The Pennsylvania State University
The Graduate School

A REFORMULATION OF THE CSSR ALGORITHM AND APPLICATION TO OPTIMAL DECEPTION STRATEGY IN TWO PLAYER GAMES

A Thesis in
Mathematics
by
Elisabeth Paulson

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Arts

May 2015

The thesis of Elisabeth Paulson was reviewed and approved* by the following:

Christopher Griffin
Associate Proffessor of Mathemtics
Thesis Advisor

Svetlana Katok
Professor of Mathematics
Director of Graduate Studies

Jason Morton
Professor of Mathematics and Statistics

*Signatures are on file in the Graduate School.

Abstract

In this thesis we explore a reformulation of the Casual State Splitting and Reconstruction (CSSR) algorithm and an application to optimal strategies for deception in repeated two-player games. The CSSR algorithm is used to infer probabilistic finite-state machines from an input stream of data. We formulate an integer programming version of the CSSR algorithm which always results in minimal probabilistic finite-state automata given an input stream of data. This reformulation is shown to be NP-hard by comparing it to the minimal clique covering problem in graph theory. We then apply this algorithm to optimal deception strategies in repeated two-player games in which one player seeks to deceive the other by making use of a deceptive "learning period". We find that this can be modeled by combining both linear optimization with a genetic algorithm. We present numerical examples of optimal deception as well as some theoretical results.

Table of Contents

List of Figures

List of Tables

Acknowledgments

Portions of this work were support by the Army Research Office under Grant W911NF-11-1-0487 and Grant W911NF-13-1-0271.

Chapter 1

Summary of Results in Thesis

Inferring probabilistic finite-state machines from an input data stream is an important task in many fields of mechanical engineering and machine learning. The most well known probabilistic finite-state machines are hidden Markov models, in which the process is assumed to transition between states at discrete time intervals and emit symbols at each state with certain probability. There are many algorithms for inferring probabilistic finite-state machines from a set of data. In this thesis we focus on the Causal State Splitting and Reconstruction (CSSR) algorithm, which was developed by Shalizi [1]. This algorithm is a heuristic which uses maximum likelihood estimates of conditional probabilities to create a minimal state probabilistic finite-state automata. The heuristic clusters together "similar" states in order to produce a machine with the smallest possible number of states.

However, the CSSR algorithm does not always result in the minimal number of states because it relies on the asymptotic convergence of conditional probabilities in the input stream. That is, it relies on the assumption that the input data is infinite length. This issue is the focus of Chapter 3 of this thesis. We use the idea of the CSSR algorithm and reformulate it as an integer program which guarantees minimal state machines. We prove that this integer program is NP-hard by drawing an analogy to the minimal clique covering program in graph theory.

A probabilistic finite-state machine can be summarized by a probabilistic transition function. This is essentially a transition matrix in the typical Markov chain sense, however in this case the transitions also depend on the next symbol that appears. In the context of decision-making and game theory, this transition function can also be thought of as strategy. An agent, given a set of states, can formulate a strategy to govern his actions in any given state. In Chapter 4, we show how to generate an optimal strategy given a set of states as well as a payoff function corresponding to each state.

As an application of the CSSR algorithm and the optimization presented in Chapter 4, we develop a model for optimal deception in repeated two-player games in Chapter 5. Namely, we are interested in multi-period competitive game where two actors first engage in a "learning

period" followed by a period of game play. In the learning period, one player (Player 2) has the opportunity to learn the opposing player's (Player 1) strategy by observing the other player's response to random input, generating a string of inputs and responses. Player 1 can infer a probabilistic finite state machine from this data which is equivalent to learning Player 1's strategy. Player 2 can then generate an optimal counter-strategy to utilize during the game play period which maximizes his reward against Player 1's strategy.

However, we introduce the notion of deception by allowing Player 1 to be aware of Player 2's observation and learning. Thus, Player 1 is aware of this observation, and can "trick" the other player by training them on a false strategy. That is, Player 1 can play a certain strategy during the learning period, have Player 2 optimize based on this strategy, and then switch strategies in the game play period in order to maximize his/her own reward. We present numerical results using this model for classic repeated Prisoner's Dilemma.

Chapter 2

Literature Review

This thesis focuses on a few main topics that will be covered in the literature review. First we give an overview of probabilistic finite-state machines. We then give a review of algorithms for determining these machines from an input data stream, specifically focused on the Causal State Splitting and Reconstruction algorithm. Finally, we give an overview of competitive two-player games and review literature on deception strategies.

2.1 Probabilistic Finite State Machines

Pattern recognition is a rapidly growing area of research that seeks to identify underlying processes in a set of input data. Probabilistic finite-state machines (pFSMs) are a wide class of machines used to describe such processes [2]. They are used in a wide variety of fields including speech and pattern recognition [3, 4], bioinformatics [5, 6], and machine translation [7, 8, 9] among others. We will focus on two types of probabilistic finite state machines–Hidden Markov models (HMMs) and probabilistic finite-state automata (pFSA).

Before describing these two types of pFSMs, we first present notation based on [2]. Let $\mathcal{A}$ be a finite alphabet and denote $\mathcal{A}^*$ as the set of all strings that can be created from $\mathcal{A}$. A symbolic output is a sequence $\mathbf{y} = y_0 y_1 y_2 ... y_n,\ y_i \in \mathcal{A}$ where the subscripts represent discrete time intervals.

If $\mathbf{y}$ is a sequence, then $\mathbf{x}$ is a subsequence if $\mathbf{x}$ is a sequence with symbols in $\mathcal{A}$, and there are integers i and k such that $\mathbf{x} = y_i y_{i+1} ... y_{i+k}$.

A stochastic language $\mathcal{D}$ is a probability distribution over $\mathcal{A}^*$, and $\Pr_{\mathcal{D}}(\mathbf{x})$ is the probability of string $\mathbf{x} \in \mathcal{A}^*$ occurring under $\mathcal{D}$ with $\sum_{\mathbf{x} \in \mathcal{A}^*} \Pr_{\mathcal{D}}(\mathbf{x}) = 1$. The overall goal of any pFSM is to infer the underlying process which most likely produced a symbolic series $\mathbf{y}$.

Rabiner uses the example of a coin toss to explain an HMM, which is applicable to pFSMs in general [10]: imagine that a coin tosser is hidden behind a barrier and you are on the other side. You cannot observe what is happening on the other side of the barrier, but the coin tosser

will tell you the result of each flip. Thus, the observations would consist of a string of heads and tails that could perhaps look something like

$$THHHTHTT...$$

We will see how to construct both an HMM and pFSA for this example later on.

Every pFSM consists of a set of states and associated transition probabilities, which are two important parameters of the model. Rabiner poses three main questions of interest related to HMMs which apply to all pFSMs:

1. Given an observation sequence $\mathbf{y}$, how can we compute the probability of $\mathbf{y}$ given all of the parameters of our model?

2. Given an observation sequence $\mathbf{y}$ and all of the parameters of our model, how can we choose a state sequence hat best explains our observed sequence?

3. How can we choose the model parameters to maximize the probability of $\mathbf{y}$? That is, we want to choose parameters that result in the highest likelihood of $\mathbf{y}$ being observed.

Rabiner also gives mathematical tools for answering each of these questions. For Question 1 he presents a brute-force method using independence of observations, as well as a more efficient method called the Forward-Backward Procedure which was established by Baum et al. in [11, 12], also known as the Baum-Welch algorithm. The solution to Question 2 depends on how one defines the "best state sequence." For example, we could find the state which has the highest probability for each observation. However, this sequence of states may not even turn out to be a valid sequence of state transitions based on our model [10]. There are better methods for determining an optimal state sequence which Rabiner presents. However, Question 3 is of the most relevance and interest to this thesis and will be further explored.

2.1.1 Hidden Markov models

Hidden Markov models (HMMs) are a pattern recognition tool used to construct a Markov model when the state space is unknown. The theory of Hidden Markov Models was established by Baum et al. in the 1960's and early 1970's [13, 11, 12, 14, 15], and were first used in speech processing, for example as in [16] and [17, 18]. Since then, they have also been widely used in handwriting recognition [19, 20], and tracking [21]. Rabiner gives an extensive tutorial on Hidden Markov models which is widely cited in the field [10]. Rabiner explains that Hidden Markov models generalize stochastic processes for which the observations are a probabilistic function of the state [10]. The underlying stochastic process is not observable, but can instead be *inferred* from a string of observations.

As in [22], we define a Hidden Markov model as a tuple $\mathcal{M} = \langle \mathcal{Q}, \mathcal{A}, T, E, \pi_0 \rangle$, where $\mathcal{Q}$ is a finite set of states, $\mathcal{A}$ is our finite alphabet of symbols, T is a state transition probability function, E is a state-based emission probability function, and π_0 is a an initial probability distribution

over the $\mathcal{Q}$. This process is stochastic in that it must satisfy the constraints $T(q_i, q_j) \geq 0$ and $\sum_{j=1}^{N} T(q_i, q_j) = 1$ where N is the total number of states.

In the coin flip example, we could assume that the coin is unbiased and the flipper is reporting truthful coin flips. In that case, our model would simply be Figure 2.1 where $p_h = 0.5$. State 1 represents heads, since $E(H) = 1$, and state 2 represents tails, since $E(T) = 1$.

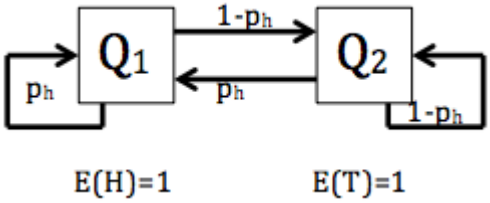

Figure 2.1: A Hidden Markov model constructed from a coin flip with one coin.

However, if the coin is biased, then we would want our Hidden Markov model to reflect such a bias. In that case, our model would still look like Figure 2.1 however $p_h \neq 0.5$ and instead the probabilities would reflect the bias.

In the above situations, we assumed that the coin flipper was only using one coin. Another possibility is that the flipper is using two different coins. In this case, we could construct a HMM similar to Figure 2.2, however now each state corresponds to a different coin that is being used. In this case, p_{11} represents the probability that the flipper continues to use coin 1, whereas $1-p_{11}$ is the probability that the flipper switches from using coin 1 to using coin 2. Coin 1 emits heads with probability p_{h1} and coin 2 emits heads with probability p_{h2}.

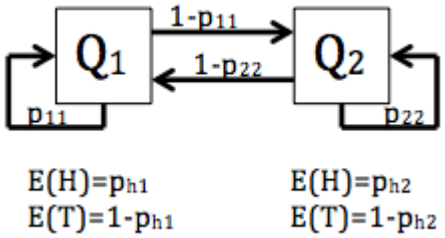

Figure 2.2: A Hidden Markov model constructed from two different, possibly biased, coins being flipped.

Rabiner points out that an important distinction between these two possible models is that the model shown in Figure 2.1 has only one parameter, whereas the model shown in Figure 2.2 has four parameters that must be estimated. While the two-coin model is more capable of modeling complicated systems, and could even be used when there is only one coin, it would be be unnecessary and overcomplicated for a processes in which there is really only one coin.

There are many different types of HMMs. In the coin toss examples, every state could be reached in one step from any other state. An ergodic HMM is one in which every state can be

reached in a *finite* number of steps from any other state [10], an example of which is shown in Figure 2.3(a). Other types of models have been found to better account for physical phenomena such as speech recognition. An example of such a model is a left-right model, in which the states progress over time. A variant of the left-right model shown in Figure 2.3(b) is used in [18] for continuous speech recognition. A similar model is a parallel path left-right model in which the states still progress over time but could follow different paths, shown in Figure 2.3(c).

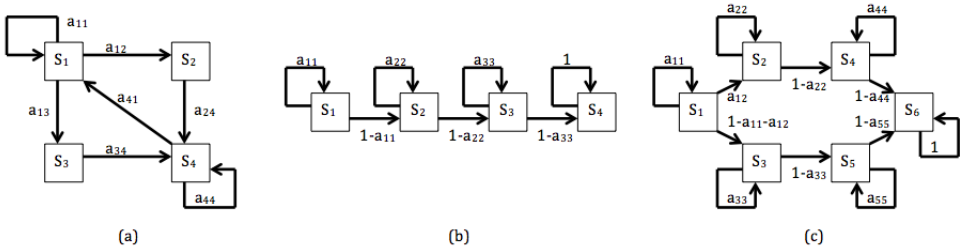

Figure 2.3: A comparison of three different types of Hidden Markov models, (a) shows an ergodic HMM, (b) shows a left-right HMM, and (c) shows a modified left-right HMM.

There are also processes that do not emit discrete symbols, but instead emit continuous vectors. These continuous output HMMs are studied extensively in [23, 24, 25]. These processes may also be modeled by HMMs, but the model parameters must be computed in a different way because we can no longer consider discrete probability densities [10].

2.1.2 Probabilistic Finite-State Automata

A very similar idea to hidden Markov models are probabilistic finite state automata (pFSA). Unlike Hidden Markov models, where each *state* emits symbolic output, in a pFSA, each *transition* corresponds to a symbolic output. Given a symbolic input stream, both hidden Markov models and pFSA can be inferred in order to explain the underlying processes [2]. A pFSA is a tuple $G = \langle Q, \mathcal{A}, \delta, p \rangle$, where Q is a finite set of states, $\mathcal{A}$ is a finite alphabet and $\delta \subseteq Q \times \mathcal{A} \times Q$ is a transition relation, and $p : \delta \to [0, 1]$ is a probability function such that

$$\sum_{a \in \mathcal{A}, q' \in Q} p(q, a, q') = 1 \quad \forall q \in Q. \tag{2.1}$$

.

A pFSA is *deterministic* (called a dpFSA) when for every state q and symbol a, there exists a new state q' such that $\delta(q, a, q') = 1$. In other words, the state transition is completely defined by the initial state q and the transition symbol a. In such cases the dpFSA can be more compactly written as $G = \langle Q, \mathcal{A}, \delta \rangle$. It is easy to see that if there is some initial probability distribution π_0 over Q, then the triple $\langle G, p, \pi_0 \rangle$ is a Markov chain with transitions labels.

In order to relate HMMs and pFSAs, we present the following two propositions from [22]:

Proposition 1. *Given a pFSA $\mathcal{G}$ with m transitions and $Pr_G(\lambda) = 0$ (where λ is the empty string), there exists an HMM $\mathcal{M}$ with at most m states such that $\mathcal{D}_\mathcal{M} = \mathcal{D}_\mathcal{G}$, where λ is the*

empty string.

Proposition 2. *Given an HMM $\mathcal{M}$ with n states, there exists a pFSA $\mathcal{G}$ with at most n states such that $\mathcal{D}_{\mathcal{G}} = \mathcal{D}_{\mathcal{M}}$.*

These two theorems speak to the fact that HMMs and pFSA can model the same processes in very similar ways. To see this, we model the coin flip example as a pFSA instead of an HMM. The HMM in Figure 2.1 corresponds to Figure 2.4 shown below. In Figure 2.4, the *transitions* are labeled with symbols, instead of considering the states to emit symbols as in the HMM in Figure 2.1.

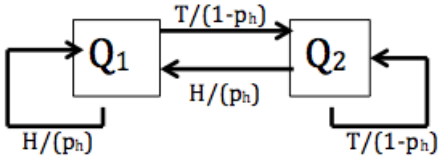

Figure 2.4: A pFSA model constructed from a coin flip with one coin.

The difference between a HMM and a pFSA are further illustrated by comparing Figure 2.2 to Figure 2.5. Both models represent the two-coin example. However, in the HMM in Figure 2.2, each state corresponds to a coin and emits either a heads or tails with certain probability. The only transitions shown are the transitions from coin one to coin two. In the pFSA in Figure 2.5, again each state represents a coin. However, each transition is labeled with either a H or T, and each transition is also given an associated probability. Notice that Figure 2.5 is nondeterministic because the transition symbol does *not* determine the next state; instead, the state transitions are probabilistic given the transition symbol. The two models represent exactly the same process, but can be thought about slightly differently.

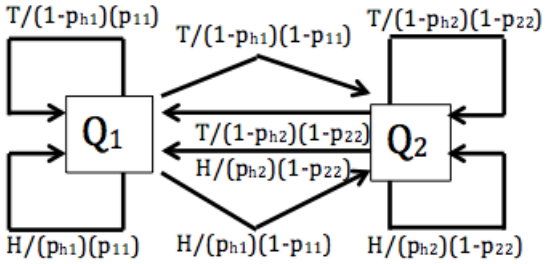

Figure 2.5: A pFSA model constructed from a coin flip with one coin.

2.2 Inferring Probabilistic Finite-State Machines

In this section we focus on methods for solving Problem 3 presented by Rabiner [10]: How to best infer the parameters of our pFSM. Given a process which produces a stream of training data, there are many algorithms that are used to infer a hidden Markov model for the process.

The standard algorithm for solving this problem is the Baum-Welch algorithm, which maximizes expectation [10, 14]. The result of this algorithm is a transition matrix T which estimates the state transitions in the HMM. The algorithm requires an initial guess for the structure of the HMM (an initial guess for T and an initial probability distribution π_0), and a sequence of output symbols. It uses dynamic programming to solve a nonlinear optimization problem [26].

2.2.1 Baum-Welch Algorithm

Given a stream of observed symbolic output $\mathbf{y} = y_0 y_1 y_2 ...,\ q_i \in \mathcal{A}$, and corresponding state transitions $\mathbf{x} = x_0 x_1 x_2 ...,\ x_i \in \mathcal{Q}$, the Baum-Welch algorithm uses a forward-backward procedure to find the parameters of the HMM that maximize the likelihood of observing $\mathbf{y}$ [14]. Let T be the state transition probability function, so $T(q_i, q_j) = \Pr(y_t = q_j | y_{t-1} = q_i)$, (which we will write T_{ij} as shorthand), where $q_i, q_j \in \mathcal{Q}$ are any two states. Let $\pi_i = P(x_0 = q_i)$ be the initial probability distribution. Finally, let E be the state-based emission probability function with elements $E_j(y_t) = Pr(y_t | x_t = q_j)$ which is the probability of observing y_t given that we are in state q_j. Letting θ be the set of parameters (T, E, π^0), the goal of the Baum-Welch algorithm is to find $\theta^* = \max_\theta Pr(\mathbf{y}|\theta)$ [14]. A description of the Baum-Welch Algorithm can be seen in Algorithm 1, adapted from [27].

Algorithm 1 – Baum-Welch

Input: Observed sequence $\mathbf{y}$; $\theta = (T, E, \pi_0)$
Repeat until convergence criteria met
Forward Procedure:

1: $\alpha_i(0) \rightarrow \pi_i E_i(y_0)$
2: $\alpha_j(t+1) = E_j(a_{t+1}) \sum_{i=1}^{N} \alpha_i(t) T_{ij}$
3: return $\alpha_i(t) \rightarrow Pr(y_0 ... y_t, x_t = q_i | \theta)$ {The probability of observing $y_0 y_1 ... y_t$ and being in state q_i at time t}

Backward Procedure:

1: $\beta_i(T) = 1$
2: $\beta_i(t) = \sum_{j=1}^{N} \beta_j(t+1) T_{ij} E_j(y_{t+1})$
3: return $\beta_i(t) = Pr(y_{t+1} ... y_T | x_t = q_i, \theta$ {The probability of the ending partial sequence $y_{t+1} y_{t+2} ... y_T$ given that we start in state q_i at time t.}

Update the Parameters:

1: $\gamma_i(t) \rightarrow Pr(x_t = q_i | \mathbf{y}, \theta) = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)}$ {The probability of being in q_i at time t given $\mathbf{y}$}
2: $\xi_{ij}(t) \rightarrow Pr(x_t = q_i, x_{t+1} = q_j | \mathbf{y}, \theta) = \frac{\alpha_i(t) T_{ij} \beta_j(t+1) E_j(y_{t+1}}{\sum_{k=1}^{N} \sum_{l=1}^{N} \alpha_k(t) T_{kl} \beta_l(t+1) E_l(y_{t+1}} = \frac{\alpha_i(t) T_{ij} \beta_j(t+1) E_j(y_{t+1})}{\sum_{k=1}^{N} \alpha_k(t) \beta_k(t)}$
{The probability of being in state q_i at time t and state q_j at time $t+1$ given $\mathbf{y}$ and θ.}
3: $\pi_i^* \rightarrow \gamma_i(0)$
4: $T_{ij}^* = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}$ {This is the expected number of transitions from state q_i to state q_j divided by the total expected number of transitions from state q_i to any state.}
5: $E_i^*(y_k) \rightarrow \frac{\sum_{t=1}^{T} \mathbb{1}_{y_t = a_k} \gamma_i(t)}{\sum_{t=1}^{T} \gamma_i(t)}$ {This is the number of times that the symbol a_k appears after being in state q_j over the total number of times that we are in state q_j}
6: **return** π^*, T^*, E

The Baum-Welch algorithm is guaranteed to increase the likelihood $Pr(\mathbf{y}|\theta)$ upon every iteration; however, convergence to a global maximum is not guaranteed. Furthermore, some prior knowledge about the Hidden Markov model must be known in order to estimate the initial parameters [22]. The optimal path approximation, known as the Viterbi approximation, can be used instead of the true forward probabilities [2]. This results in an algorithm called the Viterbi reestimation algorithm, as discussed in [2].

2.2.2 CSSR Algorithm

In this thesis, we focus on the approach developed by Shalizi [1] for producing ϵ-machines from a stream of input data. We now focus on constructing a pFSA instead of an HMM. This work is extended in [28, 29] and [30].

Shalizi's approach to constructing a pFSA is to find statistically significant groupings of the training data which then correspond to causal states in the pFSA. In this formulation, each state is really an equivalence class of unique strings. This algorithm groups unique strings based on the conditional probabilities of the next symbol as a window slides over the training data. The length of the window, L, is called the history length. The window gradually increases in length up to a maximum length L, which is the maximum history length that contains predictive power for the process. This approach is called the Casual State Splitting and Reconstruction (CSSR) algorithm. The result of the CSSR algorithm is a Markov model where each state consists of a set of histories of up to length L that all share the same conditional probability distributions on the next symbol. The CSSR algorithm is shown in Algorithms 2-4. We use the same notation as in Section 2.2.1, however we now consider constructing a pFSA:

In Section 2.1.2, we defined a pFSA as a tuple $G = \langle Q, \mathcal{A}, \delta, p \rangle$, where Q is a finite set of states, $\mathcal{A}$ is a finite alphabet, $\delta \subseteq Q \times \mathcal{A} \times Q$ is a transition relation, and $p : \delta \rightarrow [0, 1]$ is a probability function such that

$$\sum_{a \in \mathcal{A}, q' \in Q} p(q, a, q') = 1 \quad \forall q \in Q \tag{2.2}$$

We now focus on the state transition relation and associated probabilities, rather than the symbol probability distribution for a given state.

Consider a function $\eta : \mathcal{A}^{l_1} \rightarrow \mathcal{F}_{\mathcal{A}}$, where $\mathcal{F}_{\mathcal{A}}$ is the set of probability distributions with support $\mathcal{A}$. This function gives the probability distribution over the set of symbolic outputs for each state. The function η also describes a pFSA whose states are composed of strings of length l_1 from $\mathcal{A}$. In order to understand the transition relation, we introduce a new shift function $\zeta(w_1, a)$ where $a \in \mathcal{A}$ and $w_1 = x_i x_{i+1} ... x_{i+l_1-1} \in \mathcal{A}^{l_1}$, as $\zeta(w_1, a) = w_2$ if and only if $w_2 = x_{i+1} x_{i+2} ... x_{i+l_1-1} a \in \mathcal{A}^{l_1}$. The transitions of η are defined so that if $w_1, w_2 \in \mathcal{A}^{l_1}$ and $w_2 = \zeta(w_1, a)$ for $a \in \mathcal{A}$, then $p(w_1, a, w_2) = \eta(w_1)(a)$. It is worth noting that this *may* not be the smallest statistically equivalent pFSA describing the behavior of η, however it is sufficient as a model of η.

Given a sequence $\mathbf{y}$ produced by a stationary process, the Causal State Splitting and Reconstruction (CSSR) Algorithm infers a set of causal states and a transition structure for a pFSA that provides a maximum likelihood estimate of the true underlying process dynamics. In this case, a causal state is a function mapping histories to their equivalence classes, as in [31].

The states are defined as equivalence classes of conditional probability distributions over the next symbol that can be generated by the process. The set of states found in this manner accounts for the deterministic behavior of the process while tolerating random noise that may be caused by either measurement error or play in the system under observation. The CSSR Algorithm has useful information-theoretic properties in that it attempts to maximize the mutual information among states and minimize the remaining uncertainty (entropy) [31].

The CSSR Algorithm is straightforward. We are given a sequence $\mathbf{y} \in \mathcal{A}^*$ and know a priori the value $L \in \mathbb{N}$.[1] For values of i increasing from 0 to L, we identify the set of sequences W that are subsequences of $\mathbf{y}$ and have length i. (When $i = 0$, the empty string is considered to be a subsequence of $\mathbf{y}$.) We compute the conditional distribution on each subsequence $\mathbf{x} \in W$ and partition the subsequences according to these distributions. These partitions become states in the inferred HMM. If states already exist, we compare the conditional distribution of subsequence $\mathbf{x}$ to the conditional distribution of an existing state and add $\mathbf{x}$ to this state if the conditional distributions are ruled identical. Distribution comparison can be carried out using either a Kolmogorov-Smirnov test or a χ^2 test with a specified level of confidence. The level of confidence chosen affects the Type I error rate. Once state generation is complete, the states are further split to ensure that the inferred model has a deterministic transition relation δ.

Algorithm 2 will produce the function η; it is similar to the initialization portion of the CSSR algorithm presented by Crutchfield and Shalizi [32, 33, 1]. Let $\#(\mathbf{w}, \mathbf{y})$ be the number of times sequence $\mathbf{w}$ occurs in $\mathbf{y}$. Also let $\#(\mathbf{w}, \mathbf{y}, a)$ be the number of times $\mathbf{w}$ occurs in $\mathbf{y}$ and a immediately follows $\mathbf{w}$.

Algorithm 2 Initialization

Input:

Observed sequence $\mathbf{y}$
Alphabet $\mathcal{A}$
Lengths L

Output:

Map η

Procedure:

1: Let W be the set of substrings of $\mathbf{w}$ with length L
2: **for all** $\mathbf{w} \in W$ **do**
3: Compute $\Pr(a \in \mathcal{A}|\mathbf{w}_1) := \frac{\#(\mathbf{w},\mathbf{y},a)}{\#(\mathbf{w},\mathbf{y})}$.
4: **end for**
5: Define $f_{\mathbf{w}}(a) := \Pr(a \in \mathcal{A}|\mathbf{w}_1)$.
6: Define $\eta(w) := f_{\mathbf{w}}$.
7: **return** η

In Line 1 of Algorithm 2 we compute the substrings needed. In Line 3, we produce the

[1] There are some heuristics for choosing L found in [31]

maximum likelihood estimator for the probability of observing output a given observation of input sequence $\mathbf{w}_1$ and output sequence $\mathbf{w}_2$ prior to the appearance of a. Finally, η is defined as the collection of these estimators.

Algorithm 2 produces a state space $Q = \mathcal{A}^{l_1}$, where states correspond to histories of length L. At state $q = \mathbf{w} \in Q$ input $a \in \mathcal{A}$ occurs and transfers the system to a new state $q' = \mathbf{w} = \zeta(\mathbf{w}, a)$. We will write $q' = \delta(q, a)$ to denote this relationship.

For any state $q \in Q$, $\eta(q) \in \mathcal{F}_{\mathcal{A}}$. The probability distributions can be thought of as a set of vectors in $[0, 1]^{|\mathcal{A}|}$, for a fixed ordering of $\mathcal{A}$.

In [34], Algorithm A provides a way of reducing this state space that can be useful for simplifying the optimization problem presented in the next section. The technique we present is identical to the one given in [34] and is drawn from the results of Shalizi and Crutchfield [1, 33, 32]. We present it in a manner more consistent with modern pattern recognition techniques.

The approach to minimizing the state space Q is to first cluster (merge) existing states that have statistically identical conditional distributions over $\mathcal{A}$ and then to de-cluster those states that would result in a non-deterministic state transition function δ.

Let $q_1, q_2 \in Q$. Define a metric $T(q_1, q_2)$. In [34] the metric was based on the p-value of the Kolmogorov-Smirnov test when applied to distributions $\eta(q_1)$ and $\eta(q_2)$. A χ^2 test [35], multinomial comparison test [36], or simple Euclidean distance could also be used. Let t_0 be a user provided threshold and let $p = \{q_{i_1}, \ldots, q_{i_n}\}$ be a cluster of states in Q. Define

$$\xi(p) = \frac{1}{n} \sum_{j=1}^{n} \xi(q_{i_j}) \tag{2.3}$$

We can then define $T(q, p)$ using $\xi(p)$ as expected. Clustering the states of Q into a new state space P can now proceed iteratively. Algorithm 3 shows this procedure.

Clearly the size of state space P is less than or equal to the size of state space Q. However, the resulting transition function δ may be non-deterministic.

To correct the non-determinism, we must split the states of P apart into the final (reduced) state space R. Splitting is a recursive operation in which we iterate through the states of P selecting an initial pair $(\mathbf{w}_1, \mathbf{w}_2) = q$ in some state p. We create a new state $r \in R$ with this q. We then analyze the remaining elements (original states) in p to determine whether they should be added to r or whether a new state r' should be created in R to deal with non-determinism in the transition function. This procedure is repeated recursively until no new states are added to R. Algorithm 4 summarizes the procedure.

Beginning with an initial symbolic input $\mathbf{y}$ the application of Algorithms 2-4 will create a reduced state space representation of the function η. It should be noted that it is sufficient to execute only Algorithm 2 to obtain a complete representation of the function η. If the state space is too large, then the additional algorithms can be executed to enhance processing later.

Work in [33, 1, 34] is sufficient to show that the resulting η representations are minimal in state size after executing Algorithms 2-4 as the number of samples approaches infinity. In

Algorithm 3 Merging

Input:

State Space Q
Threshold $t_0 \in \mathbb{R}$

Output:

State Space P

Procedure

$P := \emptyset$
for all $q \in Q$ **do**
  if $P \neq \emptyset$ **then**
    $p^* = \arg\min_{p \in P} T(q, p)$
    if $T(q, p^*) > t_0$ **then**
      $p_{\text{new}} = \{q\}$
      $P := P \cup \{p_{\text{new}}\}$
    else
      $p := p \cup \{q\}$
    end if
  else
    $p_{\text{new}} = \{q\}$
    $P := P \cup \{p_{\text{new}}\}$
  end if
end for

Chapter 3 we show how to correct this for smaller sample sizes and also show that the minimum state estimation problem is NP-hard in this case. State minimization is useful in managing the size of the optimal control problem as we see in the sequel.

The complexity of CSSR is $\mathrm{O}(k^{2L+1})+ \mathrm{O}(N)$, where k is the size of the alphabet, L is the maximum subsequence length considered, and N is the size of the input symbol sequence [31]. Given a stream of symbols $\mathbf{y}$, of fixed length N, from alphabet $\mathcal{A}$, the algorithm is linear in the length of the input data set, but exponential in the size of the alphabet.

2.3 The Markov Decision Processes

Markov Decision Processes (MDPs) addresses how to optimize behavior and model decision making in situations where there is partial uncertainty. They were first introduced in the 1960's to solve decision-making problems when there is a finite number of possible actions and multiple time periods [37]. While there are many types of MDPs, we are primarily interested in discrete time MDPs, which are also the simplest. A discrete time MDP consists of a finite state space Q, a set of possible actions A, a transition relation $\delta \subseteq Q \times \mathcal{A} \times Q$, a transition probability function $p : \delta \to [0, 1]$, a reward function $R : Q \to \mathbb{R}$. A policy or strategy is a rule that specifies an action to be taken at every state given the system's history. If the policy only depends on the previous state, then we can call this a Markov policy since it obeys the Markov property [38]. At every stage, the *controller* or decision-maker chooses an action $a \in A(q_t)$ which results in an immediate reward $r(q_t, a)$ [39].

Similar to Section 2.2.2, we define a strategy η as in [39] as a block row vector where row

Algorithm 4 Splitting

Input:

State Space P

Output:

State Space R

Procedure

```
1: N_0 := 0
2: N := |P|
3: repeat
4:    for all p_i ∈ P do
5:       s := 0
6:       M = |p_i| {The size of cluster p_i}
7:       Choose q_{i_0} ∈ p_i
8:       Create state r_{is} = {q_{i_0}}
9:       R := R ∪ {r_{is}}
10:      for all q_{ij} ∈ p_i (j ≠ 0) do
11:         FOUND = 0
12:         for l ∈ 0, ..., s do
13:            Choose q ∈ r_{il}
14:            if δ(q, a) = δ(q_{ij}, a) for all a ∈ A then
15:               FOUND = 1
16:               r_{il} := r_{il} ∪ {q_{i_j}}
17:            end if
18:         end for
19:         if FOUND = 0 then
20:            s := s + 1
21:            Create state r_{is} = {q_{i_j}}
22:            R := R ∪ {r_{is}}
23:         end if
24:      end for
25:   end for
26:   N_0 := N
27:   N : |R|
28: until N = N_0
```

$\eta(q_i) = (\eta(q_i, a_1), \eta(q_i, a_2), ...\eta(q_i, a_{m(q_i)})$ where $\sum_{j=1}^{m_{q_i}} \eta(q_i, a_j) = 1$. It is easy to see that $\eta(q_i, a_j)$ is simply the probability that the controller chooses action a_j at state q_i. As before, a strategy is called *deterministic* if $\eta(q, a) = \{0, 1\}$ for all $q \in Q$ and $a \in A$.

Given an initial state and strategy, we can let $\{R_t\}$ be the sequence of rewards. The cumulative reward of the whole process, which will be referred to as the value $v(\eta, q_0)$ is thus $v(\eta, q_0) = \sum_{t=0}^{\infty} r(q_t, a)$. For an infinite process this sum does not converge, so we will be interested in a discounted Markov decision process, Γ_β, where we define a discount parameter $\beta \in [0, 1]$ and we define the value as $v_\beta(\eta, q_0) = \sum_{t=0}^{\infty} \beta^t r(q_t, a)$. This notion of value leads to an intuitive definition of the "optimal control problem" as stated in [39] which is to "find, if possible, a strategy η^* that maximizes $v_\beta(\eta)$".

As in [39], we will define the value vector of the decision process Γ_β as $v_\beta := v_\beta(\eta^*) =$

$\max_\eta v_\beta(\eta)$ and present the conjectured optimality equation,

$$v_\beta(q) = \max_{a \in A(q)} \{r(q,a) + \beta \sum_{q'=1}^{N} p(q'|q,a) v_\beta(q')\} \tag{2.4}$$

where $v_\beta(q)$ is the discounted value starting from state q.

In [39] it is shown that any vector satisfying the following system of linear inequalities is an upper bound on the discounted value vector:

$$v(q) \geq r(q,a) + \beta \sum_{q'=1}^{N} p(q'|q,a) v(q') \tag{2.5}$$

for all $q \in Q$ and $a \in A(q)$. This suggests that the discounted value vector of Γ_β is the solution to the optimization problem

$$\min \sum_{i=1}^{N} \frac{1}{N} v(q_i) \tag{2.6}$$

subject to the constraints in 2.5. The coefficient $\frac{1}{N}$ is present to consider starting from any initial state with equal probability. Filar and Vriesz [39] also presents the dual to this linear program

$$\begin{cases} \max & \displaystyle\sum_{q \in Q} \sum_{a \in A(q)} r(q,a) x_{qa} \\ \text{s.t.} & \displaystyle\sum_{q \in Q} \sum_{a \in A} [\delta(q,q') - \beta \Pr(q'|q,a)]\, x_{qa} = \frac{1}{N} \quad \forall q' \in Q \\ & x_{qa} \geq 0 \quad \forall q \in Q, a \in A \end{cases} \tag{2.7}$$

Where the variables x_{qa} are the dual variables.

Filar and Vriesz [39] present the following main result related to the primal and dual linear programs,

Theorem 2.3.1. *(Value vector and optimal strategy)*

1. *The primal (2.6) and dual (2.7) Problems posses finite optimal solutions.*
2. *Let* $\mathbf{v}^0$ *be an optimal solution to Problem 2.6; then* $\mathbf{v}^0 = \boldsymbol{v}_\beta$*, the value vector of* Γ_β*.*
3. *Let* $\mathbf{x}^0 = \{x^0_{qa} | a \in A(q), a \in A\}$ *be an optimal solution to Problem 2.7 and define* $x^0_a := \sum_{a \in q(A)} x^0_{qa}$ *for each* $q \in Q$*; then* $x^0_a > 0$ *and the strategy* η^* *defined by*

$$\eta^0(q,a) := \frac{x^0_{qa}}{x^0_a} \quad \forall a \in A(q), q \in Q$$

is an optimal stationary strategy for process Γ_β*.*

and the following corollary:

Corollary 2.3.2. *(Validity of Optimality Equation)*

1. *The value vector* $\mathbf{v}_\beta$ *is the unique solution of the optimality equation 2.4*
2. *For each* $q \in Q$ *select any one action* $a_q \in A(q)$ *that achieves the maximum in 2.4. Define* η^* *by*

$$\eta^* = \begin{cases} 1 \text{ if } a{=}a_q \\ 0 \text{ otherwise} \end{cases} \tag{2.8}$$

for each $q \in Q$*. Then* η^* *is an optimal deterministic strategy in* Γ_β*.*

Thus, an optimal strategy can easily be found once the value vector has been determined by solving the dual problem 2.7. In order to find the value function, a family of algorithms known as "methods of successive approximation" can be used [39]. In Chapter 4 we will solve a version of the dual Problem in the context of a competitive two-player game.

2.4 Deception in Game Theory

The use of deception in strategic interactions creates an extra layer of strategy and complication. Burgoon and Buller define deception as a "deliberate act perpetrated by a sender to engender in a receiver beliefs contrary to what the sender believes is true to put the receiver at a disadvantage" [40]. This definition clearly includes a notion of intent, and thus accidental misinformation or incomplete knowledge may not be considered deception.

Bell and Whaley, in [41, 42], have done the majority of work developing a general theory and classification of deception. They classify deception into two categories–dissimulative and simulative. Dissimilative deception attempts to hide the true information, whereas simulative deception attempts to show false information. Dissimilative deception can further be classified into three subgroups:

- **Masking** attempts to hide a feature by blending it in with the background or avoiding detection
- **Repackaging** attempts to do the same by making the feature look like something else
- **Dazzling** occurs when the feature cannot be hidden, and thus the deceiver creates other distractions in order to confuse the receiver

Simulative deception can also be subcategorized into three groups:

- **Mimicking** occurs when a feature is made to appear to be something that it is not
- **Inventing** is when false information or features are displayed by creating an alternate reality rather than mimicking an existing reality, such as when the Greeks invented the Trojan horse

- **Decoying** is used to lure the receiver away from discovering the real information or feature.

Dissimilative and simulative deception can be thought of as opposites, as are all of the three subcategories. [41, 42]

In Chapter 6 of this thesis we apply the results of previous chapters to a deception scenario which utilizes the method of decoy. While we assume that the receiver is unaware and unsuspicious of deception taking place, there are many methods and theories for effectively discovering deception that we review. Johnson et al. [43] builds on the work by Mawby and Mitchell [44] by classifying strategies for detecting deception into two broad groups: methods that attempt to detect evidence of deceptive behavior, and methods that search for deception information. In order to detect deceptive behavior, there are two main strategies:

- **Motivation based** strategies look for indications and circumstantial evidence that the agent has a motivation to deceive. This strategy can only be effective if extensive knowledge about the agent is known. However, these attributes might be easier to detect than the actual deception itself.
- **Process based** strategies look for attributes in the environment that reveal the process of manipulation. This is subtly different than looking for the actual deception itself.

Methods for detection that focus on finding actual pieces of deception information can be categorized into four strategies:

- **Recognition based** strategies look for common deceptions, or deceptions that have occurred historically.
- **Conservative based** strategies avoid committing to a specific representation and suggest that the agent should "infer less, infer more soundly, or infer later in an information processing task" [43].
- **Preemption based** strategies attempt to limit the amount of deception that can occur by limiting the control that the agent has on the environment. This strategy limits the very potential for deception to occur.
- **Intentionally based** strategies focus on the goals of the actor and search for deceptions that would aid in achieving those goals. Thus, knowledge about the actor is required for this strategy.

Understanding deception and trust from a modeling perspective is becoming increasingly important as information is being created, collected, and analyzed at increasing rates. Being able to parse true information from deceptive or false information is a difficult yet important task. In [45], Santos and Johnson discuss the need for intelligent systems (AI systems), to be able to detect deception. Santos also draws a distinction between intention deception, or misinformation, and unintentional deception, such as incomplete knowledge [45]. In a multi-agent system, the introduction of misinformation can lead incorrect information entering the system, resulting in

wrong decisions being reached by the decision-maker [45]. In [46], George and Carlson found that humans can only detect deception slightly more than 50% of the time, and even less when the interaction is via electron media.

Evaluating the trustworthiness of online information is a developing area in deception theory. In [47], Norman develops a trust scale that online consumers can use to evaluate the trustworthiness of a site. In [48], Griffin and Moore develop a model for two decision making where the potential for deception exists. This model is most applicable to online networks where information is gathered and multiple sources and must be acted upon. They provide a framework for evaluating the truth of a given statement where one player provides information to the other [48].

In this thesis we develop a general model for deciding optimal deception strategies in simple two-player games. It differs from social network deception in that we assume there are only two players, and one player wishes to deceive the other. This type of model can be applied to competitive scenarios such as war-type games and strategy games.

Chapter 3

CSSR as an Integer Program

The majority of this chapter appears in [49]. In this chapter we follow the notation presented in Section 2.1.2.

3.0.1 A Practical Problem with CSSR

As observed in Shalizi and Shalizi (2004), the CSSR converges to the minimum state estimator asymptotically [31], however it does not always result in a correct minimum state estimator for finite a finite sample. With an infinitely long string, all of the maximum likelihood estimates converge to their true value and the CSSR algorithm works correctly. That is, as $N \to \infty$, the probability of a history being assigned to the wrong equivalence class approaches zero. A formal proof of this is given in [1] and relies on large deviation theory for Markov chains.

The error in estimating the conditional probabilities with strings of finite length can cause the CSSR algorithm to produce a set of causal states that is not minimal.

The following example will clarify the problem. Consider $\mathcal{A} = \{0, 1\}$ and $\mathbf{y}$ defined as follows:

$$\begin{aligned} y_i &= 0 \quad 1 \leq i \leq 1818 \\ [y_{i-3}\ y_{i-2}\ y_{i-1}\ y_i] &= 1100 \quad 1818 < i \leq 2218 \\ [y_{i-2}\ y_{i-1}\ y_i] &= 100 \quad 2218 < i \leq 2281 \\ [y_{i-2}\ y_{i-1}\ y_i] &= 101 \quad 2281 < i \leq 2284 \end{aligned} \tag{3.1}$$

Without loss of generality, we consider exclusively strings of length two. The strings, in order of appearance, are $\{00, 01, 11, 10\}$. The conditional probability distribution on the next element for each string is shown in the following table:

And the following p-values associated with comparing each string's conditional probability distribution to that of every other string:

We now step through the initialization and splitting phases of the CSSR Algorithm, which is presented as Algorithms 2 and 3:

string	$Pr(0\|\text{string})$	$Pr(1\|\text{string})$
00	0.937	0.063
01	0.180	0.820
11	1.000	0.000
10	0.992	0.008

Table 3.1: Conditional Probabilities

string	00	01	11	10
00	1.0000	–	–	–
01	0.0000	1.0000	–	–
11	0.0096	0.0000	1.0000	–
10	0.0131	0.0000	0.3628	1.0000

Table 3.2: p-values

1. String 00 is put into its own state, q_1
2. We use z-test of proportions to test if the 01 and 00 have the same conditional distribution. This results in a p-value of approximately 0, as seen Table 3.2 so string 01 is put into a new state, q_2
3. We compare the conditional distribution of 11 to string 00. This results in a p-value of 0.0096, and comparing it to 01 results in a p-value of 0, so string 11 is put into a new state, q_3
4. We compare the conditional probability distribution of 10 to that of 00. This results in a p-value of 0.0131. Comparing it to 01 results in a p-value of 0, and comparing it to 11 results in a p-value of 0.3628. Since 0.3628 is greater than 0.0131 and greater than the chosen significance level of 0.01, the string 10 is clustered with string 11, so $q_3 = \{11, 10\}$.

Thus, at the end of the Splitting step of CSSR, we are left we three different states:

$$q_1 = \{00\}$$
$$q_2 = \{01\}$$
$$q_3 = \{11, 10\}$$

We now go on to the Reconstruction step, which is presented in Algorithm 4. Let $\mathbf{x}_{ij}$ be a state in q_i where $j \in 1, 2, ..., |q_i|$. The Reconstruction step checks whether, for each $a \in \mathcal{A}$, $\delta(\mathbf{x}_{i1}, a) = \delta(\mathbf{x}_{i2}, a) = ... = q_k$. Recall that $\delta(\mathbf{x}_{ij}, a) = q_k$ means that $\mathbf{x}_{ij}a$ produces a sequence that resides in state q_k. If this is not satisfied, then state q_i must be broken up into two or more states until we have a deterministic transition relation function.

In our example, the first two states do not need to be checked since they each only consist of one string. We check the third state: $\delta(11, 0) = q_3$ since string $10 \in q_3$, and $\delta(10, 0) = q_1$ since string $0 \in q_1$. Thus, determinism is not satisfied and state q_3 must be split into two states. The

result of the CSSR algorithm is a four-state Markov chain where each string resides in its own state.

Now we will show why this result is not the minimal state Markov generator for this input sequence. Suppose that during the Splitting step, string four was put into state q_1 instead of q_3. This could have been done soundly from a statistical point of view, since 0.0131 is also greater than our cut-off level of 0.01. However, by the CSSR heuristic, the fourth string was grouped according to the *highest* p-value. If the fourth string were put in q_1, then after the Splitting step we would have gotten the following states:

$$\begin{aligned} q_1 &= \{00, 10\} \\ q_2 &= \{01\} \\ q_3 &= \{11\} \end{aligned}$$

Now we move on to the Reconstruction step. This time, we only need to consider state q_1. Notice that $\delta(10, 0) = \delta(00, 0) = q_1$ and $\delta(00, 1) = \delta(10, 1) = q_2$. We see that state q_1 does satisfy determinism, and does not need to be split. Thus, our final number of states is only three, which is minimal. This provides us with motivation to reformulate the CSSR algorithm so that it always produces a the minimal number of states, even when given a small sample size. This paper seeks to address this issue and propose a reformulation of the CSSR algorithm which always results in minimal state models even with a finite sample.

3.1 Integer Programming Formulation

Much of this section has been adapted from Cluskey (2013) [50]. Suppose we have a string $\mathbf{y} \in \mathcal{A}$ and we assume that L is known. Let W be the set of distinct strings of length L in $\mathbf{y}$, and let $|W| = n$. When we cluster the elements of W into states we must be sure of two things: 1) That all strings in the same state have the same conditional distributions, and 2) that the resulting transition function is deterministic. Our goal is to formulate an integer programming problem that minimizes the number of states and preserves the two conditions above. Define the following binary variables:

$$x_{ij} = \begin{cases} 1 & \text{string } i \text{ maps to state } j \\ 0 & \text{else} \end{cases} \tag{3.2}$$

Remark 1. We assume implicitly that there are n states, since n is clearly the maximum number of states that could be needed. If, for some j, $x_{ij} = 0 \; \forall \; i$, then that state is unused, and our true number of states is less than n.

Let:

$$z_{il}^{\sigma} = \begin{cases} 1 & \text{string } i \text{ maps to string } j \text{ upon symbol } \sigma \\ 0 & \text{else} \end{cases} \tag{3.3}$$

For example, $z_{il}^{\sigma} = 1$ if $i = \langle x_1, ..., x_L \rangle$ and $l = \langle x_2, ..., x_L, \sigma \rangle$. Assuming j and k are state indices while i and l are string indices, we also define y_{jk}^{σ} as

$$(x_{ij} = 1) \wedge (z_{il}^{\sigma} = 1) \wedge (x_{lk} = 1) \implies (y_{jk}^{\sigma} = 1)$$

This variable ensures the encoded transition relation maps each state to the correct next state given a certain symbol. Specifically, if string i in clustered in state j and string i transforms to string l given symbol σ, and string l is clustered in state k, then (j, σ, k) must be the transition relation. This can be written as the constraint

$$(1 - x_{ij}) + (1 - z_{il}^{\sigma}) + (1 - x_{lk}) + y_{jk}^{\sigma} \geq 1 \quad \forall i, j, k, l, \sigma$$

To achieve determinism, we must have the condition that

$$\sum_{k} y_{jk}^{\sigma} \leq 1 \quad \forall j, \sigma$$

This condition ensures that for a given symbol, each state can transition to only one other state.

We ensure that the strings in each state have the same conditional probability distributions using a parameter μ, where

$$\mu_{il} = \begin{cases} 1 & \text{strings } i \text{ and } l \text{ have identical conditional probability distributions} \\ 0 & \text{else} \end{cases} \tag{3.4}$$

In order to determine if two strings have identical distributions an appropriate statistical test (like Pearson's χ^2 or Kolmogorov-Smirnov) must be used. The variables must satisfy

$$(x_{ij} = 1) \wedge (x_{lj} = 1) \implies (\mu_{il} = 1)$$

This states that strings i and l can only belong to the same state if they have statistically indistinguishable distributions. This can be written as the following constraint

$$(1 - x_{ij}) + (1 - x_{lj}) + \mu_{il} \geq 1 \quad \forall i, l, j$$

Finally, we define the variable p to be used in our objective function. Referring back to the remark under 3.2, this variable simply enumerates up the number of "used" states out of n. That is:

$$p_j = \begin{cases} 0 & \sum_i x_{ij} = 0 \\ 1 & \text{else} \end{cases} \tag{3.5}$$

This can be enforced as the following constraint

$$p_j \geq \frac{\sum_i x_{ij}}{n}, \; p_j \in \{0, 1\}$$

Note that $\frac{\sum_i x_{ij}}{n} \leq 1$. These constraint are equivalent to Equation 3.5 because when $\sum_i x_{ij} = 0$, p_j will be 0, and when $\sum_i x_{ij} > 0$, p_j will be 1. We will minimize $\sum_j p_j$ in our optimization problem. This addition extends the work done in [50], in which a series of optimization problems had to be solved.

Definition 1 (Minimum State Deterministic pFSA Problem)**.** The following binary integer programming problem, which, if feasible, defines a mapping from strings to states and a probability transition function between states which satisfies our two requirements.

$$P(N) = \begin{cases} \min \ \sum_j p_j \\ s.t. \ (1 - x_{ij}) + (1 - z_{il}^{\sigma}) + (1 - x_{lk}) + y_{jk}^{\sigma} \geq 1 \quad \forall i, j, k, l, \sigma \\ \quad \sum_k y_{jk}^{\sigma} \leq 1 \quad \forall j, \sigma \\ \quad (1 - x_{ij}) + (1 - x_{lj}) + \mu_{il} \geq 1 \quad \forall i, l, j \\ \quad p_j \geq \frac{\sum_i x_{ij}}{S} \ \ \forall i, j \\ \quad \sum_j x_{ij} = 1 \ \ \forall i \end{cases} \tag{3.6}$$

and is called the Minimum State Deterministic pFSA (MSDpFSA) Problem.

The following proposition is clear from the construction of Problem 3.6:

Proposition 3.1.1. *Any optimal solution to MSDpFSA yields an encoding of a minimum state probabilistic finite machine capable of generating the input sequences with statistically equivalent probabilities.*

Remark 2. Because this formulation does not rely on the assumption of infinite string length to correctly minimize the number of states, it succeeds where the CSSR algorithm fails. Instead of clustering strings into the state with the *most* identical conditional probability distributions, this optimization problem uses the identical distributions as a condition with the *goal* of state minimization. Thus, in the example, even though the fourth string had a higher p-value when compared to the third state than the first state, since both of the p-values are great than 0.05 this algorithm allows the possibility of clustering the fourth string into either state, and chooses the state which results in the smallest number of final states after reconstruction. Unlike CSSR, this algorithm does not rely on asymptotic convergence of the conditional probabilities to their true value. However, it is clear that as the sample size increases the probability distributions will become more exact, leading to better values of μ.

3.1.1 Resolution to the Example Problem

Using the integer program formulation to solve the example previously presented does result in the minimum state state estimator of the process represented by the given input string. The

Table 3.3: Integer Program Variables

(a) x values

–	q_1	q_2	q_3	q_4
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	1	0	0	0

(b) u values

–	00	01	11	10
00	1	0	0	1
01	0	1	0	0
11	0	0	1	1
10	1	0	1	1

(c) z^0 values

–	00	01	11	10
00	1	0	0	0
01	0	0	0	1
11	0	0	0	1
10	1	0	0	0

(d) z^1 values

–	00	01	11	10
00	0	1	0	1
01	0	0	1	0
11	0	0	1	0
10	0	1	0	0

(e) y^0 values

–	q_1	q_2	q_3
q_1	1	0	0
q_2	1	0	0
q_3	1	0	0

(f) y^1 values

–	q_1	q_2	q_3
q_1	0	1	0
q_2	0	0	1
q_3	0	0	1

(g) p values

1	1	1	0

Table 3.4: State Transition Probabilities

–	q_1	q_2	q_3
q_1	0.9341	0.0659	0
q_2	0.5789	0	0.4211
q_3	1	0	0

integer program finds the following solution for the variables z, x, y, u, and p seen in Table 3.3. Our objective function is to minimize the sum of the variables p_j. As can be seen in Table 3.3, there is only one nonzero column in the x matrix (the last column). Thus, only $p_4 = 0$, and the the value of our objective function is 3.

We can also recover a matrix of transition probabilities shown in Table 3.4. Our final reduced state space machine is shown in Figure 3.1.

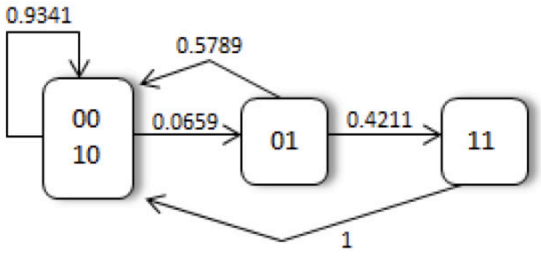

Figure 3.1: The reduced state space machine derived from the input given by Equation 3.1

3.2 Computational Complexity of MSDpFSA

In this section we assert that the integer program given by Problem 3.6 is NP-hard by reducing a less complex problem to the minimal clique covering problem. Recall given a graph, a minimal clique covering problem is to identify a set of cliques in the graph so that each vertex belongs to at least one clique and so that this set of cliques is minimal in cardinality.

While the CSSR algorithm attempts to find a minimal state *deterministic* finite-state automata, determinism is not a necessary property for accurate reconstruction. For a given input sequence, the construction of a finite-state automata which is *not* necessarily deterministic is less computationally intensive than solving the MSDpFSA problem, and is discussed by Schmiedekamp et al. (2006) [51]. Like the CSSR algorithm, the algorithm presented in [51], called the CSSA algorithm, is a heuristic which does not always results in a minimal state probabilistic FSA. The following definition provides an integer program for optimally solving a minimal state pFSA which is not necessarily deterministic. This is identical to Problem 3.6 except that we discard the constraints that ensure determinism.

Definition 2 (Minimum State Non-Deterministic pFSA Problem)**.** The following binary integer programming problem, which, if feasible, defines a mapping from strings to states and a probability transition function between states which is not necessarily deterministic.

$$P'(N) = \begin{cases} \min \ \sum_j p_j \\ s.t. \ (1 - x_{ij}) + (1 - x_{lj}) + \mu_{il} \geq 1 \quad \forall i, l, j \\ \quad p_j \geq \frac{\sum_i x_{ij}}{S} \ \ \forall i, j \\ \quad \sum_j x_{ij} = 1 \ \ \forall i \end{cases} \tag{3.7}$$

and is called the Minimum State Non-Deterministic pFSA (MSNDpFSA) Problem.

Proposition 3.2.1. *MSNDpFSA is NP-Hard.*

Proof. Let $G = (V, E)$ be the graph on which we want to find the minimal clique covering. Assume that $V = \{1, 2, ..., n\}$ and let I be a $n \times n$ matrix such that $I_{ij} = 1 \iff$ there is an edge connecting vertices i and j. We reduce Problem 3.7 by letting n be the number of unique strings of length L in $\mathbf{y}$, so we can let each string correspond to a vertex of G. Let $I_{ij} = 1 \iff \mu_{ij} = 1$. This means that two strings are connected if and if only if they have identical conditional probability distributions. We show that Problem 3.7 is equivalent to finding a minimal clique cover of G where $\sum_j p_j$ is the number of cliques. Let the set of cliques corresponding to the minimal clique cover of G be $C = \{C_1, ..., C_m\}$, where C_j is a specific clique, and $C_j = V_j \subset V$.

We can define the variables x by $x_{ij} = 1 \iff V_i \in C_j$. Thus, the constraint that $(1 - x_{ij}) + (1 - x_{lj}) + \mu_{il} \geq 1 \quad \forall i, l, j$ simply means that if two vertices are in the same clique then there

must be an edge between them. We also have that $p_j = 0$ iff there is at least one vertex in clique j, so the set of j such that $p_j = 1$ corresponds to the non-empty cliques, i.e., is identical to C (since C only consists of non-empty cliques). Thus, minimizing $\sum_j p_j$ is equivalent to minimizing the number of cliques needed to cover G. The constraint that $\sum_j x_{ij} = 1 \ \ \forall i$ simply means that each vertex belongs to exactly one clique. Thus we see that, given a sequence, finding a non-deterministic minimal finite-state machine is equivalent to a finding minimal clique covering of the corresponding graph.

Furthermore, we show that any arbitrary graph corresponds to a sequence of finitely many characters, such that the graph formulation of the string is identical to the given arbitrary graph. Recall that the graph formulation of a sequence considers all strings of length L to be the vertices, and connects two vertices if and only if they have identical conditional probability distributions.

We demonstrate this construction through an example. Suppose we are given the graph shown in Figure 3.2

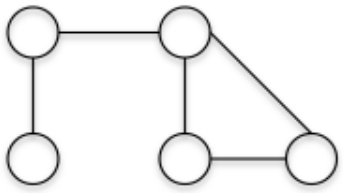

Figure 3.2: An example arbitrary graph

Since there are five vertices, we will use the characters $\{a, b, c, d, e\}$ along with $\{0, 1\}$ to constitute our alphabet. We will also have $L = 1$. Consider the sequence

$$a0a0...a0a1b0b0...b0b1c0c0...c0c1d0d0...d0d1e0e0...e0e1$$

where there are n_a $a0$'s following by one $a1$, n_b $b0$'s followed by one $b1$, etc. It is clear that we can manipulate each n_x so that we get conditional probability distributions similar to Table 3.5.

string	$Pr(0\|\text{string})$	$Pr(1\|\text{string})$
a	0.99	0.01
b	0.98	0.02
c	0.97	0.03
d	0.94	0.06
e	0.91	0.09

Table 3.5: Conditional Probabilities

We could arbitrarily alter these distributions in order to end up with a graph that looks like Figure 3.3 depending on the significance level that we choose in order to compare the probability distributions. The characters 0 and 1 could be connected to each other, but will never be connected to the graph containing the letters because they have completely different probability distributions (the string 0 and 1 always map to a letter, whereas the letters always map to a

number).

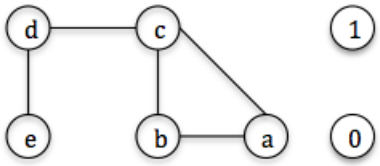

Figure 3.3: Graph formulation of the example string

Thus, any non-deterministic minimal finite-state machine corresponding to the sequence would also give a minimal clique covering of the original graph. This construction can easily be extended to any arbitrary graph. Thus, it is clear that the integer program given in Problem 3.7 is equivalent to a minimal clique covering and is NP-hard.

□

Remark 3. A proof of the NP-hardness of Problem 3.7 would be similar to the proof of Proposition 3.2.1, except it would include the deterministic element of the resulting probabilistic finite state machine. This does not add substantially to the proof and the computational complexity of the problem should not decrease when passing to the deterministic case because of the added degree of complexity due to the enforcement of determinism.

3.2.1 Example of Reduction

We will illustrate the equivalence of Equation 3.7 and the minimal clique covering through an example. Using the same example as before, the resulting graph G is shown in Figure 3.4. The string 00 and 10 have an edge in common because they have identical conditional distributions as noted by Table 3.2. By the same reasoning, 10 and 11 also have an edge in common. However, 00 and 11 do not share an edge, and 01 has no incident edges.

The integer program given by Equation 3.7 produces either of the following two results:

$$Q = \{q_1 = \{00, 10\}, q_2 = \{11\}, q_3 = \{01\}\} \text{ or}$$
$$Q' = \{q_1 = \{11, 10\}, q_2 = \{00\}, q_3 = \{01\}\}$$

Both of these two groupings results in one of two minimal clique coverings. Let $V_1 = 00,\ V_2 = 10,\ V_3 = 11,\ V_4 = 01$.

$$C = \{C_1 = \{1, 2\},\ C_2 = \{3\},\ C_3 = \{4\}\} \text{ or}$$
$$C' = \{C_1 = \{2, 3\},\ C_2 = \{1\},\ C_3 = \{4\}\}$$

When the determinism constraint is added to the integer program, the result is Q' (or equivalently C') instead of Q (or C). The solver recognizes that Q (*or* C) would have to be split in order for determinism to be satisfied, so Q' (*or* C') is chosen instead. If we think of the

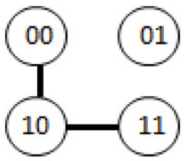

Figure 3.4: The graph G corresponding to the string y given in Equation 3.1.

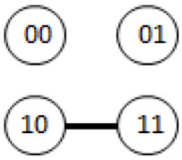

Figure 3.5: The graph G considered by the CSSR algorithm

.

CSSR algorithm in terms of our graph equivalency, the CSSR algorithm does not consider the existence of an edge between 00 and 10. Let T be Table 3.2. For a vertex i, CSSR only places an edge between i and $\text{argmax}_{j \neq i}\{T(i,j) : T(i,j) > 0.05\}$.Thus, by the CSSR algorithm, the graph is really the image shown in Figure 3.5. In this formulation, there is only one minimal state clustering (minimal clique covering), G (*or* C) which does not satisfy determinism, so G is split into four states.

3.3 Minimal Clique Covering Formulation of CSSR Algorithm

In this section we present an algorithm for determining the minimal state hidden Markov model using a minimal clique vertex covering reformulation. As shown in the previous section, the two problem formulations are equivalent if we exclude the determinism constraint. Let $\mathbf{W} = \{S_1, ..., S_n\}$ be the set of unique strings of length L in $\mathcal{A}$, so W is the set of vertices in our graph formulation. In the revised algorithm, we first use the CSSR algorithm, Algorithms 2-4, to find an upper bound on the number of cliques needed. We then use Bron-Kerbosch algorithm to enumerate all maximal cliques, $C = \{C_1, C_2, ..., C_m\}$, given as Algorithm 5 [52].

Define $H := \text{argmin}_R\{|R| : R \subset \mathbf{C} \ \wedge \ \cup_i R_i = W\}$. Thus H is the set of maximal cliques of minimal cardinality such that every vertex belongs to at least one clique. This can be found using a simple binary integer program given in Algorithm 6. We can think of Algorithm 6 as defining a mapping from every string to the set of all maximal cliques. A clique is "activated" if at least one string is mapped to it. The goal, then, is to activate as few cliques as possible. Of course, each string can only be mapped to a clique in which it appears. We define a few variables, which are present in our integer program:

$$y_i = \begin{cases} 1 & C_i \in H \\ 0 & \text{else} \end{cases} \tag{3.8}$$

The variable y can be thought of as whether or not clique C_i is activated.

$$I_{ij} = \begin{cases} 1 & S_i \in C_j \in H \\ 0 & \text{else} \end{cases} \tag{3.9}$$

$$s_{ij} = \begin{cases} 1 & S_i \text{is mapped to } C_j \in H \\ 0 & \text{else} \end{cases} \tag{3.10}$$

To distinguish between I and s, notice that each string is only mapped to one clique, but each string could appear in more than one clique.

Proposition 3.3.1. *The set H is a minimal clique vertex covering.*

Proof. Suppose there is a minimal clique vertex covering of a graph G with k cliques such that every clique is not maximal. Choose a non-maximal clique and expand the clique until it is maximal. Continue this procedure until every clique is maximal. Let our set of cliques be called R. Clearly $|R| = k$ since no new cliques were created. Also note that $R \subset C$ since R consists of maximal cliques. Further, $\cup_i R_i = W$ since we started with a clique vertex covering. Finally, notice that $R = \text{argmin}_H\{|H| : H \subset \mathbf{C} \ \wedge \ \cup_i H_i = W\}$ because $|R| = k$ and k is the clique covering number of G, so it is impossible to find an H with cardinality less than k. □

Algorithm 5 –Finding all maximal cliques

Input: Observed sequence $\mathbf{y}$; Alphabet $\mathcal{A}$, Integer L
Setup:
Call the set of strings that appear in $\mathbf{y}$ $\mathbf{W} = \{S_1, ..., S_n\}$, so $\mathbf{W}$ is our set of vertices
Determine $f_{\mathbf{x}|\mathbf{y}}(a)$ for each unique string
Generate an incident matrix, $\mathbf{U} \in \mathbb{R}^{nxn}$, $\mathbf{U}_{ij} = u_{ij}$, where u_{ij} is defined in 3.4
$P \leftarrow [1, 2, ..., n]$ {P holds the prospective vertices connected to all vertices in R}
$R \leftarrow \emptyset$ {Holds the currently growing clique}
$X \leftarrow \emptyset$ {Holds the vertices that have already been processed}
function Bron-Kerbosch(R,P,X)
$k \leftarrow 0$
if $P = \emptyset$ and $X = \emptyset$ **then**
    report R as a maximal clique, $R = C_k$
    $k \leftarrow k + 1$
end if
for each vertex v in P **do**
    Bron-Kerbosch($R \cup v, P \cap N(v), X \cap N(v)$) {where $N(v)$ are the neighbors of v}
    $P \leftarrow P \setminus v$
    $X \leftarrow X \cup v$
end for
return $\{C_1, C_2, ..., C_m\}$

Proposition 3.3.2. *Any solution to Q(C) results in a minimal clique vertex covering.*

Algorithm 6 –Minimal Clique Vertex Covering

Input: Observed sequence $\mathbf{y}$; Alphabet $\mathcal{A}$, Integer L
Set-up:
1. CSSR
   run the CSSR algorithm (Algorithms 2-4)
   return k, the number of cliques found by CSSR
2. Find all maximal cliques
   run Algorithm 5
   return $\{C_1, C_2, ..., C_m\}$

Finding all minimal clique coverings:
Let $I_{ij} = 1$ denote string S_i belonging to clique C_j
Let $s_{ij} = 1$ denote string S_i being mapped to clique C_j

$$Q(C) = \begin{cases} \min \sum_i y_i \quad \{y_i \text{ indicates whether clique } C_i \text{ is being used or not}\} \\ s.t. \;\; y_i \geq \dfrac{\sum_j s_{ij}}{|C_j|} \\ \qquad y_i \leq \sum_j s_{ij} \\ \qquad \sum_i y_i < k \\ \qquad s_{ij} \leq I_{ij} \\ \qquad \sum_j s_{ij} = 1 \end{cases}$$

Proof. This is clear by noting that $\operatorname{argmin} Q(C)$ results in the subset, H, of $C = \{C_1, C_2, ..., C_m\}$ where H is defined as $\operatorname{argmin}_R\{|R| : R \subset \mathbf{C}\ and\ \cup_i R_i = W\}$ in conjunction with Proposition 3.3.1. □

Before discussing the rest of the algorithm, which concerns determinism, we first state an important remark about Algorithm 6. It is possible that the set H is such that some vertices are in more than one maximal clique. However, we are actually interested in the set of minimal clique vertex coverings for which each vertex only belongs to *one* clique, which can be extracted from H. See Figures 3.6 and 3.7 for an example. The set H is shown in Figure 3.6. From this set H, we can deduce the two minimal clique vertex coverings shown in 3.7. In initializing Algorithm 7, which imposes determinism on each covering, we find the set of all minimal coverings for which each vertex belongs to one clique.

Once we have determined the set of minimal clique covers of our original graph where each vertex only belongs to one clique, we select a final clique covering which is minimal and deterministic. This is done by considering each minimal clique covering individually and then restructuring it when necessary to be deterministic. This corresponds to Algorithm 4, the reconstruction portion of the CSSR algorithm. Note in Algorithm 7 each vertex corresponds to a string of length L, which came from an initial string $\mathbf{y}$. Also recall that we have previously defined the binary variables $\mathbf{z}$ as

$$z_{il}^{\sigma} = \begin{cases} 1 & \text{string } i \text{ maps to string } j \text{ upon symbol } \sigma \\ 0 & \text{else} \end{cases}$$

Algorithm 7 – Minimal Clique Covering Reconstruction

Input: Observed sequence $\mathbf{y}$; Alphabet $\mathcal{A}$, Integer L;
Set-up:

Perform Algorithm 5 on $\mathbf{y}$ and obtain a minimal clique covering
From this initial clique covering, find the set of all minimal coverings such that each vertex belongs to exactly one clique, an example of which is seen in Figure 3.7. Let this set of minimal coverings be $\mathbf{T} = \{T_1, ..., T_l\}$
From $\mathbf{y}$, $\mathcal{A}$, and the set $\{S_1, ..., S_n\}$ as found in Algorithm 5, determine the matrix z.

Minimal Clique Covering Reconstruction:

for $h = 1$ **to** l **do**
  $i = 1$
  $N =$ the number of cliques in each T_h {n}ote that all T_h have the same number of cliques because they are all minimal
  $M = N$
  while $i \leq M$ **do**
    $N = M$
    $F =$ the set of all vertices (strings) which are in clique i
    if length$(F) > 1$ **then**
      for $j = 2, 3, ..., \#\{F\}$ **do**
        Use $\mathbf{z}$ to find what clique $F(j)$ maps to for each $\sigma \in \mathcal{A}$
        if $F(j)$ does not map to the same clique as $F(1)$ upon each σ **then**
          if $M > N$ and $F(j)$ maps to the same clique as some $F(k)$ upon each σ, $k \in [N+1, M]$ **then**
            Add $F(i)$ to the clique containing $F(k)$
          else
            Create a new clique containing $F(i)$
            $M = M + 1$
          end if
        end if
      end for
    end if
    $i = i + 1$
  end while
end for
FinalCovering = min$\{C_h | C_h$ has the minimum number of columns $\forall h\}$
return FinalCovering

where σ is any element of our alphabet $\mathcal{A}$. The following proposition is clear from the construction of Algorithms 6 and 7.

Proposition 3.3.3. *Algorithm 6 along with Algorithm 7 produces an encoding of a minimum state probabilistic finite machine capable of generating the input sequence.*

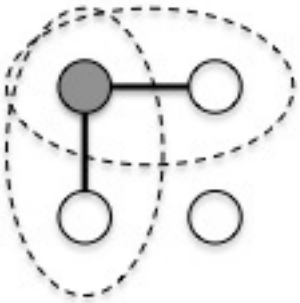

Figure 3.6: An example output from Algorithm 6

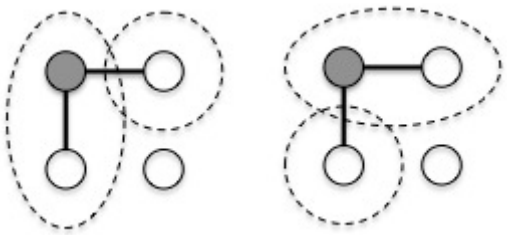

Figure 3.7: Minimal clique coverings where each vertex is only in one clique

3.4 Comparing run times of modified CSSR algorithms

This paper has discussed three different approaches for determining minimal state hidden Markov models from a given input sequence: the CSSR algorithm, CSSR as an integer program, and the minimal clique covering reformulation. We now compare the run times of all three algorithms. We find that the CSSR algorithm has the fastest run time, however it does not always produce a minimal state model as we have seen in a prior example. The minimal clique covering reformulation is relatively fast and always results in a minimal state model. The integer program reformulation is extremely slow, but also always results in a minimal state model. This makes CSSR a useful heuristic for solving the NP-hard MSDpFSA problem.

The CSSR integer program, using only a two-character alphabet and sequence of length ten as input, takes about 100 seconds to run. We can see in Figure 3.8 that the minimal clique covering reformulations takes less than 0.2 seconds for sequences up to length 10,000. For only two character alphabets, the run time appears to be a nearly linear function of the sequence length, with spikes occurring presumably in cases where more reconstruction was needed. In Figure 3.8 we see the run time of the minimal clique covering reformulation for a three-character alphabet. For sequences up to length 100, the algorithm took no more than 120 seconds, which is remarkably better than the integer program. It is much less linear than shown in Figure 3.8 because reconstruction is needed more frequently for three characters.

For two-, three-, and 4-character alphabets, we also compare the minimal clique covering formulation time to that of the original CSSR algorithm. This can be seen in Figure 3.9 where we take the average of the run times for the CSSR and clique covering formulation to compare the two. Note that with a two-character alphabet, the clique covering formulation is slightly faster, but for the three- and four-character alphabets the CSSR algorithm is significantly faster. This is due to the fact that the CSSR algorithm does not actually guarantee a minimal state Markov model.

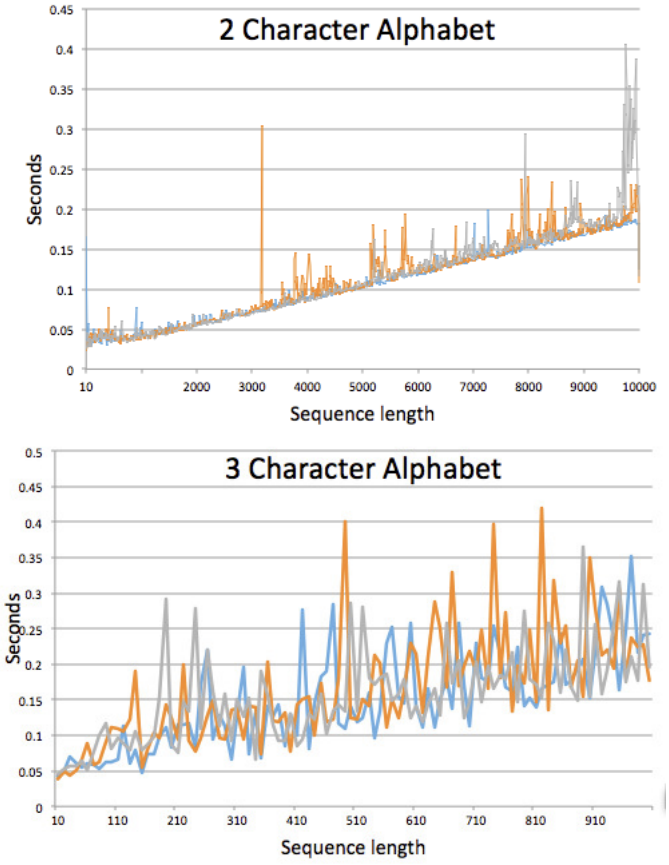

Figure 3.8: Clique covering reformulation times

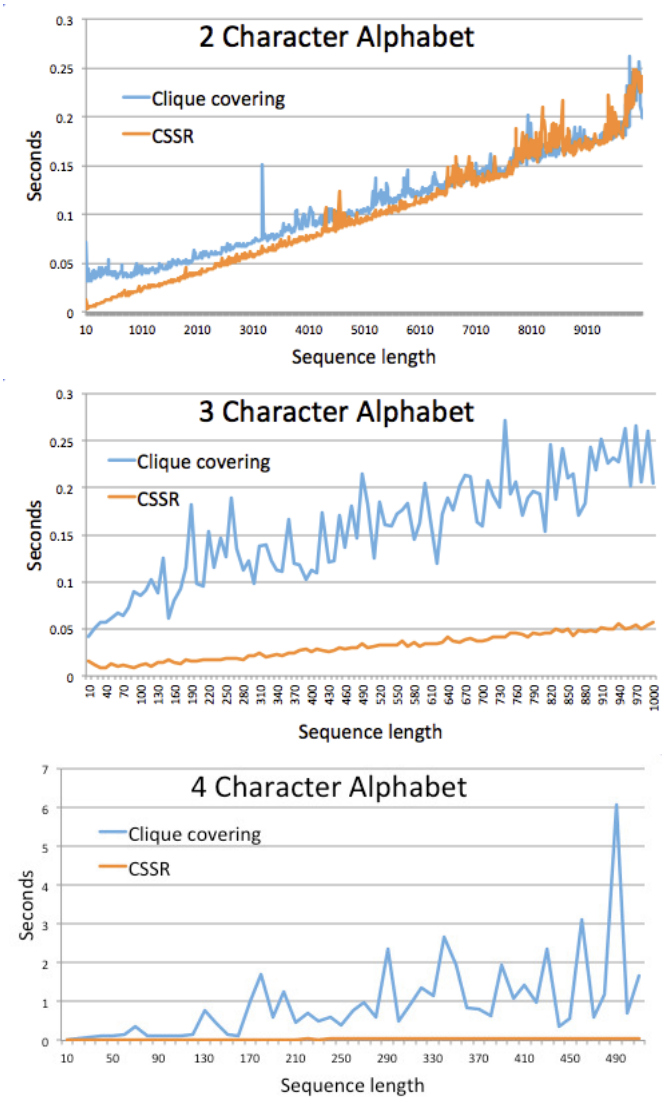

Figure 3.9: Minimal clique covering run times compared to CSSR

Chapter 4

Optimal Process Control of Symbolic Transfer Functions

4.1 Introduction

This chapter extends the discussion in Sections 2.2.2 and 2.3. The majority of this chapter appears in [53]. We now discuss Mealey machines, which are very similar to pFSA except that we now have an input and an output alphabet, instead of just an input alphabet. A *polygenic Mealy machine* is a tuple $M = \langle Q, \mathcal{A}, \mathfrak{A}, \delta \rangle$ where Q and $\mathcal{A}$ are as above and $\mathfrak{A}$ is a second *output* alphabet and $\delta \subseteq Q \times \mathcal{A} \times \mathfrak{A} \times Q$ is a transition relation with input alphabet $\mathcal{A}$ and output alphabet $\mathfrak{A}$. Determinism of the transition relation is defined just as it was for a PFA. If we assign a probability function to the transition relation (as we did for probabilistic PFA), then we obtain a probabilistic Mealy machine. (Note for the sake of brevity, we remove the term polygenic.)

The function $\xi : \mathcal{A}^{l_1} \times \mathfrak{A}^{l_2} \rightarrow \mathcal{F}_{\mathfrak{A}}$, describes a *formal* probabilistic Mealey machine whose states are composed of pairs of strings in $\mathcal{A}^{l_1} \times \mathfrak{A}^{l_2}$ and whose transitions are defined so that if $(w_1, v_1), (w_2, v_2) \in \mathcal{A}^{l_1} \times \mathfrak{A}^{l_2}$ and $w_2 = \zeta(w_1, a)$ and $v_2 = \zeta(v_1, \alpha)$, then:

$$p\left[(w_1, v_1), a, \alpha, (w_2, v_2)\right] = \xi(w_1, v_1)(\alpha) p_a$$

where p_a is a *variable* holding the probability that a occurs.

The method for generating ξ is extremely similar to the Algorithms presented in Section 2.2.2 which were used to generate η. Assume we are given $\mathbf{x}$, the input and $\mathbf{y}$, the output. We will assume that the input signal is generated randomly and not as a function of the observed output (i.e., $\mathbf{x}$ is dithered). If not, we will be modeling the coupled dynamics of an existing control system and we will not obtain a true representation of the impact of a control signal on system output. This is consistent with classical transfer function modeling.

Let $\mathbf{w}$ be a subsequence of $\mathbf{x}$ and let $\mathbf{z}$ be a subsequence of $\mathbf{y}$. By $(\mathbf{w}, \mathbf{z})$ we mean the pair of input/output sequences. Assume that we know there is a lag of k time observation symbols before an output is observed. Then, for example if we began generating input symbols $x(1)x(2)\cdots$, and the lag is 2, then the output $y(1)$ will occur as symbol $x(3)$ is generated. From now on, we assume that the two sequences are appropriately aligned, so that $y(2)$ corresponds to the input $x(1)$, and previous output $y(1)$ even though though $y(2)$ appears at time $k+1$.

The main difference in generating ξ instead of η lies in Algorithm 2–the initialization phase. Let $\#(\mathbf{w}_1, \mathbf{x}, \mathbf{w}_2, \mathbf{y})$ be the number of times sequence $\mathbf{w}_1$ occurs in $\mathbf{x}$ and at the same time sequence $\mathbf{w_2}$ occurs in $\mathbf{y}$ so that the ends of the two sequences coincide. (Recall, $\mathbf{w}_1$ may have different length $\mathbf{w}_2$). Also let $\#(\mathbf{w}_1, \mathbf{x}, \mathbf{w}_2, \mathbf{y}, \alpha)$ be the number of times $\mathbf{w}_1$ occurs in $\mathbf{x}$ and at the same time sequence $\mathbf{w_2}$ occurs in $\mathbf{y}$ so that the ends of the two sequences coincide and α immediately follows $\mathbf{w}_2$.

Algorithm 8 Initialization

Input:

Symbolic time series $\mathbf{x}$ and $\mathbf{y}$
Alphabets $\mathcal{A}$ and $\mathfrak{A}$
Lengths l_1 and l_2

Output:

Map ξ

Procedure:

1: Let W be the set of pairs of substrings of $(\mathbf{w}_1, \mathbf{w}_2)$ with length l_1 and l_2 respectively, synchronized in ending position.
2: **for all** $(\mathbf{w}_1, \mathbf{w}_2) \in W$ **do**
3: Compute $\Pr(\alpha \in \mathfrak{A} | \mathbf{w}_1, \mathbf{w}_2) := \frac{\#(\mathbf{w_1}, \mathbf{x}, \mathbf{w_2}, \mathbf{y}, \alpha)}{\#(\mathbf{w_1}, \mathbf{x}, \mathbf{w_2}, \mathbf{y})}$.
4: **end for**
5: Define $f_{\mathbf{w}_1, \mathbf{w}_2}(\alpha) := \Pr(\alpha \in \mathfrak{A} | \mathbf{w}_1, \mathbf{w}_2)$.
6: Define $\xi(w_1, w_2) := f_{\mathbf{w_1}, \mathbf{w_2}}$.
7: **return** ξ

Algorithm 8 now produces a state space $Q = \mathcal{A}^{l_1} \times \mathfrak{A}^{l_2}$, where states correspond to pairs of histories of length l_1 and l_2. At state $q = (\mathbf{w}_1, \mathbf{w}_2) \in Q$ input $a \in \mathcal{A}$ occurs with corresponding output $\alpha \in \mathfrak{A}$ and transfers the system to a new state $q' = (\mathbf{w}_1', \mathbf{w}_2') = (\zeta(\mathbf{w}_1, a), \zeta(\mathbf{w}_2, \alpha))$. We will write $q' = \delta(q, a, \alpha)$ to denote this relationship.

For any state $q \in Q$, $\xi(q) \in \mathcal{F}_{\mathfrak{A}}$. The probability distributions can be thought of as a set of vectors in $[0,1]^{|\mathfrak{A}|}$, for a fixed ordering of $\mathfrak{A}$. For the remainder of this section, we will treat $\xi(q)$ as such a vector.

Algorithm 9 corresponds to Algorithm 3 of Section 2.2.2 in which there is only one alphabet. Similarly, Algorithm 10 corresponds to Algorithm 4 but is modified for two alphabets. Beginning with an initial symbolic input $\mathbf{y}$ and output $\mathbf{x}$ the application of Algorithms 8-10 will create a reduced state space representation of the symbolic transfer function η. It should be noted that it is sufficient to execute only Algorithm 2 to obtain a complete representation of the symbolic transfer function. If the state space is too large, then the additional algorithms can be executed to enhance processing later.

Algorithm 9

Input:

State Space Q
Threshold $t_0 \in \mathbb{R}$

Output:

State Space P

Procedure

```
1: P := ∅
2: for all q ∈ Q do
3:    if P ≠ ∅ then
4:       p* = arg min_{p∈P} T(q, p)
5:       if T(q, p*) > t_0 then
6:          p_new = {q}
7:          P := P ∪ {p_new}
8:       else
9:          p := p ∪ {q}
10:      end if
11:   else
12:      p_new = {q}
13:      P := P ∪ {p_new}
14:   end if
15: end for
```

For the remainder of this chapter we assume known and fixed alphabets $\mathcal{A}$ and $\mathfrak{A}$. We further assume that l_1, l_2 and k are known and fixed. An algorithm for inferring l_1 is available in [28]. It can be extended to infer l_2 if needed. As noted previously, we will assume that $k = 1$, which will simplify the notation significantly.

Let Q be the state space of ξ and let $r_q : \mathcal{A} \times \mathfrak{A} \to \mathbb{R}$ be a state parameterized reward function. At each discrete time t the system is in some state $q \in Q$. A control function chooses an input $a \in \mathcal{A}$ which causes an output symbol α to be generated at the next time step. This output is a function of previous inputs up to (but not necessarily including a, when there is a $k = 1$ lag). We may assume a reward $\beta^t r_{q(t)}$ results where $\beta \in (0, 1)$ is a discounting factor. The state then becomes q' as a result of the input and output.

For all $q \in Q$ define:

$$R_q = \begin{bmatrix} r_q(a_1, \alpha_1) & \cdots & r_q(a_1, \alpha_n) \\ \vdots & \ddots & \vdots \\ r_q(a_m, \alpha_1) & \cdots & r_q(a_m, \alpha_n) \end{bmatrix} \tag{4.1}$$

where $|\mathcal{A}| = m$ and $|\mathfrak{A}| = n$. For state $(\mathbf{u}, \mathbf{v}) \in q \in Q$ let $\xi(q) = \xi(\mathbf{u}, \mathbf{v})$ be the vector of probabilities that the various $\alpha \in \mathfrak{A}$ will occur.

If we are given the symbolic transfer function ξ and the objective is to derive a policy η so that the long run pay-off is optimized under the β-discounting rule, then this problem can be coded as a Markov Decision Problem [54]. Following [39], we may write the problem of maximizing

Algorithm 10

Input:

State Space P

Output:

State Space R

Procedure

$N_0 := 0$
$N := |P|$
repeat
  for all $p_i \in P$ **do**
    $s := 0$
    $M = |p_i|$ {The size of cluster p_i}
    Choose $q_{i_0} \in p_i$
    Create state $r_{is} = \{q_{i_0}\}$
    $R := R \cup \{r_{is}\}$
    for all $q_{ij} \in p_i$ $(j \neq 0)$ **do**
      FOUND = 0
      for $l \in 0, \ldots, s$ **do**
        Choose $q \in r_{il}$
        if $\delta(q, a, \alpha) = \delta(q_{i_j}, a, \alpha)$ for all $(a, \alpha) \in \mathcal{A} \times \mathfrak{A}$ **then**
          FOUND = 1
          $r_{il} := r_{il} \cup \{q_{i_j}\}$
        end if
      end for
      if FOUND = 0 **then**
        $s := s + 1$
        Create state $r_{is} = \{q_{i_j}\}$
        $R := R \cup \{r_{is}\}$
      end if
    end for
  end for
  $N_0 := N$
  $N : |R|$
until $N \neq N_0$

long run reward subject to β discounting as:

$$\begin{cases} \min \quad \sum_{q' \in Q} \pi^0_{q'} v_{q'} \\ \text{s.t.} \;\; v_q \geq [R_q \xi(q)]_a + \beta \sum_{q' \in Q} \Pr(q'|q, a) v_{q'} \;\; \forall q \in Q, a \in \mathcal{A} \end{cases} \tag{4.2}$$

Here $\mathbf{v}$ is a vector in $\mathbb{R}^{|Q|}$ with elements v_q and for vector y, $[y]_a$ indicates the element of y corresponding to $a \in \mathcal{A}$. This allows us to use the MDP framework for the more general

problem. Problem 4.2 has known dual, which was first presented as Problem 2.7 in Chapter 2:

$$\begin{cases} \max & \sum_{q\in Q}\sum_{a\in\mathcal{A}}[R_q\xi(q)]_a x_{qa} \\ \text{s.t.} & \sum_{q\in Q}\sum_{a\in\mathcal{A}}\left[\delta(q,q') - \beta\Pr(q'|q,a)\right]x_{qa} = \pi^0_{q'} \;\; \forall q' \in Q \\ & x_{qa} \geq 0 \;\; \forall q \in Q, a \in \mathcal{A} \end{cases} \tag{4.3}$$

Here x_{qa} are the dual variables corresponding to the $|Q| \times |\mathcal{A}|$ constraints in Problem 4.2 and $\delta(q,q')$ is the Dirac delta function. It should also be noted that $Pr(q'|q,a)$ is precisely $\xi(q)(\alpha)$ where $q' = \delta(q,a,\alpha)$. We also present the following proposition which comes from Theorem 2.3.1.

Proposition 4.1.1. *Let $\mathbf{x}^*$ be an optimal (vector) solution to Problem 4.3. For fixed $q \in Q$, let*

$$x_q = \sum_{a\in\mathcal{A}} x_{qa} \tag{4.4}$$

Then the optimal policy η is given by:

$$\eta(q)(a) = \frac{x^*_{qa}}{x^*_q} \tag{4.5}$$

For fixed $\mathbf{u} \in \mathcal{A}^{l_1}$, if for all $\mathbf{v}_1, \mathbf{v}_2 \in \mathfrak{A}^{l_2}$ we have $\eta(\mathbf{u},\mathbf{v}_1) = \eta(\mathbf{u},\mathbf{v}_2)$ then η is an open loop controller. Note Problem 4.3 has a constraint space with size equal to the number of states in Q, making state minimization of the η model useful for minimizing the size of the optimization problem.

4.2 Symbolic Statistical Process Control

We define a learning and optimization strategy for symbolic dynamical systems analogous to the statistical process control process identified by Box and Jenkins and discussed in detail in [55].

Assuming an N-time step learning period, the following algorithm can be used to derive a symbolic transfer function and optimal open-loop η controller:

1. While $t \leq N$
   (a) Choose a completely random (i.e., dithered) input (i.e., a random η function) and record the output.
   (b) Builds symbolic sequences $\mathbf{x}$ and $\mathbf{y}$.
2. At time $t = N$, use Algorithms 8-10 (or Algorithm A from [34] to derive function ξ, the symbolic transfer function.
3. Using function ξ, compute η^* using Problem 4.3.

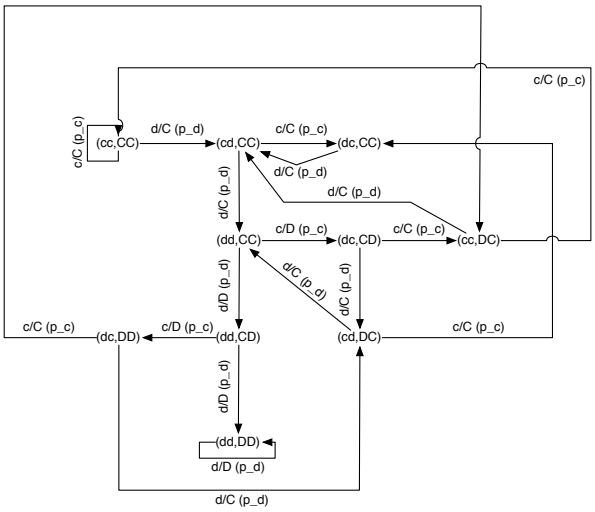

Figure 4.1: The formal Mealey machine state space model of ξ derived using Algorithm 8 from a Tit-for-Two-Tats game scenario. In this figure, the symbols p_c and p_d indicate the probability that Player 1 strategies c or d will be played. The deterministic nature of the Tit-for-Two-Tats algorithm makes it deterministic in Player 2's strategy.

4.2.1 Example: Learning to Play against Tit-for-2-Tats

Repeated games offer a simple symbolic input / output system on which to build an example. This modeling approach could also be used in other computational systems in which symbolic (categorical) inputs lead to symbolic (categorical) outputs like in I/O protocols.

The Prisoner's Dilemma Game is a well known symmetric bimatrix game describing the behavior of two captured criminals. There are two strategies for each player, Collude (C) and Defect (D). ([56], Page 25) has an excellent overview. In the repeated Prisoner's Dilemma Game, two players repeat this game in an attempt to maximize their long run pay-off.

In Tit-for-2-Tats, a forgiving strategy is used to avoid unending defection. If Player 1 plays C in round i, then Player 2 will play C in round $i+1$. If Player 1 plays D in round i and had previously played C in round $i-1$, then Player 2 still plays C in Round $i+1$. Otherwise, Player 2 plays D. If Player 2 is using a fixed strategy (protocol) $\mathcal{S}$ such as Tit-for-2-Tats, it may be possible to *game* the strategy thus deriving a better long run payoff for Player 1. This is exactly the problem of generating an optimal control law for Player 1.

4.2.1.1 Learning ξ from Input Data

We apply Algorithms 8-10 to identify a ξ function for the Tit-for-2-Tats strategy assuming Player 1 executed a random strategy in order to fully explore the search space. As in [34], we used 25 iterations of a repeated game to extrapolate ξ using a randomized input.

After executing Algorithm 8 on the input sequence, we obtain the state space model for the behavior of ξ shown in Figure 4.1. This model shown in Figure 4.1 contains 10 states, corresponding to the 10 observed pairs of strategy sequences observed for Players 1 and 2, each

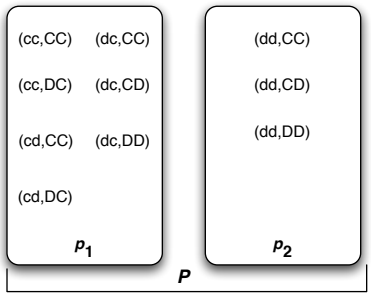

Figure 4.2: The clustering of states from the formal Mealey state machine shown in Figure 4.1.

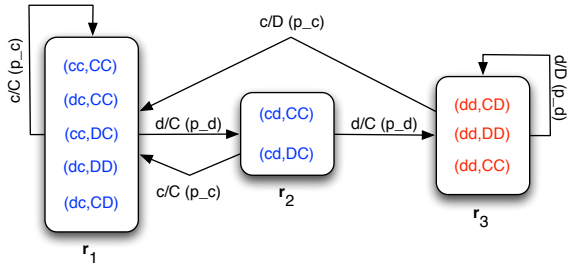

Figure 4.3: The reduced state space formal Mealey machine modeling ξ derived from an input sequence of player moves. This machine is the result of executing Algorithms 8-10 .

with length 2. We can cluster theses states using Algorithm 9. The results of running Algorithm 9 are shown in Figure 4.2. Having identified the clustered state space, we can execute Algorithm 10 to identify the reduced state space and formal Mealey machine modeling ξ. This is shown in Figure 4.3.

4.2.1.2 Finding the Optimal Response η

Let:

$$R_q = \begin{bmatrix} 1 & -2 \\ 2 & \frac{1}{2} \end{bmatrix}$$

for all $q \in Q$. This is a standard Prisoner's Dilemma type game matrix. We suppose that the strategies are read from left to right as collude or defect and from top to bottom in the same order. If we assume an initial state to contain $([c, c], [C, C])$, i.e., one in which all players have cooperated up to this point, then when using the derived ξ function, the specific instance of

Problem 4.3 is:

$$\begin{cases} \max & x_{1,c} + 2x_{1,d} + x_{2,c} + 2x_{2,d} - 2x_{3,c} + \frac{1}{2}x_{3,d} & \\ s.t. & (1-\beta)x_{1,c} + x_{1,d} - \beta x_{2,c} - \beta x_{3,c} = 1 & (q' = 1) \\ & -\beta x_{1,d} + x_{2,c} + x_{2,d} = 0 & (q' = 2) \\ & (1-\beta)x_{3,d} - \beta x_{2,d} + x_{3,c} = 0 & (q' = 3) \\ & x_{1,c}, x_{2,c}, x_{3,c}, x1, d, x_{2,d}, x_{3,d} \geq 0 & \end{cases}$$

The state q' to which each constraint corresponds is shown in the right most column. For large enough (e.g., $\beta > 1/2$) the optimal solution yields behavior:

$$\begin{aligned} \eta([c,c])(d) &= 1.0 \\ \eta([c,d])(c) &= 1.0 \\ \eta([d,c])(d) &= 1.0 \end{aligned}$$

and $\eta(q)(a) = 0$ for all other $q \in Q$ and $a \in \mathcal{A}$. These dynamics create a cycle in which Player 1 (who constructs η) will repeatedly defect and then collude, thus taking advantage of the altruism of Player 2 in employing the Tit-for-2-Tats strategy. This is illustrated in Figure 4.4. It is worth

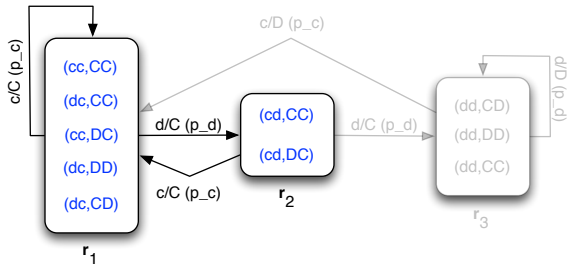

Figure 4.4: The cycle induced by the dynamics provided in the optimal response to Tit-for-Two-Tats illustrated as a sub-machine of the reduced state space shown in Figure 4.3.

noting that for smaller values of β (i.e., $\beta < 0.45$) the optimal strategy is to always defect. Thus when the award decay is too great, cyclic cooperation and defection is not worth it. In the following chapter we extend the ideas of this chapter explore optimal strategies of deception. We use the same framework as the two-player game presented in Section 4.2.1, however we assume that one player us attempting to deceive the other by teaching them a "fake" strategy to learn.

Chapter 5

Optimal Deception Strategies in Two-Player Games

In this chapter we explore optimal strategies of deception in two-player games. Namely, we are interested in multi-period competitive game where two actors first engage in a "learning period" following a period of game play. In the learning period, one player (Player 2) has the opportunity to learn the opposing player's (Player 1) strategy by observing the other player's response to random input, generating a string of inputs and responses. Player 1 can infer a probabilistic finite state machine from this data which is equivalent to learning Player 1's strategy. Player 2 can then generate an optimal counter-strategy to utilize during the game play period which maximizes their reward against Player 1's strategy.

However, we introduce the notion of deception by allowed Player 1 to be aware of Player 2's observation and learning. Thus, Player 1 is aware of this observation, and can "trick" the other player by training them on a false strategy. That is, Player 1 can play a certain strategy during the learning period, have Player 2 optimize based on this strategy, and then switch strategies in the game play period in order to maximize their own reward. We now formalize this idea.

Let $\Gamma = (\{P_1, P_2\}, \Sigma_1 \times \Sigma_2, \pi_1 \times \pi_2)$ be a two player game where P_1 and P_2 are the players, $\Sigma_1 \times \Sigma_2$ is the state space, and $\pi_1 \times \pi_2$ are the payoff functions. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be, respectively, the set of possible actions for each player. Then we can definite a state $\sigma_i \in \Sigma_1$ as $\{a_1, a_2, ..., a_{L1}\}$ where $a_i \in \mathcal{A}_1$ and L_1 is the history length for $P1$. We can do the same for $P2$.

Suppose we extend Γ to a repeated game with payoff functions $\Pi_1 \times \Pi_2$. Let $\xi : \Sigma_1^{L1} \times \Sigma_2^{L2} \to \mathcal{F}_{\Sigma_1}$ be the goal strategy of Player 1, where $\mathcal{F}_{\Sigma_1}$ is the set of discrete probability distributions over Σ_1 giving the next (probabilistic) move for Player 1 given the prior state of the game. Suppose that Player 1 plays $\mathbf{y} : \Sigma_1^{L1} \times \Sigma_2^{L2} \to \mathcal{F}_{\Sigma_1}$, and generates $(\boldsymbol{\alpha}, \boldsymbol{\beta}) \in \Sigma_1^{L1} \times \Sigma_2^{L2}$ where β is generated randomly.

Player 2 learns $\hat{\mathbf{y}} : \Sigma_1^{L1} \times \Sigma_2^{L2} \to \mathcal{F}_{\Sigma_1}$ and develops η^* which is an optimal response to $\hat{y}$.

Then the resulting outcome of play is

$$\widetilde{\Pi_1} = \Pi_1(\eta^*, \xi) - \lambda||\xi - y||,$$

and

$$\widetilde{\Pi_2} = \Pi_2(\eta^*, \xi),$$

where λ is the marginal cost of deception and $||\xi - y||$ is a metric measuring the difference between strategies ξ and y. In this thesis we define $||\xi - y||$ as $\sum_i |\xi_i - y_i|$.

The problem for Player 1 is to compute

$$Y = \arg\max_y \widetilde{\Pi_1}(\eta^*(y), \xi) \tag{5.1}$$

where η^* is computed endogenously, as a linear program.

5.1 Zero cost of deception

For now we will assume that $\lambda = 0$. Let $\eta' = \max_\eta \Pi_1(\eta, \xi)$. That is, η' is the strategy for Player 2 that results in the best outcome for Player 1. The ultimate goal of Player 1 is to deceive Player 2 into playing η'. In order to do so, Player 1 wishes to find a strategy y such that $||\eta^*(y) - \eta'||$ is minimized. This way, Player 1 can "train" Player 2 on strategy y so that Player 2 develops the optimal response η^* which is as close to η' as possible.

First, Player 1 must determine η' based on ξ. This can be coded as a Markov Decision Problem as in Problem 4.3. After solving Problem 4.3, Proposition 4.1.1 tells us that the optimal policy η' is given by:

$$\eta'(q)(a) = \frac{x^*_{qa}}{x^*_q} \tag{5.2}$$

The following proposition is clear from the model:

Proposition 5.1.1. *Let $\eta' = arg\max_\eta \Pi_1(\eta, \xi)$. If y^* solves:*

$$\min_y ||\eta^*(y) - \eta'(\xi)||$$

then y^* is the optimal deception when $\lambda = 0$.

The problem of Player 1 is to find such a strategy y. When $\lambda = 0$ this can be accomplished with a genetic algorithm described in Algorithm 11. Algorithm 11 starts with a random set of possible strategies, $\{y_1, ..., y_n\}$–the "parent" set. Each strategy is evaluated according to an objective function which is described in Algorithm 12. In this case, the "score" of each strategy is given in line 3 of Algorithm 12 by $\max_\eta ||\eta - \eta'|| - ||\eta^*(y_i) - \eta'||$. Notice that $\max_\eta ||\eta - \eta'||$ is a constant for all y_i, and is simply dependent on the chosen metric. In our case, $\max_\eta ||\eta - \eta'|| = 2\sum_i |\eta'_i| = 2|Q|$. When $\eta^* = \eta'$, the objective function is maximized.

The number of offspring for each parent is given by lines 5-9 of Algorithm 11. Thus, the number of offspring of a parent strategy is a direct result of how it scores compared to average.The new set of strategies consists of rep_i copies of strategy y_i. Finally, we utilize a crossover and mutation period in order to diversify the possible strategies considered. Each strategy is randomly crossed with another strategy, and mutated with probability P_{mutate}. This set of strategies then becomes the "parent" set, and the process is repeated. This loop is repeated until all of the scores for each strategy are the same.

Algorithm 11 –Finding y

Input: a state space Q, alphabet $\mathcal{A}_2$, payoff function Π_2, a goal optimal strategy η'
Genetic Algorithm

1: Start with a set of n possible strategies y, $\{y_1, ..., y_n\}$, where $y_i \in \mathcal{F}_{\Sigma_1}^{L1}$
2: obj $\leftarrow$ **objFunction**$(y, Q, \mathcal{A}_2, \Pi_2, \eta')$ {See Algorithm 12}
3: **while** $\text{obj}_i \neq \text{obj}_j \ \forall i, j$ **do**
4: **for** $i = 1 : n$ **do**
5: **if** $\text{obj}_i > \text{avg}_i(\text{obj}_i)$ **then**
6: rep_i =Round$(\frac{\text{obj}_i}{\text{avg}_i(\text{obj}_i)})$
7: **else**
8: $\text{rep}_i = 0$
9: **end if**
10: **end for**
11: $n \leftarrow \sum_i \text{rep}_i$
12: Create the set of new strategies $y' = \{y'_1, ..., y'_n\}$ consisting of rep_i copies of y_i
13: Randomly choose a crossover partner and location for each y'_i
14: Mutate y'_i with probability P_{mutate}
15: $y \leftarrow y'$
16: obj $\leftarrow$ **objFunction**$(y, Q, \mathcal{A}_2, \Pi_2, \eta')$
17: **end while**

Algorithm 12 –Objective Function for $\lambda = 0$

obj =**objFunction**$(y, Q, \mathcal{A}_2, \Pi_2, \eta')$

1: **for** i=1:n **do**
2: Solve Problem 4.3 for $\eta^*(y_i)$
3: $\text{obj}_i \leftarrow \max_\eta ||\eta - \eta'|| - ||\eta^* - \eta'||$ {This transformation is done because we want to maximize obj_i instead of minimize}
4: **end for**

For the remained of this section we will explore a numerical example where the game is Prisoner's Dilemma, and Player 1 wishes to use a "tit-for-two-tat " strategy in the final round, which is strategy ξ. For this strategy, $L_1 = 2$, meaning that Player 1's action depend on the previous two actions by both players. We also assume that play continues for infinite time. This strategy is shown in Figure 5.1 as a probabilistic state transition function dependent on Player 2's strategy. Capital letters are Player 1's actions and lowercase letters are Player 2's actions. Table 5.1 shows the payoff function for each player. Π_1 and Π_2 can be elicited by taking the first and second values of Π, respectively.

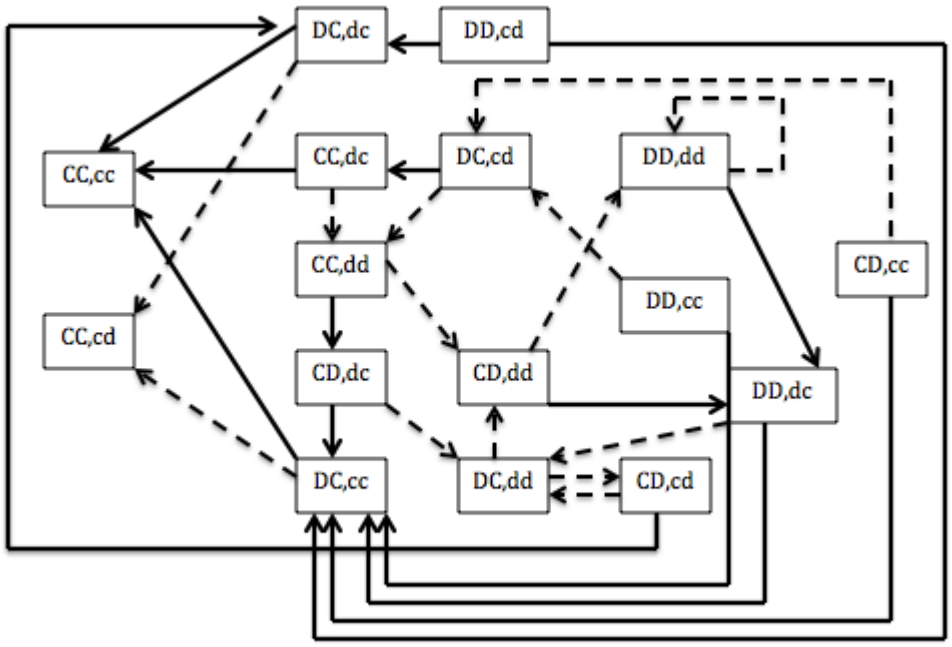

Figure 5.1: The "tit-for-two-tat" strategy for Player 1. The states depend on Player 2's actions. A solid line shows the state transition when Player 2 cooperates and a dashed line shows the state transitions when Player 2 defects.

Table 5.1: Payoff Matrix Π

	d	c
D	-1,-1	3,-2
C	-2,3	1,1

Given strategy ξ, Player 1 can determine the strategy η' such that $\Pi_1(\eta', \xi)$ is maximized by solving Problem 4.3. For this specific problem, assuming an initial probability distribution of $\pi^0_{\{C,c\}} = 1$, the set of constraints of Problem 4.3 simplify to:

$$
\begin{cases}
.1x_{\{CC,cc\},c} + x_{\{CC,cc\},d} - .9x_{\{CC,dc\},c} - .9x_{\{DC,cc\},c} - .9x_{\{DC,dc\},c} = 1 & \{CC,cc\} \\
-.9x_{\{CC,cc\},d} + x_{\{CC,cd\},c} + x_{\{CC,cd\},d} - .9x_{\{CC,dc\},dc} - .9x_{\{DC,cc\},d} - .9x_{\{DC,dc\},d} = 0 & \{CC,cd\} \\
-.9x_{\{CC,cd\},c} + x_{\{CC,dc\},c} + x_{\{CC,dc\},dc} - .9x_{\{DC,cd\},c} = 0 & \{CC,dc\} \\
-.9x_{\{CC,cd\},d} + x_{\{CC,dd\},c} + x_{\{CC,dd\},d} - .9x_{\{DC,cd\},d} = 0 & \{CC,dd\} \\
x_{\{CD,cc\},c} + x_{\{CD,cc\},d} = 0 & \{CD,cc\} \\
x_{\{CD,cd\},c} + x_{\{CD,cd\},d} = 0 & \{CD,cd\} \\
-.9x_{\{CC,dd\},c} + x_{\{CD,cd\},c} + x_{\{CD,cd\},d} - .9x_{\{DC,dd\},c} = 0 & \{CD,dc\} \\
-.9x_{\{CC,dd\},d} + x_{\{CD,dd\},c} + x_{\{CD,dd\},d} - .9x_{\{DC,dd\},d} = 0 & \{CD,dd\} \\
-.9x_{\{CD,cc\},c} - .9x_{\{CD,dc\},c} + x_{\{DC,cc\},c} + x_{\{DC,cc\},d} - .9x_{\{DD,cc\},c} - .9x_{\{DD,dc\},c} = 0 & \{DC,cc\} \\
-.9x_{\{CD,cc\},d} - .9x_{\{CD,dc\},d} + x_{\{DC,cd\},c} + x_{\{DC,cd\},d} - .9x_{\{DD,cc\},d} - .9x_{\{DD,dc\},d} = 0 & \{DC,cd\} \\
-.9x_{\{CD,cd\},c} + x_{\{DC,dc\},c} + x_{\{DC,dc\},dc} - .9x_{\{DD,cd\},c} = 0 & \{DC,dc\} \\
-.9x_{\{CD,cd\},d} + x_{\{DC,dd\},c} + x_{\{DC,dd\},d} - .9x_{\{DD,cd\},d} = 0 & \{DC,dd\} \\
x_{\{DD,cc\},c} + x_{\{DD,cc\},d} = 0 & \{DD,cc\} \\
x_{\{DD,cd\},c} + x_{\{DD,cd\},d} = 0 & \{DD,cd\} \\
-.9x_{\{CD,dd\},c} + x_{\{DD,dc\},c} + x_{\{DD,dc\},d} - .9x_{\{DD,dd\},c} = 0 & \{DD,dc\} \\
-.9x_{\{CD,dd\},d} + x_{\{DD,dd\},c} + .1x_{\{DD,dd\},d} = 0 & \{DD,dd\} \\
x_{qa} \geq 0 \ \ \forall a \in Q, a \in \mathcal{A}
\end{cases}
\tag{5.3}
$$

Solving Problem 4.3 with the set of constraints 5.3 results in:

$$\eta'(\{CC,cc\})(c) = 1$$

Notice that the optimal response function was only specified for states that were reached. Since our initial probability distribution was $\pi^0_{\{CC,cc\}} = 1$, the game stays in $\{CC,cc\}$ forever. In order to determine η' completely, we must solve Problem 4.3 starting from every state. This yields the optimal response function:

$$\eta'(q)(c) = 1 \ \ \forall q \in \Sigma_1^{L1} \times \Sigma_2^{L2}$$

which means that η' is the strategy "always cooperate."

Thus, Player 1 wishes to train Player 2 on some strategy y such that $\eta^*(y) = \eta'$ ="always cooperate." As it happens, the best response to the tit-for-tat strategy is to always cooperate. This strategy only has history length one. The tit-for-tat strategy is shown in Figure 5.2. Thus, in our example, Algorithm 11 could return y ="tit-for-tat" (note: there may be other strategies y for which η^*="always cooperate" besides tit-for-tat).

Once y is determined, Player 1 generates $(\alpha, \beta) \in \Sigma_1^* \times \Sigma_2^*$ using strategy y. This could look something like Table 5.2 for the tit-for-tat strategy. Using the observed sequence (α, β), Player 2 learns $\hat{y}$ using an algorithm such as CSSR. Player 2 then chooses an optimal response such that $\widetilde{\Pi_2} = \Pi_2(\eta^*, \xi)$ is maximized by solving Problem 4.3. Of course, this optimal response was already predetermined by Player 1 to be as close to η' as possible since y was determined by running Algorithm 11.

Table 5.2: Input and Output Strings

β	c	c	d	d	c	d	d	d	c	c
α	-	C	C	D	D	C	D	D	D	C

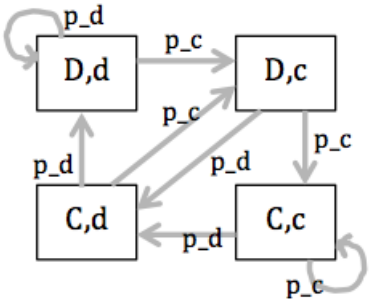

Figure 5.2: The tit-for-tat strategy for Player 1. The states depend on Player 2's actions which are given as probabilities.

When the two players play against each other, Player 1 plays ξ and Player 2 plays η^*. In this case, since $y =$"tit-for-tat" results in $\eta^* = \eta'$ exactly and Player 2 ends up playing "always cooperate." Meanwhile, Player 1 plays ξ which was "tit-for-two-tat." The resulting play is simply stationary in state $\{CC, cc\}$. The payoff for both players is $\sum_n^\infty .9^n = 10$.

In order to measure how much the deception aided Player 1, we can also determine what Player 1's payoff would have been without deception, i.e. if Player 2 could optimize against "tit-for-two-tat" instead of optimizing against "tit-for-tat". In this case, the optimal strategy is for Player 2 to alternate between cooperating and defecting, taking advantage of the forgiving tit-for-two-tat strategy. Using this alternating strategy instead, the game play alternates between state $\{CC, cc\}$ and $\{CC, cd\}$. In this case, the long run payoff of Player 2 is

$$1 + .9(2) + .9^2(1) + .9^(3)(2) + etc = \sum_{n\ even} .9^n(1) + \sum_{n\ odd} .9^n(2) = 14.73$$

and the long run Payoff for Player 1 is

$$1 + .9(-3) + .9^2(1) + .9^(3)(-3) + etc = \sum_{n\ even} .9^n(1) + \sum_{n\ odd} .9^n(-3) = -8.95$$

Apparently, the use of deception greatly helps Player 1 in this case. Without deception, long-run payoff would always be negative. However, with deception, $\{CC, cc\}$ is the only absorbing state, yielding high positive payoffs for Player 1 and Player 2.

5.2 Nonzero cost of deception

When $\lambda \neq 0$, i.e. we assume that there is a cost of deception, the strategy y chosen by Player 1 can shift so that

$$\widetilde{\Pi_1} = \Pi_1(\eta^*, \xi) - \lambda||\xi - y||$$

is maximized. That is, Player 1 now must balance the game-play payoff with how much they are willing to deviate from strategy ξ. In this case, the genetic algorithm in Algorithm 11 can still be used, however the objective function must change. We can no longer simply seek a strategy such that $||\eta^*(y) - \eta'||$ is minimized. The new objective function can be seen in Algorithm 13. For every strategy y_i, we must compute the payoff for that strategy, $\Pi_1(y_i, \eta^*(y_i))$ (Line 3), as well as the cost of deception, $||\xi - y_i||$ (Line 4). The overall score of a certain strategy is the payoff minus the cost of deception (Line 5). The strategy that scores the best must balance these two components. Because the values of obj_i could be negative, we also perform the transformation in Line 6 to ensure strict positivity.

Algorithm 13 –Objective Function for $\lambda \neq 0$

obj =**objFunction**$(y, \xi, Q, \mathcal{A}_2, \Pi_2, \Pi_1)$

1: **for** i=1:n **do**
2: Solve Problem 4.3 for $\eta^*(y_i)$
3: $\text{payoff}_i \leftarrow \Pi_1(y_i, \eta^*(y_i))$
4: $\text{cost}_i \leftarrow ||\xi - y_i||$
5: $\text{obj}_i \leftarrow \text{payoff}_i - \text{cost}_i$
6: $\text{obj}_i \leftarrow \text{obj}_i - \min_i(\text{obj}_i) + 1$ {This ensures that the objective function is positive and nonzero for each strategy y_i}
7: **end for**

Now we return to the tit-for-two-tat example. When $\lambda = 0$, we know that Algorithm 13 will return a strategy y which results in $\eta^*(y) =$"always cooperate". (Note: tit-for-tat is one such strategy, however there could be another strategy z such that $\eta^*(z) =$"always cooperate" as well. Algorithm 11 could return any such strategy.) In this case, we were able to find a strategy y such that $\eta' = \eta^*(y)$ exactly. By Proposition 5.1.1, the strategy y is our optimal deception with $\lambda = 0$.

However, when $\lambda \neq 0$, Algorithms 11 and 13 produce a slightly different strategies y' seen in Table 5.3, which in turn also results in different strategies $\eta^*(y')$. Algorithms 11 and 13 now search for a strategy which not only maximizes $\Pi_1(y', \eta'^*(y'))$ but also simultaneously minimizes $\lambda||\xi - y||$. Therefore we might expect a slightly smaller value of $\Pi_1(y', \eta'^*(y'))$, but a strategy more similar to ξ, so that $||y' - \xi||$ is smaller. Table 5.3 shows the resulting strategies y' produced by Algorithms 11 and 13 for $\lambda = 1, 1.5, 2$, and 20 as well as the goal strategy ξ.

From Table 5.3 we see that for $\lambda = 1, 1.5$, and 2, y' lies "between" y and ξ in the sense that $||y - \xi|| > ||y' - \xi|| > 0$. As the value λ increases, the strategy resulting from Algorithm 13 looks more and more like strategy ξ and eventually become ξ at a critical value λ^*. As λ increases, the

Table 5.3: Strategies y' when $\lambda = 1, 1.5, 2, 20$

state	$\lambda = 1$	$\lambda = 1.5$	$\lambda = 2$	$\lambda = 20$	ξ
$\{CC, cc\}$	c	c	c	c	c
$\{CC, cd\}$	d	d	d	c	c
$\{CC, dc\}$	c	c	c	c	c
$\{CC, dd\}$	d	d	d	d	d
$\{CD, cc\}$	c	c	c	c	c
$\{CD, cd\}$	c	c	c	c	c
$\{CD, dc\}$	c	c	c	c	c
$\{CD, dd\}$	d	d	d	d	d
$\{DC, cc\}$	c	c	c	c	c
$\{DC, cd\}$	d	d	c	c	c
$\{DC, dc\}$	c	c	c	c	c
$\{DC, dd\}$	d	d	d	d	d
$\{DD, cc\}$	c	c	c	c	c
$\{DD, cd\}$	d	c	c	c	c
$\{DD, dc\}$	c	c	c	c	c
$\{DD, dd\}$	d	d	d	d	d

payoff for Player 1 is more highly weighted by the cost of deception. That is, as $\lambda \to \infty$,

$$\widetilde{\Pi_1} = \Pi_1(\eta^*, \xi) - \lambda||\xi - y|| \approx -\lambda||\xi - y||$$

Thus, as λ increases, Algorithm 11 along with Algorithm refalg:optlambda will result in a strategy that is more and more similar to ξ in order to minimize $||\xi - y||$. However, since these strategies are discrete, there is a finite number λ^* at which Algorithms 11 and 13 will result in ξ exactly. In this example, the critical value is approximately $\lambda^* \approx 20$ and we can see in Table 5.3 that when $\lambda = 20$, the resulting strategy y' is exactly equal to ξ, which is tit-for-two-tat.

Chapter 6

Conclusions and Future Work

In Chapter 3 of this thesis we illustrated the the problem stated by Shalizi and Crutchfield [1] of determining a minimal state probabilistic finite state representation of a data set is NP-hard for finite data sets. As a by-product, we formally proved this to be the case for the non-deterministic case as studied in [51]. As such, this shows that both the CSSR algorithm of [1] and CSSA algorithm of [51] can be thought of as low-order polynomial approximations of NP-hard problems. Future work in this area includes studying, in detail, these approximation algorithms for the problems to determine what the heuristic properties of the CSSR algorithm are. We could also explore further reformulations of the integer programming MSDpFSA Problem in order to find faster run times.

In Chapter 4 we have provided a method for optimizing a control process in which the inputs and output are symbolic and the transfer function is stochastic. This work attempts to apply the notions of optimal control in the Box Jenkins framework [57] to the case of purely symbolic processes. We drew upon work in [34] to obtain symbolic transfer function estimators. We then applied results from Markov Decision Processes [54] to determine closed loop controls in this case. For future work, we note that in computing an estimator for ξ, we applied Algorithms 8 - 10 (Algorithm A of [34]). These algorithms provides a pointwise predictor for the multinomial distributions that comprise the symbolic transfer functions. In [58] it is shown how to use confidence intervals to improve recognition in hidden Markov models [10] and how to derive confidence intervals on the transitions of a specific HMM process. The same technique can be applied here to derive confidence intervals on the transition probabilities of the symbolic transfer function. We could use this information to find a solution that is robust to our imprecise knowledge of the second player using a competitive Markov decision process [39] in this case as well. This is particularly useful for online learning when the convergence of the symbolic transfer function is slow.

In Chapter 5, we developed a model which utilized the CSSR algorithm to form optimal strategies for deception in two-player games. Using this model, a numerical example was pre-

sented which explored deception in the Prisoner's Dilemma game. Future directions for this work include exploring how the length of the learning period affects the effectiveness of deception. Similarly, we hope to determine how long the learning period must be in order to effectively infer the strategy that is being utilized in that period. We also wish to apply this model to more complicated game structures, and possible to real-world war-like scenarios. Another area of future research includes either limiting the game-play to a finite number of stages instead of assuming infinite play, or allowing the deceived player to adjust his strategy during the game-play period as he realizes that he is being deceived. This may be a more realistic model formulation because it does not assume that the deceived player is completely naive.

Bibliography

[1] C. Shalizi, K. Shalizi, and J. Crutchfield, "An algorithm for pattern discovery in time series," arXiv:cs.LG/0210025 v3, November 2002.

[2] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. Carrasco, "Probabilistic finite-state machines - part i," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 7, pp. 1013–1025, July 2005.

[3] K. Fu, *Syntactic Pattern Recognition and Applications*. Prentice Hall, 1982.

[4] L. Miclet, *Structural Methods in Pattern Recognition*. Springer-Verlag, 1987.

[5] N. Abe and H. Mamitsuka, "Predicting protein secondary structure using stochastic tree grammars," *Machine Learning*, vol. 29, no. 2-3, pp. 275–301, 1997.

[6] Y. Sakakibara, M. Brown, R. Hughley, I. Mian, K.Sjolander, R. Underwood, and D. Haussler, "Stochastic context-free grammars for trna modeling," *Nuclear Acids Research*, vol. 22, pp. 5112–5120, 1994.

[7] S. D. H. Alshawi, S. Bangalore, "Learning dependency translation models as collections of finite state head transducers," *Computational Linguistics*, vol. 26, 2000.

[8] S. Bangalore and G. Riccardi, "Stochastic finite-state models for spoken language machine translation," *Proc. Workshop Embedded Machine Translation Systems, NAACL*, pp. 52–59, May 2000.

[9] F. P. M. Mohri and M. Riley, "The design principles of a weighted finite-state transducer library," *Theoretical Computer Science*, vol. 231, pp. 17–32, 2000.

[10] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," vol. 77, no. 2, pp. 257–286, 1989.

[11] L. Baum and J. Egon, "An inequality with applications to statistical estimation for probabilistic functions of a markov process and to a model for ecology," *Bull. Amer. Meteorol. Soc.*, vol. 73, pp. 360–363, 1967.

[12] L. Baum and G. Sell, "Growth functions for transformations on manifolds," *Pac. J. Math.*, vol. 27, no. 2, pp. 211–227, 1968.

[13] L. Baum and T. Petrie, "Statistical ingerence for probabilistic functions of finite state markov chains," *Annals of Mathematics Statistics*, vol. 37, pp. 1554–1563, 1966.

[14] L. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains," *Annals of Mathematics Statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[15] L. Baum, "An inqueality ans associated maximization technique in statistical estimation for probabilistic functions of markov processes," *Inequalities*, vol. 3, pp. 1–8, 1972.

[16] J. Baker, "The dragon system-an overview," *IEEE Trans. Acoust. Speech Signal Processing*, vol. ASSP-23, no. 1, pp. 24–29, 1975.

[17] F. Jelinek, L. Bahl, and R. Mercer, "Design of a linguistic statistical decoder for the recognition of continuous speech," *IEEE Trans. Informat. Theory*, vol. IT-21, pp. 404–411, 1975.

[18] F. Jelinek, "Continuous speech recognition by statistical methods," *Proc. IEEE*, vol. 64, pp. 532–536, 1976.

[19] H. Xue and V. Govindaraju, "Hidden markov models combining discrete symbols and continuous attributes in handwriting recognition," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 28, no. 3, pp. 458–462, 2006.

[20] J. Hu, M. K. Brown, and W. Turin, "Mm-based online handwriting recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 18, no. 10, pp. 1039–1045, 1996.

[21] S. Lefevre, E. Bouton, T. Brouard, and N. Vincent, "A new way to use hidden markov models for object tracking in video sequences," in *Process. III-117-III-120*, 2003.

[22] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. Carrasco, "Probabilistic finite-state machines - part ii," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 7, pp. 1026–1039, July 2005.

[23] L. Liporace, "Maximum likelihood estimation for multivariate observations of markov sources," *IEEE. Trans. Informat. Theory*, vol. IT-28, no. 5, pp. 729–735, 1982.

[24] B. Juang, "Maximum likelihood estimation for mixture multivariate stochastic observations of markov chains," *ATT Tech. J.*, vol. 64, no. 6, pp. 1235–1249, 1985.

[25] B. Juang, S. Levinson, and M. Sondhi, "Maximum likelihood estimation for multivariate mixture observations of markov chains," *IEEE Trans. Informat. Theory*, vol. IT-32, no. 2, pp. 307–309, March 1986.

[26] J. Schwier, R. Brooks, C. Griffin, and S. Bukkapatnam, "Zero knowledge hidden markov model inference," *Pattern Recognition Letters*, vol. 30, no. 14, pp. 1273 – 1280, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865509001597

[27] A. McCallum, "Hidden markov models: Baum welch algorithm," March 2004, notes for Introduction to Natural Language Processing. [Online]. Available: https://people.cs.umass.edu/ mccallum/courses/inlp2004a/lect10-hmm2.pdf

[28] R. R. Brooks, J. Schwier, C. Griffin, and S. Bukkapatnam, "Zero knowledge hidden markov model inference," *Pattern Recognition Letters*, vol. 30, pp. 1273–1280, 2009.

[29] J. Schwier, R. R. Brooks, and C. Griffin, "Methods to window data to differentiate between markov models," *IEEE Trans. Sys. Man and Cyber. B*, vol. 41, no. 3, pp. 650–663, 2010.

[30] Y. Lu, J. Schwier, R. Craven, R. R. Brooks, and C. Griffin, "Inferring statistically significant hidden markov models," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 7, pp. 1548–1558, 2013.

[31] C. R. Shalizi and K. L. Shalizi, "Blind construction of optimal nonlinear recursive predictors for discrete sequences," in *Proc. ACM International Conference on Uncertainty in Artificial Intelligence*, 2004.

[32] C. Shalizi and J. Crutchfield, "Computational mechanics: Pattern and prediction, structure and simplicity," *J. Statistical Physics*, vol. 104, no. 3/4, 2001.

[33] C. Shalizi, K. Shalizi, and J. Crutchfield, "Pattern discovery in time series, part i: Theory, algorithm, analysis, and convergence," Santa Fe Institute, Tech. Rep., 2002.

[34] C. Griffin, R. Brooks, and J. Schwier, "Determining a purely symbolic transfer function from symbol streams: Theory and algorithms," in *Proc. American Control Conference*, Seattle, WA, June 2008, pp. 4065–4067.

[35] R. Hogg and E. Tanis, *Probability and Statistical Inference*, 7th ed. Pearson/Prentice Hall, 2006.

[36] C. Quesenberry and D. Hurst, "Large sample simultaneous confidence intervals for multinomial proportions," *Technometrics*, vol. 6, no. 191-195, 1964.

[37] *Dynamic Programming and Markov Decision Processes.* Cambridge, MA: MIT Press, 1960.

[38] Q. Hu and W. Yue, *Markov decision processes with their applications.* New York: Springer, 2008, vol. 14. [Online]. Available: www.summon.com

[39] J. Filar and K. Vrieze, *Competitive Markov Decision Processes.* New York, NY, USA: Springer-Verlag, 1997.

[40] J. K. Burgoon and D. B. Buller, "Interpersonal deception: Iii. effects of deceit on perceived communication and nonverbal behavior dynamics," *Journal of Nonverbal Behavior*, vol. 18, no. 2, pp. 155–184, 1994.

[41] J. Bell and B. Whaley, *Cheating and Deception.* Transaction Publ., 1991. [Online]. Available: https://books.google.com/books?id=ojmwSoW8g7IC

[42] B. Whaley, "Toward a general theory of deception," *Journal of Strategic Studies*, vol. 5, no. 1, pp. 178–192, 1982. [Online]. Available: http://dx.doi.org/10.1080/01402398208437106

[43] P. Johnson and S. Grazioli, "Fraud detection: Intentionality and deception in cognition," *Accounting, Organizations and Society*, vol. 25, pp. 355–392, 1993.

[44] R. Mawby and R. W. Mitchell, *Deception: Perspectives on Human and Nonhuman Deceit.* State University of New York Press, 1986, ch. Feints and Ruses: An Analysis of Deception in Sports.

[45] E. Santos Jr and G. Johnson Jr, "Toward detecting deception in intelligent systems," in *Defense and Security.* International Society for Optics and Photonics, 2004, pp. 130–141.

[46] J. F. George and J. Carlson, ""group support systems and deceptive communication"," in *Proceedings of the 32nd Hawaii International Conference on System Sciences.* IEEE, January 1999.

[47] D. R. Norman, "How to identify credible sources on the web," 2001, master's thesis, Joint Military Intelligence College.

[48] C. Griffin and K. Moore, "A framework for modeling decision making and deception with semantic information," in *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on.* IEEE, 2012, pp. 68–74.

[49] E. Paulson and C. Griffin, "Computational complexity of the minimum state probabilistic finite state learning problem on finite data sets," December 2014, submitted to Journal of Applied Mathematics and Computation.

[50] C. Cluskey, "A linear integer program for the causal state splitting and reconstruction algorithm," Honor's Thesis, Schreyer Honors College, Penn State Univeristy, Spring, 2013.

[51] M. Schmiedekamp, A. Subbu, and S. Phoha, "The clustered causal state algorithm: Efficient pattern discovery for lossy data-compression applications," *Comput. Sci. Eng.*, vol. 8, no. 5, 2006.

[52] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Comm. ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[53] C. Griffin and E. Paulson, "Optimal process control of symbolic transfer functions." International Feedback Computing Workshop, April 2015.

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

[55] *Statistical Process Adjustment for Quality Control.* Wiley Interscience, 2002.

[56] P. Morris, *Introduction to Game Theory.* Springer, 1994.

[57] G. Box and G. Jenkins, *Time Series Analysis: Forecasting and Control.* Holden-Day, 1976.

[58] R. Brooks, J. Schwier, and C. Griffin, "Behavior detection using confidence intervals of hidden markov models," *IEEE Trans. Sys. Man and Cyber. B*, vol. 39, no. 6, pp. 1484–1492, December 2009.