue λ_2 in a similar capacity, ensuring that deviations from the stationary distribution are diminished quickly in successive steps of the process.

2.2 Matrix Exponential

The exponential of a square matrix X can be defined by

$$\exp(X) = \sum_{i=0}^{\infty} \frac{X^i}{i!} \tag{2.1}$$

[2]The definition of mixing time is usually given in ℓ_1-distance, but the two distances are related by a factor of $\sqrt{n}$. Again, we refer the reader to [54] for a more in-depth and careful discussion.

It may also be defined, as in the scalar exponential, by the limit

$$\exp(X) = \lim_{n\to\infty} \left(\mathbf{I} + \frac{X}{n}\right)^n \tag{2.2}$$

The proof of equality of these definitions follows as in the scalar case, but for a thorough treatment of the matrix exponential, we refer the reader to [6]. Instead, we will only state and prove a few facts about the matrix exponential pertaining to stochastic matrices. We start with a definition.

Definition 2.1. *A* stochastic rate matrix *is a square matrix Q which satisfies the following conditions:*

1. *The rows of Q sum to 0.*
2. *the off-diagonal entries of Q are non-negative.*

Note that a positive scalar multiple of a rate matrix is still a rate matrix. In addition, we have the following:

Fact 2.2. *Let Q be a stochastic rate matrix, then* $\exp(Q)$ *is a stochastic matrix.*

Proof. We will make use of the second definition (equation 2.2). Note that for sufficiently large n, $(\mathbf{I}+Q/n)$ is a stochastic matrix: pick n larger than each diagonal entry of Q. Then the diagonal entry of $(\mathbf{I}+Q/n)$ is positive and each off-diagonal entry is the same as the entry of Q/n which is non-negative. Since the rows of Q sum to 0, this still holds for Q/n, and so the rows of $(\mathbf{I}+Q/n)$ sum to 1. Then $(\mathbf{I}+Q/n)^n$ is also stochastic, and so the limit $\exp(Q)$ is stochastic. □

As a result, if $t \geq 0$, $\exp(tQ)$ is also a stochastic matrix[3] We state a few more facts.

Fact 2.3. *Let Q be a stochastic rate matrix, and $s, t \geq 0$.*

1. $\exp((s+t)Q) = \exp(sQ)\exp(tQ)$

[3]The stochastic rate matrix is important in the study of continuous-time Markov chains, which feature prominently in the study of phylogenies. However, our treatment of phylogenies will skip the continuous-time angle and have a retrospective look, in which the process will be separated into discrete time steps.

2. *If λ and $\vec{v}$ are an eigenvalue and eigenvector pair of Q, then $e^{t\lambda}$ and $\vec{v}$ are an eigenvalue and eigenvector pair of* $\exp(tQ)$.

Proof. 1. This follows as in the scalar case, using the first definition (equation 2.1)

$$\exp((s+t)Q) = \sum_{i=0}^{\infty} \frac{(s+t)^i Q^i}{i!} = \sum_{i=0}^{\infty} \sum_{j=0}^{i} \frac{s^i Q^i}{i!} \frac{t^{i-j} Q^{i-j}}{(i-j)!} = \sum_{i=0}^{\infty} \frac{s^i Q^i}{i!} \sum_{k=0}^{\infty} \frac{t^k Q^k}{k!} = \exp(sQ)\exp(tQ)$$

2. We again use the first definition.

$$\exp(tQ)\vec{v} = \left(\sum_{i=0}^{\infty} \frac{t^i Q^i}{i!} \right) \vec{v} = \sum_{i=0}^{\infty} \frac{t^i Q^i \vec{v}}{i!} = \sum_{i=0}^{\infty} \frac{t^i \lambda^i \vec{v}}{i!} = e^{t\lambda}\vec{v}$$

□

Chapter 3

Detecting Dependencies in Evolutionary Models

In this chapter we consider the question: *given two characters, can we decide if they are independent*? Our answer is *yes*, under biologically reasonable assumptions about the tree Markov random field (that is the model of evolution) and the type of dependence allowed. In this work, we show a proof of concept that dependence can be detected using just census data, though we make no claims to the statistical efficiency of this approach, compared to the likelihood tests mentioned above. We note that there are problems such as the reconstruction problem on trees in the Ising model, where a census is a sufficient statistic ([61]).

In addition, we introduce some simple models of dependence among characters that seem well-suited to the biological application. If two characters are independent, then on each edge of the tree the matrix governing their joint evolution is just the tensor product of the marginal matrices. Two characters are dependent if this does not hold, i.e., that there are edges where the transition matrix for the joint character differs (significantly) from the tensor product of the marginals. Thus dependence of two characters is captured by their joint transition matrix M deviating significantly from the tensor product T of their marginal matrices. The difference between the i-th rows of M and T is the difference in next-state probabilities when the process is in the joint state, i. If there is a biological reason that certain next-states are preferred, this reason should persist across all the rows of the deviation matrix, $M - T$. Thus we expect all rows of $M - T$ to point roughly in the same direction. Similarly, this biological reason should persist throughout the phylogeny, and hence we expect deviations on each edge of the tree to also point in roughly the same direction. This is just as well because, mathematically it would be impossible to detect dependencies when the deviations on different edges could potentially cancel out the dependence signal. Our bounds on how well we can detect dependence are a function of how well-aligned the deviation vectors are to some specific direction.

Another remark is in order about our restriction to detecting dependence of *pairs* of characters. In biologically realistic scenarios even dependencies across sets of characters of cardinality greater than 2 will manifest as dependencies of pairs of characters from this set. In other words, it is hard to conceive of biological scenarios where a set of k characters (for $k > 2$) are dependent, while every subset of $k - 1$ characters from this set are independent.

To detect dependency among characters, we have to overcome two challenges - first that the topology of the tree imposes a confounding dependence between the states of nearby leaves and second that the mutation process could cause the 'dependence' signal from higher up in the tree to cancel with dependence signals from lower down. A technical contribution of this work is a concentration bound for tree Markov random fields. Let Z be the random variable counting the number of occurrences of a character in a particular state at the leaves of a rooted tree. We show that as long as a certain natural norm of the transition matrices is bounded away from 1 (by an arbitrarily small constant), the variance of Z is sub-quadratic in the number of leaves, and the expectation of Z is linear in the number of leaves.

3.1 Preliminaries

Stochastic Model of Evolution. Let T rooted at r denote the underlying tree in a tree Markov random field. With little loss of generality we assume that the root has degree 2 and every other internal node has degree 3. A *character* maps the nodes of the tree to a set $\mathbf{C}$ of s *states* (for example, $\{0, 1\}$, $\{A, C, G, T\}$, $\{20$ amino acids$\}$, etc. A single character evolves ‘down’ the tree as follows. At the root r it has some distribution over its states, which need not be uniform. However, it is sufficient to consider the initial distribution to be uniform due to the mixing properties of the stochastic process. With every edge $e = (u, v)$ of T, is associated a stochastic $s \times s$ transition matrix M_e that governs the evolution of the character. More precisely, $\Pr[X_v = b | X_u = a] = M_e(a, b)$.

Specific biological models assume that these matrices are drawn from special types of stochastic matrices. For instance, the Cavender-Farris-Neyman (CFN) ([9], [23], [64]) model for binary states ($s = 2$) assumes that on each edge all characters have the same symmetric transition matrices. Thus a single scalar (the probability of mutation) determines the transition matrix on any edge; this scalar is usually a measure of the time duration represented by the edge. The Jukes-Cantor ([45]) model is a simple generalization to 4-state characters and the Kimura ([49]) model is determined by 2 parameters rather than 1. In our work, we consider models of evolution at 3 levels of generality, listed below. All the above biological models lie in the most restrictive level. One reason we consider the more general models is because they lead to mathematically interesting problems whose solutions might be applicable in other contexts beyond phylogenies. Note that in this work, we distinguish between an independent case and a dependent case; the required properties listed below apply to the transitions in the independent case. The transitions in the dependent case differ from transitions which satisfy the listed properties by an ‘error matrix’ which we specify below.

Shared Eigenbasis. Our most restrictive model assumes that all transition matrices are positive semi-definite (PSD) and have the same eigenbasis on every edge; this is true for all biological models studied so far.

PSD. At a greater level of generality, we do not require the PSD matrices for a character to have the same eigenbases on all edges.

Doubly stochastic. In this model we only assume all transition matrices are doubly stochastic. Thus they could be asymmetric, unlike in the previous two models.

Rate Matrix. In this biologically-motivated model, we have rate matrices Q_1, Q_2 corresponding to the first and second characters. Each edge has a length t_e, and the transition matrices on e are $\exp(tQ_1)$ and $\exp(tq_2)$. Note that the matrices in this model share eigenbases, and the eigenvalues are all positive, making this seem more restrictive. However, this model allows for nonuniform stationary distributions over states.

The parameter that governs our results is the following $1 \to 1$ norm of transition matrices: $||M|| := \sup_{0 \neq x \perp \mathbf{1}} ||x^\top M||_1/||x||_1$. *We assume* $||M|| \leq \lambda < 1$ *for some constant* λ. It is easy[1] to see that $||M||$ is always at least the second eigenvalue (in absolute value) of M; our above assumption implies $\lambda_2(M) \leq \lambda$ as well. In order to detect dependence we will need increasingly tighter upper bounds on $\lambda_2(M)$ as we move to more general models of evolution.

The Dependence Model. Let X and Y be two characters and let X_u and Y_u denote the states of these characters at node u of T. If X and Y evolve independently, then the transition matrix governing the evolution of the joint variable (X, Y) across edge e of the tree is given by the matrix $M_e \otimes N_e$ where M_e and N_e are the $s \times s$ transition matrices associated with the individual characters. Note that we allow different characters to have different transition matrices. If X and Y are not independent, then we assume the following dependence model. Firstly, we assume that the joint random variable (X, Y) evolves via a Markovian process. This is standard in biology where mutation is assumed to be history independent. So, for every edge e there exists an $s^2 \times s^2$ transition matrix P_e such that $\Pr[(X_v, Y_v) = (a', b') | (X_u, Y_u) = (a, b)] = P_e((a, b), (a', b'))$. Furthermore, we assume a *consistent preferred direction* dependence model where the joint evolution of the two characters tends to bias probabilities in a preferred direction in comparison to the situation when they evolve independently. We model this by making assumptions on the 'deviation' matrix $D_e := P_e - M_e \otimes N_e$. In the simplest, but already non-trivial case that we call the **u**niform rank-1 dependence model, we assume that the deviation matrix $D_e = D$ for all edges, and furthermore, D is the outer product $\mathbf{1}d^\top$ for some s^2-dimensional vector d with $||d||_1 \geq \delta > 0$ for some known parameter δ. This stringently models situations where there are preferred states and every transition biases the distribution by the same vector in favor

[1]v be an eigenvector corresponding to eigenvalue $\lambda < 1$. M is stochastic, so $v \perp \mathbf{1}$ and $Mv = \lambda v$ implies $||M|| \geq |\lambda|$.

of the preferred states regardless of the starting state or edge. We also investigate a generalization of the uniform rank-1 dependence that we call the **d**irectional-drift dependence model. Here we assume there exist some direction d^* such that every row of every deviation matrix D_e has an inner product of at least δ with d^*. In addition, the norm of any row of any of these matrices is at most a constant. For ease of presentation we use the uniform rank-1 model almost throughout this chapter, only discussing the more general model in the last section.[2]

Informal Statement of Results for uniform rank-1 dependence:

1. In the shared eigenbasis model, we can detect dependence with no further assumptions. As stated above, this includes all the major models of evolution studied so far.

2. In the PSD model, if all single-character transition matrices have $\lambda_2 < 0.797$, then we can detect dependence. However, there exists examples of trees with PSD transition matrices with $\lambda_2 \geq 0.832$ and yet the distribution on the leaves is indistinguishable from the case of independent evolution.

3. In the doubly-stochastic model, we can detect dependence if all transition matrices have $\lambda_2 \leq \frac{1}{2}$. We cannot prove 'better ' negative results than for the PSD case.

4. In the rate matrix model, as in the shared eigenbasis model, we require no further assumptions.

Informal Statement of Result for directional-drift dependence in the PSD model:
If each row of D_e has length at most δ/β, then we can allow $\lambda_2 \leq \frac{\beta}{\frac{1}{2}+\beta}$.

3.2 The Tester, Analysis Roadmap, and Technical Challenges

The input to our dependency testers are the values of the characters at the leaves of the phylogeny. For two characters X and Y, denote their leaf data as $\{X_r\}$ and $\{Y_r\}$, respec-

[2]Throughout the chapter, we are concerned with detecting dependence between a pair of characters. However, it is straightforward to generalize our models and results to a constant subset of characters where instead of pairs $\mathbf{C} \times \mathbf{C}$, we would be dealing with random variables over a larger domain. For simplicity, we will stick to pairs.

tively. In addition, we take an ε as an accuracy parameter. Our tester is extremely simple: For each ordered pair of states, we count the number of leaves that have that pair. If there is a 'large discrepancy' in this number, the characters are dependent.

Algorithm 1 Dependence Detection

```
1: procedure DependenceDetection($\{X_r\}, \{Y_r\}, \varepsilon$)
2:     for $(i, j) \in \mathbf{C} \times \mathbf{C}$ do
3:         $Z_{i,j} \leftarrow |\{l \mid (X_l, Y_l) = (i, j)\}|$
4:     end for
5:     for $(i, j) \in \mathbf{C} \times \mathbf{C}$ do
6:         if $|Z_{i,j} - 1/|\mathbf{C}|^2| \geq \varepsilon n$ then
7:             output dependent
8:         end if
9:     end for
10:    output independent
11: end procedure
```

We now briefly outline the analysis and the challenges involved. The correctness of this algorithm relies on both a concentration bound on the number of leaves in each state pair, as well as a significant and guaranteed discrepancy of the distribution of each leaf from uniform.

We prove a concentration bound for the overall distribution of state pairs at the leaves. For all ordered pairs i, we need that the number of leaves Z_i that have the state i, is concentrated around its mean. This is not trivial since Z_i is a sum of indicator random variables that are not independent, because in a tree Markov random field, even the state of one character at 'near by' leaves are highly correlated. We obtain concentration by upper-bounding the second moment (Theorem 5) which is done via a coarse but sufficiently good upper bound on the variance (Lemma 3.2). We also show that when the characters are independent, we *expect* each joint state to be almost equally likely (Lemma 3.5). Since the norms of the transition matrices are bounded away from 1, we expect rapid mixing and the leaf state to be close to stationary distribution, which by the assumption of double-stochasticity is uniform.

In the case of dependent characters, we show that a large discrepancy indeed occurs (in expectation) for at least one pair of states (Lemma 3.6) . This is nontrivial since different edges have different transition matrices, and the effect of one matrix's deviation may cancel the effect of its predecessors. Indeed, in the PSD model, we show that this can happen even when $\lambda_2 \geq 0.832$ (Lemma 3.10). However, if all matrices share the same eigenbasis, then such a 'bad case' cannot occur (Lemma 3.7), and so the shared eigenbasis model doesn't need any further assumptions. In the doubly stochastic model, an upper bound of 0.5 suffices (Lemma 3.11) to detect dependency. For the PSD model, an upper bound $\lambda_2 \leq 0.797$ suffices, and this is more subtle to show. To do so, we prove a lower bound on a quantity $v^\top Av$ where A is a product of k PSD matrices (and therefore, not necessarily PSD) and v is a vector perpendicular to the all ones vector. We show (Lemma 3.9) that this quantity is at least $-(\lambda \cos(\pi/k+1))^k ||v||_2^2$; this result may be of independent interest. We leave open the question of finding the exact value in $[0.797, 0.832]$ at which dependence can be detected in the PSD model. Finally, in Theorem 7, we show that in the directional-dependence model, we can detect dependence when λ^* is bounded by a function of δ, β.

3.3 Bounding the Variance

Fix an $i \in \mathcal{P} = \mathbb{C} \times \mathbb{C}$. Let Z be the random variable counting the number of leaves of T in state i. For a random variable X, let $\mathbb{E}[X]$ denote its expectation and $\mathrm{Var}(X)$ its variance. Recall $\lambda < 1$ is an upper bound on the norm of any of the transition matrices P_e on the edges e. In this section, we prove the following theorem.

Theorem 5. *Given any n leaf trivalent tree T, $Var(Z) = O(n^{2-2\log_2(1/\lambda)})$.*

For any vertex v in T, let Z_v to be the number of leaves in the sub-tree of T rooted at v in state in i; so $Z = Z_{root}$. Let L_v denote the leaves in the subtree rooted at v. For a leaf $\ell \in L_v$, let $\mathsf{dist}(v, \ell)$ denote the number of edges on the path from v to ℓ in the tree. Define

$$\Lambda(v) \triangleq 2 \sum_{\ell \in L_v} \lambda^{\mathsf{dist}(v,\ell)} \tag{3.1}$$

The following claim bounds $\Lambda(v)$ at any vertex.

Proposition 3.1. *For any vertex u with n leaves in its subtree, $\Lambda(u) \leq O(n^{1-\eta})$ where $\eta = \log_2(1/\lambda)$.*

Proof. Note that $\Lambda(u) = 2\sum_{i\geq 1}|L_i|\lambda^i$ where L_i is the set of leaves at a distance i from u. Since $\lambda < 1$, an n leaf tree which maximizes $\Lambda(u)$ will make the tree as balanced in height as possible. (This can be proved by a "swapping" argument similar to the proof of optimality of Huffman trees.) In particular, the maximizing tree has all leaves at distance $\lfloor \log n \rfloor$ or $\lfloor \log n \rfloor + 1$. Therefore, $\Lambda(v) \leq \frac{2}{\lambda} \cdot n\lambda^{\log n} = \frac{2}{\lambda} \cdot n^{1-\log(1/\lambda)}$. □

The next lemma bounds the variance in terms of the Λ's. We will use the lemma to prove the theorem.

Lemma 3.2. *$Var(Z) \leq \frac{1}{2}\sum_{v\in V(T)\setminus root}(\Lambda(v))^2$.*

Proof. For a vertex v and two states $j, k \in \mathcal{P}$, define

$$\Delta_v(j,k) \triangleq |\mathbb{E}[Z_v|X_v = j] - \mathbb{E}[Z_v|X_v = k]| \tag{3.2}$$

The following claim relates Δ_v with $\Lambda(v)$.

Claim 3.3. *For any vertex v, and for any two states $j, k \in \mathcal{P}$, we have $\Delta_v(j,k) \leq \Lambda(v)$.*

Proof. Fix a vertex v and a leaf $\ell \in L_v$. Let $e_1, e_2, \ldots, e_{\mathsf{dist}(v,\ell)}$ be the edges on the path from v to ℓ. Let P denote the matrix $P_{e_1} \cdot P_{e_2} \cdots P_{e_{\mathsf{dist}(v,\ell)}}$; this is the transition matrix from v to leaf ℓ. In particular, P is row-stochastic (row entries add up to 1). We use the following simple fact about row stochastic matrices; another proof of this can be found in Lemma 4.12 in [55].

Claim 3.4. *For any two row stochastic matrices P_1 and P_2, we have $||P_1P_2|| \leq ||P_1||\cdot||P_2||$.*

Proof. Let x be the vector with $||x||_1 = 1$ and $x\top\mathbf{1} = 0$ such that $||P_1P_2|| = ||x^\top P_1P_2||_1$. Note that $y^\top = x^\top P_1$ also satisfies $y^\top\mathbf{1} = 0$ since P_1 is row stochastic. Therefore, $||y^\top P_2||_1 \leq ||y||_1 \cdot ||P_2||$. Also by definition, $||y||_1 = ||x^\top P_1||_1 \leq ||P_1||$. □

This claim implies that $||P||$ is at most $\lambda^{\mathsf{dist}(v,\ell)}$. In particular, this shows that for any vertex v, for any leaf $\ell \in L_v$ at a distance $\mathsf{dist}(v,\ell)$, and for any states $j,k \in \mathbb{C}$, we have $|\Pr[X_\ell = i|X_v = j] - \Pr[X_\ell = i|X_v = k]| \leq ||x^T P||_1 \leq 2\lambda^{\mathsf{dist}(v,\ell)}$, where x is the vector with

$x_j = 1$, $x_k = -1$, and $x_s = 0$ otherwise. The claim follows by noting

$$
\begin{aligned}
\Delta_v(j,k) &= \Big|\sum_{\ell \in L_v} \left(\Pr[X_\ell = i | X_v = j] - \Pr[X_\ell = i | X_v = k]\right)\Big| \\
&\leq \sum_{\ell \in L_v} |\Pr[X_\ell = i | X_v = j] - \Pr[X_\ell = i | X_v = k]| \leq 2 \sum_{\ell \in L_v} \lambda^{\mathsf{dist}(v,\ell)}
\end{aligned}
$$

□

Now we can finish the proof of Lemma 3.2. Fix any vertex u. Recall that T_u denotes the subtree of T rooted at u and Z_u is the number of leaves in L_u in state i. We now show using induction on the height of T that for any state $j \in \mathcal{P}$, $\mathrm{Var}(Z_u | X_u = j) \leq \frac{1}{2}\sum_{v \in V(T_u)\setminus u} \Lambda^2(v)$. This proves the lemma with u as the root, and summing over all the conditional events.

Note that the claim is vacuously true when u is a leaf since both LHS and RHS are 0. Let u have children $v_1, \ldots, v_q$ (if the tree is binary, $q = 2$, but this lemma holds for any tree). Assume we have proved the inductive claim for the v_i's. Note that conditioned on X_u, the random variables $Z_{v_1}, Z_{v_2}, \ldots$ are independent, since they count over leaves on disjoint subtrees. Therefore, for any $j \in \mathcal{P}$, $\mathrm{Var}(Z_u | X_u = j) = \sum_{i=1}^{q} \mathrm{Var}(Z_{v_i} | X_u = j)$.

We now show that for any parent-child pair $e = (u, v_i)$ and any state $j \in \mathcal{P}$, we have

$$\mathrm{Var}(Z_{v_i} | X_u = j) = \sum_{k \in \mathcal{P}} P_{jk} \mathrm{Var}(Z_{v_i} | X_v = k) + \frac{1}{2} \sum_{k \neq k' \in \mathcal{P}} P_{jk} P_{jk'} \Delta^2_{v_i}(k, k') \qquad (3.3)$$

where $P_{jk} = \Pr[X_{v_i} = k | X_u = j] = P_e(j,k)$. (3.3) suffices to complete the proof. By induction, the first summand in the RHS is at most $\frac{1}{2}\sum_{w \in T_{v_i}\setminus v_i} \Lambda^2(w)$. From Claim 3.3, we have $\Delta_{v_i}(k,k') \leq \Lambda(v_i)$ and $\sum_{k \neq k'} P_{jk} P_{jk'} \leq (\sum_{k \in \mathcal{P}} P_{jk})^2 = 1$, thereby giving that the second summand in the RHS of (3.3) is at most $\frac{1}{2}\Lambda^2(v_i)$. Together, we get $\mathrm{Var}(Z_{v_i} | X_u = j) \leq \frac{1}{2}\sum_{w \in T_{v_i}} \Lambda^2(w)$, and by adding over all v_i, $1 \leq i \leq q$, we complete the proof of Lemma 3.2. □

Proof of theorem 5. Let $\mathcal{V}(n)$ be a function that denotes the maximum value of $\sum_{v \in V(T)\setminus r} \Lambda^2(v)$ over all n-leaf binary trees. By Lemma 3.2, we want a subquadratic upper bound on $\mathcal{V}(n)$. Let u be the *centroid* of T. That is, $n/3 \leq |L_u| \leq 2n/3$. It is easy to see this is well defined. Let T_u denote the subtree of T rooted at u, and let T'_u denote the subtree of T with all descendants of u deleted. Note that both T_u and T'_u are binary trees, and have

ρn and $(1-\rho)n$ leaves for $\rho \in [1/3, 2/3]$. By definition, $\sum_{v \in V(T_u) \setminus u} \Lambda^2(v) \leq \mathcal{V}(\rho n)$ and $\sum_{v \in V(T'_u) \setminus r} \Lambda^2(v) \leq \mathcal{V}((1-\rho)n)$.

Suppose $u = u_0, u_1, \ldots, u_r = r$ is the unique path from u to r in T. Note that the $\Lambda(v)$'s in tree T'_u are the same as in tree T for all vertices except the u_i's. For each u_i, $\Lambda(u_i)$ in the tree T is that in T'_u plus $\lambda^i \cdot \Lambda(u)$ Thus, we have

$$\begin{aligned}\sum_{v \in V(T) \setminus r} \Lambda^2(v) \leq \quad & \mathcal{V}(\rho n) + \mathcal{V}((1-\rho)n) \ + \ \sum_{i=0}^{r} \Big((\Lambda(u_i) + 2\lambda^i \Lambda(u))^2 - \Lambda^2(u_i) \Big) \\ = \quad & \mathcal{V}(\rho n) + \mathcal{V}((1-\rho)n) \ + \ 4\Lambda(u) \sum_{i=0}^{r} \lambda^i \Lambda(u_i) \ + \ 4\Lambda^2(u) \sum_{i=0}^{r} \lambda^{2i}\end{aligned}$$

From Proposition 3.1, we can bound $\Lambda(u_i)$ by $O(n^{1-\eta})$ for $i = 0, ..., r$. So we get the following recurrence for $\mathcal{V}(n)$:

$$\mathcal{V}(n) \leq \mathcal{V}(\rho n) + \mathcal{V}((1-\rho)n) + O(n^{2-2\eta})$$

which evaluates to $\mathcal{V}(n) = O(n^{2-2\eta})$. $\square$

3.4 Analysis of the Tester in Uniform Rank-1 Model

Let μ_0 be the state at the root. Let μ be the uniform distribution over the s^2 states. Let $r_0 = \mu_0 - \mu$ be the *error vector* at the root. Recall P_e is the transition matrix of the joint random variable (X, Y) at edge e. We write $P_e = R_e + D$ where $R_e = M_e \otimes N_e$ and D is the zero-matrix if the characters are independent, and $D = \mathbf{1}d^\top$ in case the characters are dependent.

Our goal in this section is to establish the following theorem[3], asserting the correctness of the tester for the uniform rank-1-model.

Theorem 6. *Under each of the following evolutionary models, under the listed assumptions on the norm on the transition matrices R_e, Algorithm **D**ependence Detection is correct with $1 - 1/\text{poly}(n)$ probability.*

Shared Eigenbasis. *No extra assumption on R_e is needed.*

[3]Note that $\lambda_2(R_e) \leq \max(\lambda_2(M_e), \lambda_2(N_e))$

PSD. *If* $\lambda_2(R_e) \leq 0.797$.

Doubly stochastic. *If* $\lambda_2(R_e) \leq 0.5$.

For a leaf ℓ, let μ_ℓ be the distribution at the leaf, and $r_\ell = \mu_\ell - \mu$ be the error vector at the leaf. Let $(e_1, e_2, \ldots, e_{\mathsf{dist}(\ell)})$ be the path from the root to the leaf ℓ. Then, if the characters are independent, we get

$$r_\ell^\top = r_0^\top \left(\prod_{k=1}^{\mathsf{dist}(\ell)} R_{e_k} \right) \tag{3.4}$$

and if the characters are dependent, we get

$$r_\ell^\top = d^\top + \sum_{i=1}^{\mathsf{dist}(\ell)} d^\top \left(\prod_{k=i+1}^{\mathsf{dist}(\ell)-1} R_{e_k} \right) + r_0^\top \left(\prod_{k=1}^{\mathsf{dist}(\ell)} R_{e_k} \right) \tag{3.5}$$

We first prove a lemma to show that when the character pair evolves independently, the distribution of state pairs at the leaves is close to uniform.

Lemma 3.5. *If the characters are independent, then for all $i \in \mathcal{P}$, we have $|\mathbb{E}[Z_i] - n/s^2| \leq O(n^{1-\beta})$ for some constant β depending on λ, the upper bound on the norms of all the transition matrices.*

Proof. By our assumption, $||R_{e_i}|| \leq \lambda$ for all i. Substituting in (3.4), and using Fact 3.4, we get $||r_\ell||_1 \leq \lambda^{\mathsf{dist}(\ell)}||r_0||_1 \leq 2\lambda^{\mathsf{dist}(\ell)}$ since $||r_0||_1 \leq ||\mu||_1 + ||\mu_0||_1 = 2$. In turn, this implies $|\langle r_\ell, e_i\rangle| \leq 2\lambda^{\mathsf{dist}(\ell)}$. Note for any i, we have $\mathbb{E}[Z_i] = \sum_\ell \langle \mu_\ell, e_i \rangle = n/s^2 + \sum_\ell \langle r_\ell, e_i\rangle \leq n/s^2 + \Lambda(\text{root})$, and the lemma follows from Proposition 3.1. □

Next, we prove a contrasting lemma for the dependent case, depending on the model of evolution. In each case, we show that there is a deviation from the uniform in the distribution at the leaves. In particular, we exhibit $r^* \in \mathbb{R}^{s^2}$ whose coordinates sum up to zero with each entry in $[-1, +1]$ such that there is some $\varepsilon > 0$ satisfying

$$\text{For all } \ell, \text{ we have} \quad \langle r_\ell, r^* \rangle \geq \varepsilon. \tag{Deviation}$$

We prove the following lemma, under the assumption that this equation is satisfied. In later subsections we demonstrate r^* in every model of evolution.

Lemma 3.6. *If under a model of evolution we obtain an r^* and ε satisfying* (Deviation)*, then there exists $i, j \in \mathcal{P}$ such that $|\mathbb{E}[Z_i] - \mathbb{E}[Z_j]| \geq \varepsilon n$ where ε is a constant depending on δ and s.*

Proof. Let $\bar{\mu} := \frac{1}{n}\sum_\ell \mu_\ell$. Observe that $\langle \bar{\mu}, r^* \rangle \geq \varepsilon$ as well. Since r^* is a convex combination of vectors of the form $\{e_i - e_j\}$ where e_i is the indicator vector for pair i, we get there exists (i, j) such that $\langle \bar{\mu}, (e_i - e_j) \rangle \geq \varepsilon$. But $\langle \bar{\mu}, e_i \rangle$ is precisely $\mathbb{E}[Z_i]/n$ since $\langle \mu_\ell, e_i \rangle$ indicates the probability leaf ℓ is in state pair i. Therefore (Deviation) implies that there exists a pair i and j such that $\mathbb{E}[Z_i] - \mathbb{E}[Z_j] \geq \varepsilon n$. □

Proof of Theorem 6. These follow from Lemma 3.5 and Lemma 3.6 (with appropriate use of Equation (Deviation)) using Lemma 3.7, Lemma 3.8, and Lemma 3.11 given below, Theorem 5 and Chebyshev's inequality. □

Shared eigenbasis model

Recall in this model we assume if characters are independent, then the transition matrices M_e are PSD and share the same eigenbasis over all edges. Thus matrix $R_e = M_e \otimes N_e$ also is PSD and have the same eigenbasis across all edges.

Lemma 3.7. *In the shared-eigenbasis model, for each leaf ℓ, $\langle r_\ell, d \rangle \geq \|d\|^2(1 - \lambda^{\mathsf{dist}(\ell)})$. Thus in (Deviation), $r^* = d$ and $\varepsilon = \|d\|_2^2(1 - \lambda) \geq \delta^2(1 - \lambda)/s^2$ suffices.*

Proof. We can multiply both sides of (3.5) by d to get $r_\ell^\top d = d^\top d + \sum_{i=1}^{\mathsf{dist}(\ell)} d^\top A_i d + r_0^\top B d$, where $A_i = \prod_{k=i+1}^{\mathsf{dist}(\ell)-1} R_{e_k}$ while $B = \prod_{k=1}^{\mathsf{dist}(\ell)} R_{e_k}$. The main observation is that if the R_e's share an eigenbasis, then products of these matrices are also PSD. Thus, each A_i is PSD implying the second sum is ≥ 0. The final term $|r_0^\top B d| \leq \|r_0^\top\|_\infty \|Bd\|_1 \leq \lambda^{\mathsf{dist}(\ell)}$, by Cauchy-Schwarz, and the second inequality follows since $\|B\| \leq \lambda^{\mathsf{dist}(\ell)}$. Thus,

$$\langle r_\ell, d \rangle \geq \|d\|_2^2(1 - \lambda)$$

□

Positive semi-definite model

Recall that in this model each M_e is PSD, and thus R_e is PSD as well.

Lemma 3.8. *In the PSD model, if $\lambda_2(R_e) \leq \lambda^* < 0.797$, then for each leaf ℓ, $\langle r_\ell, d\rangle \geq \epsilon(\lambda^*) > 0$.*

Proof. We expand (3.5) to get (we ignore the last term since it vanishes with $\mathsf{dist}(\ell)$.)

$$\langle r_\ell, d\rangle = \|d\|_2^2 + \sum_{i=1}^{\mathsf{dist}(\ell)-1} d^\top \left(\prod_{k=i+1}^{\mathsf{dist}(\ell)} R_{e_k}\right) d \tag{3.6}$$

Note that each term in the sum is correlated, since they use the same matrices R_{e_k}. To lower bound this product, we will relax this restriction, and allow that each term choose its own matrices. In particular, we will use the following lemma.

Lemma 3.9. *Suppose $A_1, \ldots, A_k$ are k positive semi-definite transition matrices, with second eigenvalue bounded by λ^*, and let v be a vector with entries summing to 0. Then*

$$v^\top (A_1 \cdots A_k) v \geq -(\lambda^*)^k \cos^{k+1}\left(\frac{\pi}{k+1}\right) \|v\|_2^2 \tag{3.7}$$

Proof. We note that to minimize the left-hand side, we are essentially looking to make $A_1 \cdots A_k v$ be a long vector pointing away from v. To do this, we can assume that each A_i is a scaled projection onto a fixed vector u_i. Suppose some A_i is not. Then let u_i be a unit vector in the direction of $A_i \cdots A_k v$, and replace A_i with a projection onto u_i and a scaling by λ^* without reducing the length of the resulting vector. Then we can let θ_i be the angle between $A_i \cdots A_k v$ and $A_{i+1} \cdots A_k v$, and θ_0 the angle between $A_1 \cdots A_k v$ and $-v$. Then we have $|v^\top A_1 \cdots A_k v| = (\lambda^*)^k \left(\prod_{i=0}^k \cos(\theta_i)\right) \|v\|_2^2$.

Finally, using the concavity and monotonicity of the cosine function in the domain $[0, \pi/2]$, and the fact that the total projections go from v to $-v$, so $\sum \theta_i \geq \pi$, we conclude that to minimize this, each θ_i should be equal and so each $\theta_i = \frac{\pi}{k+1}$. $\square$

We use this lemma to bound $r_\ell^\top d \geq \|v\|_2^2 \left(1 - \sum_{k=2}^\infty (\lambda^*)^k \cos^{k+1}\left(\frac{\pi}{k+1}\right)\right)$. Note that the parenthesized expression in the RHS can be lower bounded, for any integer $N \geq 2$, by $\left(1 - \sum_{k=2}^N \cos^{k+1}\left(\frac{\pi}{k+1}\right) - \frac{(\lambda^*)^{N+1}}{1-\lambda^*}\right)$. For instance if $N = 2$, we get that if $\lambda^* \leq 2/3$, then the expression is lower bounded by 1/18. Numerically, we obtained the best tradeoff at $N = 8$ where $\lambda^* < 0.797$ implies the expression is > 0. □

Note that the value 0.797 is not exact, even for this bound we have given, and better bounds may exist. However, we cannot allow λ^* to be arbitrarily close to 1 which is encapsulated in the following lemma.

Lemma 3.10. *In the PSD model it is not always possible to detect dependence at the leaves, even if* $\lambda_2(R_e) \leq 0.832$ *for all* e.

If the second eigenvalue is allowed to be close to 1, then there exists a sequence of transforms which causes the state-pair distribution at a leaf to be uniform, and thus indistinguishable from the independent case. In this section, we will focus on a 2-dimensional subspace of the error space containing $\vec{d}$. It is easy to check that as long as we choose a positive semi-definite transformation in this subspace of dimension 2, it is realizable in the full state-pair space of s^2 dimensions. We will let $d = (1, 0)$ in this 2-dimensional subspace.

Now consider k scaled projections $A_1, \ldots, A_k$ where

$$A_i = \lambda^* \begin{pmatrix} \cos^2\left(\frac{i\pi}{k}\right) & \cos\left(\frac{i\pi}{k}\right)\sin\left(\frac{i\pi}{k}\right) \\ \cos\left(\frac{i\pi}{k}\right)\sin\left(\frac{i\pi}{k}\right) & \sin^2\left(\frac{i\pi}{k}\right) \end{pmatrix}$$

is a scaled projection onto a vector making an angle $i\pi/k$ with $\vec{d}$, the x-axis.

To make the deviation $\vec{r} = 0$ after the last transform ($\vec{r}^\top \mapsto \vec{r}^\top M + \vec{d}^\top$), we will first apply a large number of transforms where one of the eigenvectors is in the direction of $\vec{d}$, with a corresponding eigenvalue of λ^*. This will allow us to get our deviation $\vec{r}$ to be arbitrarily close to $\frac{1}{1-\lambda^*}\vec{d}$. Then we will apply $A_1, \ldots, A_k$. We claim that if λ^* is sufficiently large, this will be 0. Note that A_k is a projection onto the x-axis, so we only have to examine the x coordinates.

Before A_1, we have $\vec{r} = \frac{1}{1-\lambda^*}\vec{d}$. After applying the transforms for $A_1, \ldots, A_k$, we have

$$r^\top A_1 \cdots A_k + d^\top (A_2 \cdots A_k + \ldots + A_k + \mathbf{I}_2)$$

where $\mathbf{I}_2$ is the 2-dimensional identity. We will examine the x coordinates of these terms. For the first term, we see that each A_i is a projection over an angle π/k and includes a scaling λ^*, thus the x coordinate of the first term is

$$\frac{-1}{1-\lambda^*}(\lambda^* \cos(\pi/k))^k$$

We will split the next part into two, as some of them will be negative and some will be positive. For $i = 1, \ldots, \lceil k/2 \rceil - 2$, these will contribute negatively the amount

$$-(\lambda^* \cos(\pi/k))^{n-i-1} \cos((i+1)\pi/k)$$

For $i = \lceil k/2 \rceil - 1, \ldots, k-1$, this will contribute positively the amount

$$(\lambda^* \cos(\pi/k))^{n-i-1} \cos((n-i-1)\pi/k)$$

Finally, $\vec{d}^\top \mathbf{I}_2$ contributes 1. In total, this gives

$$- \left(\frac{1}{1-\lambda^*}(\lambda^* \cos(\pi/k))^k + \sum_{i=1}^{\lceil r/2 \rceil - 2} (\lambda^* \cos(\pi/k))^{n-i-1} \cos((i+1)\pi/k) \right)$$
$$+ \left(1 + \sum_{i=\lceil r/2 \rceil - 1}^{k-1} (\lambda^* \cos(\pi/k))^{n-i-1} \cos((n-i-1)\pi/k) \right)$$

Finally, for a fixed k we can solve for this to be 0 to get an upper bound on allowable λ^*. For $k = 9$, this gives $\lambda^* < 0.832$.

Doubly-stochastic model

In this model we simply assume the transition matrices are doubly stochastic. We show that if $\lambda_2(R_e) < 1/2$, then we can detect dependence.

Lemma 3.11. *In the doubly-stochastic model, each leaf ℓ has that r_ℓ satisfies $\langle r_\ell, d \rangle \geq \left(1 - \frac{\lambda^*}{1-\lambda^*}\right) \|d\|_2^2$. Thus (Deviation) is satisfiable for constant $\varepsilon > 0$ if $\lambda^* < 1/2$.*

Proof. We will use a similar approach to Lemma 3.8. We will again use (3.6), and write $\langle r_\ell, d\rangle =$

$$\|d\|_2^2 + \sum_{i=1}^{\mathsf{dist}(\ell)-1} d^\top \left(\prod_{k=i+1}^{\mathsf{dist}(\ell)} R_{e_k}\right) d \geq \|d\|_2^2 \left(1 - \sum_{i=1}^{\infty} (\lambda^*)^i\right) = \|d\|_2^2 \left(1 - \frac{\lambda^*}{1-\lambda^*}\right)$$

where we lower bounded $d^\top R_{e_{i+1}} \cdots R_{e_{\mathsf{dist}(\ell)}} d$ by $-(\lambda^*)^{\mathsf{dist}(\ell)-i-1}\|d\|_2^2$, since all eigenvalues in the space of error vectors are bounded in absolute value by λ^*.

□

3.5 Directional-drift Dependence Model

Here, we generalize the error model, in the PSD evolution model, using the directional-drift dependence model which we now describe. We recall that PSD model generalizes the Shared Eigenbases model, which in turn generalizes all stochastic models studied in the literature. In this model, there is a fixed direction d^*, such that every row of each error matrix has the following properties: (1) $\|d^\top\| \leq \delta/\beta$, and (2) $\langle d, d^*\rangle \geq \delta$ for a significant δ and a constant β.

Theorem 7. *In the PSD evolutionary model with the directional-drift dependence model above, if all transition matrices have norm bounded by $\lambda^*(\beta) = \frac{\beta}{\frac{1}{2}+\beta}$, then **D**ependence detection is correct.*

This theorem will again follow from Lemmas 3.6 and 3.5, through the use of Equation (Deviation), with $r^* = d^*$ and $\varepsilon(\beta, \lambda)$.

Let us first examine now what happens in one step, when we start with a vector $\vec{\mu} + \vec{r}$, and apply the transform $P_e = R_e + D_e$,

$$(\mu + r)^\top \mapsto \mu^\top + r^\top R_e + (\mu + r)^\top D_e$$

When $D_e = \mathbf{1}d^T$, we see the last term is precisely $d^\top$. Now, however, we get some vector d_e which has $\|\vec{d}\|_2 \leq \delta/\beta$ and $\langle \vec{d_e}, \vec{d^*}\rangle \geq \delta$. We will use primarily the fact that this added vector has these properties. As before, we will view the transform in the error space, where the transform is

$$r^\top \mapsto r^\top R_e + d_e$$

Our approach to show that the detection of dependence is possible here will be by induction. In particular, we will show that for each node other than the root, there is some $x^* = x^*(\beta)$ such that if the distribution at the node v is $\mu + r_v$, then $\langle r_v, d^* \rangle \geq x^* \delta$. This is true for the direct children of the root, as the distribution is precisely $\mu + d_e$ where e is the edge connecting to the root. By hypothesis, $\langle d_e, d^* \rangle \geq \delta$. This gives us the base case for induction.

Before we prove the general case, we will first observe that $\|r\|_2 \leq \frac{1}{1-\lambda^*}\frac{\delta}{\beta}$. This is clear since every transform R_e reduces the length by a factor λ^*, and then we add a vector of length at most δ/β. The length bound then is just a geometric series.

To prove the general case of the induction, we first show that it suffices to examine the problem in 2 dimensions. So suppose that we have a deviation r which satisfies that $\langle r, d^* \rangle \geq x^*$ for some constant x^* to be determined later. We want to show that for any positive semi-definite R with all eigenvalues at most λ^* and any $\vec{d}$ satisfying the length and inner product requirements above, that $\langle R^\top r + d, d^* \rangle \geq x^*$.

It is clear that we only need to concern ourselves with the space of at most 3 dimensions spanned by $d^*, r, R^\top r$, since the added d will add some fixed amount in the direction of d^*. We are concerned with how negative $\langle R^\top r, d^* \rangle$ can be. We know that for any direction z, if $R^\top r$ is in the direction z, the largest length it can have is $\|r\|_2 \cos\theta$ where θ is the angle between z and r. Then in taking the inner product with d^*, we gain another factor ϕ where ϕ is the angle between z and r. So if $R^\top r$ is not in the same plane as d^* and r, then since cos is increasing in the range $[0, \pi/2)$, we can replace z with z' which is the projection of z into the plane of d^* and r then both θ and ϕ increase, and the resulting $\langle R^\top r, d^* \rangle$ is more negative. Since we are concerned here with the worst case, it suffices here to consider only when $R^\top r$ lies in the plane of r and d^*, thus reducing the induction step to 2 dimensions.

Now we prove the general step of the induction. We know that we can view this in 2 dimensions, so let us take d^* to be the x-axis (recall that we defined it to be unit length, so we can do this without distorting lengths). Again, we are concerned with minimizing $\langle R^\top r, d^* \rangle$. As we have seen, we can assume R is a scaled projection. So if it is onto a vector

z which forms an angle θ with r and ϕ with the negative x-axis, then

$$\langle R^\top r, d^* \rangle \geq \lambda^* \|r\|_2 \cos(\theta)\cos(\phi) \geq \lambda^* \|r\|_2 \cos^2(\psi/2)$$

where ψ is the angle between r and the negative x-axis. Let $r = (x, y)$ now. Our inductive hypothesis is that $x \geq x^*\delta$. We also have $\phi = \tan^{-1}(y/x)$. Standard trigonometric manipulations (and careful choice of sign) give us that

$$\|r\|_2 \cos^2(\psi/2) \geq \frac{1}{2}(x - \sqrt{x^2 + y^2}) \geq \frac{1}{2}\left(x^*\delta - \frac{\delta}{1-\lambda^*}\right)$$

Our goal is to get $\langle R^\top r + d, d^* \rangle \geq \frac{1}{2}\left(x^*\delta - \frac{\delta}{1-\lambda^*}\right) + x^*\delta$ to be $\geq x^*\delta$ Thus, to see what x^* works, we solve for x^* and get this is true if

$$x^* \geq \frac{\beta - (\frac{1}{2} + \beta)\lambda^*}{(1-\lambda^*)(1 - \frac{1}{2}\lambda^*)}$$

The expression on the right is positive when $\lambda^* < \frac{\beta}{\frac{1}{2}+\beta}$. In other words, when λ^* satisfies this equation, an x^* exists satisfying what we want. This proves (Deviation) for this generalized error model, using positive semi-definite matrices, and completes the proof of Theorem 7

3.6 Biological Rate Matrix Model

Whereas previous sections were focused on mathematically-motivated definitions in order to prove general statements, our goal in this section is to apply our results and methods in a more biologically-motivated setting. In particular, previous sections used evolutionary models where the transition matrices were 'positive semi-definite' or 'double-stochastic', terms which rarely arise in the biological literature. We will instead focus on rate-based evolutionary models. In these models, a stochastic rate matrix Q encodes a stationary distribution, as well as the transition rates between each pair of states. Although our motivation in this section is to state explicit results in models favored by biologists, we will also provide results in this section which apply even when the stationary distribution is nonuniform.

Recall that a stochastic rate matrix Q (definition 2.1) satisfies that each row sums to 0 and each off-diagonal entry is non-negative. Of particular interest is the general time-reversible rate matrix proposed by Tavare in 1986 [82]. We will reproduce his four-state rate matrix here as a concrete example. Let $(\pi_1, \pi_2, \pi_3, \pi_4)$ be the stationary probabilities of the four states, and let $x_1, x_2, x_3, x_4, x_5, x_6$ be non-negative parameters. The rate matrix is given by

$$Q = \begin{pmatrix} * & x_1 & x_2 & x_3 \\ \pi_1 x_1/\pi_2 & * & x_4 & x_5 \\ \pi_1 x_2/\pi_3 & \pi_2 x_4/\pi_3 & * & x_6 \\ \pi_1 x_3/\pi_4 & \pi_2 x_5/\pi_3 & \pi_3 x_6/\pi_4 & * \end{pmatrix}$$

where the diagonal entries are determined by the necessity of each row summing to 0. It can be checked that this rate matrix produces transition matrices with stationary distribution given by $\pi = (\pi_1, \pi_2, \pi_3, \pi_4)$. It can also be checked that this results in a time-reversible transition matrix (that is, at equilibrium, you will see a transition from state i to state j as often as a transition from state j to state i), with the rates governed by the x_i. Although biologists prefer to consider such time-reversible instances, our following result will not make use of time-reversibility.

The rate-matrix model that we work with is the following: each character exhibits a rate matrix, which we label Q_1 and Q_2. We do not necessarily need to know these rate matrices, but we must know the stationary distribution $\pi = (\pi_1, \ldots, \pi_{\mathbf{C}})$ for each character. These may differ between the two characters in question, but for simplicity, we will suppose they are identical – generalizing to different stationary distributions is straightforward. Note that this does not deviate significantly from our previous models; our previous models had implicit knowledge of the stationary distributions as all the stationary distributions were uniform. We now allow for non-uniform stationary distributions, but we will require them to be known. Note that in this evolutionary model, the transition matrices all share their eigenbases, and all eigenvalues are positive (both a consequence of Fact 2.3).

We first modify our algorithm.

The difference now is that we are also given the stationary distribution of the underlying

Algorithm 2 Dependence Detection 2

```
1: procedure DependenceDetection2($\{X_r\}, \{Y_r\}, \varepsilon, \pi = (\pi_1, \ldots, \pi_{|\mathbf{C}|})$)
2:     for $(i, j) \in \mathbf{C} \times \mathbf{C}$ do
3:         $Z_{i,j} \leftarrow |\{l \mid (X_l, Y_l) = (i, j)\}|$
4:     end for
5:     for $(i, j) \in \mathbf{C} \times \mathbf{C}$ do
6:         if $|Z_{i,j} - n\pi_i\pi_j| \geq \varepsilon n$ then
7:             output dependent
8:         end if
9:     end for
10:    output independent
11: end procedure
```

biological model. Consequently, the main test of the algorithm compares the counts of state-pairs against the expected counts from the underlying biological models if the two characters evolved independently, instead of against the counts of other state-pairs. The latter made sense in the earlier iteration because all state pairs had the same expected count due to the stationary distributions being necessarily uniform.

We prove the following analogue of Theorem 6.

Theorem 8. *Algorithm **D**ependence Detection 2 is correct with $1 - 1/\text{poly}(n)$ probability in the rate matrix model, with the uniform rank-1 dependence model.*

As before, we will need that in the independent case, the counts are close to the expected products $\pi_i\pi_j$ of the marginal stationary distributions.

Lemma 3.12. *If the characters are independent, then for all $(x, y) \in \mathbf{C} \times \mathbf{C}$, $|\mathbb{E}[Z_{(x,y)}] - n\pi_x\pi_y| \leq O(n^{1-\beta})$ for some constant β depending on λ, the upper bound on the norms of all the transition matrices.*

The proof is identical to the one of 3.5.

We will similarly need that in the dependent case, the counts deviate in a significant manner. In particular,

Lemma 3.13. *If the characters are dependent under the uniform rank-1 dependence model, then there exists $(x, y) \in \mathbf{C} \times \mathbf{C}$ such that $|\mathbb{E}[Z_{(x,y)}] - n\pi_x\pi_y| \geq \varepsilon n$.*

Proof. Let $d^\top$ be the row for the uniform rank-1 dependence model. Let r_ℓ be the distribution of state pairs at leaf ℓ in the dependent case. We will show that $\langle r_\ell, d\rangle \geq \|d\|_2^2(1-\lambda)$, so $\epsilon = \|d\|_2^2(1-\lambda)/s^2$ is sufficient.

Recall that

$$r_\ell^\top = d^\top + \sum_{i=1}^{\mathsf{dist}(\ell)} d^\top A_i + r_0^\top B$$

where, as before, $A_i = \prod_{k=i+1}^{\mathsf{dist}(\ell)} Q_{e_k}$ and $B = \prod_{k=1}^{\mathsf{dist}(\ell)} Q_{e_k}$. Since the Q_{e_k} share an eigenbasis, and their eigenvalues are all non-negative, we see that $d^\top A_i d$ for any i is positive. $d^\top d = \|d\|_2^2$ is clear. And as before, $|r_0^\top B d| \leq \|r_0^\top\|_\infty \|Bd\|_1$ by Cauchy-Shwartz and the norm bound of our transition matrices. Thus,

$$\langle r_\ell, d\rangle \geq \|d\|_2^2(1-\lambda)$$

□

The proof of Theorem 8 is then identical to the proof of Theorem 6, substituting Lemmas 3.12 and 3.13 for Lemmas 3.5 and 3.6, respectively.

3.7 Discussion and Future Research

In the previous sections, we have shown both positive and negative results for detecting a specific form of dependence of characters. However, there is a gap between the positive and negative results in the positive semi-definite world. In fact, both are approximate numerical results, and having an analytic solution in either case would be more satisfying.

As noted in the preliminaries, we use a very primitive test. There are few instances where a census is really as good as a likelihood method. As mentioned in the background information, works on likelihood exist [31, 66], and another direction is to actually apply their techniques in our models to see how we do comparatively.

There are also new directions to consider on the bigger picture. In this chapter, we have considered testing the dependence of two characters. As mentioned at the beginning of the chapter, dependency across more than two characters in biologically realistic scenarios will still be visible as a dependency of two characters. From a purely theoretic standpoint though, we might ask if there is an efficient way to detect dependency in large sets. It is trivial to extend the result further to any constant number of characters, and thus to solve the following problem by brute force: given a collection of characters, find all the sets up to some constant size which are dependent.

Other directions could involve other error models. In particular, one could hope to solve an issue with our error model: the errors are in some sense blind to the 'edge lengths' of the underlying tree. In biology, the edges are often given a length which reflects the time between events. Our upper bound on the second eigenvalue becomes a lower bound on this length. We wonder if there is an adaptation of our model to better mesh with short edge lengths.

One attempt at a treatment that allows for short edge lengths, is to consider the rate matrix transitions that we addressed in the last section, and instead of altering the transition matrices, alter the joint rate matrix. Since we deal with joint-transition matrices, the rate matrix Q is replaced by a joint rate matrix of the form $Q \otimes I_n + I_n \otimes Q'$, and an error is considered on top of this joint rate matrix. However, detection in the manner we have used in this chapter becomes trivial in that the dependence either manifests obviously in the stationary distribution or is undetectable by our census methods. A more in-depth analysis which attempts to analyze tree structure would be required.

Chapter 4

Identifying and Diagnosing Faults in Read-Once Formulas

We now consider the question of diagnosing a read-once formula. In this problem, we are given a read-once formula F, as well as access to an implementation of F which may be faulty. First, we would like to know if we can efficiently detect errors in any blackbox implementation of a blueprint. Second, we would like to know where the errors occur when the implementation and the blueprint differ. In this chapter, we will give algorithms to answer both of these questions, though the efficiency of the algorithm for the second depends on a balance condition on the implementation. These algorithms will show that the ability to diagnose a formula is dependent only on the ability to diagnose the gates which comprise the formula, and not specifics of the formula itself.

We will call the tree underlying a read-once formula its *topology.* The classical model of read-once formulas uses a basis consisting of AND, OR, and NOT gates, leading to the use of the term 'AND-OR trees.' In this chapter, however, we will not necessarily use this basis (although it is a clean example basis to keep in mind). We will permit more general bases, consisting of a set of 'identifiable' gates, a notion we make more precise in Section 4.1. The intuition is that the allowable gates are those for which we can solve our problem on a depth-1 formula consisting only of that gate.

Our error model in this chapter will be deterministic: a 'faulty' or 'bad' gate will always output precisely the wrong answer, relative to the inputs received from its children. One may think of the fault as adding a NOT gate to the output of the bad gate. On the other hand, a 'good' gate will always output the correct answer relative to the inputs received from its children. Note that the output of a gate is dependent upon errors occurring in its descendents, and errors themselves may cancel out, as in [39][86]. In addition, we will permit the inputs themselves to be faulty – they are negated consistently in the same manner.

Our model and problem can be stated as follows. Suppose we are given a read-once formula, which we will refer to as the *blueprint*. This corresponds to an intended design for the formula. In addition, we are given an *implementation* of the formula, where some of the gates of the blueprint are replaced with faulty versions as described above. We are given only black box access to the implementation – if we set all the inputs, we are allowed to see the output, but we are given full access to the blueprint – we can see exactly its topology and its gates. Our goal is to use our knowledge of the blueprint to determine first if the implementation is faithful to the blueprint, and if it is not, diagnose or locate the faults in the implementation.

We recall that the implementation has an identical topology to the blueprint. Furthermore, we are armed with knowledge of the gate in the corresponding position of the blueprint.

In Section 4.1, we will prove our two positive results. The first is that we can always determine whether the implementation is identical to the blueprint – that is, every gate in the implementation behaves identically to its corresponding gate in the blueprint.

Theorem 9. *Given a blueprint read-once formula f and an implementation $\hat{f}$, it is always possible to test equality, $f = \hat{f}$, in a number of probes linear in the sizes of the formulas.*

We will also show a matching lower bound.

Proposition 4.1. *There exists a blueprint read-once formula and an implementation of this blueprint, such that testing equality requires a number of probes linear in the size of the formulas.*

The second goal is to diagnose the errors. Suppose that we are promised there is some constant bound c on the number of errors. Then there are only $O(n^c)$ possible implementation. Treating each of these as an ‘blueprint’, we can compare it to the actual implementation using the same method as testing equality. Thus in $O(n^{c+1})$ it is possible to diagnose the formulas, if there is a constant bound c on the number of errors.

However, if the number of errors is not so bounded, then our positive result requires the existence of a $\{0,1\}$-oracle for the implementation. This oracle will, given a partial setting of the inputs, produce two inputs, each extending the given partial setting, one which causes the implementation to output 0 and one which causes the implementation to output 1, if such inputs exist.

Theorem 10. *Given a blueprint read-once formula f, an implementation $\hat{f}$, and a $\{0,1\}$-oracle for $\hat{f}$, we can exactly identify the erroneous gates in $\hat{f}$ in a number of probes and oracle queries linear in the sizes of the formulas.*

To actually make use of this result, we provide a probabilistic method for implementing this oracle, under a ‘balance’ condition, whose formal definition is given later in the paper. Informally, an implementation is balanced if at every gate, the probability of computing a 1 and the probability of computing a 1 are lower bounded by an inverse polynomial, where the probability is over uniformly random choices for the input bits. Using the theorem above, we provide a probabilistic method of exact diagnosis when the balance *of the implementation* is at least 1/poly.

However, we will show in Section 4.2 a negative result that we essentially require 1/poly balance. One could hope first that a 1/poly balance of the blueprint is sufficient as we have no effective control over the balance of the implementation, or even that no balance condition is necessary. We provide a blueprint which has 1/poly balance, for which we can get an implementation using just $O(\log n)$ errors that has worse than 1/poly balance, and in fact requires $2^{\Omega(n)}$ probes to diagnose exactly.

Theorem 11. *There exists a blueprint read-once formula with 1/poly balance and an implementation that has only $O(\log n)$ errors, which requires $2^{\Omega(n)}$ queries to locate all the errors.*

In Section 4.3, we will view this problem as a learning problem, and adapt our results as learning problems.

4.1 Testing Equality and Diagnosis of Errors

We define a notion of identifiability for a family of read-once formulas.

Definition 4.2. *Let $\mathcal{F}$ be a family of read-once formulas with the same topology. $\mathcal{F}$ is* identifiable *if for every two formulas $f_1, f_2 \in \mathcal{F}$ that are different, there exists an input x such that $f_1(x) \neq f_2(x)$.*

Definition 4.3. *For a formula f define the neighborhood, $\mathcal{E}(f)$ to be the set of read-once formulas that are constructed by modifying f by doing an arbitrary subset of the modification operations listed below:*

- *For an input x to f, replace x with $\overline{x}$*
- *For a gate g in f, replace g with NOT-g.*

We can only expect to diagnose the errors of a formula f exactly if $\mathcal{E}(f)$ is identifiable. We now define the same notion of identifiability for gates.

Definition 4.4. *Let g be an arbitrary gate with input arity r. Let f_g be the depth-one read-once formula on r inputs, containing just the gate g. We will say g is* identifiable *if $\mathcal{E}(f_g)$ is identifiable.*

We note a few facts about identifiable gates. First, every identifiable gate depends on all of its inputs – if it does not depend on some input, then there is no way to distinguish between negating that input or not. Second, for any even arity r, all threshold gates are identifiable (a threshold gate is one which counts the number of inputs which are 1, and outputs 1 if it exceeds some fixed threshold). For any odd arity r, all non-trivial threshold gates, except the $(r+1)/2$-threshold gate, are identifiable. In particular, AND and OR gates of any arity are identifiable. Finally, if g is an identifiable gate, then $\hat{g}$ given by negating one or more of the inputs to g and possibly the output to g, is also identifiable. NOT-gates are not identifiable, but if the output of a NOT-gate f, feeds into an identifiable gate g, then the formula with gates f and g is identifiable.

On the other hand, XOR gates are not identifiable. The set $\mathcal{E}(XOR)$ realizes just two truth tables, corresponding essentially to the parity of the number of faults surrounding the XOR gate, while there are 2^{r+1} possible formulas in $\mathcal{E}(XOR)$.

A basis consists of a set of allowed types of gates. For example, a standard basis consists of AND, OR, and NOT gates; Another consists of just NAND gates.

Definition 4.5. *A basis $\mathcal{G}$ of gates will be called* identifiable *if every read-once formula f over the basis $\mathcal{G}$ is identifiable.*

We observe that an easy consequence of this definition is that if a basis $\mathcal{G}$ is identifiable, then each gate $g \in \mathcal{G}$ must be identifiable, as there is a formula consisting of just the gate g over that basis. Conversely, we will show (Theorem 13) if a read-once formula f consists of only identifiable gates, then $\mathcal{E}(f)$ is identifiable. Thus, a basis $\mathcal{G}$ is identifiable if and only if each gate in $\mathcal{G}$ is identifiable – that identifiability of a set of gates means that any formula constructed from those gates is also identifiable.

While our results hold for an arbitrary basis, it may be helpful for intuition to regard the gates in the formula as AND, OR, NAND, and NOR. In the remainder of this paper, we will fix a finite basis consisting of identifiable gates $\mathcal{G}$.

Testing Equality of the Implementation

Let r^* be the maximum input arity of a gate in $\mathcal{G}$. We begin by showing that for any blueprint read-once formula f over an identifiable basis $\mathcal{G}$, and an implementation $\hat{f} \in \mathcal{E}(f)$, it is always possible to test equality of f and $\hat{f}$ in $O(2^{r^*}|f|)$ probes to the blackbox. Recall that equality here means that every gate of $\hat{f}$ behaves identically to the corresponding gate in $\hat{f}$.[1]

Theorem 12 (Restatement of Theorem 9)**.** *Let f be a blueprint read-once formula, and $\hat{f} \in \mathcal{E}(f)$ an implementation. Then there exists an algorithm to determine the equality of f and $\hat{f}$ in $O(2^{r^*}|f|)$ probes.*

[1]It is possible to modify the algorithm in this result to test equality of f and $\hat{f}$ as boolean formulas instead and drop the identifiability requirement.

Proof. The algorithm begins at the root of f, verifies that the root is correct, and then recurses on each subformula.

Let g be the gate at the root of f, and suppose g has input arity r. Let $f_1, \ldots, f_r$ be the subformulas of the children of g in the blueprint, and let $F_1, \ldots, F_r$ be the subformulas of the children of g in the implementation.

Choose vectors $\vec{x}_{i,b}$ for $1 \leq i \leq r$ and $b \in \{0, 1\}$ such that $f_i(\vec{x}_{i,b}) = b$, where $\vec{x}_{i,b}$ is a setting of exactly those variables in the subformula f_i. Since the blueprint f is known, we can work backwards from each f_i to find such inputs. Then perform all 2^r probes of $\hat{f}(\vec{x}_{1,b_1}, \ldots, \vec{x}_{r,b_r})$ for $b_1, \ldots, b_r \in \{0, 1\}$.

We will now show that if all 2^r probes of $\hat{f}$ agree with their corresponding evaluations on f, then the gate g at the root is correct. First, based on the fact that the probes agree with f, we infer some conditions on the values at the F_i's, which we cannot observe directly.

First, for every i, $F_i(\vec{x}_{i,0}) \neq F_i(\vec{x}_{i,1})$. Suppose otherwise. Since g is identifiable, it depends on each input. Thus, there is some setting of the inputs other than its i-th input such that g is either the identity or negation on its i-th input. So if $F_i(\vec{x}_{i,0}) = F_i(\vec{x}_{i,1})$, then g will appear to not depend on its i-th input in $\hat{f}$, where we know it does in f, proving that f and $\hat{f}$ are not equal, but our assumption here is that all 2^r probes of $\hat{f}$ agree with the corresponding evaluation on f.

Now we know that for each i, $F_i(\vec{x}_{i,b})$ is equal to either b or $1 - b$, which is effectively saying the i-th input to g is negated or not. And g itself is possibly negated. However, by hypothesis, g is identifiable, and so we can determine now if g has been replaced by its negation in the implementation. If it has, we can stop and say they are not equal. If has not been negated, then we can also apply the identifiability of g to claim that for all i and b, $F_i(\vec{x}_{i,b}) = b$.

Now we can recurse on each subformula F_i. Since we know that g is correct and we have inputs $\vec{x}_{j,b}$ for $j \neq i$ to control the other inputs to g, we can effectively isolate and look at the subformula F_i on its own.

It is clear that for each gate g of the read-once formula f, we perform one set of 2^r probes where r is the input arity of g. Thus, the total number of probes is linear in the size of f and 2^{r^*} where r^* is the maximum arity of a gate in $\mathcal{G}$. □

Note that in the algorithm we have given, all the probes are chosen only on the assumption that the previous tests have not proven that the implementation differs from the blueprint. In fact, we can choose for each subformula f', the inputs $\vec{x}_{f',i}$ ahead of time, and use these inputs any time they are needed in the algorithm above. Thus, the algorithm we have given is non-adaptive. In addition, we note that testing a node g in the formula actually tests both g and its children, so in a model containing both high and low arity gates, we may not need to pay the full cost for isolated high-arity gates.

We now state a proposition, showing that we cannot in general solve this problem in a sublinear number of probes.

Proposition 4.6 (Restatement of Proposition 4.1)**.** *There exists a read once formula f using only arity-2 OR gates such that given an unknown implementation $\hat{f}$, $\Omega(|f|)$ probes are required to test equality of $\hat{f}$ with f.*

Proof. Define a 'caterpillar' as follows: a caterpillar on 1 gate is simply a formula of one gate. A caterpillar on n gates is a tree where the left subtree of the root is a caterpillar on $n-1$ gates, and the right child of the root is an input. Let f be a caterpillar on n arity-2 OR gates.

Consider the subset $\mathcal{F} \subset \mathcal{E}(f)$ of possible implementations, consisting of read-once formulas with exactly two errors. Label the nodes from the root down the caterpillar as $v_1, \ldots, v_n$. For $1 \leq i < n$, denote by F^i, the implementation that contains errors exactly at v_i and v_{i+1}. Let $\mathcal{F} = \{F^i | i = 1, \ldots, n-1\}$.

We may assume that any algorithm for testing equality is non-adaptive – if we examine the computational tree based on the probe responses, we see that for every probe, one branch rejects and one branch continues (or accepts). Thus, we may consider an algorithm for equality as a set of probes S.

We note that the only input on which f outputs 0 is the all 0's input – each $F^i \in \mathcal{F}$ also outputs 0 on this input.

Thus to find a difference, for each $F^i \in \mathcal{F}$, S must contain an input on which F^i outputs 0, and f outputs 1. Fix an F^i. The errors in F^i are at v_i and v_{i+1}. We know that an input distinguishing F^i from f must contain a 1, otherwise both of these will output 0. If an input attached to a gate above v_i is set to 1, then both formulas will output 1. Thus some input below v_i must be set to 1. In addition, if the right child input of v_i is set to 0, then some input below v_{i+1} is set to 1. In this case, v_{i+1} (being in error) will output 0, leading to v_i (again being in error) outputting 1. Thus, in order for an input to cause F^i to output 1, the probe must have a 1 at the input that is the right child of v_i, and have no 1's at inputs above v_i.

Thus, each $f_i \in \mathcal{F}$ requires a different input to distinguish it from f, and so the set S contains at least $\mathcal{F}$ probes. $\square$

Exact Diagnosis of Errors in the Implementation

Now we approach the main question, of exact diagnosis of the errors in the implementation. Here, we are not only interested in whether there *are* any errors, but also identifying exactly where the errors occur. Precisely, we are again given a blueprint f and an implementation $\hat{f}$. Our aim is to identify exactly which gates in $\hat{f}$ are faulty compared to their counterparts from f.

We define a $\{0,1\}$-*oracle* for the implemented read-once formula $\hat{f}$. The rough idea is that the oracle is given a partial assignment to the inputs of $\hat{f}$, and returns two inputs both matching this partial assignment, one causing the implementation to output 0, and the other causing it to output 1. The definition below restricts the class of partial assignments that are allowed.

Definition 4.7. *A* $\{0,1\}$-oracle *for a read-once formula $\hat{f}$ is an oracle that does the following: it takes as input a node g in $\hat{f}$, and a set C of the children of g. In addition, it receives a partial assignment fixing exactly those inputs that are not in the subtrees rooted at nodes in C. Given these, it outputs two settings of the inputs under C such that combined with the given partial setting: one setting causes the formula $\hat{f}$ to output 0, and the other causes the formula $\hat{f}$ to output 1. If one or the other such setting does not exist, then it outputs which one does not exist.*

We now show how to use such an oracle to exactly diagnose an implementation $\hat{f}$.

Theorem 13 (Restatement of Theorem 10)**.** *Let f be a blueprint read-once formula, and $\hat{f}$ be an implementation. Let A be a $\{0,1\}$-oracle for $\hat{f}$. Then there exists an algorithm to exactly diagnose the errors of $\hat{f}$, using $O(n2^{r^*})$ probes and $O(nr^*2^{r^*})$ oracle calls.*

Proof. The key point to our algorithm will be that the identifiability of a node allows us to determine the faultiness of each of its children.

Suppose that $\vec{x}$ and $\vec{y}$ are partial assignments to the inputs of f such that each input is given an assignment in at most one of $\vec{x}$ or $\vec{y}$. Then let $\vec{x} \circ \vec{y}$ denote the partial assignment that assigns the values of $\vec{x}$ and the values of $\vec{y}$. If $\vec{x}$ and $\vec{y}$ exactly partition the inputs, then $\vec{x} \circ \vec{y}$ is a full assignment.

In order to apply identifiability at a node g, we will need the following partial assignments:

1. A partial assignment $\vec{y}_g$ for all inputs not under g such that either, for every partial assignment $\vec{x}$ for exactly the inputs under g, $f(\vec{y}_g \circ \vec{x}) = g(\vec{x})$, or for every partial assignment $\vec{x}$ for exactly the inputs under g, $f(\vec{y}_g \circ \vec{x}) = 1 - g(\vec{x})$.

2. For each child c_i of g, two partial assignments to the inputs under c_i, $\vec{z}_{c_i,0}$ and $\vec{z}_{c_i,1}$, such that $c_i(\vec{z}_{c_i,0}) = 1 - c_i(\vec{z}_{c_i,1})$ (that is, one will lead to c_i outputting 0 and the other to outputting 1, though we will not need to know which is which at this point).

We will implement a recursive approach involving both a top-down phase and a bottom-up phase. In the top-down phase, we will maintain the following *entry condition*: when the recursion reaches a node g, we will have a partial assignment $\vec{y}_g$ as described above. And in the bottom-up phase, we will maintain the following *exit condition*: when we finish at g, we will have diagnosed the errors at every node in the subtree of g, except for g itself.

Note that if we have diagnosed a subtree, then we know exactly where the errors occur and thus have full knowledge of the subtree. As a result, we can generate inputs under those subtrees as needed.

The base case for the entry condition is clear – the first node we reach is the root, and every input is under the root, so an empty 'partial assignment' suffices.

Now we show how we satisfy the entry condition for nodes other than the root. Let g be the current node, for which we already have $\vec{y}_g$ as a partial assignment to satisfy the entry condition to g. Let $c_1, \ldots, c_r$ be the children of g. Suppose we have already completed the exit conditions at some (possibly empty subset) of the children of g. For notational convenience, let us say the completed children are $c_1, \ldots, c_k$. For each completed child c_i, we can generate $\vec{z}_{c_i,0}$ and $\vec{z}_{c_i,1}$ as described above because the exit condition applies. Note that we can only guarantee that $c_i(\vec{z}_{c_i,0})$ and $c_i(\vec{z}_{c_i,1})$ are different but not exactly their values since the exit condition at c_i does not require us to know whether c_i itself is faulty.

For each choice of $b_1, \ldots, b_k \in \{0, 1\}$, we provide A, the oracle, with the following arguments: the node g, a subset of its children $\{c_{k+1}, \ldots, c_r\}$, and the partial assignment $\vec{y}_g \circ \vec{z}_{c_1,b_1} \circ \cdots \vec{z}_{c_k,b_k}$ which assigns to all inputs not under $c_{k+1}, \ldots, c_r$. Note that when we explore the first child under a node g, the partial assignment will just be $\vec{y}_g$.

Since g is identifiable and thus depends on all of its inputs, there is some choice of $b_1, \ldots, b_k$ for which the oracle succeeds and returns two inputs $\vec{x}_0$ and $\vec{x}_1$ compatible with the given partial assignment. In fact, we can find two such inputs that have Hamming distance 1 using the oracle. If $\vec{x}_0$ and $\vec{x}_1$ are at distance greater than 1, simply query the oracle on every input on a shortest path between them on the hypercube and we will find two successive points where the answers change. For notation, assume this input on which $\vec{x}_0$ and $\vec{x}_1$ differ occurs under c_{k+1}. Then let $\vec{y}_{c_{k+1}}$ be a partial assignment agreeing with both $\vec{x}_0$ and $\vec{x}_1$, assigning to all inputs not under c_{k+1}. $\vec{y}_{c_{k+1}}$ must satisfy the entry condition for c_{k+1} since $f(\vec{x}_0)$ and $f(\vec{x}_1)$ differ. (Note, therefore that the oracle's answers control the order in which the children of g are explored in the top-down phase.)

The base case for the exit condition is also simple: if g is an input (a leaf), then the exit condition is vacuous and is thus satisfied automatically.

In general, suppose again that we are at a node g, with children $c_1, \ldots, c_r$, where we have now satisfied the exit condition for all the children. Then we are able to construct the partial assignments $\vec{z}_{c_i,b_i}$ for each $1 \leq i \leq r$ and $b_i \in \{0, 1\}$. We also have the partial assignment $\vec{y}_g$ from the entry condition to g. For each choice of $b_1, \ldots, b_r \in \{0, 1\}$, we query the implementation $\hat{f}(\vec{y}_g \circ \vec{z}_{c_1,b_1} \circ \cdots \circ \vec{z}_{c_r,b_r})$ to construct a truth table for g, on the b_i's and the output value f. Identifiability then applies and tells us exactly which of the children are faulty. Since the exit conditions at the children were already satisfied, the nodes in each of their subtrees were already diagnosed. We have now diagnosed the children of g, which completes the exit condition for g.

Observe that at if g is not the root, we cannot distinguish between a fault at g and a fault between g and the root, so we do not yet diagnose g. However, at the root, we can exactly diagnose g since any error appearing to occur at g must occur at there since there is nothing above g. This allows us to complete the diagnosis at the root.

To analyze oracle and query complexity, we see that we use at most 2^{r^*} oracle calls for the entry condition to each node, and at most 2^{r^*} probes for the exit condition at each node.

□

Optimizations can be made in this recursion. We do not need to re-find the inputs for each child in C' every time, and we do not need to iterate over all $2^{|C'|}$ possibilities each time. If c was the last child we recursed on, then we have a setting which already works for $C' \setminus \{c\}$, and thus only need to test the two values of c. This reduces the number of oracle calls to $2r^*$ per node, from $r^* 2^{r^*}$, so we use only $O(nr^*)$ oracle calls overall.

Now we provide a probabilistic implementation of this oracle, under an additional assumption on the *implementation* read-once formula $\hat{f}$. To state this assumption, we define a notion of balance.

Definition 4.8. *The* balance *of a gate g in a read-once formula is the lesser of the probabilities that the value of the gate is 0 or 1, when the inputs below that gate are chosen uniformly at random. The* balance *of a read-once formula is the minimum of the balances of all of its gates.*

If the implementation $\hat{f}$ has balance which is lower bounded by an inverse polynomial $1/p(n)$, then the following lemma will provide a probabilistic algorithm for a $\{0, 1\}$-oracle for $\hat{f}$.

Lemma 4.9. *If $\hat{f}$ is an implementation with balance lower bounded by an inverse polynomial $1/p(n)$, then there is a probabilistic implementation of a $\{0, 1\}$-oracle for $\hat{f}$ using $(p(n))^{r^*} \log(2/\varepsilon)$ probes, which fails with probability at most ε.*

Proof. The implementation is simple. On input g, C and some partial setting $\vec{x}$ of the inputs not under C, we will randomly choose values for the inputs under C until we have a setting that has $\hat{f}$ output 0 and a setting that has $\hat{f}$ output 1, or until we have tried $(p(n))^r \log(2/\varepsilon)$ inputs.

To prove correctness, it suffices to show that when such settings of the inputs under C exist, we find them with high probability. We will lower bound the probability of finding each such setting in a single probe. For simplicity, let us speak of finding a setting of the inputs under C which causes $\hat{f}$ to output 0 (the other case is obviously identical). Since such a setting exists, there exists at least one setting of the specified children C of g such that $\hat{f}$ would output 0 if we just shortcut the formula and forced the specified children to take certain values.

Then the probability when choosing the inputs under C at random of causing the specified children C to take those values is at least $1/(p(n))^{r^*}$. Thus, the probability of success of one probe is at least $1/(p(n))^{r^*}$, and the probability of failure after $(p(n))^{r^*} \log(2/\varepsilon)$ probes is at most $\varepsilon/2$. A union bound over failing to find the inputs to produce 0 and the inputs to produce 1 yields an error probability of at most ε..

□

Finally, we note that a modification of this algorithm may also be used to test equality of the blueprint and implementation. The oracle simply uses the blueprint to pick inputs that yield 0 and 1, and if either disagrees in the implementation, then we stop and reject.

4.2 Lower bounds for Diagnosis

In this section, we show that it is not always possible to locate all errors, or even one error, in polynomially-many queries if the balance of the implementation is not polynomially-bounded, even if there are only $\log n$ errors in the implementation and even if we assume that we start from a blueprint that is balanced. In addition, we will use only the arity-2 AND and OR gates. In particular, we now construct a blueprint of polynomial balance, and show a corresponding class of implementations with $\log n$ errors which can not be distinguished in any polynomial number of queries.

The tree for our formula f on n nodes will be a complete binary tree. The top $\log n - \log\log n$ layers will be all AND nodes, and the lower $\log\log n$ layers will be all OR nodes.

Proposition 4.10. *f has polynomial balance.*

Proof. Let r be a node in the lower $\log\log n$ layers. Then r outputs 0 if and only if all inputs under it are 0. Since it is in the lower $\log\log n$ layers, it has at most $2^{\log\log n} = \log n$ inputs under it, and thus $\frac{1}{n}$ of its inputs output 0, and all other inputs output 1. Thus it has polynomial balance.

Now let r be a node in the upper $\log n - \log\log n$ layers. Let $s_1, \ldots, s_k$ be its descendents at level $\log\log n$. Then r is 0 if any of $s_1, \ldots, s_k$ is 0. This is thus with probability at least $1/n$. So we need to guarantee the probability r is 1 is high enough. r is 1 if and only if all of $s_1, \ldots, s_k$ are 1. Thus the probability that r is 1 is $(1 - 1/n)^k$. $k < n$, so the probability r is 1 is at least $\frac{1}{e}$. □

Let $r_1, \ldots, r_k$ for $k = n/\log n$ be the nodes at height $\log\log n$ (the maximal height OR nodes). The class of implementations will be those where $\log n$ errors are located in the set $\{r_1, \ldots, r_k\}$. Note there are $\binom{\frac{n}{\log n}}{\log n}$ such formulas, which is super-polynomial.

Proposition 4.11. *The class $\mathcal{E}(f)$ of implementations can not be distinguished in polynomially many queries.*

Proof. We allow the distinguishing algorithm more power. In particular, we also permit the algorithm to set the values of $r_1, \ldots, r_k$ (before being modified by errors), instead of the inputs. The only way the algorithm can gain information is if the algorithm can set each of the faulty r_i's to 0 (so that after fault, they output 1), and the non-faulty r_i's to 1. Any other values at the r_i will cause the implementation to output 0, and offer no information. Then the best that an algorithm can do to try to diagnose the implementation is to try every possibility – but there are super-polynomially many.

□

4.3 Error Diagnosis as Learning

Work by Angluin et al.[3] and Bshouty et al.[7] provide algorithms for learning read-once formulas in a general sense. However, their results rely on the use of equivalence queries. This allows them to compare to a constant circuit to produce an input which causes the implementation to output a specific value. In particular, this allows them to seek inputs which yield an output of 0, and inputs which yield an output of 1.

In our model, equivalence queries would have to be between the implementation and another read-once formula in $\mathcal{E}(f)$, since our concept class is no longer any read-once formula, but a read-once formula in $\mathcal{E}(f)$. This would appear to be much less powerful. In fact, we can simulate such equivalence queries using a linear number of membership queries in our model. The algorithm is simple – run the algorithm of Theorem 12 on the two formulas. Either the algorithm will complete, affirming equivalence, or it will halt at some point where we have an input which causes the formulas to have differing outputs. Instead of equivalence queries, we rely on a $\{0,1\}$-oracle which gives us inputs that cause the implementation to output specific outputs.

PAC Learning under a Product Distribution

Schapire [75] shows that it is possible to learn read-once (arithmetic) formulas under a PAC model, if the inputs are drawn from a product distribution. Note that Pitt and Valiant established that unless RP=NP, even read-once formulas cannot be learned in a distribution-free model. We adapt to provide a PAC-like-learning algorithm for inputs drawn from a product distribution, under our model. This yields a more combinatorial alternative to the algorithm given by Schapire, that functions in our setting.

The intuitive idea to our adaptation is that we can approximately learn the product distribution over the inputs, and then we can implement a probabilistic oracle like the one in Lemma 4.9. Now the balance will be dependent upon the product distribution, instead of uniform. If there are no unbalanced nodes, then this oracle and the algorithm in Theorem 13 simply complete. If there are unbalanced nodes however, we will take advantage of the fact that we only need to learn approximately. So instead of diagnosing under unbalanced nodes, we will simply guess an unbalanced implementation, which will be approximately close to the actual implementation because both will have low balance.

Proposition 4.12. *Suppose we are given an error parameter δ and an accuracy parameter ϵ, along with black box access to a real formula $\hat{f}$ as before, and $\mathcal{P}$ a product distribution on the inputs. Then with probability at least $1-\delta$, we can output a real formula $\tilde{f} \in \mathcal{E}(f)$ for which $\Pr_{x \ \mathcal{P}}[\tilde{f}(x) \neq \hat{f}(x)] \leq \epsilon$.*

Proof. First we draw from $\mathcal{P}$ to learn the product distribution using standard techniques, until our hypothesis $\tilde{\mathcal{P}}$ is with probability at least $1-\delta/2$, within variational distance $\epsilon/(8n^2/\epsilon \cdot \log(2n/\delta) \cdot 2^{r^*})$ of the $\mathcal{P}$.

Define a node to be balanced if its balance is at least $\epsilon/4n$, and unbalanced otherwise. Each time we need to use the $\{0,1\}$-oracle, we will use the parameters of $\tilde{\mathcal{P}}$ to generate samples as needed.

We define a conditional balance of a node to be the balance of the node, when we have fixed some of its inputs. We will say a node is conditionally balanced under some setting of some of its inputs, if its conditional balance when those inputs are fixed is at least $\epsilon/4n$, and conditionally unbalanced otherwise.

We now describe the changes to the algorithm. Suppose we are at a gate g with children C, and have diagnosed some of its children $C' \subset C$. Instead of querying the oracle $2^{|C'|}$ times, we will skip the role of the oracle. We will sample $4n/\epsilon \cdot \log(2n/\delta)$ inputs for the inputs under $C \setminus C'$. Fix one of the settings of the inputs under C', and see if the sampled inputs under $C \setminus C'$ will produce both a 0 and a 1. If yes, we proceed as before. Otherwise, we simply pick one of the sampled inputs arbitrarily, fix those as the inputs under $C \setminus C'$. Now note that we can still pick inputs setting g to 0 and to 1 by using the fact that C' is non-empty and is diagnosed (so we must have been able to find inputs setting g to 0 and to 1). If C' were empty, then we would never have recursed to g.

Note that we have not diagnosed $C \setminus C'$ and their descendents at this point. Instead, at the end, we will simply pick a distribution of faults under each $C \setminus C'$ so as to minimize the balance at each node of $C \setminus C'$ and by deciding whether the root is in error or not we can determine which input is biased against.

We note that the balance of a node can be written as the minimum of the probability a node is 0 and the probability a node is 1. Each is linear in the probabilities of each of its children being 0 or 1. Thus, a dynamic program which for each node remembers an arrangement of errors under that node which minimizes the probability of the node being 0, and an arrangement which minimizes the probability of the node being 1, will also allow us to minimize the balance of each node.

First, we analyze the probability of failure. The first source of failure is a probability $\delta/2$ of failure from learning the product distribution. The other source is if some node g with children C, and some subset $C' \subset C$ of diagnosed children, was conditionally balanced but we decided it was not. Since it is conditionally balanced, we should find an output for 0 and for 1 in each trial with probability at least $\epsilon/4n$. Since we run $4n/\epsilon \cdot \log(2n/\delta)$ trials, the probability of failure is at most $\delta/2n$. Since we can fail at at most n gates, the total probability of this type of failure is $\delta/2$. A union bound gives us an overall probability of failure of at most δ.

Second we analyze the quality of approximation in the absence of failure. Note that the approximation comes from two sources: our approximating the product distribution $\mathcal{P}$, and the replacing of subformulas with computed minimum balance formulas. Since our distribution is variational distance $\epsilon/(8n^2/\epsilon \cdot \log(2n/\delta))$ and we use the learned distribution at most $4n^2/\epsilon \cdot \log(2n/\delta)$ times (each set of $4n/\epsilon \cdot \log(2n/\delta)$ trials either lets us diagnose a node or lets us skip diagnosing a node, since the conditional balance was bad, and there are n nodes), the total error caused by approximating $\mathcal{P}$ is at most $\epsilon/2$.

Now at each node g for which we could not diagnose some of its children $C \setminus C'$, the conditional balance for all settings of C' was at most $\epsilon/4n$, so the distribution on $C \setminus C'$ is $\epsilon/4n$-close to constant. Since our choice of a guess for the nodes under $C \setminus C'$ minimizes balance, it must also be $\epsilon/4n$-close to constant, so the distance between the implementation and our guess is at most $\epsilon/2n$. Since there are at most n nodes, the probability that for some input we are wrong at one or more of them is at most $\epsilon/2$, giving us an overall probability of ϵ of being incorrect. $\square$

4.4 Discussion and Future Research

In the previous sections, we showed how we can diagnosis the faulty gates in a read-once formula when the gates are wrong in a deterministic version of the von Neumann model. We also showed how we can adapt it into a learning result. This gives two overall directions for further research.

First, we could consider different models of faulty gates, or models where gates can fail in different ways. The notion of identifiability that we gave is quite general though, and due to identifiability being both necessary and sufficient, further results should relax or change the goals. One option, which would extend our result, is to ask that faults be identified 'as well as possible'. That is, if there is a fault, but the cause is ambiguous, simply identify that there is a fault and give the possibilities (to contrast, in the previous sections we assumed there was a unique possible cause). This was a direction we considered because of XOR gates. In our model, we can at best see the parity of the number of faults occurring at the inputs and the output of an XOR gate. In a relaxed model, we would be happy with an output stating that an error exists at this location, and the possible explanations. However, we have not completed a result on this path.

The second direction involves exploring the idea of the learning version of the result, and what we call 'blueprint learning' in general. In traditional PAC learning (and variants), the aim is to learn a concept from an entire class. In blueprint learning, we no longer learn from an entire concept class, but an unknown drawn drawn from a specific 'local' area surrounding a given 'blueprint'. Our goal is to be able to learn either more efficiently, or learn where it is otherwise not possible.

We have examined the idea of learning DNFs under a similar model – a blueprint DNF is given along with a black box containing a DNF which is made from the blueprint by negating some (or none or all) of the literals in each term. The goal is to discern which literals are negated in each term. However, our progress in this direction has been limited, even when the DNFs are further restricted. The departure from trees is certainly a source of the difficulty – because variables occur multiple times, we have significantly less control.

We will also note an easier version of the DNF blueprint learning problem. If instead of having to adhere to the topology of the blueprint (as we would if our goal is only to discern which literals in each term are negated), we simply had to produce *any* DNF which approximates the black box (the set of concepts that can be realizable by the black box relative to the blueprint remains as above). The blueprint is a huge hint. For each term of the blueprint DNF, we retain every set of negations of its literals which is consistent with the drawn samples (that is, for every sample that satisfies the particular set of negations, the result from the black box is TRUE), and the DNF we learn is the union of all of these terms. This leads to a consistent formula, which is sufficient in blueprint learning as it is in the traditional PAC model. The difficulty then in adhering to the given topology is deciding which of these settings needs to be kept, as we may not be able to keep them all.

An interesting possibility here is that the blueprint allows us to output a hypothesis which is a DNF. This is impossible in the traditional PAC model under standard hardness assumptions. However, since we are given the blueprint, we would still like to conform to the topology of the blueprint if it is possible to do so.

Chapter 5

Optimizing Read-Once Formulas with Faulty Gates

We turn our attention now to the question of optimizing a read-once formula consisting of faulty gates. Specifically, we are given the blueprint of a read-once formula, and we are to implement it with real gates. Each gate may be faulty as in the von Neumann fault model – recall that this means that each gate has a probability ϵ of outputting the complement of the correct answer instead of what is expected based on the inputs it has received. Note we are dealing with a probabilistic error model rather than the deterministic fault model used in the previous chapter.

We are given gates of two types with known failure probabilities, which we will call 'good' and 'bad'. It can be assumed that the good gates fail with lower probability than the bad gates. The good gates need not be faultless. We are given only a limited supply of good gates, so we cannot simply cover the entire formula with good gates. Instead, we must choose where to place them in the formula so as to optimize the average correctness probability of the implementation over a known input product distribution.

In particular, let F be a read-once formula with n gates, and we are given $k < n$ good nodes. In addition, we are given a product distribution $\mathcal{D}$ over the inputs – that is, each input has a probability of being 0 or 1, independent of all the other inputs. Our goal is to use up to k good nodes, along with as many bad nodes as we need, to construct an implementation $\hat{F}$ such that we maximize the correctness:

$$\Pr_{x \leftarrow \mathcal{D}}[F(x) = \hat{F}(x)]$$

We call this the general allocation problem. We will present an approximate algorithm which finds an allocation with a correctness that is optimal to within factor $(1 + \epsilon)$ for $\epsilon > 0$ of the true maximum. Note that we do not know if there is an exact polynomial-time algorithm, or if this problem is NP-hard.

We make some simplifications initially, and will discuss how we may remove them later. We assume that each good gate costs the same amount, regardless of what the gate is. In addition, we will essentially treat the input arity of each gate as a constant, though as long as the arities are all bounded by some constant, the algorithm we present in the next section will still work. However, it does cause us to assume that a good gate is the same cost regardless of the actual arity of the gate.

5.1 Dynamic Program

In this section, we will provide a dynamic program to solve the general allocation problem approximately, and prove the following theorem.

Theorem 14. *There exists an algorithm which finds a $(1 + \epsilon)$-approximation in time polynomial in the size of the formula n and $1/\epsilon$ when all gates have bounded arity.*

We do not know currently if there is an exact algorithm, though any exact algorithm will likely require a different approach than the one presented here.

The good and bad gates will have reliability p_g and p_b, respectively. Let r be the maximum (input) arity of any node in the formula. We will have an exponential dependence on r, but it is reasonable to assume in most applications that r is a constant.

We now give some terms and some notation.

Definition 5.1. *An* allocation *to a tree (or subtree) is a map $A : T \to \{0, 1\}$ where for $x \in T$, $A(x) = 0$ indicates we have assigned a bad gate to x and $A(x) = 1$ indicates we have assigned a good gate to x. The cost of an allocation is $|\{x \in T \mid A(x) = 1\}|$, the number of* *good gates it assigns. A* suballocation *of A to a node y under x is the allocation A restricted to the subtree rooted at y.*

We will use the following convention, similar to the one used in the previous chapter: x will denote both a node and its output in a perfect formula, and $\hat{x_A}$ will denote the output of x when the subtree T_x rooted at x is given the allocation A. We will drop the subscript and write simply $\hat{x}$ where there is no ambiguity.

For a given allocation A to a tree T_x rooted at x, let $a_x^0(A)$ (resp. $a_x^1(A)$) denote the probability that the output $\hat{x} = 0$ when $x = 0$ (resp. $\hat{x} = 1$ when $x = 1$). Let p_x^0 (resp. p_x^1) be shorthand for $\Pr[x = 0]$ (resp. $\Pr[x = 1]$).

We are ready now to present the dynamic program. Let T be the tree representing the read-once formula on n inputs. For each $1 \le i \le n$, let $p_i(0)$ (resp. $p_i(1)$) denote the probability that the input i is 0 (resp. 1). Let ϵ be given for our approximation factor of $(1 + \epsilon)$. Let $\delta = 1 + \frac{\log 1+\epsilon}{n}$.

Suppose we are given k good gates. Our dynamic program will generate a large table $M[x, l, i, j]$. For a node x and for each $0 \le l \le k$, we will generate a table indexed by (i, j) containing allocations to the tree rooted at x of cost *at most l*. Each table entry will contain at most one allocation A, and this allocation A will satisfy that $\delta^{-(i+1)} < a_x^0(A) \le \delta^{-i}$ and $\delta^{-(j+1)} < a_x^1(A) \le \delta^{-j}$.

It is possible for multiple allocations to attempt to occupy the cell in the table – this is the reason for the table and the reason this algorithm can only be approximate. We keep only a single allocation per table cell, and are guaranteed that whichever we keep has both parameters $a_x^0(A)$ and $a_x^1(A)$ very close to the parameters of the other potential occupants of that cell.

We will now prove that this table suffices. The following observation follows from the construction of the table.

Observation 5.2. *Let A be an allocation to x. Let $y_1, \ldots, y_s$ be the $s \le r$ children of x and $A_1, \ldots, A_s$ the suballocations to $y_1, \ldots, y_s$. Then for $t \in \{0, 1\}$, $a_x^t(A)$ is a multivariate polynomial of degree at most r, in the $a_{y_i}^0(A_i)$ and the $a_{y_i}^1(A_i)$. Furthermore, the exponent for any of those is at most 1.*

The main consequence of this is that our table is polynomial in size.

Observation 5.3. *Let i^* and j^* be the largest i and j, respectively, indexing occupied cells in any table for any node in the tree. Then $i^*, j^* = O(n, 1/\delta)$.*

A second consequence, of the second part of Observation 5.2, is that we actually only need to retain a convex hull on the table since the function is multilinear. This allows us to reduce the number of cells kept per node from quadratic to linear.

In order to populate the table of a node x, we first populate the tables of its children, $y_1, \ldots, y_s$. Then to populate the table for x with cost l, we first consider all tuples $(l_1, \ldots, l_s)$ with $l = \sum l_i$ – that is we take x to be a bad gate and divide the allocation of good gates to the children of x. Consider a tuple of allocations to the children of x: $(B_1, \ldots, B_s) \in M[y_1, l_1] \times \cdots M[y_s, l_s]$. This setup is also an allocation to x, allocating under y_i as in the allocation B_i – call this allocation A. Let $a^{\beta}_{y_i}[B_i]$ denote the probability that $\hat{y_i}$, the output of y_i in our faulty implementation, is β when y_i, the true value of y_i relative to the inputs, is β.

Then for this tuple of allocations to the children of x, we can calculate the values $a^0_x(A)$ as follows:

$$\begin{aligned} a^0_x(A) \quad = \quad & \sum_{b_1,\ldots,b_s \,|\, x(b_1,\ldots,b_s)=0} \left[\prod_{i=1}^{s} p^{b_i}_{y_i}\right] \\ & \cdot \left[p_b \sum_{c_1,\ldots,c_s \,|\, x(c_1,\ldots,c_s)=0} \prod_{i=1}^{s} (\chi_{(b_i=c_i)} a^{b_i}_{y_i}(B) + \chi_{(b_i \neq c_i)}(1 - a^{b_i}_{y_i}(B_i))) \right. \\ & \left. + (1-p_b) \sum_{c_1,\ldots,c_s \,|\, x(c_1,\ldots,c_s)=0} \prod_{i=1}^{s} (\chi_{(b_i=c_i)} a^{b_i}_{y_i}(B) + \chi_{(b_i \neq c_i)}(1 - a^{b_i}_{y_i}(B_i))) \right] \end{aligned}$$

where $\chi_{(b_i=c_i)}$ is an indicator variable, taking value 1 when its condition of b_i equaling c_i is satisfied, and 0 otherwise (and similarly for $\chi_{(b_i \neq c_i)}$). The outer sum is over the correct values $b_1, \ldots, b_s$ for the children $y_1, \ldots, y_s$, and the inner sums deal, respectively, with when the realized values $c_1, \ldots, c_s$ for $\hat{y_1}, \ldots, \hat{y_s}$ give us the correct answer 0 and when they do not. A similar calculation, replacing the test of $x = 0$ with $x = 1$ gives us $a^1_x(A)$. This covers all allocations A where x is not chosen to be a good node. In order to cover allocations where x *is* chosen to be good, we repeat the process, except we take tuples $(l_1, \ldots, l_s)$ with $l - 1 = \sum l_i$ (since we have l total gates, and we use one for x, there are $l - 1$ left for the children of x). We can then use the same formula, replacing occurrences of p_b with p_g since we now take x to be a good gate.

It is clear that the combinations of these allocations and assigning x to be good or bad guarantees that the cost of each allocation to T_x we consider is cost at most l. As mentioned, we place each of the considered allocations into the table $M[x, l]$, keeping only one allocation per cell. This operation takes time polynomial in n and δ, but exponential in r. We do not know how to circumvent this exponential dependence on r and suspect it is not possible to do so in general.

Once we have populated the table at the root r, we can simply consider every allocation left in the table at $M[r, k]$, and calculate the average correctness probability at r as $p_r^0 a_r^0 + p_r^1 a_r^1$.

Theorem 15. *This algorithm produces a* $(1 + \epsilon)$*-approximation.*

Proof. The intuition here is that there is some optimal allocation, but since in our algorithm we only keep one allocation per table cell, we have made a series of mistakes and discarded correct allocations in favor of incorrect allocations. However, each time we do so, we have traded the correct allocation for an incorrect allocation which is very close in results.

Formally, let us first fix a post-order traversal of the nodes of the formula, $v_1, \ldots, v_n$. Now, let us construct the following sequence of allocations: $A_0, \ldots, A_n$. A_0 will be the true optimal allocation, and A_n will be the allocation that results from our dynamic program. A_i for $i \geq 1$ will be the optimal allocation, except that the available allocations at subtrees rooted at $v_1, \ldots, v_i$ are limited to those which appear in the table of our dynamic program. This does affect available allocations above these nodes, but this will not be an issue.

Now we observe that the success probability of A_{i+1} is at most a factor δ off from the success probability of A_i. Consider the suballocations selected in A_i and A_{i+1} for the subtree rooted at v_{i+1}. If the suballocation chosen for A_i is still available, then the whole allocation A_i is available for A_{i+1}. Thus, the correctness of A_{i+1} is at least the correctness of A_i. It is also not more, since every choice for A_{i+1} is available for A_i.

Now suppose that the suballocation chosen for A_i is not available for A_{i+1}. This occurs because there is another suballocation at v_{i+1} which is in the same table cell as the suballocation used by A_i. The parameters of the suballocations which are in the same cell are within factor δ by our choice of δ. By repeated application of Observation 5.2 from v_{i+1} up to the root, this implies that there is an allocation available to A_{i+1} whose success probability is within factor δ of A_i, by replacing the suballocation at v_{i+1} with the other suballocation in that table cell and leaving all other allocations the same. Thus, the success probability of A_{i+1} is at least the success probability of this modified version of A_i, which is at most factor δ off from the success probability of A_i, proving the claim.

As a result, going from A_0, the optimum, to A_n, our output, we lose at most a factor δ^n, which is

$$\delta^n = \left(1 + \frac{\log(1+\epsilon)}{n}\right)^n \leq e^{\log(1+\epsilon)} = 1 + \epsilon$$

□

5.2 Discussion and Future Research

As mentioned in the description, our algorithm seems to only be approximate, since otherwise the number of allocations we may have to consider in this brute-force fashion is very large. It would be interesting if there is a clever way to circumvent some portion of these allocations to arrive at an exact solution in polynomial-time. Note that we do not know if this problem is NP-complete, though we do not believe it is.

We noted that we only need to retain a convex hull. A natural question is why this is necessary – perhaps a pareto-optimal curve would suffice. Perhaps the natural inclination would indeed be that we want to make each gate as good as possible. However, an OR gate with one input very likely to be 1 and the other very likely to be 0, would actually prefer poor accuracy at the input likely to be 0.

Finally, this algorithm is exponential in r. This seems necessary if we follow the previous theme of having 'read-once' formulas which allow for a variety of gates and not simply AND and OR gates, as the function to calculate a_x^t recursively could have a number of terms which is exponential in r. The way we construct the possible allocations is also exponential in r though, so we would also need to find a clever way to cut that number down.

In another direction, we can generalize beyond just two levels of gates (good and bad), to more levels of gates. The algorithm essentially does not change, even if instead of allocations of each gate, they are given costs. The dynamic program then has to iterate through all the valid combinations for each cost level, but the numbers we have to consider are bounded by the size of the overall formula and have no potential of being exponential. Indeed, a further extension allows us to handle a spectrum of gates, for which the relationship between cost and reliability is uniformly continuous. We turn this into a discrete set of gates by approximation and solve as above, and correctness essentially follows from uniform continuity of the cost function.

As mentioned at the start of the chapter, the dynamic program does not care about the specific arity of a gate, as long as it is bounded. However, modeling the good/bad gate world realistically with varying arities seems like it should be doable with reasonable cost functions.

Our current goal with this research is to extend the dynamic program to work for formulas in general (as opposed to read-once formulas). It is easy to augment the current dynamic program to deal with the simple case where only a constant number of variables have fan-out greater than 1. Instead of keeping a_x^0 and a_x^1, we keep more specific versions of these values, which condition on the specific values of all the variables of fan-out greater than 1. So for example, if x_1 and x_2 were the only variables which had fan-out greater than 1, at each node we would keep analogues of a_x^0 and a_x^1 except conditioned on each possible setting of (x_1, x_2) as $(0, 0)$, $(0, 1)$, $(1, 0)$ or $(1, 1)$. Clearly, this blows up the size of the grids we keep since we must continue to consider all possibilities.

However, there are nodes which are not ancestors of any x_1 or x_2 values, and at these nodes, we do not need to condition on the settings of (x_1, x_2). As a result, each node conditions on some subset of the variables which have fan-out greater than 1, not necessarily all of them. This, and the approximate nature of our dynamic program, give some hope to the possibility of extending to formulas.

Bibliography

[1] P. G ács and A. Gál. Lower bounds for the complexity of reliable boolean circuits with noisy gates. *IEEE Transactions on Information Theory*, 40:579–583, 1994.

[2] Saad Alrawaf, Igor Chikalov, Shahid Hussain, and Mikhail Moshkov. Diagnosis of constant faults in iteration-free circuits over monotone basis. *Discrete Applied Mathematics*, 166:287–291, 2014.

[3] Dana Angluin, Lisa Hellerstein, and Marek Karpinski. Learning read-once formulas with queries. *Journal of the ACM (JACM)*, 40(1):185–210, 1993.

[4] S. Assaf and E. Upfal. Fault-tolerant sorting networks. *SIAM J. Disc. Math.*, 4:472–480, 1991.

[5] Louis Bachelier. *Louis Bachelier's theory of speculation: the origins of modern finance.* Princeton University Press, 2011.

[6] Richard Bellman, Richard Ernest Bellman, Richard Ernest Bellman, Richard Ernest Bellman, and Etats-Unis Mathématicien. *Introduction to matrix analysis*, volume 960. SIAM, 1970.

[7] Nader H Bshouty, Thomas R Hancock, and Lisa Hellerstein. Learning arithmetic read-once formulas. *SIAM Journal on Computing*, 24(4):706–735, 1995.

[8] Joseph H Camin and Robert R Sokal. A method for deducing branching sequences in phylogeny. *Evolution*, pages 311–326, 1965.

[9] James A Cavender. Taxonomy with confidence. *Mathematical Biosciences*, 40(3):271–280, 1978.

[10] Deeparnab Chakrabarty, Sampath Kannan, and Kevin Tian. Variance on the leaves of a tree markov random field: Detecting character dependencies in phylogenies. *arXiv preprint arXiv:1112.5508*, 2011.

[11] Joseph T Chang. Full reconstruction of markov models on evolutionary trees: identifiability and consistency. *Mathematical biosciences*, 137(1):51–73, 1996.

[12] CK Chow and CN Liu. Approximating discrete probability distributions with dependence trees. *Information Theory, IEEE Transactions on*, 14(3):462–467, 1968.

[13] Constantinos Daskalakis, Elchanan Mossel, and Sébastien Roch. Optimal phylogenetic reconstruction. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 159–168. ACM, 2006.

[14] Roland L'vovich Dobrushin and SI Ortyukov. Lower bound for the redundancy of self-correcting arrangements of unreliable functional elements. *Problemy Peredachi Informatsii*, 13(1):82–89, 1977.

[15] Roland L'vovich Dobrushin and SI Ortyukov. Upper bound on the redundancy of self-correcting arrangements of unreliable functional elements. *Problemy Peredachi Informatsii*, 13(3):56–76, 1977.

[16] Anthony WF Edwards and Cavalli LL Sforza. The reconstruction of evolution. *Heredity*, 18, 1963.

[17] Albert Einstein. Über die von der molekularkinetischen theorie der wärme geforderte bewegung von in ruhenden flüssigkeiten suspendierten teilchen. *Annalen der physik*, 4, 1905.

[18] Peter L Erdos, Michael A Steel, László A Székely, and Tandy J Warnow. A few logs suffice to build (almost) all trees (i). *Random Structures and Algorithms*, 14(2):153–184, 1999.

[19] Péter L Erdös, Michael A Steel, LászlóA Székely, and Tandy J Warnow. A few logs suffice to build (almost) all trees: Part ii. *Theoretical Computer Science*, 221(1):77–118, 1999.

[20] W. Evans and L. Schulman. Signal propagation, with application to a lower bound on the depth of noisy formulas. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, pages 594–603. IEEE, 1993.

[21] William S Evans and Leonard J Schulman. On the maximum tolerable noise of k-input gates for reliable computation by formulas. *IEEE Transactions on Information Theory*, 49(11):3094–3098, 2003.

[22] Martin Farach and Sampath Kannan. Efficient algorithms for inverting evolution. *Journal of the ACM (JACM)*, 46(4):437–449, 1999.

[23] James S Farris. A probability model for inferring evolutionary trees. *Systematic Biology*, 22(3):250–256, 1973.

[24] T. Feder. Reliable computation by networks in the presence of noise. *IEEE Transactions on Information Theory*, 35(3):569–571, 1989.

[25] Joseph Felsenstein. Maximum-likelihood estimation of evolutionary trees from continuous characters. *American journal of human genetics*, 25(5):471, 1973.

[26] Joseph Felsenstein. Cases in which parsimony or compatibility methods will be positively misleading. *Systematic Biology*, 27(4):401–410, 1978.

[27] Joseph Felsenstein. Alternative methods of phylogenetic inference and their interrelationship. *Systematic Biology*, 28(1):49–62, 1979.

[28] Joseph Felsenstein. Evolutionary trees from dna sequences: a maximum likelihood approach. *Journal of molecular evolution*, 17(6):368–376, 1981.

[29] Joseph Felsenstein. Confidence limits on phylogenies: an approach using the bootstrap. *Evolution*, pages 783–791, 1985.

[30] Joseph Felsenstein. Inferring phylogenies. 2004.

[31] Joseph Felsenstein and Gary A Churchill. A hidden markov model approach to variation among sites in rate of evolution. *Molecular Biology and Evolution*, 13(1):93–104, 1996.

[32] Joseph Felsenstein and Joseph Felenstein. *Inferring phylogenies*, volume 2. Sinauer Associates Sunderland, 2004.

[33] Walter M Fitch, Emanuel Margoliash, et al. Construction of phylogenetic trees. *Science*, 155(3760):279–284, 1967.

[34] Stuart Geman and Christine Graffigne. Markov random field image models and their applications to computer vision. In *Proceedings of the International Congress of Mathematicians*, volume 1, page 2, 1986.

[35] Nick Goldman. Phylogenetic information and experimental design in molecular systematics. *Proceedings of the Royal Society of London B: Biological Sciences*, 265(1407):1779–1786, 1998.

[36] Sally A Goldman, Michael J Kearns, and Robert E Schapire. Exact identification of read-once formulas using fixed points of amplification functions. *SIAM Journal on Computing*, 22(4):705–726, 1993.

[37] Judy Goldsmith, Robert H Sloan, Balázs Szörényi, and György Turán. Theory revision with queries: Horn, read-once, and parity formulas. *Artificial Intelligence*, 156(2):139–176, 2004.

[38] Vladimiar A Gurvich. Repetition-free boolean functions. *Uspekhi Matematicheskikh Nauk*, 32(1):183–184, 1977.

[39] B. Hajek and T. Weller. On the maximum tolerable noise for reliable computation by formulas. *IEEE Transactions on Information Theory*, 37(2):388–391, 1991.

[40] Thomas Hancock and Lisa Hellerstein. Learning read-once formulas over fields and extended bases. In *Proceedings of the fourth annual workshop on Computational learning theory*, pages 326–336. Morgan Kaufmann Publishers Inc., 1991.

[41] MD Hendy and David Penny. Branch and bound algorithms to determine minimal evolutionary trees. *Mathematical Biosciences*, 59(2):277–290, 1982.

[42] Xuedong D Huang, Yasuo Ariki, and Mervyn A Jack. *Hidden Markov models for speech recognition*, volume 2004. Edinburgh university press Edinburgh, 1990.

[43] John P Huelsenbeck and Keith A Crandall. Phylogeny estimation and hypothesis testing using maximum likelihood. *Annual Review of Ecology and Systematics*, pages 437–466, 1997.

[44] E. Ising. Beitrag zur Theorie des Ferromagnetismus. *Zeitschrift fur Physik*, 31:253–258, February 1925.

[45] Thomas H Jukes and Charles R Cantor. Evolution of protein molecules. *Mammalian protein metabolism*, 3:21–132, 1969.

[46] Edin Kadric, Kunal Mahajan, and André DeHon. Energy reduction through differential reliability and lightweight checking. In *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*, pages 243–250. IEEE, 2014.

[47] Sampath Kanna, Elchanan Mossel, and Kevin Tian. Optimal allocations of resources in faulty formulas. Unpublished manuscript, 2013.

[48] Sampath Kannan and Kevin Tian. Locating errors in faulty formulas. Unpublished manuscript, 2016.

[49] Motoo Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of molecular evolution*, 16(2):111–120, 1980.

[50] Ross Kindermann and J. L. Snell. *Markov Random Fields and Their Applications.* AMS, 1980.

[51] Dan Kleitman, Tom Leighton, and Yuan Ma. On the design of reliable boolean circuits that contain partially unreliable gates. *JCSS*, 55(3):385–401, 1997.

[52] Zvi Kohavi and Niraj K Jha. *Switching and finite automata theory.* Cambridge University Press, 2009.

[53] Steffen L Lauritzen. Time series analysis in 1880: A discussion of contributions made by tn thiele. *International Statistical Review/Revue Internationale de Statistique*, pages 319–331, 1981.

[54] David Asher Levin, Yuval Peres, and Elizabeth Lee Wilmer. *Markov chains and mixing times*. American Mathematical Soc., 2009.

[55] David Asher Levin, Yuval Peres, and Elizabeth Lee Wilmer. *Markov chains and mixing times*. American Mathematical Soc., 2009.

[56] Wayne P Maddison. A method for testing the correlated evolution of two binary characters: are gains or losses concentrated on certain branches of a phylogenetic tree? *Evolution*, pages 539–557, 1990.

[57] Tim Massingham and Nick Goldman. Detecting amino acid sites under positive selection and purifying selection. *Genetics*, 169(3):1753–1762, 2005.

[58] Faruck Morcos, Biman Jana, Terence Hwa, and José N Onuchic. Coevolutionary signals across protein lineages help capture multiple protein conformations. *Proceedings of the National Academy of Sciences*, 110(51):20533–20538, 2013.

[59] Faruck Morcos, Andrea Pagnani, Bryan Lunt, Arianna Bertolino, Debora S Marks, Chris Sander, Riccardo Zecchina, José N Onuchic, Terence Hwa, and Martin Weigt. Direct-coupling analysis of residue coevolution captures native contacts across many protein families. *Proceedings of the National Academy of Sciences*, 108(49):E1293–E1301, 2011.

[60] Elchanan Mossel. Phase transitions in phylogeny. *Transactions of the American Mathematical Society*, 356(6):2379–2404, 2004.

[61] Elchanan Mossel. Survey-information flow on trees. *DIMACS series in discrete mathematics and theoretical computer science*, 63:155–170, 2004.

[62] Elchanan Mossel and Sébastien Roch. Learning nonsingular phylogenies and hidden markov models. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 366–375. ACM, 2005.

[63] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms.* Chapman & Hall/CRC, 2010.

[64] Jerzy Neyman. Molecular studies of evolution: a source of novel statistical problems. *Statistical decision theory and related topics*, pages 1–27, 1971.

[65] Borivoje Nikolic. Design in the power-limited scaling regime. *IEEE transactions on Electron Devices*, 55(1):71–83, 2008.

[66] Mark Pagel. Detecting correlated evolution on phylogenies: a general method for the comparative analysis of discrete characters. *Proceedings of the Royal Society of London B: Biological Sciences*, 255(1342):37–45, 1994.

[67] N. Pippenger. On networks of noisy gates. In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 30–36. IEEE, 1985.

[68] N. Pippenger. Reliable computation by formulae in the presence of noise. *IEEE Transactions on Information Theory*, 34:194–197, 1988.

[69] Leonard Pitt and Leslie G Valiant. Computational limitations on learning from examples. *Journal of the ACM (JACM)*, 35(4):965–984, 1988.

[70] Christopher J Preston. *Gibbs states on countable sets: Gibbs states and Markov random fields.* 1974.

[71] R. Reischuk and B. Schmeltz. Reliable computation with noisy circuits and decision trees — a general $n \log n$ lower bound. In *Proc. 32nd Annual IEEE Symp. Found. of Comput.*, pages 602–611. IEEE, 1991.

[72] Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987.

[73] Yurii A Rozanov. *Probability theory: a concise course.* Courier Corporation, 2013.

[74] Naruya Saitou and Masatoshi Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425, 1987.

[75] Robert E Schapire. Learning probabilistic read-once formulas on product distributions. *Machine Learning*, 14(1):47–81, 1994.

[76] Michael Schöniger and Arndt Von Haeseler. A stochastic model for the evolution of autocorrelated dna sequences. *Molecular phylogenetics and evolution*, 3(3):240–247, 1994.

[77] Charles Semple and Mike Steel. Phylogenetics. 2003.

[78] Cavalli LL Sforza and Anthony William Fairbank Edwards. Analysis of human evolution. *Genet. Today*, 3:923–933, 1964.

[79] Robert R Sokal and Charles D Michener. A statistical method for evaluating systematic relationships. *Univ Kans Sci Bull*, 38:1409–1438, 1958.

[80] Frank Spitzer. Markov random fields and gibbs ensembles. *American Mathematical Monthly*, pages 142–154, 1971.

[81] Mike Steel. My favourite conjecture.
http://www.math.canterbury.ac.nz/ m.steel/files/misc/conjecture.pdf, 2001.

[82] Simon Tavaré. Some probabilistic and statistical problems in the analysis of dna sequences. *Lectures on mathematics in the life sciences*, 17:57–86, 1986.

[83] Elisabeth RM Tillier and Richard A Collins. Neighbor joining and maximum likelihood with rna sequences: addressing the interdependence of sites. *Molecular Biology and Evolution*, 12(1):7–15, 1995.

[84] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[85] John A Vlontzos and Sun-Yuan Kung. Hidden markov models for character recognition. *IEEE transactions on image processing: a publication of the IEEE Signal Processing Society*, 1(4):539–543, 1991.

[86] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. pages 43–98, 1956.

[87] Marian Von Smoluchowski. Zur kinetischen theorie der brownschen molekularbewegung und der suspensionen. *Annalen der physik*, 326(14):756–780, 1906.

[88] Ziheng Yang, Sudhir Kumar, and Masatoshi Nei. A new method of inference of ancestral nucleotide and amino acid sequences. *Genetics*, 141(4):1641–1650, 1995.