Computational Complexity of the Minimum State Probabilistic Finite State Learning Problem on Finite Data Sets∗

Elisabeth Paulson† Christopher Griffin†

Abstract

In this paper, we study the problem of determining a minimum state probabilistic finite state machine capable of generating statistically identical symbol sequences to samples provided. This problem is qualitatively similar to the classical Hidden Markov Model problem and has been studied from a practical point of view in several works beginning with the work presented in: Shalizi, C.R., Shalizi, K.L., Crutchfield, J.P. (2002) An algorithm for pattern discovery in time series. Technical Report 02-10-060, Santa Fe Institute. arxiv.org/abs/cs.LG/0210025. In this paper, we show that the underlying problem is NP-hard and thus all existing polynomial time algorithms must be approximations on finite data sets. Using our NP-hardness proof, we show how to construct a provably correct algorithm for constructing a minimum state probabilistic finite state machine given data and empirically study its running time.

1 Introduction

Hidden Markov models (HMMs) are a pattern recognition tool used to construct a Markov model when the state space is unknown. HMMs are used for a wide variety of purposes, such as speech recognition [1], handwriting recognition [2, 3], and tracking [4]. Given a process which produces some string of training data, there are many algorithms that are widely used to infer a Hidden Markov model for the process. In this paper, we focus on the approach developed by Shalizi [5] for producing $\epsilon$-machines from a string of input data, which can be thought of as a kind of HMM. This work is extended in [6, 7] and [8].

Shalizi’s approach to constructing a HMM is to find statistically significant groupings of the training data which then correspond to causal states in the HMM. 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 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.

Assuming an input string of infinite length where the conditional probability distributions converge to their true values, the CSSR algorithm produces a minimal-state $\epsilon$-machine for the given process. That is, any time two states could be combined while still maintaining the desired properties of the $\epsilon$-machine, they are. However, for input sequences of finite length, the CSSR algorithm

∗Portions of Dr. Griffin’s and Ms. Paulson’s work were supported by the Army Research Office under Grant W911NF-11-1-0487.
†Applied Research Laboratory, Penn State University, University Park, PA 16802
does not guarantee a minimal-state machine due to the technique by which the strings are grouped. Consequently, state explosion can occur. This paper seeks to develop a new approach, based on the CSSR algorithm, which always guarantees a minimal-state HMM. We use a very similar idea to the CSSR algorithm, but formulate it as an integer programming problem whose goal is to minimize the number of states. We then provide a reformulation of the integer programming problem as a minimal clique covering problem, which provides a faster algorithm for finding the minimal-state HMM.

2 Notation and Preliminaries

Let \( \mathcal{A} \) be a finite set of symbols or alphabet representing the observable events of a given system. Let \( y \) be the symbolic output of our system, so \( y \) is a sequence composed of the elements in \( \mathcal{A} \). We will write \( y = y_1 y_2 \ldots y_n \) to denote the individual elements in \( y \), so \( y_i \in \mathcal{A} \).

Let \( \mathcal{A}^* \) be the set of all sequences composed of symbols from \( \mathcal{A} \), and let \( \lambda \) be the empty sequence. If \( y \) is a sequence, then \( x \) is a subsequence if

1. \( x \) is a sequence with symbols in \( \mathcal{A} \) and
2. There are integers \( i \) and \( k \) such that \( x = y_i y_{i+1} \cdots y_{i+k} \).

The subsequences of \( y \) of length \( k \) constitute the sliding data windows of length \( k \) over \( y \).

For us, a hidden Markov model 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, \( \delta \subseteq Q \times \mathcal{A} \times Q \) is a transition relation, and \( p : \delta \to [0, 1] \) is a probability function such that

\[
\sum_{a \in \mathcal{A}, q' \in Q} p(q, a, q') = 1 \quad \forall q \in Q
\] (1)

This is slightly different than the standard definition of an HMM as observed in [1] because we are particularly interested in the state transition relation and associated probabilities, rather than the symbol probability distribution for a given state.

The Baum-Welch Algorithm is the standard expectation maximization algorithm used to determine the transition relation and probability function of a hidden Markov model. However, this algorithm requires an initial estimate of the transition structure, so some initial knowledge of the structure of the Markov process governing the dynamics of the system must be known.

Given a sequence \( y \) 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 hidden Markov model 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 [9].

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) [9].

The CSSR Algorithm is straightforward. We are given a sequence \( y \in \mathcal{A}^* \) and know a priori the value \( L \in \mathbb{N}^+ \). For values of \( i \) increasing from 0 to \( L \), we identify the set of sequences \( W \) that are subsequences of \( y \) and have length \( i \). (When \( i = 0 \), the empty string is considered to be a subsequence of \( y \).) We compute the conditional distribution on each subsequence \( x \in W \) and partition

---

\[ \text{1There are some heuristics for choosing } L \text{ found in [9].} \]
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 \( x \) to the conditional distribution of an existing state and add \( x \) to this state if the conditional distributions are ruled identical. Distribution comparison can be carried out using either a Kolmogorov-Smirnov test or a \( \chi^2 \) 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 \( \delta \). Algorithm 1 shows the CSSR Algorithm.

### Algorithm 1 – CSSR Algorithm

#### Initialization:
1: Define state \( q_0 \) and add \( \lambda \) (the empty string) to state \( q_0 \). Set \( Q = \{q_0\} \).
2: Set \( N := 1 \).

#### Splitting (Repeat for each \( i \leq L \))
1: Set \( W = \{x||y| \in Q(x \in q \wedge |x| = i - 1)\} \) \{The set of strings in states of the current model with length equal to \( i - 1 \)\}.
2: Let \( N \) be the number of states.
3: for each \( x \in W \) do
   4:   for each \( a \in A \) do
      5:     if \( ax \) is a subsequence of \( y \) then
      6:       Determine \( f_{ax|y}: A \rightarrow [0, 1] \) \{the probability distribution over the next input symbol\}.
      7:       Let \( f_{q|y}: A \rightarrow [0, 1] \) be the unified state conditional probability distributions; that is, the probability given the system is in state \( q_i \), that the next symbol observed will be \( a \). For each \( j \), compare \( f_{q_j|y} \) with \( f_{ax|y} \) using an appropriate statistical test with confidence level \( \alpha \). Add \( ax \) to the state that has the most similar probability distribution as measured by the \( p \)-value of the test.
      8:     end if
   9:   end for
10: end for

#### Reconstruction
1: Let \( N_0 = 0 \).
2: Let \( N \) be the number of states.
3: while \( N_0 \neq N \) do
4:   for each \( i \in 1, \ldots, N \) do
5:     Set \( k := 0 \).
6:     Let \( M \) be the number of sequences in state \( q_i \). Choose a sequence \( x_0 \) from state \( q_i \).
7:     Create state \( p_{ik} \) and add \( x_0 \) to it.
8:     for all sequences \( x_j \) \( (j > 0) \) in state \( q_k \) do
9:       For each \( a \in A \), \( x_j.a \) produces a sequence that is resident in another state \( q_k \). Let \( \delta(x_j.a) = q_k \).
10:      For \( l = 0, \ldots, k \), choose \( x \) from \( p_{lk} \). If \( \delta(x_j.a) = \delta(x, a) \) for all \( a \in A \), then add \( x_j.a \) to \( p_{ik} \). Otherwise, create a new state \( p_{ik+1} \) and add \( x_j \) to it. Set \( k := k + 1 \).
11:    end for
12: end for
13: Reset \( Q = \{p_{ik}\} \); recompute the state conditional probabilities \( f_{q|y} \) for \( q \in Q \) and assign transitions using the \( \delta \) functions defined above.
14: Set \( N_0 = N \).
15: Set \( N \) to be the number of states in our current model.
16: end while
17: The model of the system has state set \( Q \) and transition probability function computed from the \( \delta \) functions and state conditional probabilities.

The complexity of CSSR is \( O(k^{2L+1}) + O(N) \), 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. Given a stream of symbols \( y \), of fixed length \( N \), from alphabet \( A \), the algorithm is linear in the length of
the input data set, but exponential in the size of the alphabet.

2.1 A Practical Problem with CSSR

As observed in Shalizi and Shalizi (2004), the CSSR converges to the minimum state estimator asymptotically [9], 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 $N \rightarrow \infty$, the probability of a history being assigned to the wrong equivalence class approaches zero. A formal proof of this is given in [5] 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 $y$ defined as follows:

\begin{equation}
\begin{align*}
y_i &= 0 \quad 1 \leq i \leq 518 \\
[y_{i-3} y_{i-2} y_{i-1} y_i] &= 1100 \quad 518 < i \leq 582 \\
[y_{i-2} y_{i-1} y_i] &= 100 \quad 582 < i \leq 645 \\
[y_{i-2} y_{i-1} y_i] &= 101 \quad 645 < i \leq 648
\end{align*}
\end{equation}

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:

| string | $Pr(0|\text{string})$ | $Pr(1|\text{string})$ |
|--------|-----------------------|-----------------------|
| 00     | 0.9314                | 0.0686                |
| 01     | 0.5789                | 0.4211                |
| 11     | 1.0                   | 0                     |
| 10     | 0.9737                | 0.0263                |

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

| string | 00   | 01   | 11   | 10   |
|--------|------|------|------|------|
| 00     | 1.0000 | –   | –   | –   |
| 01     | 0.0067 | 0.0000 | – | –   |
| 11     | 0.0944 | 0.7924 | 1   | –   |
| 10     | –   | –   | –   | 1   |

We now step through the Splitting phase of the CSSR algorithm:

1. String 00 is put into its own state, $q_1$

2. We use Pearson’s Chi-Squared test to test if the 01 and 00 have the same conditional distribution. This results in a $p$-value of approximately 0, as seen Table 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 0.0067, 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 0.0944. Comparing it to 00 results in a p-value of 0, and comparing it to 11 results in a p-value of 0.7924. Since 0.7924 is greater than 0.0944 and greater than the chosen significance level of 0.05, 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 with three different states:

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

We now go on to the Reconstruction step. Let \( x_{ij} \) be a state in \( q_i \) where \( j \in 1, 2, \ldots, |q_i| \). The Reconstruction step checks whether, for each \( a \in A \), \( \delta(x_{i1}, a) = \delta(x_{i2}, a) = \ldots = \delta(x_{ik}, a) \). Recall that \( \delta(x_{ij}, a) = q_k \) means that \( x_{ij}a \) 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 0.0944 is also greater than our alpha-level of 0.05. 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:

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

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 Integer Programming Formulation

Much of this section has been adapted from Cluskey (2013) \[10\]. Suppose we have a string \( y \in A \) and we assume that \( L \) is known. Let \( W \) be the set of distinct strings of length \( L \) in \( y \), 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:

\[ x_{ij} = \begin{cases} 
1 & \text{string } i \text{ maps to state } j \\
0 & \text{else} \end{cases} \quad (3) \]

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:

\[ z_{il}^\sigma = \begin{cases} 
1 & \text{string } i \text{ maps to string } j \text{ upon symbol } \sigma \\
0 & \text{else} \end{cases} \quad (4) \]

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

\[ (x_{ij} = 1) \land (z_{il}^\sigma = 1) \land (x_{lk} = 1) \implies (y_{jk}^\sigma = 1) \]

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 \( \sigma \), and string \( l \) is clustered in state \( k \), then \((j, \sigma, k)\) 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 \( \mu \), where

\[ \mu_{il} = \begin{cases} 
1 & \text{strings } i \text{ and } l \text{ have identical conditional probability distributions} \\
0 & \text{else} \end{cases} \quad (5) \]

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

\[ (x_{ij} = 1) \land (x_{lj} = 1) \implies (\mu_{il} = 1) \]

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, this variable simply enumerates up the number of “used” states out of \( n \). That is:

\[ p_j = \begin{cases} 
0 & \sum_i x_{ij} = 0 \\
1 & \text{else} \end{cases} \quad (6) \]
This can be enforced as the following constraint

\[ p_j \geq \frac{\sum_i x_{ij}}{S}, \quad p_j \in \{0, 1\} \]

Note that \( \sum_i x_{ij} \leq 1 \). These constraints are equivalent to Equation \( 6 \) 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 [10], 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.

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

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

The following proposition is clear from the construction of Problem 7:

**Proposition 3.1.** 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 greater 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 \( \mu \).

### 3.1 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 integer program finds the following solution for the variables \( z, x, y, u \), and \( p \) seen in Table 3. Our objective
Table 3: Integer Program Variables

|       | \(q_1\) | \(q_2\) | \(q_3\) | \(q_4\) |
|-------|--------|--------|--------|--------|
| 00    | 1      | 0      | 0      | 0      |
| 01    | 0      | 1      | 0      | 0      |
| 11    | 0      | 0      | 1      | 0      |
| 10    | 1      | 0      | 0      | 0      |

|       | \(u\)   | 11     | 10     |
|-------|--------|--------|--------|
| 00    | 1      | 0      | 0      |
| 01    | 0      | 1      | 0      |
| 11    | 0      | 0      | 1      |
| 10    | 1      | 0      | 0      |

|       | \(z^0\) | 11     | 10     |
|-------|--------|--------|--------|
| 00    | 1      | 0      | 0      |
| 01    | 0      | 0      | 0      |
| 11    | 0      | 0      | 1      |
| 10    | 1      | 0      | 0      |

|       | \(z^1\) | 11     | 10     |
|-------|--------|--------|--------|
| 00    | 0      | 1      | 0      |
| 01    | 0      | 0      | 1      |
| 11    | 0      | 0      | 1      |
| 10    | 1      | 0      | 0      |

|       | \(y^0\) | \(y^1\) |
|-------|--------|--------|
| \(q_1\) | 1      | 0      |
| \(q_2\) | 0      | 1      |
| \(q_3\) | 0      | 0      |

|       | \(p\)   |
|-------|--------|
| 1     | 0.9341 |
| 1     | 0.0659 |
| 1     | 0.5789 |

Table 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      |

function is to minimize the sum of the variables \(p_j\). As can be seen in Table 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 4. Our final reduced state space machine is shown in Figure 1.

![Figure 1](image)

Figure 1: The reduced state space machine derived from the input given by Equation 2

4 Computational Complexity of MSDpFSA

In this section we assert that the integer program given by Problem 7 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) [11]. Like the CSSR algorithm, the algorithm presented in [11] 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 7 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.

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

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

**Proposition 4.1.** 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 8 by letting \( n \) be the number of unique strings of length \( L \) in \( y \), 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 only if they have identical conditional probability distributions. We show that Problem 8 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 \( (1 - x_{ij}) + (1 - x_{lj}) + \mu_{il} \geq 1 \forall i, l, j \) 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, it is clear that the integer program given in Problem 8 is equivalent to a minimal clique covering and is NP-hard.

**Remark 3.** A proof of the NP-hardness of Problem 8 would be similar to the proof of Proposition 4.1 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.

**4.1 Example of Reduction**

We will illustrate the equivalence of Equation 8 and the minimal clique covering through an example. Using the same example as before, the resulting graph \( G \) is shown in Figure 2.
10 have an edge in common because they have identical conditional distributions as noted by Table 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 8 produces either of the following two results:

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

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

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

![Figure 2: The graph \(G\) corresponding to the string \(y\) given in Equation 2.](image1)

![Figure 3: The graph \(G\) considered by the CSSR algorithm.](image2)

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 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 2. For a vertex \(i\), CSSR only places an edge between \(i\) and \(\arg\max_{j \neq i} \{T(i, j) : T(i, j) > 0.05\}\). Thus, by the CSSR algorithm, the graph is really the image shown in Figure 3. 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.

5 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 \(W = \{S_1, \ldots, S_n\}\) 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, Algorithm 1, 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 2 [12].

Define $H := \text{argmin}_R\{|R| : R \subset C \land \cup_i R_i = W\}$. 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 3. We can think of Algorithm 3 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:

\[
y_i = \begin{cases} 
1 & C_i \in H \\
0 & \text{else} 
\end{cases} \tag{9}
\]

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

\[
I_{ij} = \begin{cases} 
1 & S_i \in C_j \in H \\
0 & \text{else} 
\end{cases} \tag{10}
\]

\[
s_{ij} = \begin{cases} 
1 & \text{S}_i\text{is mapped to } C_j \in H \\
0 & \text{else} 
\end{cases} \tag{11}
\]

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 5.1.** 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 $R = \text{argmin}_H\{|H| : H \subset C \land \cup_i H_i = W\}$ because $|R| = k$ and $k$ is the clique covering number of $G$, so it is impossible to find an $H$ with cardinality less than $k$. 

**Proposition 5.2.** Any solution to $Q(C)$ results in a minimal clique vertex covering.

**Proof.** This is clear by noting that argmin $Q(C)$ results in the subset, $H$, of $C = \{C_1, C_2, ..., C_m\}$ where $H$ is defined as $\text{argmin}_R\{|R| : R \subset C \land \cup_i R_i = W\}$ in conjunction with Proposition 5.1.

Before discussing the rest of the algorithm, which concerns determinism, we first state an important remark about Algorithm 3. 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 4 and 5 for an example. The set $H$ is shown in Figure 4. From this set $H$, we can deduce the two minimal clique vertex coverings shown in 5. In initializing Algorithm 4 which imposes determinism on each covering, we find the set of all minimal coverings for which each vertex belongs to one clique.
Algorithm 2 – Finding all maximal cliques

Input: Observed sequence $y$; Alphabet $A$, Integer $L$

Setup:
1: Call the set of strings that appear in $y$ $W = \{S_1, \ldots, S_n\}$, so $W$ is our set of vertices
2: Determine $f_{x|y}(a)$ for each unique string
3: Generate an incident matrix, $U \in \mathbb{R}^{n \times n}$, $U_{ij} = u_{ij}$, where $u_{ij}$ is defined in $\[a\]
4: $P \leftarrow [1, 2, \ldots, n]$ { $P$ holds the prospective vertices connected to all vertices in $R$ }
5: $R \leftarrow \emptyset$ { Holds the currently growing clique }
6: $X \leftarrow \emptyset$ { Holds the vertices that have already been processed }

function Bron-Kerbosch($R, P, X$)
1: $k \leftarrow 0$
2: if $P = \emptyset$ and $X = \emptyset$
3: report $R$ as a maximal clique, $R = C_k$
4: $k \leftarrow k + 1$
5: end if
6: for each vertex $v$ in $P$
7: Bron-Kerbosch($R \cup v, P \cap N(v), X \cap N(v)$) { where $N(v)$ are the neighbors of $v$ }
8: $P \leftarrow P \setminus v$
9: $X \leftarrow X \cup v$
10: end for
11: return \{ $C_1, C_2, \ldots, C_m$ \}

Algorithm 3 – Minimal Clique Vertex Covering

Input: Observed sequence $y$; Alphabet $A$, Integer $L$

Set-up:
1. CSSR
   run the CSSR Algorithm (Algorithm 1)
   return $k$, the number of cliques found by CSSR
2. Find all maximal cliques
   run Algorithm 2
   return $\{C_1, C_2, \ldots, C_m\}$

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 $C_j$

$$ Q(C) = \min \sum_i y_i \quad \{ y_i \text{ indicates whether clique } C_i \text{ is being used or not} \} $$

s.t. $y_i \geq \sum_j s_{ij} |C_j|$  
$y_i \leq \sum_j s_{ij}$  
$\sum_i y_i < k$  
$s_{ij} \leq I_{ij}$  
$\sum_j s_{ij} = 1$
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 the reconstruction part of Algorithm 1. Note in Algorithm 4 each vertex corresponds to a string of length $L$, which came from an initial string $y$. Also recall that we have previously defined the binary variables $z$ as

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

where $\sigma$ is any element of our alphabet $\mathcal{A}$. The following proposition is clear from the construction of Algorithms 3 and 4.

**Proposition 5.3.** Algorithm 3 along with Algorithm 4 produces an encoding of a minimum state probabilistic finite machine capable of generating the input sequence.

![Figure 4: An example output from Algorithm 3](image1)

![Figure 5: Minimal clique coverings where each vertex is only in one clique](image2)
Algorithm 4 – Minimal Clique Covering Reconstruction

Input: Observed sequence $y$; Alphabet $A$, Integer $L$;

Set-up:
Perform Algorithm 2 on $y$ 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 5. Let this set of minimal coverings be $T = \{T_1, \ldots, T_l\}$.

From $y$, $A$, and the set $\{S_1, \ldots, S_n\}$ as found in Algorithm 2, determine the matrix $z$.

Minimal Clique Covering Reconstruction:
1: for $h = 1$ to $l$ do
2: $i = 1$
3: $N =$ the number of cliques in each $T_h$ note that all $T_h$ have the same number of cliques because they are all minimal
4: $M = N$
5: while $i \leq M$ do
6: $N = M$
7: $F =$ the set of all vertices (strings) which are in clique $i$
8: if $\text{length}(F) > 1$ then
9: for $j = 2, 3, \ldots, \#F$ do
10: Use $z$ to find what clique $F(j)$ maps to for each $\sigma \in A$
11: if $F(j)$ does not map to the same clique as $F(1)$ upon each $\sigma$ then
12: if $M > N$ and $F(j)$ maps to the same clique as some $F(k)$ upon each $\sigma$, $k \in [N+1, M]$ then
13: Add $F(i)$ to the clique containing $F(k)$
14: else
15: Create a new clique containing $F(i)$
16: $M = M + 1$
17: end if
18: end if
19: end for
20: end if
21: $i = i + 1$
22: end while
23: end for
24: FinalCovering = $\min\{C_h|C_h$ has the minimum number of columns $\forall h\}$
return FinalCovering

6 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 6 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 6 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 6 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 6 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.

7 Conclusion and Future Directions

In this paper we illustrated the the problem stated by Shalizi and Crutchfield [5] 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 [11]. As such, this shows that both the CSSR algorithm of [5] and CSSA algorithm of [11] 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 there approximation properties are. In addition to this work, determining weaknesses in this approach to modeling behavior are planned. We are particularly interested in the affects deception can have on models learned in this way. For example, we are interested in
Figure 7: Minimal clique covering run times compared to CSSR
formulating a problem of constrained optimal deception in which a learner, using an algorithm like the one described here or the CSSR or CSSA approximations, is optimally confused by an input stream that is subject to certain constraints in its incorrectness.

A Computing $f_{q_i|y}$ and $f_{x|y}$

The following formulas can be used to compute $f_{q_i|y}$ and $f_{x|y}$ in Algorithm 1. Let $(x, y)$ be the number of times the sequence $x$ is observed as a subsequence of $y$.

$$f_{x|y}(a) = \Pr(a|x, y) = \frac{\#(xa, y)}{\#(x, y)} \quad (12)$$

$$f_{q_i|y}(a) = \Pr(a|q_i, y) = \frac{\sum_{x \in q_i} \#(xa, y)}{\sum_{x \in q_i} \#(x, y)} \quad (13)$$

References

[1] L. R. Rabiner, “A tutorial on hidden markov models and selected applications in speech recognition,” Proceedings of the IEEE, vol. 77, no. 2, pp. 257–286, 1989.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] C. Cluskey, “A linear integer program for the causal state splitting and reconstruction algorithm,” Honor’s Thesis, Schreyer Honors College, Penn State University, Spring, 2013.
[11] 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.

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