uOttawa

Faculty of Graduate and Postdoctoral Studies

Computing most probable sequences of state transitions in continuous-time Markov systems

---

Author:

Pavel Levin

Supervisor:

Dr. Theodore Perkins

A thesis submitted to University of Ottawa in partial fulfillment of the requirements of the degree of Master of Science.

Abstract

Continuous-time Markov chains (CTMC's) form a convenient mathematical framework for analyzing random systems across many different disciplines. A specific research problem that is often of interest is to try to predict maximum probability sequences of state transitions given initial or boundary conditions. This work shows how to solve this problem exactly through an efficient dynamic programming algorithm. We demonstrate our approach through two different applications – ranking mutational pathways of HIV virus based on their probabilities, and determining the most probable failure sequences in complex fault-tolerant engineering systems. Even though CTMC's have been used extensively to realistically model many types of complex processes, it is often a standard practice to eventually simplify the model in order to perform the state evolution analysis. As we show here, simplifying approaches can lead to inaccurate and often misleading solutions. Therefore we expect our algorithm to find a wide range of applications across different domains.

Contents

Chapter 1

Introduction

A ubiquitous problem in engineering and science is to reconstruct the trajectory of a dynamical system based on a series of incomplete or noisy observations. Even though the laws governing the system's dynamics may be understood quite well, it is often necessary to make an inference about the system's unobserved behaviour. At any point in time the system can be described by its state, i.e. well-defined configuration of information describing the system. Dynamical systems are different from static ones in that their states evolve, or change, over time.

Random dynamical systems, as the name suggests, evolve randomly in time. This randomness can be associated with the future states and with the rates at which the system evolves. Depending on a particular application, the meaning of the word "state" can vary significantly.

When we talk about random processes that describe real, physical trajectories, a single state can be identified by a set of coordinates in the physical space, or some other definition of location. The problems of motion tracking or robot navigation can be formulated in this way. A typical problem involving these types of systems would be estimating the most likely path of a vehicle when the data describing its location is inherently noisy (e.g. coming from low precision tracking devices).

It also makes sense to define states and trajectories in more abstract ways. Phylogenetics is a branch of biology that tries to relate species according to their genetic sequences. In this context, evolution can be viewed as a random process where the genetic sequences change over time, and therefore states can be associates with particular configurations of those sequences. For example, a researcher may be presented with a part of a genome of an extinct species obtained from a fossil. The researcher may want to know the most probable evolutionary path for that specie, and its most likely surviving descendants.

Pattern recognition problems (e.g. speech recognition, gesture recognition, etc.) can also be viewed as inference problems in random systems. For example, in speech recognition, syllables (or even words) can be states of the system. These states can only be inferred through the sounds produced by the speaker. Interpreting speech, then, is a problem of estimating the most

likely sentence (sequence of syllables), given the sound. The problem turns out to be far from straightforward, given the peculiarities of human voice, general noisiness of the sound medium and the lack of unique correspondence between sounds and syllables in most languages.

Any instance of a random system evolving over time can be described in one of two ways—either by its *trajectory*, or by its *state sequence*. Trajectories describe how the system's states change, and *when* they change, whereas state sequences (or "paths") describe the system's evolution only in terms of its *visited states*, and the exact timing of events is not acounted for. Since we are dealing with random systems, the exact deterministic solution of any kind is impossible by definition. What we can infer, however, are the probabilities of different trajectories or state sequences. This work presents an algorithm for computing the most probable sequences of states in certain types of random systems, called continuous-time Markov chains (CTMCs). We are not interested in how much time the system spends in each state that it visits, we only want to know the sequence of visited states in their exact order.

CTMCs form a very general class of probabilistic models (see Section 2.2), and have found numerous applications across many domains. Their formalism has been used extensively to describe various aspects of mathematical models in biophysics [36], biochemistry [19], queuing theory [20],

robotics [13], epidemiology [22], ecology [40] and economics [25], just to name a few. CTMCs owe their success to the fact that they present the most general way to model memoryless systems with a finite number of states (see Section 2.1) that evolve continuously in time (see Section 2.2).

To simplify problems that involve continuous-time systems, it is customary to translate them into the discrete-time domain [30, 29, 7]. This is an attractive approach since the problem of finding the most probable state sequences can be solved quite easily (see Section 2.3.3) in the discrete-time world. Also, there is a well-known solution (Viterbi algorithm) to the problem of finding the most likely sequence of states when we assume a "hidden" discrete-time model (Hidden Markov Model), in which the actual states of the system (the so-called latent states) are inferred from observations. However, as will be shown in this work, time discretization will often produce wrong and misleading results. *Our research presents the first exact solution to the problem of efficiently finding the most probable sequences of state transitions in CTMCs.* The exact solution that is described here does not require any model simplifying assumptions and provides the solution in continuous-time domain. In the discussion below, including application examples, we will show why a state sequence is a more meaningful notion of a high-probability trajectory than the one obtained by looking at discrete-time models or other types of information. We also show why finding maximum likelihood trajec-

tories (see Section 2.3.4) is usually less informative than finding maximum probability state sequences.

First we will define CTMCs and overview the state of the art in CTMC-related prediction problems (Chapter 2). The algorithm that predicts the most likely state sequences based on boundary conditions, along with the proof that the algorithm always terminates (provided we are dealing with a finite state space), will be presented in Chapter 3. Chapter 4 illustrates one possible application of the algorithm to the problem of determining the most likely mutational pathways of the HIV virus, while Chapter 5 applies the algorithm in the domain of reliability engineering, demonstrating how our approach can be used to create a novel method for fault diagnosis in complex fault-tolerant systems. Finally, the last chapter discusses directions for further research including some possible extensions to the algorithm.

When I joined the project in September 2010, my supervisor, Dr. Perkins, had an abstract idea for an algorithm for computing most likely state sequences for CTMCs. However, he had no good way to compute the probability curves (explained below) that the algorithm requires. My main contributions to the project included: a sound and numerically stable approach to computing the probability curves, an improved method for filtering out suboptimal probability curves, an extension of the method to compute not just the most likely state sequence but the k-most likely sequences for any k,

completion of correctness proofs for all of the above, implementation of the entire approach (in Matlab and R), and detailed applications of the method in the HIV and fault diagnosis domains mentioned above.

Chapter 2

Background

The purpose of this chapter is to be a fairly self-contained primer on continuous-time Markov chains as well as on the state of the art in prediction problems for continuous-time Markov chains. It is assumed that the reader is familiar with basic probability concepts, such as random variable, conditional probabilities, etc. The chapter is split into three major sections. First, we overview stochastic systems and define the Markovian property. Second, we explain what continuous-time Markov chains are. Then, finally, we describe the four major trajectory inference approaches, and discuss the need for a new tool that will be introduced in the consequent chapter.

2.1 Stochastic systems and Markov chains

Real world dynamical systems can be modelled as either deterministic or stochastic. Deterministic systems entail well defined rules that *uniquely* determine how the system will evolve given its current state. That is to say, if we can measure the state of our system with infinite precision we know exactly what is going to happen to it assuming there is no unknown interaction with its environment. Any simple mechanical system is a good example. If we only assume the Newtonian mechanics, then the state of the system at any point in the future can be determined solely by its current state, assuming of course that we know the current state *precisely*.

Continuous-time Markov chains represent stochastic systems. Unlike in deterministic ones, in stochastic systems the current state only offers information on the probability distribution of the future states, not their exact values. Whether it is our limited understanding of the system (e.g. most biological systems), uncertainties in measurements (e.g. weather prediction), or their inherent stochasticity (e.g. quantum mechanics, molecular biology), random systems are only defined by laws that model the probability distributions of their states [44, 45]. In other words if we observe a random system in state x we can say how likely it is that it will be in state y at some point in the future, but we cannot say with certainty (unless the aforementioned likelihood is 1 or 0) whether or not it actually will be in state y, no matter

how precise our measurements are.

Formally, a stochastic (or random) process is a collection of random variables $\{X_t, t \in T\}$. If T is countable, the process is called *discrete*, otherwise it is *continuous*. When X_t's come from an uncountable set (e.g. $\mathbb{R}$, $\mathbb{R}^n$, etc.) then we say the process has a *continuous state space*. Otherwise (i.e. if all possible values of $\{X_t\}$ can be enumerated), the process is said to have a *discrete* state space. This work will only deal with discrete state spaces, in fact we will only be concerned with *finite* state systems, i.e. systems that not only have countable but also finite state space.

A special type of a discrete stochastic process is the so-called *Markov chain*, or a discrete-time Markov chain (DTMC for short). A Markov chain is a discrete stochastic process where the next state of the system only depends on its current state, and any knowledge of the system's past behaviour does not provide any new information regarding its future evolution. Formally, a stochastic process $\{X_n, n \in \mathbb{N}, X_n \in \mathbf{X}, \mathbf{X} \subseteq \mathbb{N}\}$ is a Markov chain if

$$P(X_{n+1} = j \mid X_n = i, X_{n-1} = i_{n-1}, \ldots, X_1 = i_1, X_0 = i_0) = P_{ij}. \quad (2.1)$$

That is, whenever the process is in state i there is a fixed probability P_{ij} of transitioning into state j. This probability does not depend on anything but the current state i and next immediate state j. In particular, it does not depend on any other states the system may have visited before now.

Unless mentioned otherwise, Markov chains are generally assumed to be *time-homogeneous*, meaning that the P_{ij}'s do not change over time.

It is natural to represent a DTMC in matrix form. A matrix $\mathbf{P}$ is said to represent a Markov chain if each entry p_{ij} equals the probability of transitioning from state i to state j. Conversely, any non-negative square matrix whose rows sum to 1 represents a DTMC.

To demonstrate, consider the following example. Suppose that whether or not it rains tomorrow depends only on today's weather. If today is sunny, it will rain tomorrow with probability 0.1, and if it rains today it will rain tomorrow with probability 0.6. If we denote the "sunny" state by 1 and the "rainy" state by 2, this system defines a DTMC whose matrix of transition probabilities is given by

$$\mathbf{P} = \begin{pmatrix} 0.9 & 0.1 \\ 0.4 & 0.6 \end{pmatrix} \tag{2.2}$$

A very useful tool for solving a wide range of Markov chain related problems are the so-called Chapman-Kolmogorov equations. In order to appreciate the Chapman-Kolmogorov equations we first need to define the n-step transition probability. We say that P_{ij}^n is n-step transition probability of an N-state DTMC if it is the probability that this chain in state i will be in state j after n transitions, i.e.

$$P_{ij}^n = P(X_{n+k} = j \mid X_k = i), \qquad n \geq 0, \quad 1 \leq i, j \leq N \tag{2.3}$$

It is worth noting that Eqn. 2.3 says nothing about the path that the system would go through within these n transitions. The total probability, then, is the sum of all possible disjoint trajectories that the DTMC can in principle take.

Chapman-Kolmogorov equations provide a method for computing these probabilities. The equations are

$$P_{ij}^{n+m} = \sum_{k=1}^{N} P_{ik}^{n} P_{kj}^{m}, \qquad \forall\ n, m \geq 0, \quad 1 \leq i, j \leq N \tag{2.4}$$

To see why (2.4) is true, we just use the definition of a Markov chain, in particular its "memorylessness" (also known as the *Markovian* property):

$$\begin{aligned} P_{ij}^{n+m} &= P(X_{n+m} = j \mid X_0 = i) \\ &= \sum_{k=1}^{N} P(X_{n+m} = j, X_n = k \mid X_0 = i) \\ &= \sum_{k=1}^{N} P(X_{n+m} = j \mid X_n = k, X_0 = i) \\ &\qquad \times P(X_n = k \mid X_0 = i) \qquad (2.5) \\ &= \sum_{k=1}^{N} P_{kj}^{m} P_{ik}^{n} \end{aligned}$$

The Markovian property was used in (2.5).

As was noted earlier, Markov chains can be represented by matrices. If our DTMC is defined by a square matrix $\mathbf{P}$, then Eqn. 2.4 can very naturally translate into the matrix form:

$$\mathbf{P}^{(n+m)} = \mathbf{P}^{(n)} \cdot \mathbf{P}^{(m)} \tag{2.6}$$

In particular,

$$\mathbf{P}^{(2)} = \mathbf{P}^{(1+1)} = \mathbf{P} \cdot \mathbf{P} = \mathbf{P}^2,$$

and hence by induction:

$$\mathbf{P}^{(n)} = \mathbf{P}^{(n-1+1)} = \mathbf{P}^{n-1} \cdot \mathbf{P} = \mathbf{P}^n. \tag{2.7}$$

This can be useful for all sorts of Markov chain related calculations. For example, going back to our weather example, if we would like to know the probability of rain in four days given it is raining today, we can just apply Eqn. 2.7 to our DTMC defined by linear transformation (2.2):

$$\begin{aligned}
\mathbf{P}^{(4)} = (\mathbf{P}^2)^2 &= \begin{pmatrix} 0.9 & 0.1 \\ 0.4 & 0.6 \end{pmatrix}^2 \cdot \begin{pmatrix} 0.9 & 0.1 \\ 0.4 & 0.6 \end{pmatrix}^2 \\
&= \begin{pmatrix} 0.85 & 0.15 \\ 0.6 & 0.4 \end{pmatrix} \cdot \begin{pmatrix} 0.85 & 0.15 \\ 0.6 & 0.4 \end{pmatrix} \\
&= \begin{pmatrix} 0.8125 & 0.1875 \\ 0.75 & 0.25 \end{pmatrix}
\end{aligned} \tag{2.8}$$

Matrix (2.8) gives us the 4-step transition probability transformation, and so to determine the probability of rain in 4 days given rain now, we simply look up $\mathbf{P}^{(4)}_{22}$ (recall that we labeled 2 to be the "rainy" state), which happens to be 0.25. Without the linear algebra shortcut (2.7) that is just the restatement of the Chapman-Kolmogorov equations (2.4), solving similar

questions directly, by just using the definitions of conditional probabilities would be immensely more difficult [34].

2.2 Continuous-time Markov chains

A discrete time domain is not only a realistic, but often a very natural mathematical framework for many real-life phenomena, for example whenever time advances in well-defined "steps" or whenever we are dealing with an observational time series [29]. However, most physical systems obey more general state transition laws, where the system can change its state at *any* time, not just at some predetermined time points.

Continuous-time Markov chains (CTMCs) present a natural generalization of discrete-time Markov chains into the continuous time domain. An example of a continuous-time discrete state stochastic model would be a frog leaping from one lily pad to another to get across the pond. States of the system can be associated with the lily pads (i.e. lily pad #1, lily pad #2, and so on). If the time spent at any lily pad is random and *exponentially distributed* (with not necessarily the same parameters for different lily pads), and if the probability of the next lily pad is *only* dependent on which lily pad the frog is currently sitting on, then such a process defines a CTMC. In this work we will only be dealing with *finite* CTMCs, i.e. CTMCs with finite

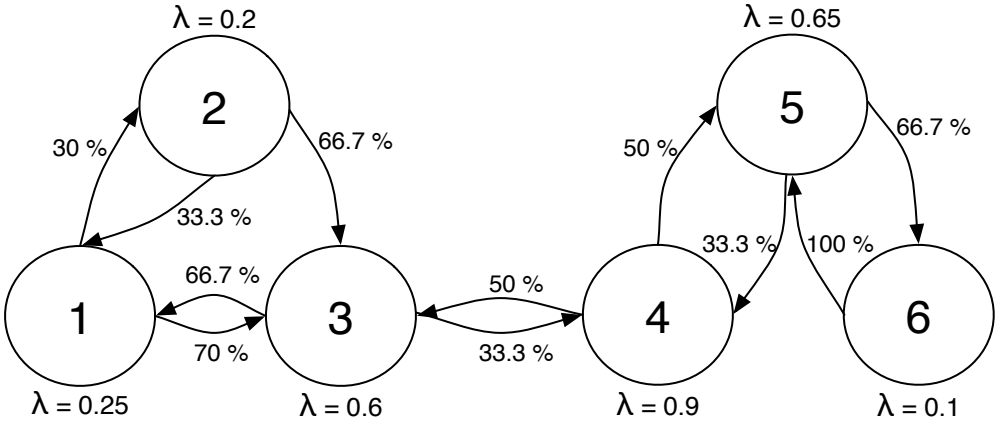

Figure 2.1: A hypothetical 6-state continuous-time Markov chain. Arrows with percentages indicate legal transitions the next state probabilities, and λ's are the dwell time parameters.

state space.

Formally, such a process can be described by the transition matrix T and the dwell time parameters $\{\lambda_i\}_{i=1,2,...,N}$ (N is the size of the state space). Dwell parameters define exponential distributions describing the holding times, e.g. $\lambda_3 = \frac{1}{3}sec^{-1}$ would mean that once the system enters state 3, it will stay there for a random time that is exponentially distributed with the average of 3 *sec*. The transition matrix T is a square matrix of dimension N. $T_{i,j}$ defines the probability of system's next state being j, given its current state is i. We also require $Ti, i = 0$ for any state i. T bears no time-related information, i.e. it says nothing about *when* the transitions happen. That information is contained in λ's, the dwell parameters

An equivalent way of representing a CTMC is with the so-called transition rate matrix Q of the same dimension as T, i.e. N. While λ_i and T_{ij} respectively define how much time the system will spend in state i and what the probability is that the next state will be j, Q_{ij} corresponds to the "probability rate" with which our system goes from state i into state j. The rates of leaving a state are always positive, while the rates of staying are negative. For each state, the (negative) rate of staying should equal the sum of outgoing (positive) rates in magnitude, which also means that the sum of all probability rates for any given state should sum to 0. When $i \neq j$, the two representations are related through formula

$$Q_{ij} = T_{ij} \times \lambda_i \tag{2.9}$$

When $i = j$, $Q_{i,i} = -\lambda_i$, which implies that $\sum_{j \neq i} Q_{ij} = -Q_{ii} = \lambda_i$. So, a finite-state CTMC can be either represented by a single matrix Q that defines the transition rates, or by a combination of transition probabilities matrix T and dwell times $\{\lambda_i\}_{i=1..N}$. While it is often more compact to use the single transition rate matrix Q, it is usually harder to interpret, and for the sake of clarity CTMCs here will be represented through the combination of T and λ's. To give an example, consider the CTMC in Fig. 2.1. This is a 6-state system given by:

$$T = \begin{pmatrix} 0 & 3/10 & 7/10 & 0 & 0 & 0 \\ 1/3 & 0 & 2/3 & 0 & 0 & 0 \\ 2/3 & 0 & 0 & 1/3 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1/3 & 0 & 2/3 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.10}$$

$$\lambda = \begin{pmatrix} 0.25 \\ 0.2 \\ 0.6 \\ 0.9 \\ 0.65 \\ 0.1 \end{pmatrix} \tag{2.11}$$

or, equivalently:

$$Q = \begin{pmatrix} -0.250 & 0.075 & 0.175 & 0 & 0 & 0 \\ 0.067 & -0.200 & 0.133 & 0 & 0 & 0 \\ 0.400 & 0 & -0.600 & 0.200 & 0 & 0 \\ 0 & 0 & 0.450 & -0.900 & 0.450 & 0 \\ 0 & 0 & 0 & 0.217 & -0.650 & 0.433 \\ 0 & 0 & 0 & 0 & 0.100 & -0.100 \end{pmatrix} \tag{2.12}$$

2.3 Prediction problems in continuous-time Markov chains

This section will talk about the state of the art in predicting CTMCs. Four major tools will be covered. First we will look at the continuous-time analog of Eqn. 2.4, called Kolmogorov's forward equation and how it can be used to estimate the future probabilities of the system's states. Then we will look at Gillespie's algorithm which is used to simulate continuous-time Markov chains. Simulations are often used to infer various things about systems when exact solutions are difficult to obtain, either analytically or numerically. Next, we will look at the possibility of translating the problem into the discrete time domain and use this more convenient setting to try to predict things about our CTMCs. The last alternative approach we will look at will be the recently proposed algorithm for estimating maximum likelihood trajectories. The final section of this chapter will summarize the four approaches emphasizing their strengths and weakness. It will also discuss the need for a new approach that will be introduced in the following chapter.

2.3.1 State distributions as a function of time

In a CTMC we can denote the probability of transitioning from state i to state j in time t by $P_{ij}(t)$. Then, similar to the discrete case (Eqn. 2.4), we

have:

$$P_{ij}(s+t) = \sum_k P_{ik}(t)P_{kj}(s) = \sum_k P_{ik}(s)P_{kj}(t) \tag{2.13}$$

Intuitively this holds true in the continuous case as well because in order to transition into j in time $t+s$, the system needs to occupy a certain state k in time t, and by summing over all possible k's we ensure that the identity holds. If we want to be rigorous:

$$\begin{aligned} P_{ij}(t+s) &= P(X(t+s) = j \mid X(0) = 1) \\ &= \sum_{k=1}^{N} P(X(t+s) = j \mid X(s) = k, X(0) = i) \\ &\qquad \times P(X(s) = k \mid X(0) = i) \\ &= \sum_{k=1}^{N} P(X(t+s) = j \mid X(s) = k) \\ &\qquad \times P(X(s) = k \mid X(0) = i) \\ &= \sum_k P_{kj}(t)P_{ik}(s) \end{aligned} \tag{2.14}$$

Since CTMCs evolve in continuous time we will naturally want to derive the expression for the time derivative of $P_{ij}(t)$. Using a standard trick [2], we take a small time increment h and subtract $P_{ij}(t)$ from $P_{ij}(t+h)$:

$$\begin{aligned} P_{ij}(t+h) - P_{ij}(t) &= \sum_k P_{ik}(t)P_{kj}(h) - P_{ij}(t) \\ &= \sum_{k \neq j} P_{ik}(t)P_{kj}(h) - [1 - P_{jj}(h)]P_{ij}(t) \end{aligned}$$

Dividing both sides by h and taking the limit $h \to 0$ yields:

$$\begin{aligned}
\frac{dP_{ij}}{dt} &= \lim_{h\to 0} \frac{P_{ij}(t+h) - P_{ij}(t)}{h} \\
&= \lim_{h\to 0} \{\sum_{k\neq j} P_{ik}(t)\frac{P_{kj}(h)}{h} - \frac{1-P_{jj}(h)}{h}P_{ij}(t)\} \\
&= \sum_{k\neq j} T_{kj}\lambda_k P_{ik}(t) - \lambda_j P_{ij}(t) \qquad (2.15)
\end{aligned}$$

Equation (2.15) is called *Kolmogorov's forward equation.* Eqn. 2.15 can be thought of as a coninuous-time analog of Eqn. 2.4. In particular we can use it to derive the probability distribution of our state space at any point in the future, provided that the initial (current) state is known.

Further,

$$\begin{aligned}
\frac{dP_{ij}}{dt} &= \sum_{k\neq j} T_{kj}\lambda_k P_{ik}(t) - \lambda_j P_{ij}(t) \qquad (2.16) \\
&= \sum_{k\neq j} q_{kj} P_{ik}(t) + q_{jj} P_{ij}(t) = \sum_{k} P_{ik}(t) q_{kj}, \qquad (2.17)
\end{aligned}$$

and so

$$\begin{pmatrix} \frac{dP_{1j}}{dt} \\ \frac{dP_{1j}}{dt} \\ \vdots \\ \frac{dP_{Nj}}{dt} \end{pmatrix}^T = \begin{pmatrix} \sum_k P_{1k}(t)q_{kj} \\ \sum_k P_{2k}(t)q_{kj} \\ \vdots \\ \sum_k P_{Nk}(t)q_{kj} \end{pmatrix}^T \qquad (2.18)$$

If we let $\mathbf{P(t)} = (P(X(t) = 1 \mid X(0) = X_0), P(X(t) = 2 \mid X(0) = X_0), \ldots, P(X(t) = N \mid X(0) = X_0))$ to be the distribution of the states at

time t, given initial state X_0, then we can re-write (2.3.1) as

$$\frac{d}{dt}\mathbf{P}(\mathbf{t}) = \mathbf{P}(\mathbf{t}) \times \mathbf{Q}, \mathbf{P}(\mathbf{0}) = \mathbf{1}_{X=X_0}. \tag{2.19}$$

This is a linear system that is solved by $\mathbf{P}(\mathbf{t}) = \mathbf{P_0}e^{\mathbf{Q}t}$ [2], where $e^{\mathbf{Q}t}$ is the matrix exponential defined by the Taylor series

$$e^{\mathbf{Q}t} = \sum_{i=0}^{\infty} \frac{(\mathbf{Q}t)^i}{i!}$$

Equation 2.19 gives an explicit way of computing the distribution of states at any time t. Here we described a common scenario when our initial state is known with certainty, in which case $\mathbf{P_0}$ will have 1 in the slot corresponding to that state and 0's everywhere else. Of course, this does not have to be the case, and we can only be given the initial probability distribtion of states. Then, the only restriction on $\mathbf{P_0}$ is that it is a vector with non-negative entries that all sum to 1.

Consider the hypothetical example from the previous section (Eqn.2.2). Solving Eqn. 2.19 for our toy example with $t = 10$ *sec* and $\mathbf{P_0} = (1, 0, 0, 0, 0, 0)$ gives $\mathbf{P}(\mathbf{10}) = (0.4286, 0.1654, 0.2115, 0.0589, 0.0499, 0.0856)$, i.e. if we evolve the system for 10 seconds starting from state 1, there is about 42.86% chance of ending up in state 1, 16.54% chance of ending up in state 2, etc. While this information can be extremely useful, solving Eqn. 2.19 provides no information as to *how* the system will get to the possible final states.

2.3.2 Simulating continuous-time Markov chains

One way of obtaining path related information of a system's evolution is through repeated simulations of that system [19, 18]. Continuous-time Markov chains are fairly intuitive to understand and they can be simulated quite efficiently. Recall that in a CTMC the system transitions between a discrete set of states. Each transition takes a random exponentially distributed time with the average parameter associated with that state. If we want to simulate the system from initial state X_0 for a total time of t_F, then it can be done through the following effective computer algorithm:

1. Initialize current system's state X to X_0 and current time t to 0
2. Initialize trajectory S to $\{(0, X_0)\}$
3. While $t < t_F$ do:

    4. Update $t = t + \delta t$, where δt comes from $Exp(\lambda_X)$ probability distribution

    5. Generate next state X according to transition probability matrix T

    6. Add the pair of values (t, X) to the trajectory S

The above pseudocode describes the so-called Gillespie algorithm [19], named after Dan Gillespie who published it in 1977 as an efficient method

to simulate chemical and biochemical reactions that are often modelled as CTMCs.

Line 1 initializes the state and time variables. In line 2 we are initializing a data structure that contains a set of pairs of time-state values. Each pair denotes the exact time the chain transitions to a new state, and the state to which it transitions. By construction each observation happens *exactly* when the system transitions from one state to another. Line 3 checks whether the time t_F has been reached in which case no further simulation is required. Lines 4 through 6 describe the main "Monte Carlo" step of the algorithm. First we generate δt according to an exponential distribution with parameter λ_X, X being system's current state. One efficient way to do this is through inverse transform sampling, which in this case would mean taking a negated natural logarithm of a uniform random variable, scaled by λ_X (i.e. $\delta t = \frac{-ln(u)}{\lambda_X}$, $u \sim U(0,1)$).

The next step (line 5) is to generate the next state X. For that we use the transition probability matrix T, more precisely we are using the vector $(T_{i,1}, T_{i,2}, \ldots, T_{i,N})$, where i is the system's current state, and N is the size of our state space. Effectively we are rolling a weighted N-sided die, whose sides are weighted according to the transition probabilities of our current state. Practically it can be easily implemented by generating a single uniform random variable from $(0,1)$ interval, and then choosing the next state

according to which region of $(0, 1)$ our random variable happened to fall into. The last step inside the loop, line 6, updates our current trajectory with the new state transition.

This process will generate one random trajectory according to the probability laws of our CTMC. Clearly, a single trajectory can rarely be very indicative of the system's behaviour. In order to get a more representative picture, multiple simulations are required. When the number of simulations is sufficiently large it may be possible to statistically analyze the resulting data set of trajectories and assign them probabilities according to their observed frequencies.

Needless to say this approach has a number of drawbacks. First of all, as we increase our simulation time t_F the expected number of state transitions in a trajectory will increase dramatically. In order for statistics to be meaningful a much larger number of simulations will be required (each of which would take longer due to increased t_F). Secondly, although empirical trajectory frequencies will approach correct probability values as we increase our sample (number of simulations), those frequencies are by no means *guaranteed* to be correct, especially if we are forced to take shortcuts due to limited computational power. Thus, although better than nothing, the simulation approach is far from perfect when it comes to inferring hidden trajectory behaviour.

2.3.3 Discretizing time

There are obvious difficulties in tackling computational problems that are defined in the continuous time domain. For one thing, continuity makes it more difficult to subdivide the problem into more trivial ones and then obtain the "grand solution" (either through dynamic programming paradigm or otherwise) by combining the simpler solutions. Discrete-time Markov chains have been studied quite a bit more extensively than their continuous-time counterparts. In particular there is a well known way to solve the discrete-time analog of our problem [21, 29]. The reason trajectory inference is easier in discrete-time Markov chains is because unlike in CTMCs there is no time dependence in the expression for probability of a trajectory in a fixed number of time steps (see Section 2.1 for details).

This is a two-step approach [7]. First we define n, the number of subintervals we want to subdivide $(0, t_F]$ into. This also fixes $\delta t = \frac{t_F}{n}$, the time step length. We can then construct a DTMC that is equivalent to our CTMC for that exact time step δt.[1] That is, the solutions of the DTMC Chapman-Kolmogorov equations should match the solutions of the CTMC Chapman-

[1] Unfortunately, there is no simple rule as to how this number n should be picked. In fact, as our HIV example shows (see Section 4.3), chosing n that is too low will result in missing some short time scale dynamics, whereas chosing n that is too large will result in missing some longer time scale dynamics. This is one of the main disadvantages of the discretization approach.

Kolmogorov equations for our chosen time step δt. The second stage is to compute a maximum likelihood trajectory of our new DTMC for n time steps.

Computing the DTMC is reletivelty straightforward.[2] Recall that a DTMC is completely defined by transition matrix P, in which entry $p_{i,j}$ equals the probabilities of transitioning from state i to state j within 1 time step. CTMC, on the other hand, is determined by the transition matrix T and dwell parameters $\lambda_1, \ldots, \lambda_N$. T is the matrix of "next state" probabilities, and λ's are the exponential wait time parameters for each state. In order for our DTMC probabilities to match the CTMC probabilities (for time δt), we first compute the "self-loop" probabilities for each state (i.e. the probabilities of not leaving the state within time δt). For any state X, this is just given by

$$p_{x,x} = P(X \ \textit{at time}\ \delta t \mid X \ \textit{at time}\ 0) = e^{-\lambda_X \times \delta t} \tag{2.20}$$

Then all the other DTMC transition probabilities will be distributed proportionally to their transition rates in CTMC, i.e.

[2]The method for computing self-loop probabilities that is described here assumes that the time step δt is small enough for the system to make at most one transition with non-negligible probability. This is usually the case, since coarse discretization will inevitably lead to significant loss in accuracy. However, if δt is large, then the self-loop probabilities should be computing using Chapman-Kolmogorov equations.

$$p_{x,y} = P(Y \ \textit{at time}\ \delta t \mid X \ \textit{at time}\ 0) = [1 - e^{-\lambda_X \times \delta t}] \times T_{X,Y} \qquad (2.21)$$

After we obtain the DTMC transition probability matrix $\mathbf{P}$ as described above, the next step is to compute the maximum likelihood trajectory using the standard dynamic programming technique:

1. Create a table F indexed by states i and times t with 2 fields: $Prob$ and $Prev$
2. Initialize $t = 0,\ F(X_0, t).Prob = 1\ F(i, t).Prob = 0\ \forall i \neq X_0$
3. For $t = \delta t,\ 2\delta t,\ 3\delta t,\ \ldots,\ t_F$ do:
    4. For all states i do:
        5. $F(i, t).Prob = max_j F(j, t - \delta t) \times p_{j,i}$
        6. $F(i, t).Prev = \arg\max_j F(j, t - \delta t) \times p_{j,i}$

Here F is a 2-dimensional data structure with two fields. One field, *Prob* contains the probability values, and the other filed *Prev* contains the trajectories. The structure is indexed by states i and (integer) time steps t. $F(i, t).Prob$ contains the probability of the most likely trajectory ending with state i at time t and $F(i, t).Prev$ contains the last transition in the most likely trajectory (without the last state i).

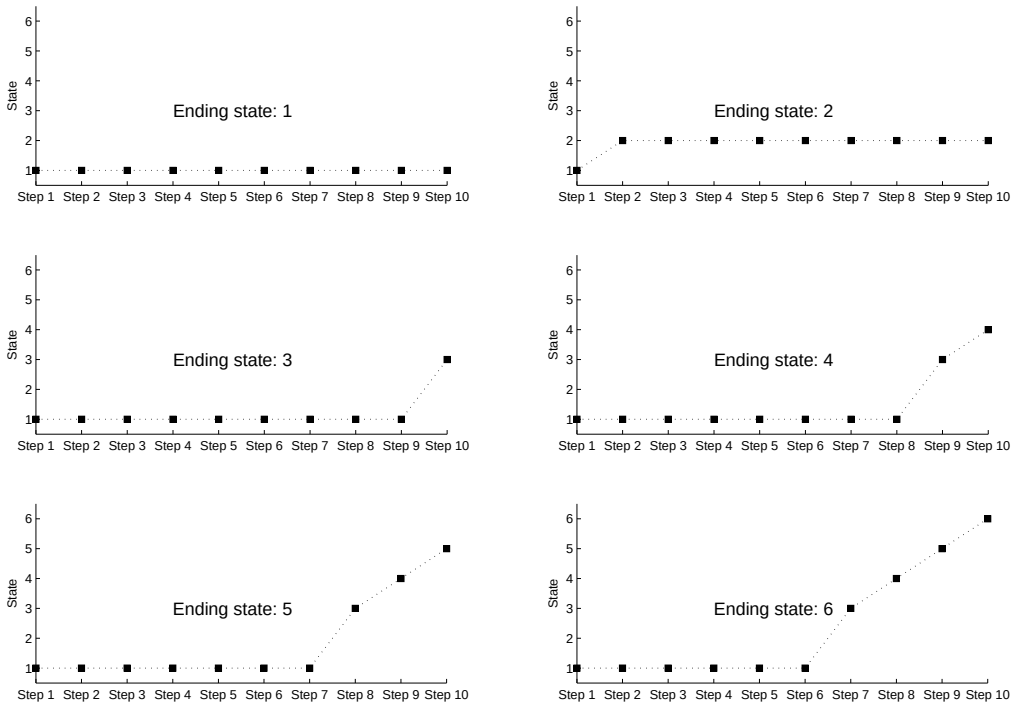

Figure 2.2: Most probable trajectories of the discrete-time version for our toy example. Typical behaviour of such solutions is to dwell in one state for most of the simulation period and only spend a single time step in all other states of the trajectory. Clearly this is not very representative of real world systems.

Although the technique is obviously attractive, this inference method is not only imprecise,[3] but can often produce misleading results (see Section 4.3). Thus, this alternative approach is not suitable as a replacement for, or even approximation to, the exact solution of our trajectory inference problem. To demonstrate, Fig. 2.2 shows the most probable trajectories of the discretized version of our toy example ($t_F = 10\ sec$, $\delta t = 1\ sec$). The time step of $\delta t = 1\ sec$ was chosen in order for the graphics to be clear. Decreasing the time step does not change the results in any way. The system still spends almost all of the time in a single state and only 1 time step in each of the other "required" states.

2.3.4 Maximum likelihood estimation of trajectories

The final inference trajectory approach we will discuss here is a polynomial time algorithm due to T. J. Perkins [27] that estimates the maximum likelihood trajectory for a given CTMC when the initial state is given. Unlike the approach in the previous section, the following solution does not simplify the problem by discretizing the time. Any CTMC trajectory $\mathbf{U}$ can be described by the sequence of its states and dwell times (see Section 2.2), i.e.

[3]The method is imprecise in the sense that a DTMC only approximate the underlying CTMC. Given the DTMC, the most-probable path computation due to the described method is exact.

$\mathbf{U} = (X_0, t_0, X_1, t_1, \ldots, X_{k-1}, t_{k-1}, X_k)$. Further, denote $\mathbf{U}_t$ to be a random variable whose values are trajectories $\mathbf{U}$. Then the likelihood of a sequence $\mathbf{U}$ that starts at X_0 and ends at time t is given by:

$$l(\mathbf{U}_t = \mathbf{U} \mid X_0) = (\prod_{i=0}^{k-1} \lambda_{X_i} e^{-\lambda_{X_i} t_i} T_{X_i, X_{i+1}})(e^{-\lambda_{X_k}(t - \sum_i t_i)} \times \mathbf{1}_{\{X(0)=X_0 \ AND \ \sum_{i=0}^{k-1} t_i < t\}}. \tag{2.22}$$

Then finding the most likely trajectory is equivalent to finding:

$$\arg\max_{\mathbf{U}} l(\mathbf{U}_t = \mathbf{U} \mid X_0 = X) \tag{2.23}$$

To compute (2.23), we proceed in two steps. First, assuming that we know what the state sequence of the most likely trajectory is, we derive the expression for the maximum likelihood waiting times, t_i. Then we use dynamic programming to actually find such a state sequence. The two steps are given in order of derivation of the solution, not in order in the algorithmic sense.

To find the likelihood of the optimal trajectory (in the maximum likelihood sense) when the exact path (i.e. state sequence) is given, we need to optimize Eqn. 2.22 for the dwell times t_i's. The solution turns out to be of the form $(\prod_{i=0}^{k-1} \lambda_{X_i} T_{X_i, X_{i+1}}) e^{-(\min_{i=0}^{k} \lambda_{X_i}) t}$, where k is the number of states in the state sequence (see [27] for details).

For the second step, the author defines the quantity $F_k(X, \lambda)$ to be the maximum achievable $l(\mathbf{U}_t = \mathbf{U} \mid X_0)$, where $\mathbf{U}$ has k state transitions, ends

with state X and has the smallest dwell parameter λ. For the trivial sequence (X_0), the quantity $F_0(X_0, \lambda_{X_0}) = e^{-t\lambda_{X_0}}$. For longer sequences we proceed recursively:

$$F_{k+1}(X, \lambda) = \max_{X' \neq X, \lambda' \in G(X,\lambda)} F_k(X', \lambda')\lambda_{X'} T_{X',X} e^{-t(\lambda - \lambda')}, \tag{2.24}$$

where $G(X, \lambda) = \{\lambda\}$, if $\lambda < \lambda_X$, $\{\lambda_{X'} : \lambda_{X'} \geq \lambda\}$, if $\lambda = \lambda_X$, and $\{\emptyset\}$ otherwise.

Thus, to compute (2.23) we apply dynamic programming to Eqn. 2.24, very similarly to how it was done in Section 2.3.3. Note that if we also know the final state of the system, X_{final} (i.e. if we have a boundary value problem), the problem of finding the maximum likelihood trajectory is the same except for we only restrict our attention to the sequences that end with X_{final}.

While the method is attractive, it turns out [27] that the algorithm produces substantially the same solution as the time discretization approach with sufficiently small time step. The clear advantage of this approach, however, is that one does not need to decide which time step to use, thus trying to compromise between the risk of collapsing a set of transitions into a single "pseudo-transition" (when the time step is too long compared to the system's fastest time scale) and the risk of running into computational difficulties due to increasing running time complexity.

2.3.5 Summary

A continuous-time Markov chain is a mathematical model for a very general class of random systems. Therefore, being able to predict how the system will evolve (or how the system has evolved) is an important problem. This section gave an overview of the tools currently being used for that problem. Unfortunately non of the available techniques correctly solve the problem of determining the most probable state sequences. The continuous-time Chapman-Kolmogorov Equation (in particular, the Kolmogorov forward equation) can be used to exactly calculate the probability distribution of states at any point in the future, given initial conditions. This provides a rather straightforward way of ranking most likely states at any point in the future. What Chapman-Kolmogorov equation does not provide us with is any information regarding either the system's trajectory or its path (state sequence).

The other three approaches deal with the trajectories. The simulation method is attractive because it is quite easy to implement, it is straightforward, and it provides statistically correct trajectories. However, trajectories are not paths. In continuous time one path includes uncountably many trajectories. If our time frame is large compared to the system's time scale, the number of simulations required for a meaningful statistical analysis can be prohibitively large. The other two approaches (time discretization and maximum likelihood trajectory) use dynamic programming to find the most likely

trajectory. Again, a trajectory is not a path and we may only care about the exact order of transitions; "when" may be irrelevant. Both approaches produce similar results that may be unrepresentative of the system's actual behaviour. In general, a CTMC's most likely trajectory for a given path will dwell in its highest dwell time state for the entire time, and only "pass" through the other states instantaneously [27]. As we will demonstrate in our HIV example (see Section 4.3), time disecretization will inevitably simplify system's dynamics leading to inaccurate solutions.

In practice it can be important to know the path. For instance, we can imagine a setting, in which the system can arrive to the same state through different paths, with different implications for the system's environment. Later we demonstrate two such examples. In the first example we have a system in which states are different strains of a virus that infects a host organism. Needless to say, different strains can have very different effects on the host organism, so that which states the system will most likely to visit, and in which order, can be just as important as knowing the final state. If a physician were to prescribe a certain regimen to a patient for some time period, it may be important to be able to predict which "states" the patient will be going through, not just the state at the end of that time period.

The other example we provide to demonstrate the importance of path-wise information is from the reliability domain. If our system is a fault-

tolerant machine, that can be in different states, then knowing the path that the system goes through may be important for diagnosing faults. If we known that the system is not functional, and we know its initial state (say, all components working fine), we may want to know the most likely sequence of malfunctions that had taken place before the system broke down.

The algorithm presented in the following chapter finds the most probable path of a continuous-time Markov chain given the initial state and the final time. To the best of our knowledge, no previous algorithm solves the problem as we formulate it. It should be an important addition to the toolkit of methods for analyzing CTMCs that are available to modellers and researchers. Optionally, if we have it, the information about the final state and/or the exact time of the final transition can be used to better estimate the most likely path. The algorithm is also easily generalizable to find the top k most likely sequences, where k is an arbitrary positive integer. We will demonstrate the utility of our approach through two different applications: ranking viral mutational pathways, and predicting the most probable failure sequences in complex fault-tolerant systems.

Chapter 3

Predicting the most probable paths

This chapter presents the solution to our main problem of computing the most probable sequence of states in a continuous-time Markov chain. First we formally define our problem. Then we present some results about the nature of the temporal probability functions of CTMC state sequences. This is necessary to understanding the algorithm, which is explained in the three section that follow. Finally, we conclude the chapter by demonstrating the algorithm with the help of our hypothetical "toy" example.

3.1 Problem definition

The problem that we solve here is how to compute the most probable sequence of states that a continuous-time Markov chain visits. In order for the problem to be well-defined, we need a minimum of two pieces of information: the initial state X_0 and the final time t_F. The initial state X_0 defines the state the system had started from, at time $t = 0$. The final time t_F is the total time that we let our system evolve. Optionally, we may have the information as to the system's final state X_F, i.e. the state our system ends up in at t_F.

More formally, a state sequence can be written as S=(X_0, ..., X_N). Let $P_t(S)$ be the probability of such sequence. It depends on all of the transitions (i.e. $X_0, X_1, \ldots, X_{N-1}$, to X_N) and no others having occurred within time t. Letting $\tau_0, \tau_1, \ldots, \tau_k$ be amounts of time the system spends in the corresponding states – the dwell times – then we have

$$P_t(S) = \left(\prod_{i=0}^{k-1} T_{X_i X_{i+1}} \right) P \left(\sum_{i=0}^{k-1} \tau_i \leq t \text{ and } \sum_{i=0}^{k} \tau_i > t \right) \tag{3.1}$$

The formula intuitively makes sense. The product of transitions is time independent by the memorylessness of Markov chains. The second, time-dependent part just says that all the specified transitions and no others must occur within time t. The first part is affected by the transition probabilities T_{X_i,X_j}, while the second part implicitly incorporates λ's the dwell time

parameters.

The problem we address in this work is: given some time $t_F > 0$, find:

$$S^* = \arg\max_S \left\{ \left(\prod_{i=0}^{k-1} T_{X_i X_{i+1}} \right) P \left(\sum_{i=0}^{k-1} \tau_i \leq t_F \text{ and } \sum_{i=0}^{k} \tau_i > t_F \right) \right\} \quad (3.2)$$

The solution does not have to be unique, and as will be shown shortly, our approach will find all solutions satisfying (3.2). A more general problem is to find the top k most probable state sequences for arbitrary number k. Our solution is in fact easily generalizable and the approach that we propose can be used to solve this more general problem as well.

3.2 Preliminaries

The problem we are trying to solve is to find the most probable sequence of states that a continuous-time Markov chain goes through, given the initial state and final time. That is equivalent to solving (3.2). Our exact solution is a functional dynamic program that recursively computes state sequence extensions from already computed ones. To derive this dynamic program, we first define the temporal likelihood function $L_t(S)$ of a sequence S. Given a state sequence S, $L_t(S)$ can be defined as the likelihood of the sequence S occurring, with the arrival to its last state at precisely time t. When the state sequence only contains one element, we can define this as $L_t(S) = \delta_{t=0}$, where δ is the Dirac delta function. Now, let $S = (X_0, \ldots, X_k)$ for $k \geq 0$

be any state sequence, and consider a one-step extension of that sequence $S' = (X_0, \ldots, X_k, X_{k+1})$. For the chain to arrive at the last state of S' at precisely time t, it must first arrive at the last state of S at some time $\tau < t$, stay in state X_k for time $t - \tau$, and then transition to state X_{k+1} and so

$$L_t(S') = \int_{\tau=0}^{t} L_\tau(S) \lambda_{X_k} e^{-\lambda_{X_k}(t-\tau)} T_{X_k X_{k+1}} d\tau \tag{3.3}$$

The probability of the sequence S can also be related to the likelihood. For the chain to have transited S by time t, but no additional states, it must have arrived to the final state in S by some time $\tau < t$ and then remained there for at least time $t - \tau$. Thus,

$$P_t(S) = \int_{\tau=0}^{t} L_\tau(S) e^{-\lambda_{X_k}(t-\tau)} d\tau \tag{3.4}$$

combining these two equations, we observe $L_t(S') = \lambda_{X_k} T_{X_k X_{k+1}} P_t(S)$. Using this and applying Eqn. 3.4 to S', we obtain a recursive integral equation for state sequence probabilities.

$$P_t(S') = \lambda_{X_k} T_{X_k X_{k+1}} \int_{\tau=0}^{t} P_t(S) e^{-\lambda_{X_{k+1}}(t-\tau)} d\tau \tag{3.5}$$

The integral equation 3.5, together with

$$P_t(X_0) = e^{-\lambda_{X_0} t} \tag{3.6}$$

allows us to compute $P_t(S)$ for any legal trajectory S. Details on how this is done are discussed in Section 3.4.

To see what these probability functions look like let's go back to our toy example (see Section 2.2). Figure 3.1 shows a few state sequences. Until about 8 seconds the trivial trajectory (1) is the most likely path out of the five paths shown. Then, up until about second 19, the sequence (1, 3, 1) is the most likely one out of them, and so on.

Besides being able to compute the probability functions for different state sequences, we also need to be able to compare these sequences. Unfortunately it is not enough to only compare the final time probability value $P_{t_F}(S)$. Since our algorithm builds up on previous solutions, it is possible to have suboptimal sequences that can be extended to optimal ones and hence should not be viewed as unimportant. We introduce the notion of *dominance* to compare different trajectories. If S_1 and S_2 are two different state sequences beginning at X_0 and ending at the same state, we say that S_1 dominates S_2 on the interval $(0, T]$ if $P_t(S_1) > P_t(S_2)$ for all $t \in (0, T]$. If S is not dominated by any other sequence then we call it *non-dominated*. In our toy example (Fig. 3.1), the state sequence (1, 3, 4, 5, 6) dominates (1, 2, 3, 4, 5, 6) on the interval $(0, 10\ sec]$, since the graph of the probability function for the first sequence is strictly above that for the second sequence. On the other hand, even though the final probability (at $t = 60\ sec$) of state sequence (1, 3, 1) is greater than the final probability of (1), (1, 3, 1) does *not* dominate (1) on the entire interval (the probabilities of (1) are higher than the probabilities

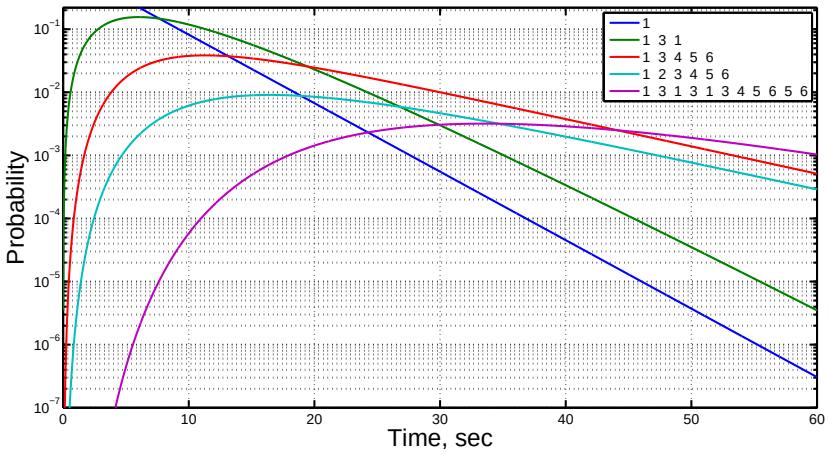

Figure 3.1: The temporal probability functions for a few selected state sequences for our toy example. The y-scale is logarithmic, so linearities on the graph corresponds to exponential decays. The trivial state sequence (1) just decays for the entire simulation period. All other sequences have unimodal probability functions that peak and then decay exponentially. It is also possible for a sequence to monotonically increase, asymptotically converging to a finite value. This happens exactly when the final state has average dwell time ∞ ($\lambda_{X_{final}} = 0$). See Section 4.3 for an example.

of (1, 3, 1) up until $t = 7.66\ sec$). Obviously, the solution to Eqn. 3.2 must only contain non-dominated sequences.

As something of an aside, there are only three different types of qualitative behaviours for such functions. The first one is the "no transition" behaviour, where the sequence consists of only its initial state. In this case the probability function is given by Eqn. 3.6 and the probability of the sequence just decays exponentially. Another non-standard case is when the final state happens to be absorbing. Here, because the dwell parameter λ is 0, the probability function asymptotically approaches a finite value. This can be easily shown to be true since if we take $\lambda_{k+1} = 0$ in Eqn. 3.5, then:

$$\begin{aligned} P_t(S') &= \lambda_{X_k} T_{X_k X_{k+1}} \int_{\tau=0}^{t} P_t(S) e^{-\lambda_{X_{k+1}}(t-\tau)} d\tau \\ &= \lambda_{X_k} T_{X_k X_{k+1}} \int_{\tau=0}^{t} P_t(S) e^{0\times(t-\tau)} d\tau \\ &= \lambda_{X_k} T_{X_k X_{k+1}} \int_{\tau=0}^{t} P_t(S) d\tau \end{aligned} \tag{3.7}$$

Since the probability function $P_t(S)$ has to be non-negative, its integral must be non-decreasing. A constant multiple of a non-decreasing function is clearly non-decreasing and since any probability function is bounded above (by 1) it is obvious that expression (3.7) must be a monotonically increasing function that asymptotically converges to a finite value.

In a more common case when the final state is not a sink, λ_{k+1} of Eqn. 3.5 is not 0. Then the qualitative behaviour of $P_t(S')$ is that it is always

decaying towards the value of $P_t(S)$. As long as $P_t(S)$ is increasing, $P_t(S')$ will also increase, "chasing" $P_t(S)$ from below. X_k is non-absorbing since $T_{k,k+1}$ must be non-zero and so $P_t(S)$ must eventually start declining asymptotically approaching zero. It would have to cross $P_t(S')$ which will be rising up to that point. Following the crossing $P_t(S)$ will have to start declining as well also approaching zero asymptotically.

Now we are ready to introduce the algorithm.

3.3 The algorithm

The idea behind the algorithm is to use dynamic programming to enumerate all non-dominated sequences, and then check each one and report the ones with the highest probability value at t_F.

1. Initialize a list A to contain the sequence $S_0 = (X_0)$.
2. Initialize a list B to be empty.
3. As long as list A is not empty
    4. Remove the first state sequence S from A.
    5. Compute $P_t(S)$ for $t \in [0, t_F]$.
    6. If S is not dominated by any sequence on list B

7. Add S to list B.

8. Remove from B any sequences dominated by S.

9. Add all single-step extensions of S to list A.

10. Sort through list B to find $\arg\max_S(P_{t_F}(S))$.

The program starts by computing $P_t(X_0)$. Then it produces all possible single state extensions of (X_0) and enqueue them. Up until the queue becomes empty we test the sequences that it contains for dominance, remove the ones that are dominated and extend the ones that are not, enqueueing them for future processing.

Provided that we are dealing with a finite state space, the above algorithm converges because the program must eventually terminate, since there are only finitely many non-dominated sequences, and hence the process must eventually exhaust itself and halt. This can be established by contradiction, i.e. suppose the assertion is not true and there are in fact infinitely many non-dominated sequences. Let's enumerate them S_1, S_2, etc. and let N_1, N_2, etc. be their lengths. Since we are assuming that there are infinitely many such sequences, $\lim_{k\to\infty} N_k$ must diverge to ∞. The probability of any

sequence S_k has to be bounded above[1] by $(1 - e^{\lambda_{max} t})^{N_k}$, where λ_{max} is the largest dwell time parameter of our CTMC. Since this quantity converges to zero pointwise as k increases, it will be strictly less than $P_t(X_0)$ for all $t \in [0, t_F]$ – contradicting the assumption that S_k is not dominated. Thus it must be the case that there are only finitely many non-dominated sequences.

3.4 Details of implementation

Although a closed form expression the integral equation (3.5) can be found, evaluating it can be computationally expensive and numerically unstable (for many realistic CTMCs computing some of the coefficients in this expression often requires dividing small numbers by very large numbers) [35]. In practice $P_t(S)$ can be computed much more efficiently numerically.[2] In order to do this we need to turn integral equation (3.5) into a differential equation. This approach is not only more efficient, it also allows to make testing for

[1]The reason for this upper bound is the following. If τ_i is the time the system spends in state i then obviously for any state sequence S, $P(S) < \prod_{i=0}^{N-1} P(\tau_i < t)$. Further, $P(\tau_i < t) = 1 - e^{\lambda_i t} \leq 1 - e^{\lambda_{max} t}$. So, if N_k is the length of S, then $P_t(S) \leq (1 - e^{\lambda_{max} t})^{N_k}$, where $\lambda_{max} = \max_i \lambda_i$.

[2]Even though we introduce numerical integration in our implementation, we claim that the solution is still exact. The equations being integrated are linear and in no way stiff, so given appropriate choice of parameters, such as tolerance and integration time step, the produced ranking of state sequences should be correct.

dominance trivial. The time dependent probability functions of all trajectories will thus be represented by vectors of those probability values that correspond to a grid of predetermined time points. If we by design make the grids match for every trajectory then computing dominance relationship will only involve comparisons of probability values at corresponding grid points.

There are a few ways to convert Eqn. 3.5 into a differential equation. Probably the most straightforward approach is by using the Laplace transform. Applying it to both sides yields:

$$\begin{aligned}\mathcal{L}\{P_t(S')\} &= \lambda_{X_k} T_{X_k,X_{k+1}} \mathcal{L}\left\{\int_{\tau=0}^{t} P_t(S) e^{-\lambda_{State_{k+1}}(t-\tau)} d\tau\right\} \\ &= \frac{\lambda_{X_k} T_{X_k,X_{k+1}}}{s+\lambda_{X_{k+1}}} \times \mathcal{L}\{P_t(S)\}\end{aligned}$$

Rearranging the terms, we get:

$$\begin{aligned}(s+\lambda_{X_{k+1}})\mathcal{L}\{P_t(S')\} &= \lambda_{X_k} T_{X_{k,k+1}} \mathcal{L}\{P_t(S)\} \\ s\mathcal{L}\{P_t(S')\} &= -\lambda_{X_{k+1}} \mathcal{L}\{P_t(S')\} \\ &\quad +\lambda_{X_k} T_{X_k,X_{k+1}} \mathcal{L}\{P_t(S)\} \qquad (3.8)\end{aligned}$$

Now, taking the inverse Laplace transform of (3.8), we get:

$$\begin{aligned}\mathcal{L}^{-1}\{s\mathcal{L}\{P_t(S')\}\} &= -\lambda_{X_{k+1}} \mathcal{L}^{-1}\{\mathcal{L}\{P_t(S')\}\} \\ &\quad +\lambda_{X_k} T_{X_k,X_{k+1}} \mathcal{L}^{-1}\{\mathcal{L}\{P_t(S)\}\} \\ \frac{dP_t(S')}{dt} &= -\lambda_{X_{k+1}} P_t(S') + \lambda_{X_k} T_{X_k,X_{k+1}} P_t(S) \qquad (3.9)\end{aligned}$$

Equation 3.9 is the linear differential equation that we use to recursively evaluate $P_t(S)$ for non-trivial trajectories. For a single state sequence $S = (X_0)$, the probability function is defined by (3.6), and it can be easily computed directly. For longer sequences, however, computing $P_t(S)$ is generally impractical and numerically computing (3.9) is a much better option [35]. Because S' is a single state extension of S, in order to compute the probability function of any sequence of length > 1 the probability functions of all of its ancestor sequences must be computed as well.

In our MATLAB implementation we use the built-in *ode45* function (Runge-Kutta 4-5 formula [41]) for numerically integrating (3.9). The evaluated function is then stored as a vector of probability values that later can be easily compared to the values of a vector for another function in order to test for dominance. S_1 is dominated by S_2 if all non-zero time values of the S_2 vector are greater than the corresponding values of S_1 vector. Recall that dominance of sequences (defined for sequences that start at the same state and end at the same state) is a relationship used to sort through sequences. Being non-dominated is a necessary condition of any solution that satisfies (3.2).

3.5 Remarks

The above algorithm efficiently computes the exact solution to our problem, as formalized by Eqn. 3.2. In fact it can be easily extended to answer the more general question of finding top k most likely trajectories. All that needs be changed in the algorithm is step 6 and step 10. In step 6, we need to retain S even if it is dominated, unless it is dominated by at least k other sequences, i.e. the maximum number of sequences that can dominate S now is $k-1$ ($k=1$ is then just a special case). Accordingly, in step 10, we would need to report the top k sequences, not just 1. Computing several most likely state sequences will be demonstrated in the following section with our toy example.

Another point that should be made is that sometimes we know *exactly* when the last transition occurred, i.e. our t_F is not just some arbitrary time point in the future, but is in fact the point where the last state of S transitions into something else. Can we take that information into account to better estimate the most likely state sequence? Yes, we can. In this case instead of looking at the *probability* that our trajectory contains a certain state sequence, we should be looking at the *likelihood* that our trajectory contains that state sequence *and* that its last transition occur at exactly time t_F. In order to do that, besides the $P_t(S)$ values the algorithm should also store the values of $L_t(S)$, the likelihood function. We can obtain the value of $L_t(S)$ by

noting that the temporal probability function of the state sequence can be written in terms of the time that the last state is entered and the probability of not leaving the last state as follows.

$$P_t(S) = \int_{\tau=0}^{t} L_\tau(S) e^{-\lambda_{X_k}(t-\tau)} d\tau \tag{3.10}$$

So if $S' = (X_0, X_1, \ldots, X_k, X_{k+1})$ is a one-transition extension of S, then the likelihood of the longer state sequence, S', can also be written in terms of the likelihood of S.

$$L_t(S') = \int_{\tau=0}^{t} L_\tau(S) T_{X_k X_{k+1}} \lambda_{X_k} e^{-\lambda_{X_k}(t-\tau)} d\tau \tag{3.11}$$

The justification is that the system transitions to state X_k at time τ, which is a random variable, then dwells there for time $t - \tau$ before transitioning to state X_{k+1}. Combining these previous two equations, we observe that

$$L_t(S') = \lambda_{X_k} T_{X_k X_{k+1}} P_t(S) \ . \tag{3.12}$$

So in order to compute the likelihood of a sequence S', we need to multiply the probability $P_t(S)$ of its parent sequence S by constant constant $\lambda_{X_k} T_{X_k X_{k+1}}$, where λ_{X_k} is the dwell time parameter of the last state of S and $T_{X_k X_{k+1}}$ is the transition probability from that state to the end state of S'.

Finally, it should be noted that an exact analytical solution to 3.1 can be obtained if desired. This is however computationally more difficult and less efficient than the numerical integration approach described here [35]. From Eqn. 3.8 it follows that:

$$\mathcal{L}\{P_t(S')\} = \frac{\lambda_{X_k} T_{X_{k,k+1}}}{(s+\lambda_{X_{k+1}})}\mathcal{L}\{P_t(S)\}.$$

Also, $P_t(S=(X_0)) = e^{-\lambda_{X_0}t}$, so that $\mathcal{L}\{P_t(S=(X_0))\} = \frac{1}{(s+\lambda_{X_0})}$. We can denote $S_k = (X_0, X_1, \ldots, X_k)$. Then,

$$P_t(S_1) = \mathcal{L}^{-1}\{\frac{\lambda_{X_0} T_{X_{0,1}}}{(s+\lambda_{X_1})} \times \frac{1}{(s+\lambda_{X_0})}\}.$$

Similarly,

$$P_t(S_2) = \mathcal{L}^{-1}\{\frac{\lambda_{X_1} T_{X_{1,2}}}{(s+\lambda_{X_2})} \times \frac{\lambda_{X_0} T_{X_{0,1}}}{(s+\lambda_{X_1})} \times \frac{1}{s+\lambda_{X_0}}\}.$$

Proceeding by induction, we get the general result for an arbitrary length state sequence:

$$\begin{aligned} P_t(S_n) &= \mathcal{L}^{-1}\{\frac{1}{s+\lambda_{X_0}} \times \frac{\lambda_{X_0} T_{X_{0,1}}}{(s+\lambda_{X_1})} \times \ldots \times \frac{\lambda_{X_n} T_{X_{n-1,n}}}{(s+\lambda_{X_n})}\} \\ &= \prod_{i=0}^{n-1} \lambda_{X_i} T_{X_i, Xi+1} \times \mathcal{L}^{-1}\left\{\prod_{j=1}^{N} \frac{1}{(s+\lambda_{i+1})^{m_j}}\right\}, \end{aligned} \tag{3.13}$$

where N is the total number of states in the system, and each m_j equals the multiplicity of state j in S_n (i.e. how many times S_n visits state j). Being a rational fraction, the above inverse Laplace transform can be solved symbolically to obtain the analytical solution [3, 47]. The general solution will always be of the form:

$$P_t(S_n) = \sum_{i=0}^{n} \sum_{j=0}^{m_i-1} c_{ij} t^j e^{-\lambda_{X_i} t}, \tag{3.14}$$

where c_{ij} are constants that only depend on CTMC parameters T and λ. Unfortunately, as was mentioned earlier, for many realistic CTMC parameters, the coefficients c_{ij} are difficult to compute. Evaluating the inverse Laplace transforms often requires dividing small numbers, that have the order of magnitude of the dwell parameters, by much larger numbers, whose orders of magnitude are often those of the factorials of the numbers of transitions in state sequences. Divisions like that tend to produce poor precision results due to the issues with floating point arithmetics. Hence, evaluating the closed form expression (3.14) is only feasible for relatively short sequences.

3.6 Example

To demonstrate, we can use our toy example (originally introduced in Section 2.2). Fig. 3.2 plots the top 7 most likely state sequences, as determined by the algorithm. The top figure shows the top 7 most likely paths for 10 second simulation period, and the bottom figure shows the top 7 most likely paths for 1 minute simulation period. For the 10 second period we can clearly see that up until about 7.66 seconds (1) dominates all other sequences. Then (1, 3, 1) takes over and remains the highest probability sequence up until $t_{final} = 10$ *sec.* Neither (1), nor (1, 3, 1), however make it to the top 7 list for 1 minute simulation. Instead the list contains much longer sequences,

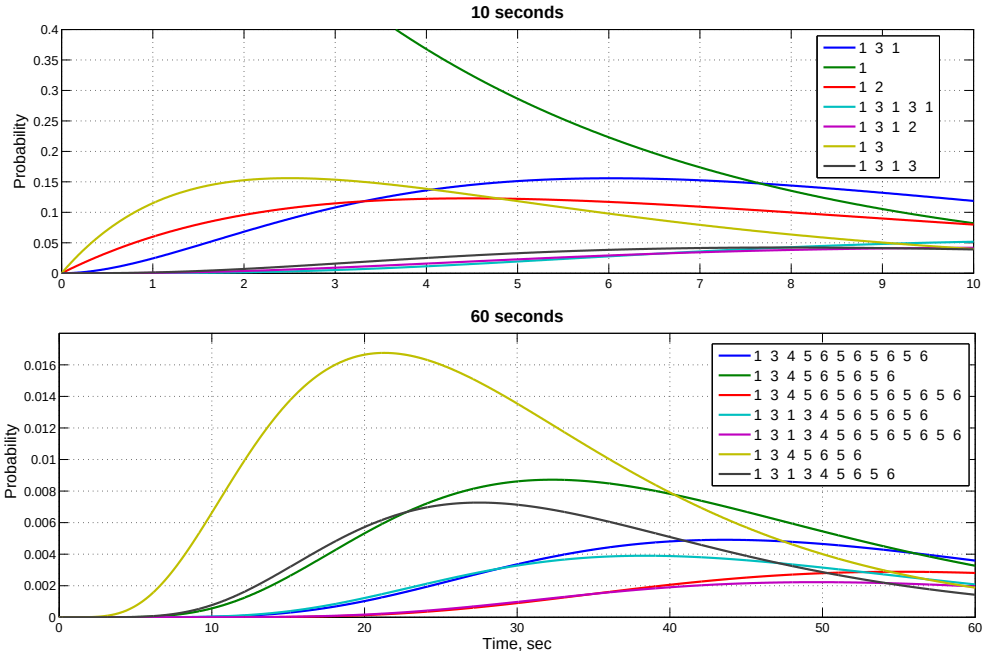

Figure 3.2: Top 7 most probable state sequences for $t_{final} = 10$ *sec* (top figure) and $t_{final} = 60$ *sec* (bottom figure). None of the paths from the top 7 most likely paths for 10 seconds feature in the top 7 most likely paths for 60 seconds and vice versa.

none of which appear in our top plot. Notice how different this is from the completely unrealistic scenarios produced by the standard discretization approach (Fig. 2.2).

Chapter 4

Application 1: HIV drug resistance

The first real application of our algorithm that we present is inferring the mutational pathways of HIV virus in patients that are undergoing antiviral treatment. First, we give some background information and the context for our problem. Next, we explain how our HIV model is estimated. Then we analyze the model using our algorithm and the more standard time-discretization approach. It should become very clear to the reader why our approach is superior. Finally, we conclude the chapter with some remarks about a previous study that used the same data set to investigate the high probability mutational pathways. However, just like the time discretization approach, their methods do not provide the exact solution to the problem.

The goal of this chapter is to demonstrate how state sequence analysis can be used in phylogenetics, and to show that the correct solutions produced by the algorithm may be different from approximate solutions of the discretization approach.

4.1 Background

Although antiviral therapies have become tremendously more successful than they had been in the past, some of the most serious infectious deceases caused by viruses, such as HIV-1, are still far from being eradicated. One of the main challenges in treating HIV patients is the patients' acquisition of drug resistance [10, 16].

The reason HIV virus is so powerful is its rapid reproduction rate and its unusually high mutation rate [11]. The half-life of an infected cell is remarkably short. This has driven the HIV virus to evolve towards higher than usual reproduction rates which in turn will ensure that there is always an enormous population of virus particles in the host body. Although blood of an infected patient will normally only contain about 2% of the total viral load, the high viral load patients can contain over 10,000 virus particles/mL. This means that a patient with 5 liters of blood will have over 50 million particles in his or her blood and 2.5 billion particles in their entire body (including lymph

nodes, spleen, brain, etc.). The average number for a patient is estimated to be between 10^7 and 10^8 copies [11, 16].

The process of HIV reverse transcription (transcribing DNA from RNA templates) is notoriously error-prone, hence the much higher than usual mutation rate, with an average of one mutation per each genome transcribed. This means that at any time patient's cells will contain an enormous number of quasispecies. By simple Darwinian logic, then, sooner or later some quasispecies will start exhibiting selective advantage over others (that the drugs are targeting) and consequently will start taking over in terms of population numbers.

The detailed mechanisms of drug resistance can be very technical and the reader is referred to some excellent review articles [10, 11, 16]. For the purpose of our work, the only important aspect to note is that there are certain sites on viral amino-chains that can cause the virus to behave quite differently in terms of drug resistance if substituted to non-wild type amino acids. That is, if we, for example, imagine HIV's RT enzyme (an important protein that our example drug will be targeting) as a sequence of 200-something amino-acids, then certain substitutions in certain parts of the sequence would yield higher (or lower) resistance to certain drugs.

4.2 Estimating the model

For our HIV example we used a dataset deposited in Genbank [6], a publicly available collection of DNA sequences, under accession numbers AY000001 to AY003708. The sequences contain 984bp from the HIV-1 pol gene that were obtained from patients in phase II clinical studies (DMP 266-003, DMP 266-004 and DMP 266-005) of Efavirenz combination therapy [4]. The studies preferentially selected patients failing Efavirenz combination therapy, thus, the model is best thought of as representing mutation dynamics among patients likely to eventually acquire resistance.

More about the dataset can be found in [4]. As Efavirenz is a reverse-transcriptase (RT) inhibitor, we only focused on the RT sections of the sequences.

To have a reasonably sized model we only included important mutations. To guide us in this process we used the HIV Drug Resistance database and the descriptive paper that accompanied the dataset [4]. The final model contains the following mutations: L100I, K101Q/E, K103N, V106M, Y188L/H, G190S/E/A, and P225H. The notation VWXYZ for a mutation indicates that at site WXY the "wild type" amino acid (i.e. the standard one, usually the one most commonly observed) V has been replace with amino acid Z. Including the wild type, we have $4 \times 32 \times 2^4 = 576$ possible mutational states. However, most of them did not occur in our data and hence did not make it

to the final model.

To estimate a continuous-time Markov chain model from the data, we first restricted attention to the 122 patients who had measurements at more than one time point. At some time points, multiple distinct HIV genotypes are present in a patient; we take the most common variant as representative of the state of the patient's HIV at that time. The states of our CTMC are combinations of the selected mutations that occurred in the 122 patients. 22 such states were identified, however 3 of them were not observed to lead to or from any other states, and they were excluded from the model. The final model contains 19 states: wild type; G190S; G190E; G190A; Y188L; Y188L +G190E; K103N; K103N +P225H; K103N +G190A; K103N +Y188H; 103N +V108I; K103N +V106M; K101E; K101E +G190S; K101Q; K101Q +G190S; K101Q +K103N; L100I +K103N; L100I +K103N +P225H. Thus the model can be represented by a 19×19 transition rate matrix.

Following previous analysis of the same data set [9], we estimated the transitions rate of our continuous-time Markov chain (CTMC) using a method due to Albert [1]. The idea behind this approach is that if $q(i, j)$ is the instantaneous transition rate from state i to state j, then it can be estimated as $\frac{N(i,j)}{A(i)}$, where $N(i, j)$ is the number of observed transitions from i into j, and $A(i)$ is the total time spent in state i.

More precisely, suppose that at time t_1 state X_1 is observed in patient X,

and at time t_2, state X_2 is observed. We make the simplifying assumption that the transition from X_1 to X_2 was a direct transition, and that the transition occurred at time t_2 (hence, that pair of observations attributes $t_2 - t_1$ waiting time to state X_1). Once we obtain the Q-matrix, it is trivial to obtain parameters T and λ's (see Eqn. 2.9) required for our algorithm.

The state space (Fig. 4.1) in our model is composed of 19 combinations of mutations including the wild type. Models like these depend on the dataset used to build them. If a certain state transition was not present in the dataset, the corresponding state transition probability in the model will be 0. Similarly, if a certain state was not observed to mutate into a different state, it would be considered a sink (or absorbing state), i.e. a state whose average dwell time is ∞. Note that we used one many possible methods to estimate CTMC rates [1], and it is possible to use other methods instead, possibly accounting for prior distribution of the rates. Different estimation methods may yield different models, however we did not investigate this since it falls outside the scope of this research. In our system the sinks are G190E, Y188L, K103N +Y188H, K103N +V106M, K101E +G190S, K101Q +G190S, L100I +K103N +P225H. The probabilities of each of these states approach some finite values as $t \to \infty$. All other states have non-zero transition probability rates to some other states.

4.3 Analysis

Our new approach to analyzing CTMCs allows us to reconstruct mutational pathways of HIV drug resistance to Efavirenz. Our CTMC is represented by combinations of selected mutations of the virus. Thus when we talk about the system transitioning from one state to another we are talking about the virus evolving from one quasispecies to another by either dropping or acquiring certain mutations. It is important to understand that we are at the dynamics within patient, as opposed to some general viral mutation patterns we might be interested in an epidemiological context. The problem we are trying to solve is: given that the patient acquired wild type HIV-1 strain at time 0, and has been undergoing Efavirenz combination therapy for time t_F, what are the most likely mutational pathways that the virus has taken assuming no additional infection incidents before t_{final}?

Our algorithm can naturally address the problem of reconstructing mutational pathway of HIV virus. The equivalent CTMC formulation of our problem would be: given that the system starts in state "wild type" and evolves for time t_{final}, what are the most likely sequences of state transitions? It turns out that the answer to this question is very time-dependent.

It can be inferred from the graphs (Fig. 4.2), that for the first 6 months or so it is most likely that the patient will not acquire any additional significant mutations. However, after approximately 6 months it becomes more probable

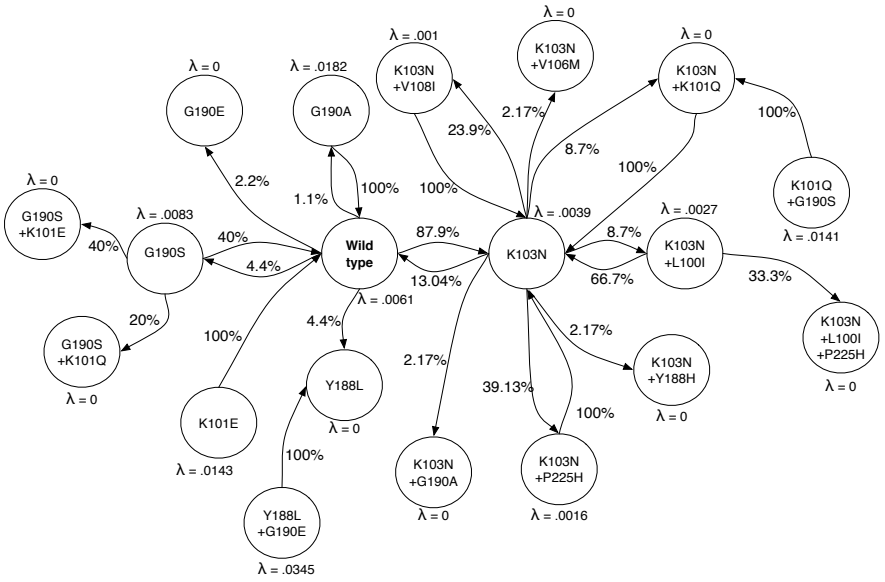

Figure 4.1: State transition diagram of our Efavirenz resistance HIV model. It is estimated based on time-series HIV genotype observations [4]. It is customary to visualize CTMCs as such state diagrams. States are labelled with mutations they include: L100I, K101Q/E, K103N, V106M, Y188L/H, G190S/E/A, and/or P225H, where the notation VWXYZ indicates that the wild-type amino acid V at position WXY in the reverse transcriptase gene is replaced by amino acid Z. Each state has its corresponding dwell parameter λ and each possible transition is represented by an arrow with a percentage value that corresponds to the transition probability. The outgoing transition probabilities for each state should sum to 100% (probability 1). The dwell times, λ's have units $days^{-1}$. States with $\lambda = 0$ are sinks. Once the system visits a sink state it cannot get out since the dwell time in a sink state is ∞. Not all states in our model are accessible from the wild type state.

that the patient's HIV will have acquired mutation K103N. In an array of studies K103N has been shown to be the single most common mutation in patients failing Efavirenz-containing treatment regimen ([4, 17, 39, 32]). It occurs in over 50% of patients failing Efavirenz combination therapy and is associated with about 25-fold reduction of Efavirenz susceptibility. Just short of two years, the most probable path in our model adds the mutation P225H, an "accessory" mutation that is known to occur almost exclusively in combination with K103N and is known to decrease Efavirenz susceptibility by more than 100-fold.

From four years on an alternative trajectory is more probable - the K103N mutation followed by acquisition of additional mutation V108I. V108I is another known "accessory" mutation associated with K103N. Both accessory mutations can subsequently be lost and regained, although in our model such paths are never more probable than some of the simpler mutation sequences. From approximately 5.5 years on, the most probable state sequence is wild type followed by the acquisition of Y188L. Y188L is considered to be a major Efavirenz-resistance mutation [32]. None of the patients in the dataset that was used appeared to lose it. Therefore the estimated CTMC model classified Y188L as a sink state in our system.

A little caveat here is in order. Like with any other maximum likelihood approach one must consider carefully how representative the maximum prob-

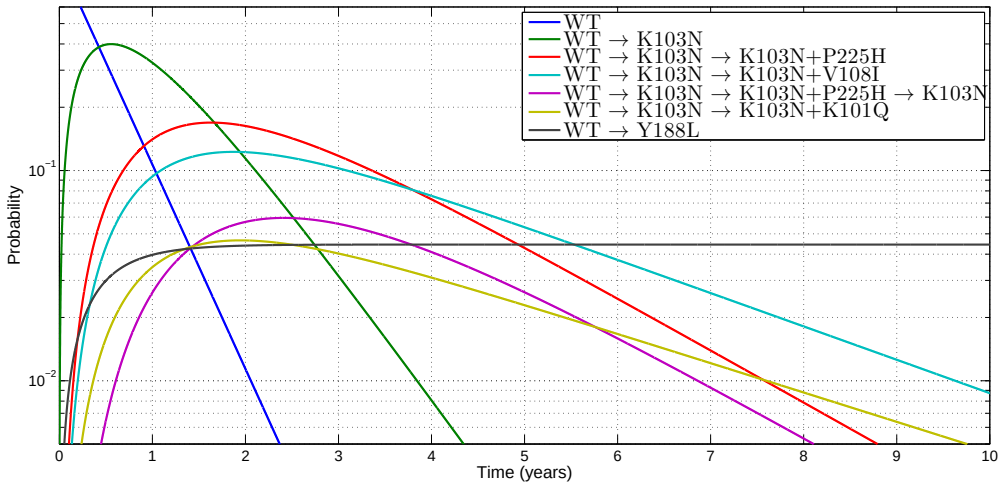

Figure 4.2: Probabilities of selected mutation sequences, starting from wild type (WT), as a function of time during the first 10 years of therapy. These include all of the most likely mutation sequences in the 10-year period. The black curve {WT $\rightarrow$ Y188L} is an example of a state sequence that ends with an absorbing state. Its probability approaches a finite value. All other probability functions in the list peak and then exponentially decay. The most probable sequence of mutations changes 5 times in a 10-year interval.

ability trajectory is. While for very short times the most probable {WT} trajectory has rather high probability values, the longer we run the simulation generally the smaller the value of the probability of most likely trajectory gets. In our example, the most likely sequnce at $t_F = 10\ years$ ({WT to Y188L}) has probability of less than 4.5%. In order to have a better picture, we need to look at the distribution of a larger portion of the total probability mass, by looking at top k, as opposed to just one top most probable state sequences. For example, in our HIV model the top 10 most probable sequences for $t_F = 10\ years$ have the total probability of 26.2%. For any particular application it should be determined how large the k needs to be.

It is interesting to compare our approach to the more standard time discretization approach (see Section 2.3.3) for estimating the most probable paths. Table 4.1 shows 3 different lists. The first list consists of correct most probable paths to all accessible states as estimated by our algorithm. The time frame chosen is ten years. The other two lists are the most probable discretized trajectories trajectories. One immediate observation is that the output of the discretization algorithm is quite sensitive to the time step chosen. While the system is likely to oscillate between states 7 and 8 (the enumeration is explained in the caption) for a 1 year time step, this oscillation disappears for discretization done with 1 day time step. This example demonstrates the main weakness of time discretization - if our system's dy-

namics evolve according to varying time frames, no single choice of a time step can capture the entire picture. While the 1 year discretization provides reasonable qualitative predictions of the correct paths for ending states 1, 2, 3, 4, 5, 7, 8, 11 and 18, it fails with states 9, 10, 12, 14, 16, 17 and 19. At the same time the 1 day discretization fails to capture the oscillations between 7 and 8 in any paths that exhibit it.

Correct path	Prob	1-Yr discretization trajectory	Prob	1-day discretization trajectory	Prob
1, 7, 8, 7, 8, 7, 1	3.78e-04	1, 8, 7, 8, 7, 8, 7, 8, 7, 1	2.19e-05	1, ..., 1 (1)	1.80e-10
1, 7, 8, 7, 8, 7, 1, 2	1.43e-05	1, 8, 7, 8, 7, 8, 7, 7, 1, 2	1.63e-06	1, ..., 1, 2 (1, 2)	4.93e-14
1, 3	2.22e-02	1, 1, 3, 3, 3, 3, 3, 3, 3, 3	1.54e-03	1, 3, ..., 3 (1, 3)	1.35e-04
1, 7, 8, 7, 8, 7, 1, 4	1.51e-06	1, 8, 7, 8, 7, 8, 7, 7, 1, 4	4.07e-07	1, ..., 1, 4 (1, 4)	1.23e-14
1, 5	4.44e-02	1, 1, 5, 5, 5, 5, 5, 5, 5, 5	3.08e-03	1, 5, ..., 5 (1, 5)	2.71e-04
1, 7, 8, 7, 8, 7	3.79e-03	1, 8, 7, 8, 7, 8, 7, 8, 7, 7	5.38e-05	1, 7, ..., 7 (1, 7)	3.80e-09
1, 7, 8, 7, 8, 7, 8	6.89e-03	1, 8, 7, 8, 7, 8, 7, 8, 7, 8	6.58e-05	1, 7, ..., 7, 8 (1, 7, 8)	5.78e-12
1, 7, 9	1.91e-02	1, 8, 7, 8, 7, 8, 7, 8, 7, 9	3.66e-06	1, 7, ..., 7, 9 (1, 7, 9)	3.21e-13
1, 7, 10	1.91e-02	1, 8, 7, 8, 7, 8, 7, 8, 7, 10	3.65e-06	1, 7, ..., 7, 10 (1, 7, 10)	3.21e-13
1, 7, 8, 7, 11	9.93e-03	1, 8, 7, 8, 7, 8, 7, 8, 7, 11	4.02e-05	1, 7, ..., 7, 11 (1, 7, 11)	3.53e-12
1, 7, 12	1.91e-02	1, 8, 7, 8, 7, 8, 7, 8, 7, 12	3.66e-06	1, 7, ..., 7, 12 (1, 7, 12)	3.21e-13
1, 2, 14	1.78e-02	1, 8, 7, 8, 7, 8, 7, 1, 2, 14	2.56e-06	1, ..., 1, 2, 14 (1, 2, 14)	1.63e-16
1, 2, 16	8.89e-03	1, 8, 7, 8, 7, 8, 7, 1, 2, 16	1.28e-06	1, ..., 1, 2, 16 (1, 2, 16)	8.17e-17
1, 7, 17	4.62e-03	1, 8, 7, 8, 7, 8, 7, 8, 7, 17	1.46e-05	1, 7, ..., 7, 17 (1, 7, 17)	1.28e-12
1, 7, 8, 7, 8, 7, 18	7.14e-04	1, 8, 7, 8, 7, 8, 7, 8, 7, 18	1.46e-05	1, 7, ..., 7, 18 (1, 7, 18)	1.28e-12
1, 7, 18, 19	2.54e-02	1, 8, 7, 8, 7, 8, 7, 7, 18, 19	5.77e-06	1, 7, ..., 7, 18, 19 (1, 7, 18, 19)	1.18e-15

Table 4.1: The 10-years most probable paths, and the most probable trajectories under discretization approach ($\delta t = 1$ *year* and $\delta t = 1$ *day*) for each accessible state. The "collapsed paths" for 1-day discretized trajectories are also given (in brackets). The states are enumerated as follows: 1 - wild type, 2 - G190S, 3 - G190E, 4 - G190A, 5 - Y188L, 6 - Y188L+G190E, 7 - K103N, 8 - K103N+P225H, 9 - K103N+G190A, 10 - K103N+Y188H, 11 - 103N+V108I, 12 - K103N+V106M, 13 - K101E, 14 - K101E+G190S, 15 - K101Q, 16 - K101Q+G190S, 17 - K101Q+K103N, 18 - L100I+K103N, 19 - L100I+K103N+P225H.

Another downside of the time discretization approach is that the probabilities of actual trajectories (as opposed to state sequences) have very small probabilities and hence are not very informative on their own. Given the inexact nature and the limited scope of discretized solution, the advantage of our exact solution should be clear now.

4.4 Remarks

The same dataset that we used [4] had been previously used to demonstrate another algorithm, called "vPhyloMM" [9]. That algorithm is designed to reconstruct mutational pathways of drug resistance, also through the analysis of continuous-time Markov chains. The algorithm first estimates the underlying CTMC much in the same way as was described in Section 4.2. The analysis of the CTMC is different, however. Just like our algorithm, "vPhyloMM" tries to estimate top most probable paths, but instead of doing it exactly, the way it is described in this work, they would chose a time interval and discretize the CTMC with δt being the entire length of the interval (see Section 2.3.3). This resulting DTMC is used to estimate the transition probabilities between states, and the probabilities of paths are approximated through the product of those probabilities. The algorithm is computationally much easier than then the one described in Section 3.3, but is obviously

not exact and, depending on particular CTMCs, may produce very different results. Unfortunately, it is not possible to directly compare our results with those of vPhyloMM becuase the state space in [9] is different from ours, and because their definition of state transition allowed for double mutations, which we did not find to be reasonable, given that the model is already in a continuous time domain.

Chapter 5

Application 2: Fault Diagnosis

This chapter shows how our algorithm for computing most probable state sequences of a continuous-time Markov chain can be used in the domain of reliability engineering to analyze fault trees. First we explain what fault trees are and how they are used in reliability engineering. Then we descibe the traditional methods used in fault tree analysis. Finally, we demonstrate our approach with two hypothetical "textbook" examples. We will try to show that our state sequence analysis can provide information about a fault-tolerant system that would not be available to other, classical, fault tree analysis methods.

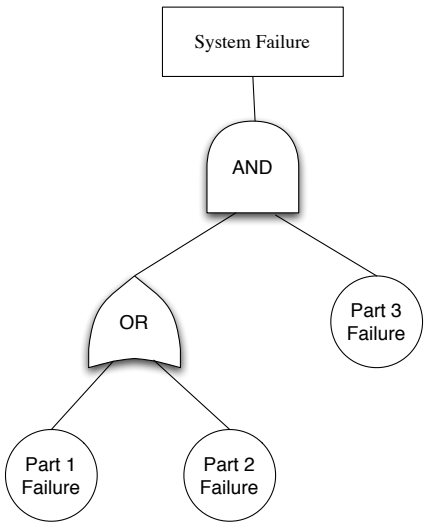

Figure 5.1: An example of a fault tree. This fault-tolerant system has 3 elementary components. The system fails if Part 3 fails together with either Part 1 or Part 2.

5.1 Introduction

Fault tree analysis is a deductive, top-down, failure-based approach to reliability engineering [23, 24]. Fault trees use boolean logic to model the logical propagation of simple failures to cause the undesired event, or top event (e.g. power shutdown, loss of vehicle, etc.). Individual failures can be thought of as low level mishaps each of which on its own would not necessarily cause the top event. The nodes of a fault tree are logic gates (e.g. AND, OR, etc.) that use outputs from other logic gates as inputs. Leaf nodes are elementary failure events.

The two most common gates in fault trees are the AND gate and the OR gate. If the top event is related to the lower level events through the AND gate then it will occur only if *all* lower level events occur. In contrast, if we have the OR gate, the top event occurs if *any* of the lower level events occur. Fig. 5.1 demonstrates a simple example of a fault tree. Here we assume that the system's functioning only depends on three parts: hypothetical Part 1, Part 2 and Part 3. The diagram basically tells us that for the system to fail, Part 3 must fail together with either Part 1 or Part 2. The system will still function if any of the three parts fails, or if both Part 1 and Part 2 fail, however any other scenario guarantees the system's failure.

Low level failures, or the leaf nodes of the fault tree, are usually modelled as components that have exponential life times. This statistical framework usually yields easily computable models that tend to be good approximations to the actual survival data. Exponentially distributed survival times with their memorylessness in turn allow us to use Markov analysis to study fault trees. It is easy to see how fault trees can be translated into the language of Markov chains. The states of a corresponding Markov chain would be represented by binary n-tuples where each slot would correspond to one of n elementary components of the system. We assume that each component can be in either of the two possible logic states (operational or failing). In such a model the top event would normally correspond to a set of different

states. If it makes the model more concise, Markov states can represent sets of binary n-tuples, not just one vector per state (e.g. all failing combinations can be grouped into a single "failing" state; similarly, in our toy example, the joint states of Parts 1 and 2 could be represented by just two values: neither failed, and one or both failed, because once one has failed the state of the other does not matter).The number of states in a full Markov model grows exponentially with the number of elementary components, but for systems of moderate size or systems where collapsing of states is possible, the Markov model can be a tractable size.

Dynamic fault trees are fault trees that have dynamic gates [15]. In dynamic gates the order in which inputs come in matter. In the reliability context it means that the order of the failures of elementary components is no longer irrelevant. An example of such a sequence dependent system is shown in Fig. 5.2. This system consists of three relevant components: the primary power source, the secondary power source and the switching mechanism. If the primary power source fails, the switch is supposed to turn on the secondary power source. In order for the system's top event to be true (high level system's failure), the system must receive no power from either of the power sources. Here we can no longer determine whether a state is failing simply by looking at which components have failed, the order of their failures must be taken into account as well. For example, If we know that

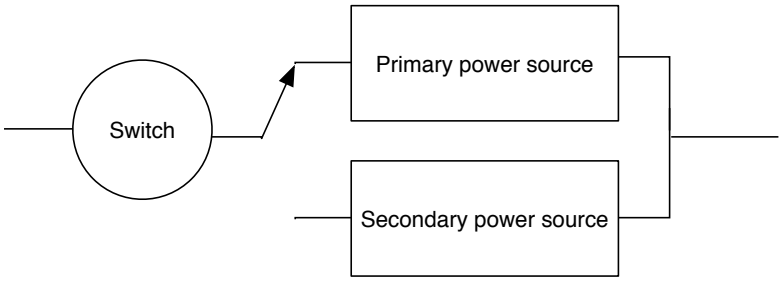

Figure 5.2: An example of a system with sequence dependent failure, adopted from [50]

the primary source and the switch are not working, is the system failing? Well, it depends. If the primary source failed before the switch, then right after the power source had failed the switch must have changed the power to the secondary source and the system would still be functional. On the other hand, if the switching mechanism had failed before the primary power source, then it could not switch to the secondary source on time and so the system is not functional. Gates that model these kind of situations are generally referred to as *functional dependency* gates.

Another important class of dynamic gates are the so-called *spare* gates. Spare gates model situations in which components have replaceable spares for when they die. Because spare parts are not operational while being in the "spare" mode, their survival must be modelled differently from that of the operational parts. Spare parts can be modelled as either hot, warm or cold. The term "hot spares" basically means that even though the component is a

spare part, it is still subject to the same aging laws as the operational parts. Cold spares are the ones that are assumed not to be subject to any aging until the moment they become operational. For example if we have a spare light bulb, for modelling purposes we can safely assume it will last forever if it is not connected to an electric circuit. Warm spares are somewhere in between, i.e. they do age, however at a slower rate.

Our algorithm can be used in fault diagnosis of such systems. In particular, unlike any other fault tree analysis technique, our approach can solve the following problem: given a certain initial configuration of the system (e.g. all components operational), and given that it is in a failing mode at time t_{final}, what are the most probable sequences of breakdowns? Actually, this question can address two different scenarios: we can know the exact time when the top event occurred, or we cannot have that information. The former case would call for the "likelihood" approach, whereas the latter one would call for the "probability" approach (see section 3.5 for more details). After having a brief look at the traditional fault tree analysis methodology, this chapter will demonstrate our new framework with two examples. The first example is a hypothetical redundant computer system that has more components than it needs. For this "textbook" example we will analyze the time dependence of the most likely failure sequences using the algorithm described in section 3.3. The second example is a simple avionics system that shows how our approach

can provide new insight.

5.2 Traditional fault tree analysis

Standard analysis of a static fault tree is fairly straightforward [23, 24]. First we identify the top event and construct the fault tree according to our objective. It is important that the scope (which events are important and which are not) and the resolution (level of detail) of the fault tree are well-established as well. Then the fault tree is evaluated qualitatively and quantitatively. In the heart of the qualitative analysis lies the notion of the *cut set*. The cut sets (more appropriately called "failure set") are sets of elementary events which together, when true, cause the top event. A minimal cut set is a cut set that cannot be reduced any further, i.e. if we take away any single event from it, it will no longer be a cut set. Cut sets can be identified by applying boolean algebra to fault trees that can be viewed as binary decision diagrams. In order to provide some information about the combinations of events that commonly lead to the system's failure, the minimal cut sets are usually sorted by the number of elements they contain (cut set order).

When doing the quantitative evaluation of a fault tree we produce the probabilities of cut sets as well as the top event. In the quantitative analysis, the cut sets are sorted by probability (not by cut set order as in the qualitative

analysis), so the *dominant* cut sets can be identified, i.e. those cut sets that contribute the most to the probability of the top event. If only limited computational resources are present, and only the top event probability is of interest, it is usually more computationally efficient to just compute the top event probability without calculating the probabilities of individual cut sets [43].

The qualitative and quantitative approaches described above would only work with static gates, such as AND, OR, and K-of-M (true when at least K out of M inputs are true) gates. If our fault tree includes any of the dynamic gates, the tree will have to be converted to a CTMC for further analysis [48, 15]. The dynamic gates are different in that the order in which the states of the inputs change matter as well. These include functional dependancy gates and hot/warm/cold spare gates, as well as the priority-AND, or sequence enforcing, gate (the output is true only if all of the inputs occur in a specified order). The analysis most commonly performed with the fault trees converted to CTMCs is evaluating the future probabilities by applying the solutions to forward Chapman-Kolmogorov equation (see Section 2.2).

When the system is very large, the sheer complexity of the resulting Markov chain can be intractable. A common strategy is the modular approach to fault tree analysis. In this approach the subtrees (modules) are

identified, thus turning the original tree into a hierarchical structure whose top level now has much lower complexity. If the nodes in the new subtree are only connected through the static gates, it is possible to solve each subtree using Markov analysis and then apply the qualitative and quantitative methodology of the static fault tree analysis. Modularization thus attempts to combine the best of the binary decision tree and the Markov analyses.

Our approach would be a suitable addition to the toolbox of Markov analysis methods available to engineers and researchers. Instead of only looking at single probabilities of failure configurations, it may be of interest to look at the *order* in which these configurations take place. As was explained in the earlier chapters, Chapman-Kolmogorov equations do not provide the means to do that. The two following sections will demonstrate how our approach can be used in practice.

5.3 Example 1: Hypothetical redundant computer system

This example, adopted from [50], illustrates the failure behaviours in a hypothetical computer system. The system under consideration consists of 3 processors (CPU1, CPU2 and CPU3), 2 memories (MEM1 and MEM2) and 1 bus (BUS). For the system to work, it needs to have at least one functional

Table 5.1: Definitions of states and dwell time parameters for fault diagnosis model.

State	λ (hr$^-$1)	Description
1	0.000161	3C2M, 3 processors, 2 memories and bus working
2	0.000161	2C2M, 2 processors, 2 memories and bus working
3	0.000161	1C2M, 1 processors, 2 memories and bus working
4	0.000131	3C1M, 3 processors, 1 memory and bus working
5	0.000131	2C1M, 2 processors, 1 memory and bus working
6	0.000131	1C1M, 1 processors, 1 memory and bus working
7	0	FAIL

CPU, a memory and a bus. Conversely, the failure mode would include any states that have all components of the same type down (i.e. both memories, a bus, or all three CPU's). The parameters of the model are given in tables 5.1 and 5.2.

The system can be visualized (Fig. 5.3) through its fault tree and its CTMC diagram. In the fault tree the nodes are the components (CPU's, MEM's and the BUS) and their failures propagate up to the top event FN through the OR and AND gates. Fig. 5.3B is the Markov diagram. ρ is the failure rate of any of the three CPU's, μ is the failure rate for either memory, and β is the failure rate for the bus. Their numerical values (adopted from [50]) are given in tables 5.2 and 5.1. Because we assume that all elementary components fail independently, the failure rate of any two or three components together will just be the sum of their individual failure rates.

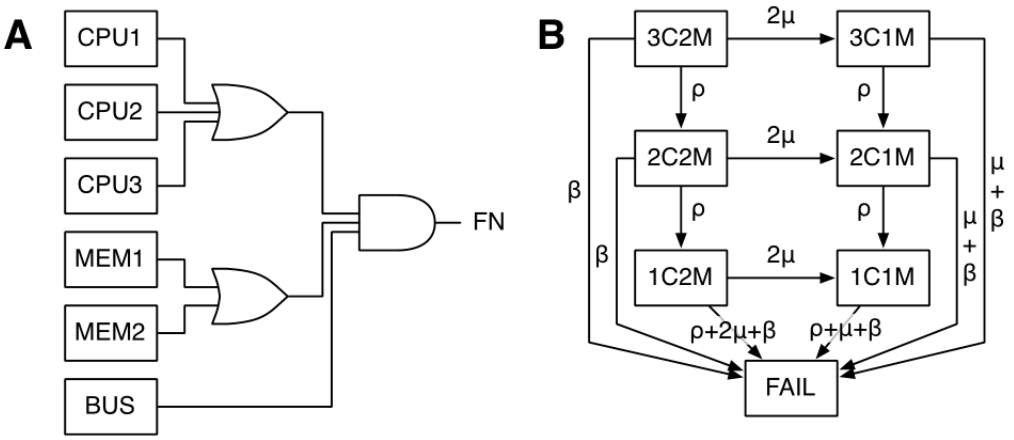

Figure 5.3: The redundant computer system. (A) The fault tree for the system. This is a negated fault tree in which the top event (FN) is true if the device is functioning. In order for the device to function any of the three CPU's (CPU1, CPU2, CPU3), the bus (BUS) and either of the two memories (MEM1, MEM2) have to be functional. (B) Markov state diagram for the associated CTMC. The notation iCjM (see table 5.1) refers to non-failing states in which i CPU's and j memories are still functional. All failing states are grouped together under a single state FAIL.

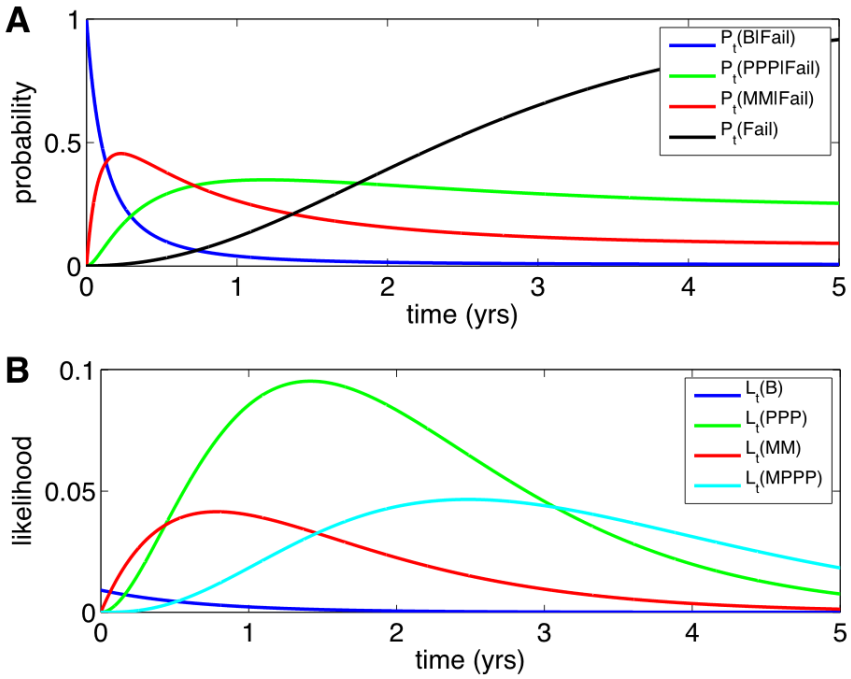

Figure 5.4: Most probable failure sequences as a function of time t, given that the failure happens at some time before t (A) or at precisely time t (B). Curves are identified by the sequence of component failures: B = bus fails, PPP = processors fail sequentially, MM = memory units fail sequentially, MPPP = one memory unit fails, followed by all three processors in sequence. $P_t(Fail)$ is the probability the system fails at or before time t.

Table 5.2: Transition probabilities for continuous-time version of fault diagnosis model.

	to 1	to 2	to 3	to 4	to 5	to 6	to 7
from 1	0	0.6211	0	0.3727	0	0	0.0062
from 2	0	0	0.6211	0	0.3727	0	0.0062
from 3	0	0	0	0	0	0.3727	0.6273
from 4	0	0	0	0	0.7634	0	0.2366
from 5	0	0	0	0	0	0.7634	0.2366
from 6	0	0	0	0	0	0	1.0000
from 7	0	0	0	0	0	0	0

We used our approach to compute most probable sequences of component failures resulting in a system failure. The first scenario (Fig. 5.4A) is when we do not know exactly when the system has gone into the failing mode. This can be the case, for example, in some autonomous systems that only get occasional check ups and status updates. In this case our algorithm is sorting non-dominated sequences (see section 3.2) according to their final time probability values. The second scenario (Fig. 5.4B) is when we know exactly when the system went into the failing mode. This can be the case if, say, our system was broadcasting a signal that was suddenly interrupted.

For short times of up a few months the most likely cause of failure is a failing BUS. In our first scenario of unknown failure time, the "failed BUS" explanation remains most likely for quite a bit longer than it does in our second scenario. For short times it is unlikely that the top event is caused

by the complete breakdown of either processors or memories since both subsystems have readily available backups. If we wait longer, however, other explanation become more likely, and just like with the HIV example (see section 4.3) the relationships between the state sequence probabilities are very time dependent.

In the second scenario, when we do know the exact failure time, it is interesting that after about 6 months the most likely explanation of failure is the complete breakdown of the CPU subsystem, however after roughly 3 years the most likely component failure sequence is memory failure followed by sequential failure of all 3 CPU's. Interestingly, in this case knowing that the system did not fail within the first three years makes it more likely that another component (a memory) failed along with the CPU's.

5.4 Example 2: Fault diagnosis within avionics system

This example is taken from [28]. This is an old reliability example that serves its purpose in demonstrating a mission-critical application. Figure 5.5 presents a Markov model of a navigational system of a fighter jet. Avionics systems consists of different subsystems, e.g. communication subsystem, navigation subsystem, sight subsystem, toss-bomb computer, etc. Here we

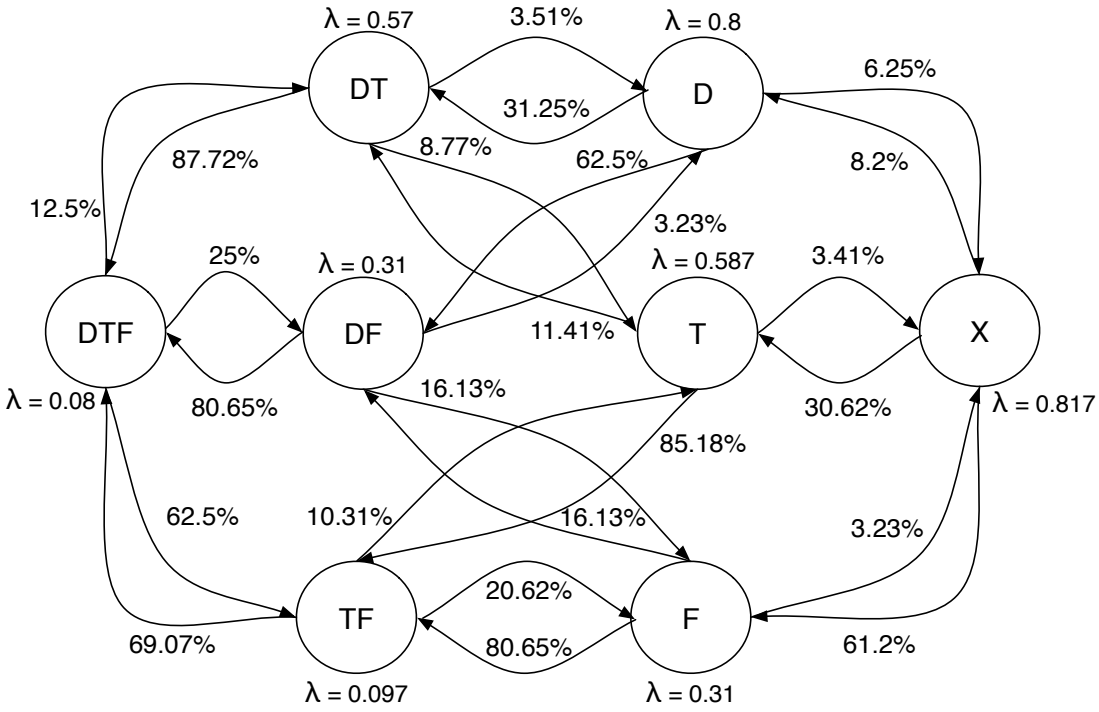

Figure 5.5: The state diagram for the navigational avionics system example. The states are explained in Table 5.3 below.

State	Doppler	TACAN	Direction Finder
DTF	Operating	Operating	Operating
DT	Operating	Operating	Down
DF	Operating	Down	Operating
TF	Down	Operating	Operating
D	Operating	Down	Down
T	Down	Operating	Down
F	Down	Down	Operating
X	Down	Down	Down

Table 5.3: Navigation Equipment Status. The states of our CTMC.

will analyze the Markov model for the navigation subsystem to demonstrate how our pathwise analysis can be used in its fault diagnosis.

The system under investigation consists of three independent subsystems which are working in parallel—Doppler Navigator, TACAN, and Direction Finder subsystems. It is assumed that all three subsystems can be repaired with certain probabilities, however we have no communication with the maintenance personnel, whose skill levels are reflected in those probabilities ("repair rates"). Figure 5.6 shows the top seven most likely state sequences of the navigational system. Up until about 24 hours it is the most likely that nothing becomes broken. For a brief period between 24 and 28 hours it is most probable that the Doppler Navigator goes down and stays that way, while from about 28 hours up until 36 hours the most likely scenario is for the Doppler Navigator to go out and then self-repair.

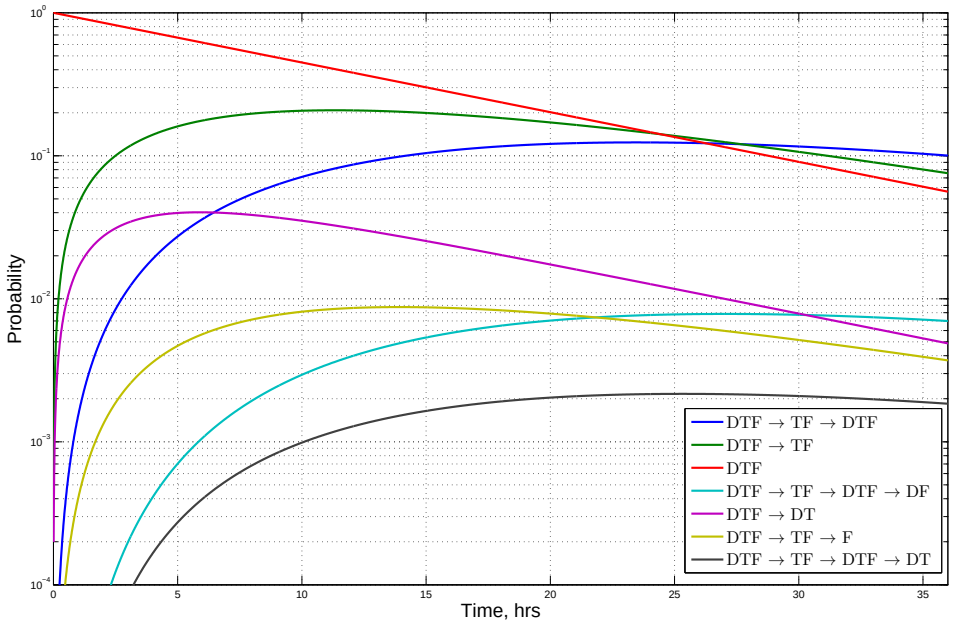

Figure 5.6: The top seven most probable state sequences for the navigational system of a fighter jet for a period of 36 hours. While it is most likely that all three parallel components of the system will be in operational mode, the chances are that Doppler Navigator went out at some point and then self-repaired.

So, if we perform a checkup after 36 hours of operation and everything seems to be working, in fact it is more probable that one of the components was out for some time period and then went back on and perhaps requires some adjustments. If we somehow know that at 36 hours exactly one component is broken, then it is most likely to be Doppler Navigator, and it is most likely that it just broke, and there had been no self-repairs before that. If we know that exactly two components are down, then the chances are those are Doppler and TACAN, and Doppler went out first. Also, if at 36 hours we observe Direction Filter in inoperable conditions, it probably went out after Doppler went out and repaired itself.

Chapter 6

Discussion

This work presented the exact solution to the problem of determining the most probable sequences of state transitions in a continuous-time Markov chain, given that the initial state and the final time are specified. The solution uses a dynamic program that searches and sorts through the finite space of non-dominated sequences (see Section 3.4). While this is a problem that arises very naturally when one deals with random systems, previous ways of tackling it would generally provide at best sub-optimal and at worst misleading solutions (see Section 4.4).

The utility of the algorithm was demonstrated through its applications to two different CTMC systems. First, it was shown how the algorithm can rank HIV mutational pathways when the virus's process of natural selection is altered by artificial pressure from antiretroviral drugs. Then, the problem

of predicting the most likely failure sequences was solved using the exact same procedure.

Other applications that we have considered but that have not been included here are predicting the most probable sequence of spacial conformations of a certain protein, and estimating the unseen conformational changes of ion channels based on patch-clamp records. As CTMC's are a popular modelling technique, it is easy to see how the algorithm can be applied to a wide range of different problems across different domains.

As was discussed in Section 3.5, our approach is general enough to solve the problem of ranking an arbitrary number of most probable sequences. Also, the information about the exact time of the last transition can be incorporated into our framework to provide a better estimate of the top most probable trajectories based on their likelihoods. There are still several ways in which the algorithm could be extended and improved upon.

In a lot of practical situations we will not be able to estimate the parameters of our model precisely. We could be given a probability distribution of certain parameters, their bounds, or they can just be missing. One way of dealing with this situation is to use multiple imputation [38]. Multiple imputation is a statistical procedure designed to deal with missing data. In our case this could work as follows. We could view our CTMC as represented by a single matrix Q that has some values missing. Whenever the algorithm

needs to use Q, the multiple imputation algorithm would create several copies of complete Q's where the missing values are imputed according to whatever information we have (of course $\sum_{j \neq i} Q_{ij} = -Q_{ii}$ would have to be satisfied). The function that calls Q is called several times for each of the copies and then a simple average of the returned values is taken. Multiple imputation methods provide simple formulas for estimating the confidence intervals for the resulting averages. Obviously, since the procedure is repeated many times during one run of the algorithm, some more complex ways to quantify our uncertainty about the final output would need to be established.

Another direction for further research is dealing with countable (not just finite) state spaces. Unfortunately, the algorithm in its current form is not guaranteed to terminate when the state space is infininte. One possible "solution" could be trimming the state space by getting rid of the states whose probabilities of being visited are below a certain threshold. Of course, this is risky and will only produce approximate solutions at best.

Establishing time complexity is another important issue. While our Matlab implementation solves all the application problems presented here in a matter of minutes or even seconds, it is currently unknown how slow the algorithm will become with more complex models. The best case scenario is when there are a few obvious "winners" that dominate most of the initial state sequences, so the others will not be extended and enqueued further

(see Section 3.3). However it is not obvious what the time complexity for the general case is. Because the high level view of the algorithm leaves flexibility as to how the probability functions are computed (see Section 3.4), the performance can vary quite a bit depending on a particular implementation. Even in our ODE-based implementation, the user is flexible to change the integration time step. While the inverse Laplace transform-based solution for solving 3.14 analytically would require dramatically less memory, the time complexity of the algorithm would grow exponentially with the number of states in the sequences. Also, having analytical instead of numerical solutions to 3.14 would turn the otherwise trivial problem of establishing dominance into a more complex optimization problem.

The main reason why establishing time complexity for our algorithm is so difficult, is because the running time seems to depend, in complex ways, on the values of probability transition rates of the CTMC, and not just the size of the state space. For example, with our current implementation the 19-state HIV example takes more than 12 times faster to compute than the 6-state toy example. The following are the running times of our Matlab implementation on a relatively old 2.99GHz Intel Core^{TM} 2 Duo CPU Lenovo PC with 1.83Ghz of RAM: toy example (6 states)—85.00 sec, HIV example (19 states)—6.60 sec, hypothetical computer system example (9 states) — 7.15 sec, fighter jet example (8 states)—15.07 sec. It is obvious that there is

no simple relationship between the number of states and the running time, therefore it is unclear how the algorithm would scale to much larger problems, and it is something that needs to be investigated further.

Dealing with partial observations could be another useful extension. Suppose we have a scenario in which we observe a certain random process. We know that the process is a continuous-time Markov chain, however the actual (latent) states of the system are unavailable to us. Instead we can only observe another set of states (observed states). An interesting problem, then, would be to find the most probable sequence of latent states given the observations. This exact problem has been solved for discrete-time Markov chains, and its solution, the Viterbi algorithm, is widely applied in engineering and science [49]. One important application of Viterbi algorithm is the domain of speech recognition [30]. The latent variables are assumed to be syllables, while the observations are sounds. Given how different the discretized solutions can be compared to the exact ones in our HIV example (see Section 4.4), it is reasonable to believe that the exact continuous-time solution could significantly improve the performance of speech recognition algorithms, as well as many others algorithms that discretize continuous-time processes.

Acknowledgements

I am indebted to Dr. Theodore Perkins for taking me into his lab and letting me work on this project. I am very grateful to have been able to work with Dr. Perkins's on the solution, which I know will have many applications and useful extensions in the future. Dr. Perkins was the most helpful advisor whose guidance and expertise helped me explore and expand my interests. At the same time I greatly enjoyed the flexibility of being a member of his lab. My thesis greatly benefited from Dr. Perkins's constructive guidance and shrewd insight.

I am also thankful to the Ottawa Hospital Research Institute (Dr. Perkins and Sprott Centre for Stem Cell Research in particular) and the Faculty of Graduate and Postdoctoral studies of University of Ottawa for the financial assistance. I am thankful to Monique Walker, the coordinator of System Science program for being very helpful all throughout my studies. Finally, I'd like to thank all my friends and family for being extremely supportive.

Bibliography

[1] A. Albert. Estimating the infinitesimal generator of a continuous time, finite state Markov process. *The Annals of Mathematical Statistics*, pages 727-753, 1962.

[2] W. J. Anderson. *Continuous-Time Markov Chains: An Applications-Oriented Approach.*, Springer-Verlag, 1991

[3] L.C. Andrews and B.K. Shivamoggi *Integral transforms for engineers and applied mathematicians*,Macmilan Publishing Company, 1986

[4] L.T. Bacheler, E.D. Anton, P. Kudish, D. Baker, J. Bunville, K. Krakowski, L. Bolling, M. Aujay, X.V. Wang, D. Ellis, M.F. Becker, A.L. Lasut, H.J. George, D.R. Spalding, G. Hollis, and K. Abremski. Human immunodeficiency virus type 1 mutations selected in patients failing efavirenz combination therapy. *Antimicrobial Agents and Chemotherapy*, 44(9):2475–84, 2000.

[5] FG Ball and JA Rice. Stochastic models for ion channels: introduction and bibliography. *Mathematical biosciences*, 112(2):189, 1992.

[6] D.A. Benson et al. GenBank *Nucleic Acids Res.*, 33:D34-D38, 2005

[7] D.P. Bertsekas. *Dynamic programming and optimal control, vol. I.* Athena Scientific, 2005.

[8] H. Boudali and J.B. Dugan. A continuous-time bayesian network reliability modeling, and analysis framework. *Reliability, IEEE Transactions on*, 55(1):86–97, 2006.

[9] P. Buendia, B. Cadwallader, and V. DeGruttola. A phylogenetic and markov model approach for the reconstruction of mutational pathways of drug resistance. *Bioinformatics*, 25(19):2522, 2009.

[10] F. Ceccherini-Silberstein, V. Svicher, T. Sing, A. Artese, M.M. Santoro, F. Forbici, A. Bertoli, S. Alcaro, G. Palamara, A. d'Arminio Monforte, et al. Characterization and structural analysis of novel mutations in hiv-1 reverse transcriptase involved in the regulation of resistance to non-nucleoside inhibitors. *Journal of Virology*, pages JVI–00303, 2007.

[11] F. Clavel et al. HIV Drug Resistance *New Eng. J. of Med.*, 350:1023-35, 2004

[12] B. Clotet. Efavirenz: resistance and cross-resistance. *International journal of clinical practice. Supplement*, 103:21, 1999.

[13] B. Damas and P. Lima Stochastic discrete event model of a multi-robot team playing an adversarial game *5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles - LAV2004*, Lisboa, Portugal

[14] D. Duffie and P. Glynn. Estimation of continuous-time Markov processes sampled at random time intervals. *Econometrica, 72(6):1773-1808, 2004*, 99:12795-12800, 2002.

[15] J. Dugan, S. Bavuso and M. Boyd Dynamic fault tree models for fault tolerant computer systems *IEEE Transactions on Reliability*, 41(3):363-377, 1992

[16] R.T. DAquila, J.M. Schapiro, F. Brun-Vézinet, B. Clotet, B. Conway, L.M. Demeter, R.M. Grant, V.A. Johnson, D.R. Kuritzkes, C. Loveday, et al. Drug resistance mutations in hiv-1. *Top HIV Med*, 11(3):92–96, 2003.

[17] J. E. Gallant et al. Efficacy and safety of tenofovir DF vs stavudine in combination therapy in antiretroviral-naive patients: a 3-year randomized trial. *Jama*292: 191-201.

[18] M.A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem. A*,104(9):1876-1889, 2000.

[19] D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81:2340-2361, 1977.

[20] D. Gross and C.M. Harris. *Fundamentals of queueing theory.* Springer-Verlag, 1985.

[21] A. Hordijk, D.L. Iglehart, and R. Schassberger Discrete time methods for continuous time Markov chains *Advances in Applied Probability*, 772-788, 1976

[22] M.J. Keeling and K.T.D. Eames. Networks and epidemic models. *Journal of the Royal Society Interface*, 2(4):295, 2005.

[23] W.S. Lee, DL Grosh, F.A. Tillman, and C.H. Lie. Fault tree analysis, methods, and applications – a review. *Reliability, IEEE Transactions on*, 34(3):194–203, 1985.

[24] W. Long, Y. Sato, and M. Horigome. Quantification of sequential failure logic for fault tree analysis. *Reliability Engineering & System Safety*, 67(3):269–274, 2000.

[25] L. Calvet and A. Fisher Forecasting multifractal volatility *Journall of Econometrics*, 105: 27-58, 2001.

[26] M. Nei and S. Kumar. *Molecular evolution and phylogenetics*. Oxford University Press, USA, 2000.

[27] Theodore J. Perkins. Maximum likelihood trajectories for continuous-time markov chains. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1437–1445. 2009.

[28] J. Pukite and P. Pukite *Modeling for Reliability Analysis*, IEEE Press, 1998

[29] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, New York, 1994.

[30] LR Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[31] M. Rausand and A. Hoyland. *System Reliability Theory: Models, Statistical Methods, and Applications*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2008.

[32] S.Y. Rhee, M.J. Gonzales, R. Kantor, B.J. Betts, J. Ravela, and R.W. Shafer. Human immunodeficiency virus reverse transcriptase and protease sequence database. *Nucleic acids research*, 31(1):298, 2003.

[33] S.Y. Rhee, J. Taylor, G. Wadhera, A. Ben-Hur, D.L. Brutlag, and R.W. Shafer. Genotypic predictors of human immunodeficiency virus type 1 drug resistance. *Proceedings of the National Academy of Sciences*, 103(46):17355, 2006.

[34] S.M. Ross *Introduction to Probability Models, 9th ed* Elsevier, 2007

[35] A. Reibman and K. Trivedi Numerical transient analysis of Markov models *Comput. Opns Res.*, 15: 19-36, 1988.

[36] B. Roux, T. Allen, S. Berneche, and W. Im. Theoretical and computational models of biological ion channels. *Quarterly reviews of biophysics*, 37(01):15–103, 2004.

[37] M. Salehi and T. J. Perkins. Maximum probability reaction sequences in stochastic chemical kinetic systems. *Frontiers in Systems Biology*, 2010.

[38] J.L. Shafer Multiple imputations: a primer *Statistical Methods in Medical Research*, 1999

[39] R.W. Shafer et al. Comparison of four-drug regimens and pairs of sequential three-drug regimens as initial therapy of HIV-1 infection *N. Engl. J. Med.*, 349: 2304-2315

[40] M. Spencer and E. Susko Continuous-time Markov models for species interactions *Eco. Soc. Amer.*, 86:3272-3278

[41] L.F. Shampine and M.W. Reichelt The MATLAB ODE Suite *SIAM Journal on Scientific Computing*, Vol. 18, 1997, 1-22

[42] D. G. Spiller, C. D. Wood, D. A. Rand, and M. R. H. White. Measurement of single-cell dynamics. *Nature*, 465:736–745, 2010.

[43] M. Stamatelatos, W. Vesely, J. Dugan, J. Fragola, J. Minarick, and J. Railsback. Fault tree handbook with aerospace applications, version 1.1. *National Aeronautics and Space Administration*, 2002.

[44] P. S. Swain, M.B. Elowitz, and E. D. Siggia. Intrinsic and extrinsic contributions to stochasticity in gene expression. *Proceedings of the National Academy of Sciences of the USA*, 99:12795-12800, 2002.

[45] V. Shahrezaei, J.F. Olliver, and P.S. Swain. Colored extrinsic fluctuations and stochastic gene expression. *Molecular Systems Biology*, 4(1), 2008.

[46] CA Vandenberg and F. Bezanilla. A sodium channel gating model based on single channel, macroscopic ionic, and gating currents in the squid giant axon. *Biophysical Journal*, 60(6):1511–1533, 1991.

[47] M.E. Van Valkenburg *Network analysis*, 2nd Edition, Prentice Hall, 1964

[48] WE Vesely. A time-dependent methodology for fault tree evaluation. *Nuclear engineering and design*, 13(2):337–360, 1970.

[49] A.J. Viterbi Error bounds for convolutional codes and an asymptotically optimum decoding algorithm *IEEE Transactions on Information Theory*, 13(2): 260-269

[50] W Vesely et al. Fault tree handbook with aerospace applications. *Space Administration, NASA*, 2002.