The Pennsylvania State University
Schreyer Honors College

Department of Computer Science and Engineering

A Linear Integer Program for the Causal State Splitting and Reconstruction Algorithm

Christopher Cluskey
Spring 2013

A thesis
submitted in partial fulfillment
of the requirements for a baccalaureate degree
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Christopher H. Griffin
Thesis Supervisor

John Hannan
Honors Adviser

*Signatures will be on file if this thesis is approved.

Abstract

We provide an introduction to Hidden Markov Models (HMM) followed by a description of the Causal State Splitting and Reconstruction Algorithm, a variation on a HMM construction method. We then describe how to write the CSSR Algorithm as a linear integer program and demonstrate a simple example.

Contents

List of Figures

1 Introduction to Hidden Markov Models and the Causal State Splitting and Reconstruction Algorithm

1.1 Hidden Markov Model Introduction

Hidden Markov Models have been around since the late 1960s and early 1970s, but are still useful today [**?**]. To get an idea of how a Hidden Markov Model can be used, lets start with an example. Imagine we are given a sequence of coin toss results and are assigned the task of predicting future coin toss results from the given data. It may be the case that it is just a fair coin being tossed over and over again, but that may not be the case. Maybe we start with a fair coin, but, after every flip there is a 50% chance of switching to a coin that is biased to come up tails 90% of the time. Then when we flip that coin there is a 50% chance we will switch back to the original coin. If our given data looks like HTTHHTTTTTTTTH we are more inclined to believe that this is the case (because of all the tails results after flipping the 3rd heads). A Hidden Markov Model tries to infer these 'hidden states' instead of just guessing that we are flipping a coin that is biased to come up heads only about 29% (4/14) of the time. (See Figure **??**.)

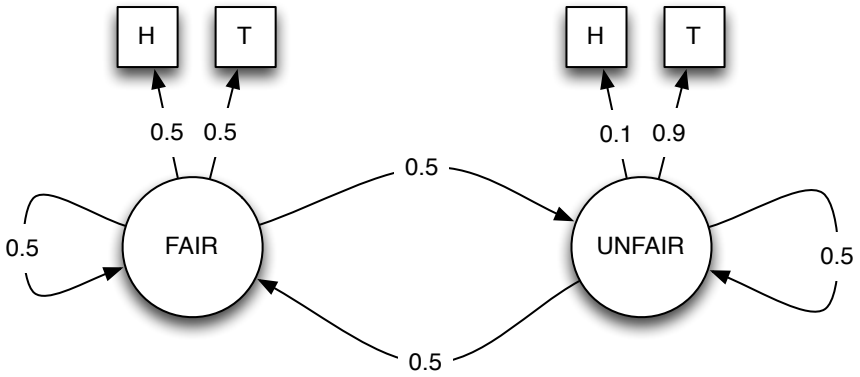

Figure 1: An example of a Hidden Markov Model.

Hidden Markov Models often use matrices to represent the probabilities of each outcome depending on the current state. For our previous coin toss example, the following matrix could be used:

$$\begin{bmatrix} 0.5 & 0.5 \\ 0.1 & 0.9 \end{bmatrix}$$

Where the first column contains the probability of flipping heads and the second column contains the probability of flipping tails. Also, the first row contains the symbol emission probabilities for state 1 (the fair coin) and the second row for state 2 (the biased coin). You also will have a second matrix to represent your state transitions:

$$\begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$$

Where the first column is the probability of moving to state 1 (where you are flipping a fair coin) and the second column is the probability of moving to state 2 (where you are flipping the biased coin). And the first row represents state 1 and the second row represents state 2.

A Hidden Markov Model can be expressed as $\lambda = (A, B, \pi)$ where A is the state transition probabilities, B is the observed symbol emission probabilities, and π is the initial state distribution. Given these, there are three problems that must be solved for the model to be useful in real world applications [**?**].

The first problem is, given an observation sequence and a model, how to compute the probability that the sequence was produced by the model. This problem can be solved using the Forward Backward Procedure [**?**].

The second problem is, given an observation sequence and a model, how to choose the state sequence that best explains how the observation sequence was produced. This can be solved using the Viterbi algorithm [**?**].

The third problem is, how to optimize the model's parameters to best describe how a given sequence comes about. This process is called *training* the Hidden Markov Model. It is by far the hardest of the three problems, and really we can only find locally optimal solutions [**?**].

1.2 Applications of Hidden Markov Models

Hidden Markov Models can be used to model many different kinds of scenarios. One application is in recognizing online handwriting. To accomplish this, a model is created that looks at a handwritten word and separates it into letters and ligatures that connect the letters. Attempts at this have had varying degrees of success due to the high variability of different handwriting styles [**?**].

Another use for Hidden Markov Models is in speech recognition. A model will analyze samples from an audio clip in order to translate the audio into text. Traditionally, these samples are extracted at discrete intervals in the clip, but it has been found that if variable intervals are used it can help distinguish between similar sounding words. In both

handwritten and speech recognition the input to the Hidden Markov Model is either letters or words [**?**].

When using Hidden Markov Models for speech recognition, it is helpful to know what part of speech each word is. This can increase the accuracy of recognizing future words. You can use Hidden Markov Models to tag words with their parts of speech. When you know a current word's part of speech as well as the previous word's part of speech, it increases the probability of correctly recognizing words [**?**].

Hidden Markov Models have also been using in gait recognition. These models focus on both gait shape and gait dynamics. This approach uses the silhouette of the subject to determine this information. This can be used to recognize an individual just from analyzing his/her gait. It has been found that there are some factors that will affect the correctness of recognition. These include the viewpoint, subjects shoes, the walking surface, if anything is being carried, as well as the speed at which the subject is walking. Of these, differing surfaces caused the model the most trouble in identifying its subject [**?**].

Hidden Markov Models have been used when identifying technology users using biometrics. Biometrics refers to identifying humans by their personal characteristics and traits. In one case study, users entered usernames and passwords into a computer and this data was used to build a Hidden Markov Model which learned about the typing patterns of each individual. This model was based not only on what letters were being typed, but also the timing between letters and which letters were clustered together. The model worked very well to identify users and predict characters because of the stochastic nature of typing patterns [**?**].

Determining credit card fraud is another area in which Hidden Markov Models have proved useful. A Hidden Markov Model can be trained with a cardholder's normal spending patterns including factors such as shipping address, billing address, and whether the cardholder is classified as a high, medium, or low spender. Then, incoming transactions are fed into the model, and if they are not accepted with sufficiently high probability, the transaction will be flagged and the bank can either decline the transaction or notify the customer of possible fraud. The right probability can be hard to find because we want to be able to find as many fraudulent transactions as possible while not accidentally rejecting genuine transactions [**?**].

Hidden Markov Models have also been used to help object tracking in video sequences. When the approximate trajectory of an object is known, a Hidden Markov Model can be used to estimate where the object will be from frame to frame of a video. Then, traditional object tracking algorithms can limit their search to the location predicted

by the Hidden Markov Model. Since a video contains multidimensional data, multidimensional Hidden Markov Models are used when estimating an object's position [**?**].

1.3 CSSR Theory

The CSSR (Causal-State Splitting Reconstruction) Algorithm was developed at Cornell University as a way to generate a Hidden Markov Model from an observed sequence [**?**]. The CSSR Algorithm is just a heuristic with the goal of state minimization. The generated Hidden Markov Model has output symbols tied to state transitions, and thus differs slightly from traditional Hidden Markov Models

To best explain how the CSSR Algorithm works, it is easiest to give an example. Consider the binary sequence 0101010101 where the alphabet contains only 0's and 1's. The Algorithm takes an input variable L which is the maximum length of a subsequence to consider. For simplicity lets choose our L to be 1. First we find the probability of encountering a 0 or a 1. There are five 0s and five 1s out of ten symbols, so the probability of choosing either a 0 or a 1, if we have no information about the previous symbol, is 0.5. Then we find the conditional probabilities. These are the probabilities of seeing a specific symbol given that we have already seen a certain subsequence. For example, given that we saw the subsequence 0, the probability of seeing a 1 is 1.0. This is calculated by dividing the number of times we see 0, in this case 5, by the number of times we see 01, also 5. So we get 5/5 or a probability of 1.0. Then we find the the probability of seeing a 0 given that we have already seen a 0. Then we do the same for a given subsequence of 1. If we had chosen $L = 2$ we would continue this for the subsequences 00, 01, 10, and 11. Now that we know these probabilities we can construct the Hidden Markov Model in Figure **??**. Once we have this figure, we begin the reconstruction step of the CSSR Algorithm to determine whether any of the 3 states we found can be combined, or if any of the states need to be split to ensure determinism of the resulting probabilistic finite state automaton. In this simple example it turns out that none of the states need to be split or combined [**?**].

See a more formal explanation of the CSSR Algorithm in Algorithm **??**.

Algorithm **??** is the CSSR algorithm presented by Crutchfield and Shalizi, with one exception. In the work in [**?**], Shalizi recommends removing transient states associated with the resulting Markov model (as they do not represent long-run dynamics.) We do not require the user to remove these states, since they may have important transient

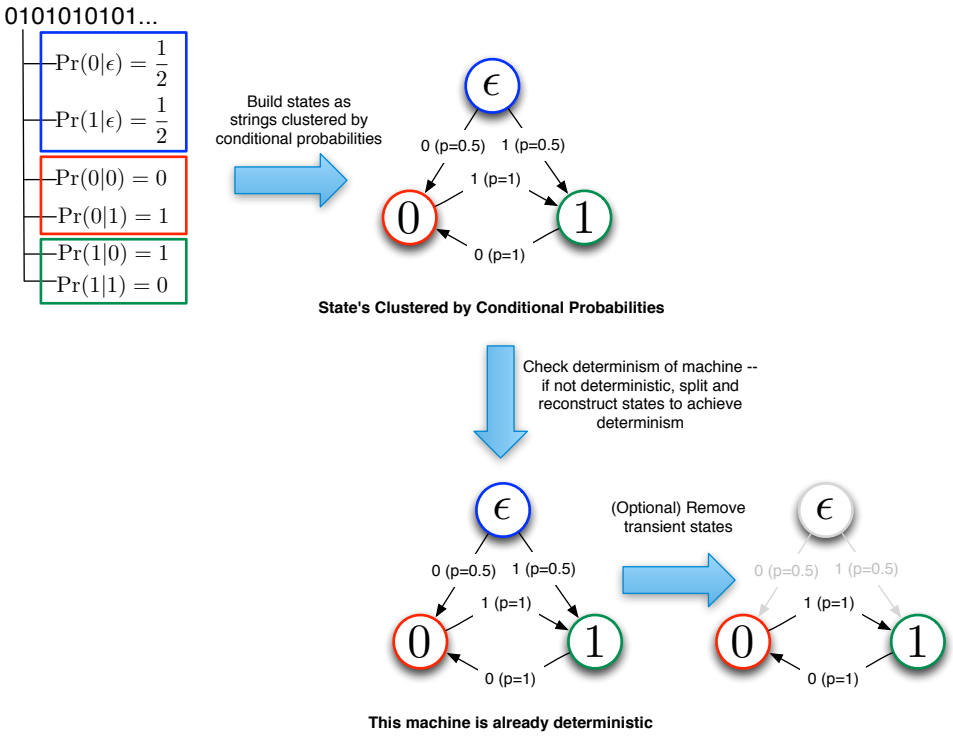

Figure 2: HMM generated by CSSR on 01 example.

Algorithm Description 1.1 – CSSR Algorithm

Input: Observed sequence $\mathbf{y}$; Alphabet $\mathcal{A}$, Integer l;
Initialization:

1. Define state q_0 and add ϵ to state q_0. Set $Q = \{q_0\}$.
2. Set $N := 1$.

Splitting (Repeat for each $i \leq l$)

1. Set $W = \{\mathbf{x} | \exists q \in Q(\mathbf{x} \in q \wedge |\mathbf{x}| = i - 1)\}$. Let N be the number of states.
2. For each $\mathbf{x} \in W$, for each $a \in \mathcal{A}$, if $a\mathbf{x}$ is a subsequence of $\mathbf{y}$, then
   (a) Determine $f_{a\mathbf{x}|\mathbf{y}} : \mathcal{A} \to [0, 1]$, the probability distribution over the next input symbol.
   (b) Let $f_{q_i|\mathbf{y}} : \mathcal{A} \to [0, 1]$ 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 i, compare $f_{q_i|\mathbf{y}}$ with $f_{a\mathbf{x}|\mathbf{y}}$ using an appropriate statistical test with confidence level α. (Generally, the Kolmogorov-Smirnov test or the χ^2-test is used.) Add $a\mathbf{x}$ to the state that has the most similar probability distribution as measured by the p-value of the test. If all tests reject the null hypothesis that $f_{q_i|\mathbf{y}}$ and $f_{a\mathbf{x}|\mathbf{y}}$ are the same, then create a new state q_{N+1} and add $a\mathbf{x}$ to it. Set $N := N + 1$.

Reconstruction

1. Set $N_0 = 0$
2. Let N be the number of states.
3. Repeat the following while $N_0 \neq N$:
   (a) For each $i \in 1, \ldots, N$: Set $k := 0$. Let M be the number of sequences in sate q_i. Choose a sequence $\mathbf{x}_0$ from state q_i. Create state p_{ik} and add w_0 to it. For all sequences $\mathbf{x}_j$ $(j > 0)$ in state q_i:
      i. For each $a \in \mathcal{A}$ $\mathbf{x}_j a$ produces a sequence that is resident in another state q_k. Let $\delta(\mathbf{x}_j, a) = q_k$.
      ii. For $l = 0, \ldots, k$, choose $\mathbf{x}$ from p_{ik}. if $\delta(\mathbf{x}_j, a) = \delta(\mathbf{x}, a)$ for all $a \in \mathcal{A}$, then add $\mathbf{x}_j$ to p_{ik}. Otherwise, create a new state p_{ik+1} and add $\mathbf{x}_j$ to it. Set $k := k + 1$.
   (b) Set $N_0 = N$.
   (c) Let N be the number of states in the current model.
4. Recompute the state probabilities and assign transitions using the δ function defined above.

information.

1.4 Computing $f_{q_i|\mathbf{y}}$ and $f_{\mathbf{x}|\mathbf{y}}$

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

$$f_{\mathbf{x}|\mathbf{y}}(a) = \Pr(a|\mathbf{x}, \mathbf{y}) = \frac{\#(\mathbf{x}a, \mathbf{y})}{\#(\mathbf{x}, \mathbf{y})} \tag{1}$$

$$f_{q_i|\mathbf{y}}(a) = \Pr(a|q_i, \mathbf{y}) = \frac{\sum_{\mathbf{x}\in q_i} \#(\mathbf{x}a, \mathbf{y})}{\sum_{\mathbf{x}\in q_i} \#(\mathbf{x}, \mathbf{y})} \tag{2}$$

1.5 Properties of Algorithm ??

Proposition 1.1. *The probabilistic finite state machine returned by Algorithm* **??** *is minimally stochastic. Further, the states produced by Algorithm* **??** *are sufficient statistics for the future symbols produces by the process.*

Proof. See Corollary 1 and Theorem 4 of [**?**]. □

It is good to note that the time complexity of this algorithm as computed by Shalizi is $O(k^2L + 1) + O(N)$ Where k is the number of symbols in the alphabet, L is the maximum length subsequence, and N is the length of the given sequence [**?**].

2 CSSR Algorithm as an Integer Program

2.1 Introduction

In this chapter we will create a CSSR like integer program that will converge on the results of the CSSR Algorithm. It will find the states of a Hidden Markov Model given an input string. Unlike the CSSR Algorithm, this integer program is not a heuristic and will return the minimum state Hidden Markov Model.

2.2 Construction of the Integer Program

To create this integer program, we need a sequence. Lets consider the sequence $\mathbf{x} = \langle x_1, \ldots, x_n, \ldots \rangle \in \mathcal{A}^+$. Where $\mathcal{A}^+$ is the alphabet that all the x_t's are taken from. For each x_t there is some p such that x_t is

a function of the previous p terms in the sequence x (p is the same as L from Chapter 1).

From this information, we can construct a function η that gives us a mapping from strings of length p into the set of multinomial distributions over $\mathcal{A}$

$$\eta\left(\langle x_{t-1},\ldots,x_{t-p}\rangle\right)(\sigma) = \Pr\left(x_t = \sigma|\langle x_{t-1},\ldots,x_{t-p}\rangle\right) \tag{3}$$

Given the sequence x the maximum likelihood estimator for the probability of seeing a symbol $\sigma \in \mathcal{A}$ can be determined if we know the number of times $\langle x_1,\ldots,x_p\rangle$ occurs in x and the number of times $\langle x_1,\ldots,x_p,\sigma\rangle$ occurs in x. The MLE is given below.

$$\eta\left(\langle x_{t-1},\ldots,x_{t-p}\rangle\right)(\sigma) = \frac{\#of\langle x_1,\ldots,x_p,\sigma\rangle}{\#of\langle x_1,\ldots,x_p\rangle} \tag{4}$$

As the observed sequence, x, becomes larger, the value of the MLE will converge on the true value of η

In order to define states for the FSA, we can create a state for each sequence $\langle x_1,\ldots,x_p\rangle$. For example, on seeing a symbol σ, state $\langle x_1,\ldots,x_p\rangle$ will transition to state $\langle x_2,\ldots,x_p,\sigma\rangle$. This way of creating states can however create excess states and we may need to cluster together. When clustering states we need to keep two things in mind. First, make sure all strings in a given state have identical conditional distributions, and second, the resulting transition function is deterministic. The following shows a direct approach to achieving these goals, unlike the clustering and splitting approach used by Shalizi [**?**].

2.3 Main Results

Assuming we know p let W be the number of all distinct subsequences of length p in a given subsequence x. We need to cluster the elements in W such that the two conditions mentioned above are satisfied. Clearly, the number of clusters created will be between 1 and $|W|$. Since the conditions are satisfied if we give each string it's own cluster, the problem is merely to minimize the number of clusters needed.

If we number the strings in W $1,\ldots,|W|$, we can define the following binary variable:

$$x_{ij} = \begin{cases} 1 & \text{String } i \text{ maps to cluster } j \\ 0 & \text{else} \end{cases} \tag{5}$$

Next, lets define another binary variable:

$$z_{il}^{\sigma} = \begin{cases} 1 & \text{string } l \text{ can be reached from string } i \text{ on symbol } \sigma \\ 0 & \text{else} \end{cases} \tag{6}$$

For example, z_{il}^{σ} would equal 1 if $i = \langle x_1, \ldots, x_p \rangle$ and $l = \langle x_2, \ldots, x_p, \sigma \rangle$

Using these variables and assuming j and k are cluster indexes we can define:

$$x_{ij} = 1 \wedge z_{il}^{\sigma} = 1 \wedge x_{lk} = 1 \implies y_{jk}^{\sigma} = 1 \tag{7}$$

Basically this expression says that there is a transition from state j to state k if there exists strings i and l such that i maps to state j and l maps to state k and l can be reached from i upon seeing the symbol σ. We can rewrite this statement as a constraint:

$$(1 - x_{ij}) + (1 - z_{il}^{\sigma}) + (1 - x_{lk}) + y_{jk}^{\sigma} \geq 1 \quad \forall i, j, k, l, \sigma \tag{8}$$

To ensure the transition function is deterministic, we must add the following constraint:

$$\sum_{k} y_{jk}^{\sigma} \leq 1 \quad \forall j, \sigma \tag{9}$$

To ensure all strings in a given state have identical conditional distributions, let's define the binary variable μ_{il}. μ_{il} will equal 1 if and only if a chosen statistical test (e.g. the Pearson's χ^2 test) indicates that string i and string l have identical distributions. This leads us to the following relation:

$$x_{ij} = 1 \wedge x_{lj} = 1 \Leftrightarrow \mu_{il} = 1 \tag{10}$$

Which can then be written as the constraints:

$$(1 - x_{ij}) + (1 - x_{lj}) + \mu_{il} \geq 1 \quad \forall i, l, j \tag{11}$$

Using these constraint, we can create the following integer program. This program, if feasible, will define a mapping of strings to states that satisfies the two conditions defined above.

$$P(N) \begin{cases} \min & \displaystyle\sum_{j,k,\sigma} y_{jk}^{\sigma} \\ s.t. & (1 - x_{ij}) + (1 - z_{il}^{\sigma}) + (1 - x_{lk}) + y_{jk}^{\sigma} \geq 1 \quad \forall i, j, k, l, \sigma \\ & \displaystyle\sum_{k} y_{jk}^{\sigma} \leq 1 \quad \forall j, \sigma \\ & (1 - x_{ij}) + (1 - x_{lj}) + \mu_{il} \geq 1 \quad \forall i, l, j \\ & \displaystyle\sum_{j} x_{ij} = 1 \quad \forall i \\ & x_{ij}, z_{il}^{\sigma}, y_{jk}^{\sigma} \in \{0, 1\} \forall i, j, k, l, \sigma \end{cases} \tag{12}$$

Integer programming is, in general, NP complete, however we will implement this problem in chapter 3 to see if it can be solved in a reasonable amount of time.

To find the smallest number of states needed, we may solve the following bi-level problem:

$$\begin{cases} \min & N \\ s.t. & P(N) \text{ is feasible} \\ & N \in \{1, \ldots, |W|\} \end{cases} \tag{13}$$

The work in this chapter proves the following proposition:

Proposition 2.1. *If there is a feasible solution to Problem* **??** *for a fixed N, but no solution for Problem* **??** *for $N-1$, then the minimum state generator (and output of CSSR) should consist of an N state Markov chain with symbol probabilities constructed according to Equation* **??**. □

3 Linear Program Application

3.1 Introduction

In this chapter we translate all the constraints described in chapter 2 to Maple code. Then we will use this code to solve a simple example and see if the results correspond to the expected results.

3.2 Code Formulation

From the linear program described in **??** we have created the Maple code displayed in Figures **??** through **??**. *GenerateConstraints1* corresponds to the first constraint from **??**. *GenerateConstraints2* corresponds to the second constraint from **??** (The sum of the y_{jk}^{σ}). *GenerateConstraints3* corresponds to the fourth constraint from **??** (The sum of the x_{ij}). Lastly, *GenerateConstraints4* corresponds to the third constraint from **??**.

After we have defined these generate constraint functions, we create constraints for the 01 example talked about in chapter 1. Since we know z_{il}^{σ} will equal 0 when $\sigma \neq l$ we only need 4 of *GenerateConstraints1*. We then run these constraints to the *LPSolve* function.

You can see the solution to this linear program in Figure **??**. The $x_{i,j}$'s are the probabilities of outputting symbol j when in state i. The $y_{i,j,k}$'s are the state transition probabilities (The probability of moving from state i to state k upon seeing the symbol j.

We can see that 2 is the minimum number of states because if we only had 1 state, then from *GenerateConstraints4* we know that either x_{00} or x_{10} would have to equal 0. However, from *GenerateConstraints3* the sum of the x_{0j}'s must be 1 which means that x_{00} must be 1 because there is only 1 state. However, the sum of the x_{1j}'s must be 1 which means that x_{10} must be 1 because there is only 1 state. This creates a contradiction because x_{00} and x_{10} cannot both be 1, so there cannot be a solution with only 1 state.

4 Conclusions and Future Directions

In this thesis, we discussed the problem of deriving a Hidden Markov Model from a stream of data and showed the CSSR algorithm, an alternative to the classical Expectation Maximization algorithm. We noted that the CSSR algorithm is a heuristic, not guaranteed to return a minimum state HMM (as it was designed to do). To compensate for this, we derived a integer linear programming problem to construct an N state deterministic finite state machine with probabilities that would accept a given string with proscribed probabilities, if such a machine exists. We then showed that a bi-level programming problem could be created whose solution is the resulting minimum state probabilistic generator for the string. An example illustrates this construction method.

As part of future work, we should execute scalability testing and derive the computational complexity of solving the proposed integer programming problem and resultant bi-level programming problem. While we are certain these problems are no worse than NP-complete (as integer programming is known to be NP complete) it may be that there is a faster (i.e., polynomial time) solution method. By way of example, we note that the CSSR algorithm runs in polynomial time on the inputs for a fixed alphabet size. This algorithm could be used to find an initial feasible solution, which could be refined in a branch-and-bound integer programming solver method to obtain solutions faster than cold start methods.

```
> with(Optimization):
> #i, l == Strings, j, k == States
> GenerateConstraints1 := proc(sigma :: integer, i :: integer, l :: integer) :: set;
  local j, k, ret:
  ret := {}:
  for j from 0 to 1 do
  for k from 0 to 1 do
  ret := ret union {(1 - x[i,j]) + (1 - x[l,k]) + y[sigma,j,k] ≥ 1}:
  end do:
  end do:
  end:
> GenerateConstraints2 := proc(sigma :: integer, j :: integer) :: set;
  local k, ret, ss:
  ret := {}:
  ss := 0:
  for k from 0 to 1 do
  ss := ss + y[sigma,j,k]:
  end:
  ret := ret union {ss ≤ 1}:
  end:
> GenerateConstraints3 := proc(i :: integer) :: set;
  local j, ret, ss:
  ret := {}:
  ss := 0:
  for j from 0 to 1 do
  ss := ss + x[i,j]:
  end:
  ret := ret union {ss = 1}:
  end:
> GenerateConstraints4 := proc(i :: integer, j :: integer, l :: integer) :: set;
  local ret:
  ret := {}:
  ret := ret union {(1 - x[i,j]) + (1 - x[l,j]) + mu[i,l] ≥ 1}:
  end:
>
> CON := GenerateConstraints1(0,0,0) union GenerateConstraints1(0,1,0)
      union GenerateConstraints1(0,1,1) union GenerateConstraints1(1,1,1)
      union GenerateConstraints2(0,0) union GenerateConstraints2(0,1)
      union GenerateConstraints2(1,0) union GenerateConstraints2(1,1)
      union GenerateConstraints3(0) union GenerateConstraints3(1)
      union GenerateConstraints4(0,0,0) union GenerateConstraints4(0,0,1)
      union GenerateConstraints4(0,1,0) union GenerateConstraints4(0,1,1)
      union GenerateConstraints4(1,0,0) union GenerateConstraints4(1,0,1)
      union GenerateConstraints4(1,1,0) union GenerateConstraints4(1,1,1)
```

$$CON := \{x_{0,0} + x_{0,1} = 1, x_{1,0} + x_{1,1} = 1, 0 \le 2 - 2x_{0,0}, 0 \le 2 - 2x_{0,1}, 0 \le 2 - 2x_{1,0}, 0 \le 2 - 2x_{1,1}, 0 \le 1 - 2x_{0,0} + y_{0,0,0}, 0 \le 1 - x_{0,0} - x_{1,0}, 0 \le 1 - 2x_{0,1} + y_{0,1,1}, 0 \quad \textbf{(1)}$$

Figure 3: Maple code created for CSSR (1)

```
    ≤ 1 − x_{0,1} − x_{1,1}, 0 ≤ 1 − 2 x_{1,0} + y_{1,0,0}, 0 ≤ 1 − 2 x_{1,1} + y_{1,1,1}, 0 ≤ 1 − x_{0,0}
    − x_{0,1} + y_{0,0,1}, 0 ≤ 1 − x_{0,0} − x_{1,0} + y_{1,0,0}, 0 ≤ 1 − x_{0,0} − x_{1,1} + y_{1,0,1}, 0 ≤ 1
    − x_{0,1} − x_{0,0} + y_{0,1,0}, 0 ≤ 1 − x_{0,1} − x_{1,0} + y_{1,1,0}, 0 ≤ 1 − x_{0,1} − x_{1,1} + y_{1,1,1}, 0
    ≤ 1 − x_{1,0} − x_{0,0} + y_{0,0,0}, 0 ≤ 1 − x_{1,0} − x_{0,1} + y_{0,0,1}, 0 ≤ 1 − x_{1,0} − x_{1,1}
    + y_{1,0,1}, 0 ≤ 1 − x_{1,1} − x_{0,0} + y_{0,1,0}, 0 ≤ 1 − x_{1,1} − x_{0,1} + y_{0,1,1}, 0 ≤ 1 − x_{1,1}
    − x_{1,0} + y_{1,1,0}, y_{0,0,0} + y_{0,0,1} ≤ 1, y_{0,1,0} + y_{0,1,1} ≤ 1, y_{1,0,0} + y_{1,0,1} ≤ 1, y_{1,1,0}
    + y_{1,1,1} ≤ 1}

> mu[0, 0] := 1 :
  mu[1, 0] := 0 :
  mu[0, 1] := 0 :
  mu[1, 1] := 1 :

> CON
{x_{0,0} + x_{0,1} = 1, x_{1,0} + x_{1,1} = 1, 0 ≤ 2 − 2 x_{0,0}, 0 ≤ 2 − 2 x_{0,1}, 0 ≤ 2 − 2 x_{1,0}, 0 ≤ 2      (2)
    − 2 x_{1,1}, 0 ≤ 1 − 2 x_{0,0} + y_{0,0,0}, 0 ≤ 1 − x_{0,0} − x_{1,0}, 0 ≤ 1 − 2 x_{0,1} + y_{0,1,1}, 0 ≤ 1
    − x_{0,1} − x_{1,1}, 0 ≤ 1 − 2 x_{1,0} + y_{1,0,0}, 0 ≤ 1 − 2 x_{1,1} + y_{1,1,1}, 0 ≤ 1 − x_{0,0} − x_{0,1}
    + y_{0,0,1}, 0 ≤ 1 − x_{0,0} − x_{1,0} + y_{1,0,0}, 0 ≤ 1 − x_{0,0} − x_{1,1} + y_{1,0,1}, 0 ≤ 1 − x_{0,1}
    − x_{0,0} + y_{0,1,0}, 0 ≤ 1 − x_{0,1} − x_{1,0} + y_{1,1,0}, 0 ≤ 1 − x_{0,1} − x_{1,1} + y_{1,1,1}, 0 ≤ 1
    − x_{1,0} − x_{0,0} + y_{0,0,0}, 0 ≤ 1 − x_{1,0} − x_{0,1} + y_{0,0,1}, 0 ≤ 1 − x_{1,0} − x_{1,1} + y_{1,0,1}, 0
    ≤ 1 − x_{1,1} − x_{0,0} + y_{0,1,0}, 0 ≤ 1 − x_{1,1} − x_{0,1} + y_{0,1,1}, 0 ≤ 1 − x_{1,1} − x_{1,0}
    + y_{1,1,0}, y_{0,0,0} + y_{0,0,1} ≤ 1, y_{0,1,0} + y_{0,1,1} ≤ 1, y_{1,0,0} + y_{1,0,1} ≤ 1, y_{1,1,0}
    + y_{1,1,1} ≤ 1}

>

> MakeObj := proc( ) :: sum;
  local sigma, k, j, ret :
  ret := 0 :
  for sigma from 0 to 1 do
  for j from 0 to 1 do
  for k from 0 to 1 do
  ret := ret + y[sigma, j, k] :
  end do:
  end do:
  end do:
  end:

> OBJ := MakeObj( )
      OBJ := y_{0,0,0} + y_{0,0,1} + y_{0,1,0} + y_{0,1,1} + y_{1,0,0} + y_{1,0,1} + y_{1,1,0} + y_{1,1,1}      (3)

>

> LPSolve(OBJ, CON, assume = binary)
                                                                                                              (4)
```

Figure 4: Maple code created for CSSR (2)

$$\left[4, \left[x_{0,0} = 0, x_{0,1} = 1, x_{1,0} = 1, x_{1,1} = 0, y_{0,0,0} = 0, y_{0,0,1} = 1, y_{0,1,0} = 0, y_{0,1,1} = 1, y_{1,0,0} = 1, y_{1,0,1} = 0, y_{1,1,0} = 1, y_{1,1,1} = 0\right]\right] \quad (4)$$

>

Figure 5: Maple code created for CSSR (3)

Bibliography

[1] Rabiner, L. R. (1989) "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. IEEE*, **77**(2), pp. 257–286.

[2] Sin, B.-K., J.-Y. Ha, S.-C. Oh, and J. Kim (1999) "Network-based approach to online cursive script recognition," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, **29**(2), pp. 321–328.

[3] Lim, S. and M. Clements (1992) "Pseudo-continuous hidden Markov modeling for automatic speech recognition," in *Southeastcon '92, Proceedings., IEEE*, pp. 482–488 vol.2.

[4] Yuan, L. (2010) "Improvement for the automatic part-of-speech tagging based on hidden Markov model," in *Signal Processing Systems (ICSPS), 2010 2nd International Conference on*, vol. 1, pp. V1–744–V1–747.

[5] Liu, Z. and S. Sarkar (2006) "Improved gait recognition by gait dynamics normalization," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **28**(6), pp. 863–876.

[6] Chang, W. (2005) "Improving hidden Markov models with a similarity histogram for typing pattern biometrics," in *Information Reuse and Integration, Conf, 2005. IRI -2005 IEEE International Conference on.*, pp. 487–493.

[7] Srivastava, A., A. Kundu, S. Sural, and A. Majumdar (2008) "Credit Card Fraud Detection Using Hidden Markov Model," *Dependable and Secure Computing, IEEE Transactions on*, **5**(1), pp. 37–48.

[8] Lefevre, S., E. Bouton, T. Brouard, and N. Vincent (2003) "A new way to use hidden Markov models for object tracking in video sequences," in *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, vol. 3, pp. III–117–20 vol.2.

[9] Shalizi, C. R. and K. L. Klinkner (2004) "Blind Construction of Optimal Nonlinear Recursive Predictors for Discrete Sequences," in *Uncertainty in Artificial Intelligence: Proceedings of the Twentieth Conference (UAI 2004)* (M. Chickering and J. Y. Halpern, eds.), AUAI Press, Arlington, Virginia, pp. 504–511. URL `http://arxiv.org/abs/cs.LG/0406011`

[10] Shalizi, C. R., K. L. Shalizi, and J. P. Crutchfield (2002) *Pattern Discovery in Time Series, Part I: Theory, Algorithm, Analysis, and Convergence*, *Tech. rep.*, Santa Fe Institute.

ACADEMIC VITA

Christopher Cluskey
cjc5319@psu.edu

Education

- B.S., Computer Science, Spring 2013, The Pennsylvania State University, University Park, PA

Honors and Awards

- Academic Excellence Scholarship, Schreyer Honors College, Fall 2009 - Spring 2013