Thesis

Consistent Hidden Markov Models

Submitted by

Pradyumna Kumar Narayana Rao Gari

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2014

Master's Committee:

Advisor: Bruce A. Draper

Ross Beveridge
Chris Peterson

Abstract

Consistent Hidden Markov Models

Activity recognition in Computer Vision involves recognizing the appearance of an object of interest along with its action, and its relation to the scene or other important objects. There exist many methods that give this information about an object. However, these methods are noisy and are independent of each other. So, the mutual information between the labels is lost. For example, an object might be predicted to be a tree, whereas its action might be predicted as walk. But, trees can't walk.

However, the compositional structure of the events is reflected by the compositional structure of natural language. The object of interest is the predicate, usually a noun, the action is the verb, and its relation to the scene may be a preposition or adverb. The lost mutual information that says that trees can't walk is present in natural language. The contribution of this thesis is a method of visual information fusion based on exploiting the mutual information from Natural language databases.

Although Hidden Markov Models (HMM) are the traditional way to smooth noisy stream of data by integrating information across time, they can't account for the lost mutual information. This thesis proposes an extension to HMM (Consistent HMM) that can integrate visual information to the lost mutual information by exploiting the knowledge from language databases.

Consistent HMM performs better than other state of the art HMMs on synthetic data generated to simulate the real world behavior. Although the performance gain of integrating the knowledge from language databases both during training phase and run-time is better,

when considered individually, the performance gain is more when the knowledge is integrated during run-time than training.

Acknowledgements

I would like to thank my advisor, Professor Bruce A. Draper, for giving me the opportunity to be involved in Computer Vision research. His guidance and encouragement in every step of my research made this thesis possible. I would also like to thank Dr. Ross Beveridge for being supportive throughout my Masters degree, and Dr. Chris Peterson for introducing me to the term Segre Variety. I am grateful to the faculty and staff of Computer Science department for making me feel the department as my second home.

I also am grateful to my colleagues Stephen O'Hara, Quanyi Mo, Maggie Wigness and Hao Zhang who provided valuable support and advice, especially in my initial days. I extend my gratitude to my friends Wimroy, Somtirtha, Hrushikesh, Rahul Dutta, Jatin for all the help and support through thick and thin. In particular, I would acknowledge Wimroy for helping me become familiarize with Linux and scripting. I am indebted to Satya Abhishek who supported me as a brother and made sure that I didn't miss my family.

I am deeply thankful to the love and support of my family. My parents and brother sacrified all their comforts and always gave me beyond their limits. They always put me first in their life and allowed me to cherish my dreams. My special thanks to my uncles Prasad and Srinivas for being a constant support to my family. I am also thankful to my grand parents, cousins and my extended family for believing and instilling confidence in me. Last but not the least, I am thankful for having Harini in my life who added colors to my plain life. Moreover, she is responsible for all the eye candy images in this thesis.

Table of Contents

List of Figures

CHAPTER 1

Introduction

1.1. Motivation

The ultimate goal of computer vision is to visualize and understand a scene as humans do. The computer vision field is far from making such a system a reality. But, the first goal of such a system would be to recognize the activities/events of the important objects in a scene. Activity recognition of an object of interest is possible by first recognizing the object, and the action it is doing, along with its relation to the scene or other important objects. The compositional structure of the events is reflected by the compositional structure of natural language. The object of interest is the predicate, usually a noun, the action is the verb, and its relation to the scene may be a preposition or adverb. There exist many methods in computer vision to recognize what the object is, what it is doing, and its relation to the scene or other objects in the scene. When these methods are independent of each other, mutual information between them is lost. For example, the object might be predicted to be a tree, whereas its action might be predicted as walk. But, trees can't walk. The mutual information that says that trees can't walk is present in natural language. The contribution of this thesis is a method of visual information fusion based on exploiting the mutual information from natural language databases.

The first step of activity recognition is to recognize the important objects in a scene. The important objects in a scene, the ones that may be the predicates of verbs, for example people and cars, typically move. The first step of focusing attention on objects that move is usually done using background subtraction [42] for fixed cameras, or motion segmentation [11, 24] when a camera is in motion. Once the objects that move are found, they can be

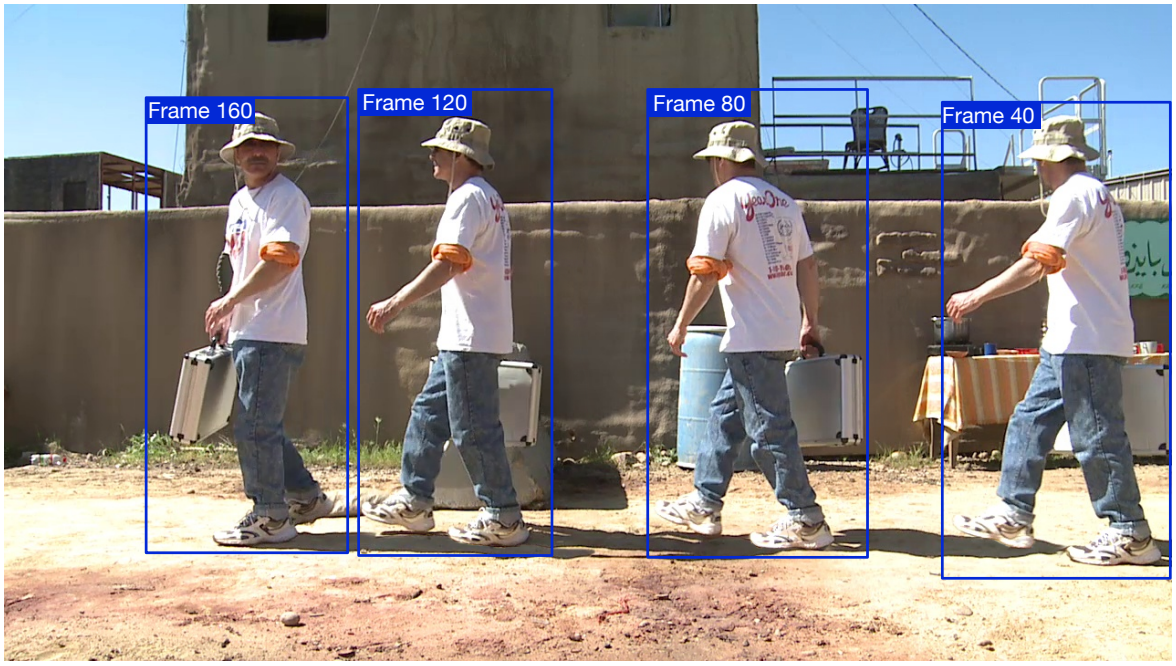

FIGURE 1.1. Person tracked over multiple frames

tracked in multiple frames using adaptive correlation filters [8]. Thus, a video can be divided into a collection of regions of interest spanning into multiple consecutive frames called tracks. Tracking a detected object over multiple frames is essential as the activities we are trying to recognize occur over multiple frames. A track of a person over multiple frames is shown in the figure 1.1. Here, the track which spans over multiple frames is overlayed in a single image for better visual representation.

Once a video is divided into tracks, the information of the tracked object is required. We consider appearance, action, and trajectory information in this thesis. Many factors influence the appearance of an object. One of the most significant factor is the viewpoint. An object may appear differently from different viewpoints. For example, a tree appears differently when viewed from above compared to its view from the ground. So, it is important to know what the detected object is irrespective of the view point. Infact it has been studied, and there exists a viewing sphere where the appearance look the same. A set of such

qualitatively similar viewpoints form an aspect [31]. An appearance is a set of qualitatively similar images from an object, which implies an aspect but also possibly lighting, expression, etc. The appearance of the object is denoted by a noun. Multiple appearances can map to a single noun, and nouns can have synonyms as well. Appearances such as leaf, branch, and trunk map to a noun tree. Car, tree, and person are some examples of the appearance of an object. In addition to the appearance of an object, it is important to to recognize what the objects are doing. It can be recognized by analyzing the motion of the track over multiple time frames and is called the action of the object. Actions are represented by verbs. Walk, jump, and drive are examples of actions performed by objects. In addition to the appearance, and action, the relation of an object to the scene is another important aspect of activity recognition. The position of the object with respect to the scene is a special kind of relation used in this thesis. This special relation of an object with respect to the scene is called trajectory. The trajectory gives information about how fast the object is moving with respect to the scene, usually an adverb, for example "quickly". Moreover, trajectory also gives information about the direction of motion of the tracked object which is a preposition. Trajectory information is important because the appearance and action of an object are related to its trajectory. For example, trees can't move and standing is not possible while moving fast.

Appearance, action, and trajectory information of a tracked object can be estimated at every instance. The appearance of a track can be estimated using clustering methods as shown by Wigness *et al.* [45]. Appearance clustering methods estimate the appearance of a track at every instance by looking at a single instance of the track. However, actions typically occur over time. Classifiers can be trained to predict the action of an object over time. For example, the action of an object can be estimated by clustering tracks on a product

manifold [27]. The trajectory of a track is calculated by measuring the displacement of the center of the track in consecutive frames. We define an appearance labeling system to be a classifier that predicts an appearance label to every instance of a track. Similarly, we define an action labeling system, and a trajectory labeling system to be the classifiers that assign action and trajectory labels at every instance of a track respectively.

The labeling systems are not perfect and the labels they assign are often noisy. For example, appearance labeling system often confuse persons with trees because both trees and persons are elongated parallel vertical structures with unpredictable textures. So, a tree may be mislabeled as a person or vice versa. Moreover, in some cases the labeling systems might not predict any label for some instances of a track. An intelligent vision system must account for noisy or missing labels along with lost mutual information. As discussed later in the thesis, Hidden Markov Models (HMMs) are the traditional way to integrate information across time by smoothing the noise. Appearance, action and trajectory labels of a foreground object do not change often and have structure in time. The noise in appearance, action and trajectory labels can be smoothed using an HMM.

However, apperance, action, and trajectory labels of a track are assigned by independent processes. As a result, the mutual information that exists between the appearance, action, and trajectory labels is lost. HMMs can integrate given information across time. But, they do not account for lost mutual information. The contribution of this thesis is to propose an extension of HMMs that can integrate visual information along with the lost mutual information between them by exploiting it from language databases.

1.2. Thesis Contributions

The contribution of this thesis is to propose a new formulation of HMMs that integrates visual information from multiple sources along with the mutual information associated with them. This mutual information can be exploited from language databases such as wordnet[17], verbnet [40], and framenet [1]. The information regarding these databases is explained in chapter 2. The mutual information exploited from natural language can be expressed in the form of binary constraints. Binary constraints have a value of 1 if a state combination is valid, 0 if not. For example, wordnet, verbnet, and framenet contain knowledge such as people can walk, trees can't walk, and standing is not possible while moving. This information can be expressed in terms of binary constraints. Alternatively, the binary constraint knowledge can also be built manually. Fusing this knowledge with visual information results in better information fusion.

The Baum-Welch algorithm is an expectation-maximization algorithm used to train HMMs. This algorithm is discussed in section 2.1.5. This thesis proposes an extended Baum-Welch algorithm that takes the knowledge from language databases into account without requiring an excessive number of training samples.

The Viterbi algorithm is a method for determining the most likely state sequence, given an observation sequence. This algorithm is discussed in section 2.1.6. This thesis also proposes an extended Viterbi algorithm that improves the prediction of states over the standard Viterbi algorithm by taking binary constraints knowledge from language databases into account. The advantage of the proposed Viterbi algorithm is that the binary constraints can be imposed without retraining the HMM.

We compare the performance of the proposed HMM with other extensions of the HMM on synthetic data generated to mimic real world behavior. Other extensions of HMM considered

in this thesis are Factorial HMM, Fully Coupled HMM, and Brand CHMM, all of which are discussed in chapter 2. The synthetic data is explained in chapter 4. It can be seen that Consistent HMM performs better on synthetic datathan other HMMs listed above. The performance gain of integrating the knowledge from language databases both during training phase and run-time is better. However, when considered individually, the performance gain is more when the knowledge is integrated during run-time.

1.3. Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 will provide necessary background to understand the HMMs. In addition it introduces the different extensions of HMM to integrate information from multiple sources along with the use of HMMs in computer vision. Moreover, the chapter also introduces the English language databases that provide knowledge constraints. Next, Chapter 3 proposes a new formulation of HMMs that takes knowledge from language databases into account. New algorithms to train the HMM and run it are also discussed in this chapter. A detailed description of the synthetic data and the experimental results are discussed in Chapter 4. Finally, concluding remarks and future work are discussed in Chapter 5.

CHAPTER 2

Background

This chapter provides the necessary background to understand the algorithms presented in chapter 3 and the experiments in chapter 4. In particular, section 2.1 begins with a tutorial introduction to the concepts and notation of Hidden Markov Models (HMMs). The algorithms to train and run HMMs are discussed next. A concrete example of an HMM is provided in section 2.2 which is in line with the data used in this thesis. Readers familiar with HMMs can directly jump to section 2.3 that surveys techniques for combining or coupling HMMs to model multiple streams of data. A special emphasis is placed on Coupled HMMs as their structure is the most appropriate one for the data used in this thesis. Constrained HMMs are discussed in section 2.4, as the HMM we are proposing share some commonalities with Constrained HMMs. Section 2.5 reviews natural language knowledge databases that can provide linguistic constraints. Finally, the uses of HMMs in Computer Vision are discussed in section 2.6.

2.1. Hidden Markov Models

Real-world processes have structure in time and produce either discrete or continuous observable outputs called signals. These signals are often corrupted with noise. Modeling such real-world processes is useful in many applications including, but not limited to, computer vision. Hidden Markov Models (HMMs) is a popular method to model such processes [9, 33].

Hidden Markov Models are a special kind of Bayesian network for modeling time series data. They represent probability distributions over sequences of observations [19, 25]. The Hidden Markov Model gets its name from two defining properties. First, an HMM can be

considered as a doubly stochastic model. A stochastic model, also known as a random model, represents the evolution of some random variable, or system, over time. Unlike deterministic models, which can only evolve in one way, stochastic models evolve with some randomness. In an HMM, one stochastic model can be observed as a sequence of observations, while another stochastic model is hidden (states). The hidden stochastic model can be evaluated only through the observations [12]. Second, it assumes that the hidden state satisfies the first order Markov property: that is, the state at time t is dependent only on the state at time $t-1$ and is independent of all the states prior to it. The observations also satisfy a first order Markov property with respect to the states: given a state, its corresponding observation is independent of all other states and observations [19]. Figure 2.1 shows a graphical representation of the dependencies between states and observations in an HMM. The first order Markov property of an HMM is evident from the graph in the figure 2.1. In the figure, the observations are represented by circles and the hidden states are represented by rectangles. This representation is common throughout this thesis.

In the following subsections, we will discuss the terminology related to HMMs along with forward variables and backward variables. Then, we will examine the three basic problems that HMMs solve. In addition, we will explore the Baum Welch algorithm and Viterbi algorithm to solve the basic problems for HMMs.

2.1.1. Elements of HMM

- Set of States: S = $\{S_1, S_2, ..., S_N\}$ where $|S|$ = N. The state at time t is denoted by $q_t = S_i \in S$.
- Set of observation symbols: V = $\{v_1, v_2, ..., v_M\}$ where $|V|$ = M. The observation symbol at time t is denoted by O_t.

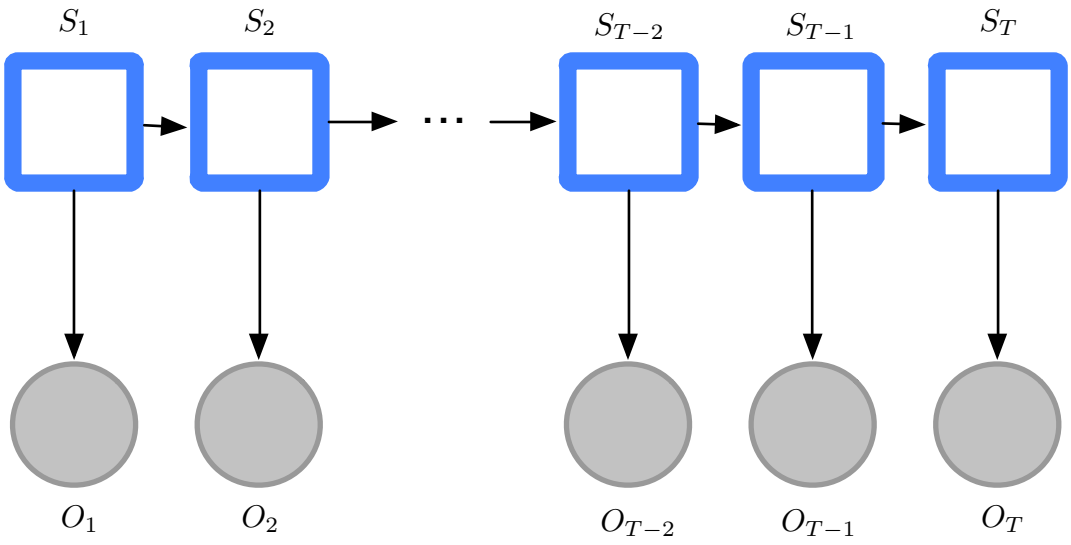

FIGURE 2.1. Dependencies between States and Observations in a Hidden Markov Model

- Transition Probabilities: A $= \{a_{ij}\}$, where $\{a_{ij}\} = P[q_{t+1} = S_j | q_t = S_i]$, $1 \leq i, j \leq N$ (Probability of going from state S_i to state S_j).
- Emission Probabilities: B $= \{b_j(k)\}$, where $b_j(k) = p[O_t = v_k | q_t = S_j]$, $1 \leq j \leq N, 1 \leq k \leq M$ (Probability of outputting symbol v_k from state S_j).
- Initial State Probabilities: $\pi = \{\pi_i\}$, where $\pi_i = P(q_1 = S_i)$, $1 \leq i \leq N$ (Probability of initial state being S_i).

Given the elements of an HMM, an HMM is defined as $\lambda = (A, B, \pi)$, where N and M are implicitly represented by A and B. An HMM also requires a sequence of observed symbols: $O = O_1 O_2 ... O_T$.

Given an HMM λ, and an observation sequence O, the first order Markov properties of states and observations allows us to reformulate the joint distribution of a sequence of states

and observations as follows:

$$P(S_{1:T}, O_{1:T}|\lambda) = P(S_1)P(O_1|S_1)\prod_{t=2}^{T} P(S_t|S_{t-1})P(O_t|S_t) \tag{2.1}$$

2.1.2. Forward Variables

Forward probability ($\alpha_t(i)$) is the probability of the partial observation sequence, $O_1O_2...O_t$, and state S_i at time t, given the model λ. In other words, it is the probability of the state sequence to end at a state S_i at time t, having seen the observations till time t.

$$\alpha_t(i) = P(O_1O_2O_3...O_t, q_t = S_i|\lambda) \tag{2.2}$$

A dynamic programming approach is used to calculate the forward probability of all states over all time steps. The forward probability($\alpha_t(i)$) is calculated inductively using dynamic programming as follows:

(1) Initialization

$$\alpha_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \tag{2.3}$$

(2) Induction

$$\alpha_{t+1}(i) = \left[\sum_{j=1}^{N} \alpha_t(j)a_{ij}\right] b_i(O_{t+1}), \quad 1 \leq t \leq T-1, 1 \leq j \leq N \tag{2.4}$$

This dynamic programming approach for forward variable calculation can be visualized as a state trellis of length T and size N. A visualization of a state trellis is shown in figure 2.2. The dynamic programming approach allows us to calculate the forward variable in the state trellis in $\mathrm{O}\left(TN^2\right)$ time.

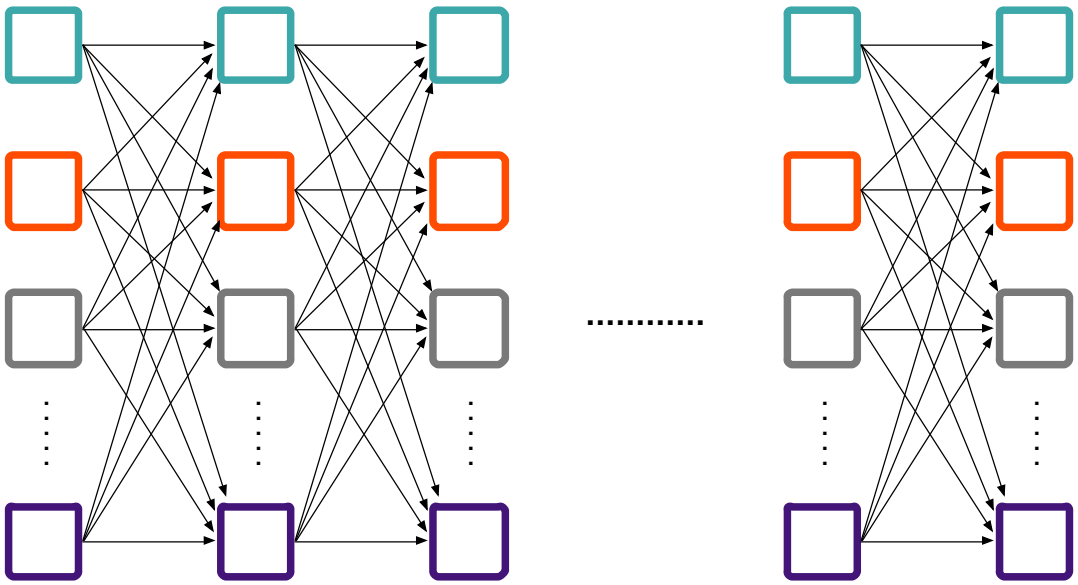

FIGURE 2.2. Example of a State Trellis

2.1.3. Backward Variables

Backward probability ($\beta_t(i)$) is the probability of the observation sequence from time t+1 to the end, given a state i at time t and model λ. It is the probability of observing an observation sequence from time t+1 to the end by starting at a state S_i at time t.

$$\beta_t(i) = P(O_{t+1}O_{t+2}O_{t+3}...O_T|q_t = S_i, \lambda) \tag{2.5}$$

Similar to the forward variable, a dynamic programming approach is used to calculate the backward probability of all states over all time steps. Backward probability ($\beta_t(i)$) is calculated inductively using dynamic programming as follows:

(1) Initialization

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \tag{2.6}$$

(2) Induction

$$\beta_t(i) = \sum_{j=1}^{N} \beta_{t+1}(j) a_{ij} b_j(O_{t+1}), \quad T-1 \geq t \geq 1, \quad 1 \leq j \leq N \tag{2.7}$$

The complexity of the backward probability calculation in a state trellis of length T and width N is O (TN^2).

2.1.4. Three Basic Problems for HMMs

HMMs address three different problems:

(1) Estimating the probability of an observation sequence given a specific HMM.

(2) Estimating the most likely state sequence, given an observation sequence.

(3) Learning the HMM parameters based on the observations.

Problem 1 calculates the probability of an observation sequence O, given the model λ, i.e.,P(O|λ). It is the measure of how likely the observation sequence is generated by the given model. This measure is important to solve the third problem of HMM, namely, finding the model that best explains the data where the HMM parameters are adjusted to maximize the value of P(O|λ). Forward variables and backward variables calculated by equations (2.4) and (2.7) are required to calculate P(O|λ) [4, 6]. Once the forward and backward variables are calculated, the probability of the observation sequence is calculated as:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_t(i) \beta_t(i) \tag{2.8}$$

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{2.9}$$

Problem 2 of an HMM is to find the most likely state sequence that produces a given observation sequence. As already stated, in an HMM, states are hidden, and the observations they generate are visible. So, the hidden states must be inferred from observations. In general, there will be multiple state sequences that generate a given observation sequence. The Viterbi algorithm is an efficient way to calculate one such most likely sequence. It is explained in subsection 2.1.6.

Problem 3 addresses the training of HMMs to maximize the probability of a given observation sequence. The HMM parameters are adjusted until convergence, so as to maximize the value of $P(O|\lambda)$. The Baum-Welch algorithm discussed in subsection 2.1.5 is used to train the HMM with the Expectation Maximization method.

2.1.5. Baum Welch Algorithm

The Baum Welch algorithm addresses the problem 3 of HMMs by training HMMs to maximize the probability of a given observation sequence. The Baum-Welch algorithm [3, 4, 6, 23] is used to train the HMM with the Expectation Maximization method [15]. The following terms calculated using forward and backward variables are used in HMM training.

- $\xi_t(i, j)$: Probability of being in state S_i at time t, and state S_j at time t+1, given the model and observation sequence.

$$\xi_t(i, j) = P(q_t = S_i, q_{t+1} = S_j|O, \lambda) \tag{2.10}$$

$$\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1}\beta_{t+1}(j)}{P(O|\lambda)} \tag{2.11}$$

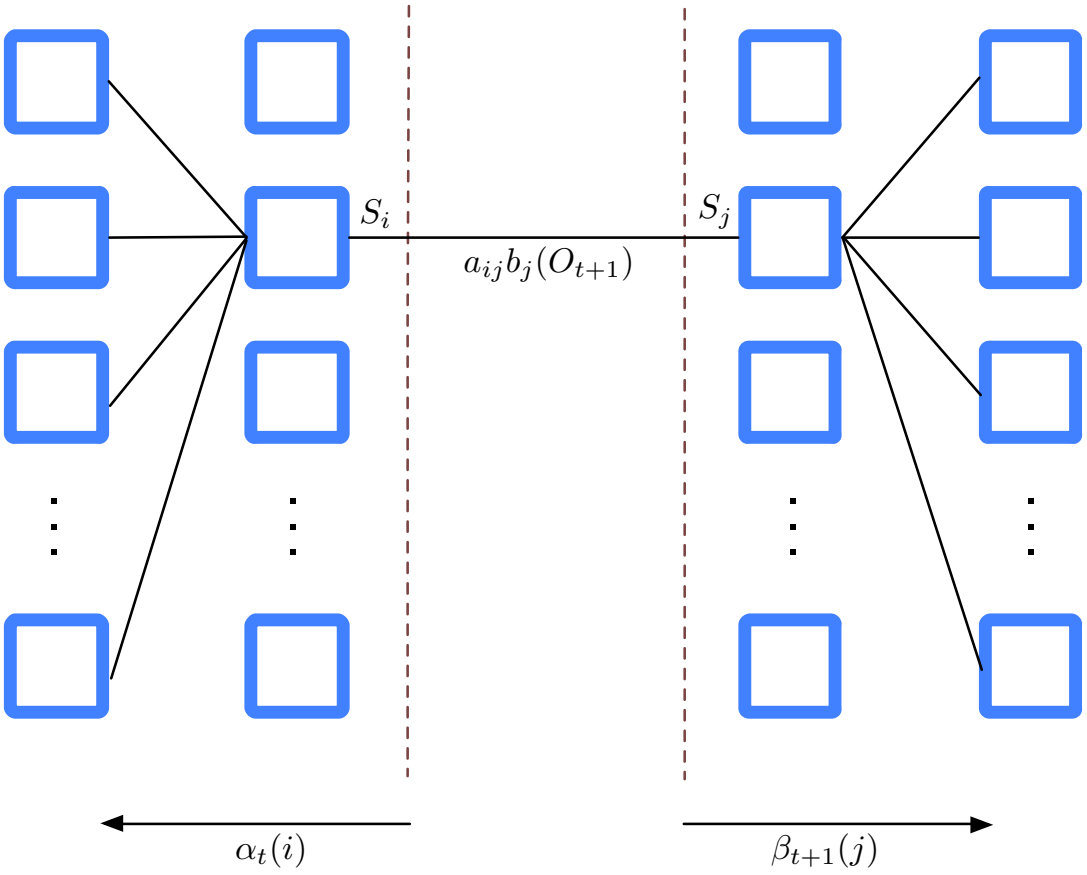

FIGURE 2.3. Probability of being in state S_i at time t and S_j at time t+1

- $\gamma_t(i)$: Probability of being in state S_i at time t, given the observation sequence O, and the model λ.

$$\gamma_t(i) = P(q_t = S_i|O, \lambda) \tag{2.12}$$

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i, j) \tag{2.13}$$

Summing $\gamma_t(i)$ over t, i.e., $\sum_{t=1}^{T-1} \gamma_t(i)$ is the expected number of times the state S_i is visited, given a model λ and an observation sequence O. Similarly, summing $\xi_t(i, j)$ over t, i.e., $\sum_{t=1}^{T-1} \xi_t(i, j)$ gives the expected number of transitions from state S_i to S_j, given a model

λ and an observation sequence O. Then, the parameters of HMM are adjusted as follows:

$$\bar{\pi}_i = \gamma_1(i) \tag{2.14}$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{2.15}$$

$$\bar{b}_j(k) = \frac{\sum_{\substack{t=1 \\ s.t. O_t = v_k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.16}$$

The updated model is represented as $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$. The process is repeated again with the updated model until $P(O|\lambda)$ converges to a local optimum. The Baum-Welch algorithm is guaranteed to converge to a local optimum [7]. The updated model, if not converged, results in better probability of given observation sequence than the previous model, i.e., $P(O|\bar{\lambda}) \geq P(O|\lambda)$ [5, 7]. The initial model is crucial for training, as the Baum-Welch algorithm finds a local optima.

2.1.6. Viterbi Algorithm

An observation sequence can be produced from many different state sequences. It is not practically feasible to find the best state sequence by examining all possible state sequences of an HMM. The Viterbi Algorithm [18, 44] allows us to estimate the most likely state sequence given an observation sequence using dynamic programming. The Viterbi algorithm is similar to the forward algorithm except that the summation in equation (2.4) is replaced with a max. The Viterbi algorithm has an additional backtracking step, as is the case with most dynamic programming algorithms, to find the most likely sequence.

Given observation sequence $O = \{O_1 O_2 ... O_T\}$, let $Q = \{q_1 q_2 ... q_T\}$ be the most likely state sequence. Let the best score probability along a single path ending at state S_i accounting

for first t observations be denoted as $\delta_t(i)$.

$$\delta_t(i) = \underset{q_1,q_2,...q_{t-1}}{\arg\max}\ P[q_1 q_2 ... q_t = S_i, O_1 O_2 ... O_t|\lambda] \tag{2.17}$$

Let ψ be the array to store the best states that maximizes δ at each step. This backpointer array is used to retrieve the most likely state sequence. Having defined δ and ψ, the Viterbi algorithm can be stated as follows:

(1) Initialization

$$\delta_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \tag{2.18}$$

$$\psi_1(i) = 0 \tag{2.19}$$

(2) Induction

$$\delta_{t+1}(j) = \max_{1\leq i\leq N} \left[\delta_t(i) a_{ij}\right] b_j(O_{t+1}), \quad 1 \leq t \leq T-1, 1 \leq j \leq N \tag{2.20}$$

$$\psi_{t+1}(i) = \underset{1\leq i\leq N}{\arg\max} \left[\delta_t(i) a_{ij}\right], \quad 1 \leq t \leq T-1, 1 \leq j \leq N \tag{2.21}$$

(3) Backtracking

$$q_T = \underset{1\leq i\leq N}{\arg\max}[\delta_T(i)] \tag{2.22}$$

$$q_t = \psi_{t+1}(q_{t+1}), \quad t = T-1, T-2, ..., 1 \tag{2.23}$$

The state sequence $\mathrm{Q} = \{q_1 q_2 ... q_T\}$ generated by the Viterbi algorithm is the most likely state sequence, given an observation sequence and an HMM. The complexity of the Viterbi algorithm is $\mathrm{O}\left(TN^2\right)$

2.2. A Concrete Example

Having defined the terminology related to an HMM, along with the algorithms to train and run it, we present a tangible example going forward. This example is also useful to explain the generation of synthetic data in section 4.1. Consider an appearance labeling system that labels a tracked object at every instance. Let the objects that the system recognizes be a person, a car and a tree. In an ideal world, the appearance of the tracked object is always the same. In other words, a person can't transform into a car or vice versa. But, due to tracker errors, a track may jump from one object to other. So, appearance transitions occur. In addition, the labels generated by the appearance labeling system are noisy. Smoothing this noisy stream data can be cast as a Hidden Markov Model problem, where the observation sequences are the noisy labels generated by the appearance labeling system and the hidden states are the actual labels of the tracked object. This problem can be considered as a special case of HMMs, where there is a one-to-one mapping from states to observations, and in fact we use the same names for states and observations, distinguished only by subscripts. For example, $person_o$ is the observation of (noisily) looking like a person, while $person_s$ is the state of actually being a person. The graphical representation of this HMM is shown in figure 2.4.

An HMM is defined by the set of states, observations, prior probabilities, transition probabilities, and observation probabilities. The structure of the HMM is already known here. The states and observations are the labels generated by the system. Prior transition table can be generated by assuming each state as being equally likely. Transition and observation probabilities are assigned randomly and the HMM is trained using the Baum-Welch algorithm until convergence as discussed in subsection 2.1.5. After training, the observation

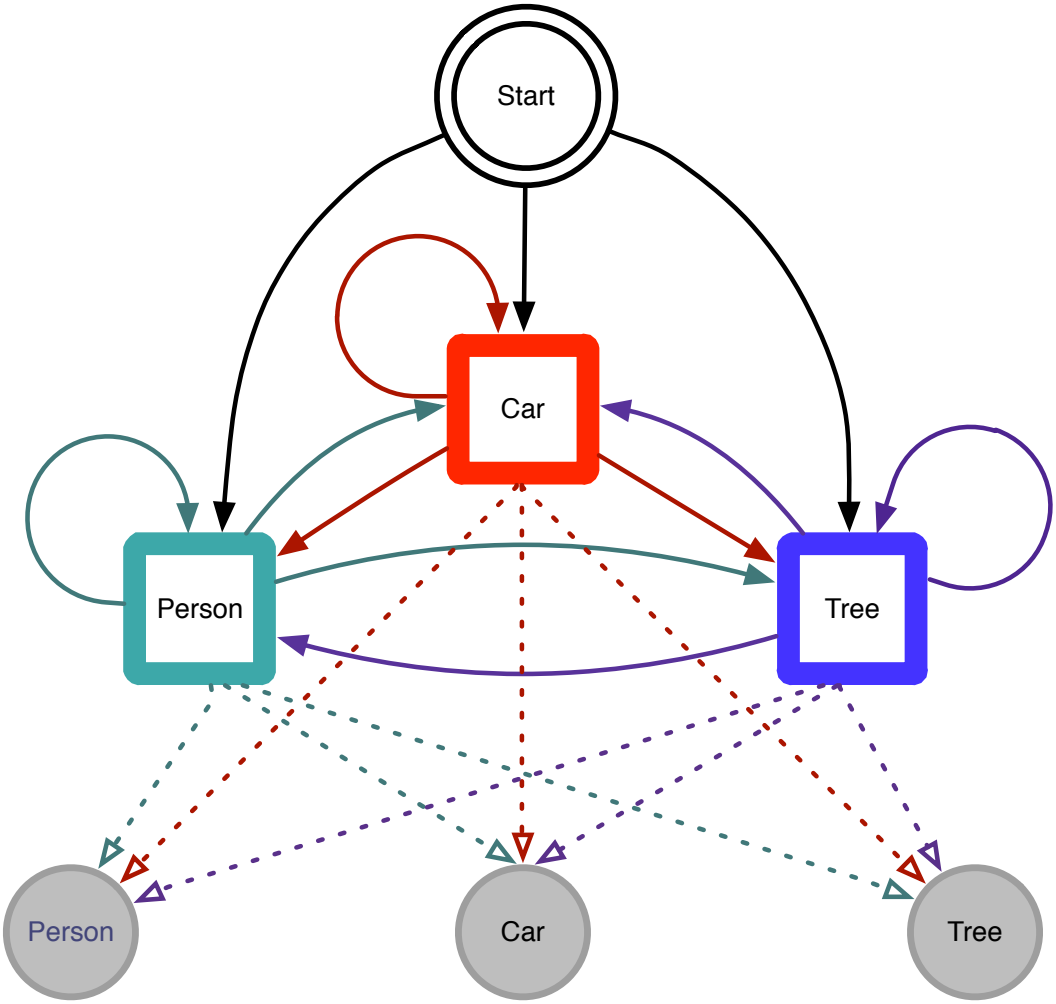

FIGURE 2.4. Example of a Hidden Markov Model

table captures the error distribution of the appearance labeling system, whereas the transition table captures how often the track jumps from one object to the other. Once the HMM is trained, the labels generated by the appearance labeling system (observation sequences) are smoothed by running the Viterbi algorithm discussed in subsection 2.1.6.

Figure 2.4 shows one HMM for object appearance. In addition, we have a similar HMM for actions, where the states and observations are walk, stand, and drive, and yet another HMM for trajectories. The rest of this thesis addresses how to best combine HMMs such as these, knowing that certain combinations of states are not allowed, e.g. trees can't walk.

2.3. Extensions to Hidden Markov Models

Although HMMs are useful to model many time series models, they have limitations in the context of interacting time series models. Many real world signals are generated by multiple processes. These signals can be considered as multiple channels of data generated by multiple processes which may be dependent or independent. The voices in a cocktail party is an example of multiple channels of independent data. Whereas, human behavior is made up of multiple interacting processes. The appearance of an object is obviously correlated to the action it does. A person can walk whereas a tree can't.

Signals from systems with multiple processes are common in the real world. Such signals have structure both in time and space. As HMMs can represent a single variable, signals from multiple processes can be modeled by an HMM by representing the states of the HMM as the combination of the states from all channels. The resulting HMM is called a fully coupled HMM (FCHMM). If there are C chains, each with N states, the FCHMM has N^C states. The complexity of FCHMM is untenable, particularly because enormous amounts of data is required to train such a system. There is often insufficient data to train a FCHMM, leading to undersampling. In other words, a large number of states result in overfitting the data. As the FCHMM is not practical, even for a small number of channels with limited states, HMMs can be extended by coupling, such that the number of states and the amount of training data is manageable. Different types of couplings are discussed in the next subsection.

2.3.1. Varieties of Couplings

Standard HMMs can be extended to model multiple chains of data from multiple processes, either by coupling the outputs or coupling the states. Factorial HMMs, Linked HMMs,

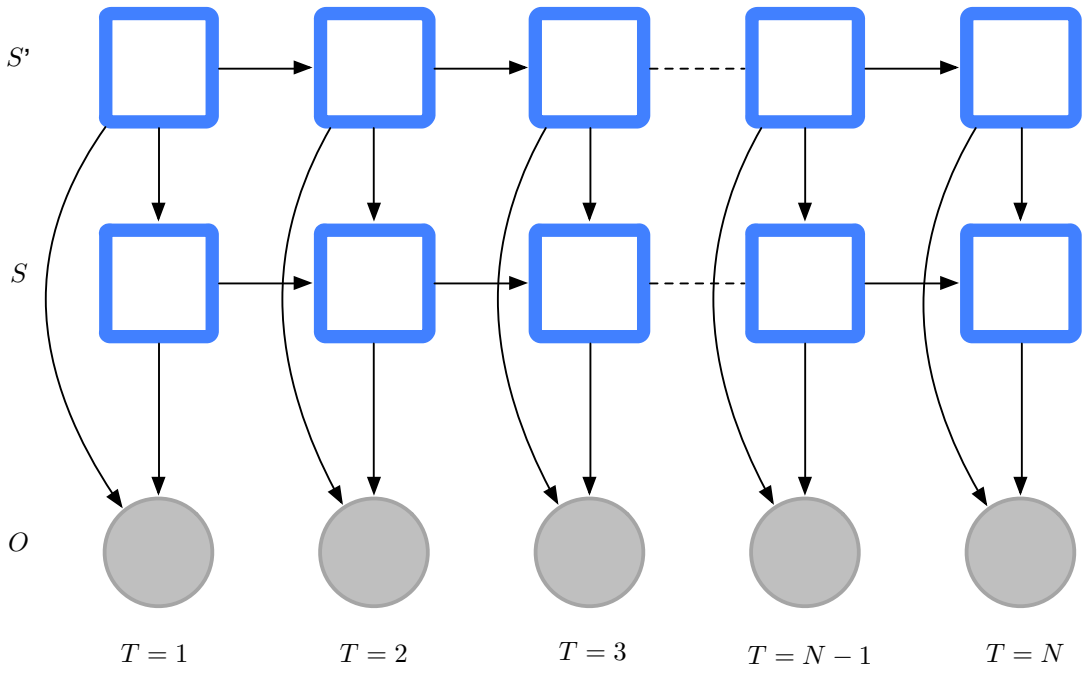

FIGURE 2.5. Factorial Hidden Markov Model

Hidden Markov Decision Trees, and Coupled HMMs are different extensions of HMMs to model multiple streams of data. They are explained below.

One extension of an HMM can be done by modeling each process as a different HMM and coupling their outputs. These models are called factorial Hidden Markov Models (FHMM) [20] and are shown in figure 2.5. This is the weakest coupling and is suited for modeling data from independent processes. FHMMs have a clear advantage over FCHMMs to model data from independent processes: to model each C processes each with N states, FCHMMs require N^C states whereas FHMMs require only NC states.

Although, FHMMs can model multiple independent processes, many interesting applications have multiple channels of data that carry complementary information. Modeling such data with FHMMs is inappropriate as any variation of the interacting processes is modeled as noise. For example, consider appearance and action of a subject as two chains of data. If

a FHMM is used to model this data, at a given instance an HMM may predict the appearance of an object as tree, whereas other HMM may predict the action of the same object as walk. But, trees can't walk. To model such multiple dependent processes, a variety of other inference graph structures have been proposed.

Linked HMMs (LHMM) were proposed by Saul and Jordan [39]. LHMMs are parallel Boltzmann chains coupled by weights that connect their hidden state nodes to exploit the correlation between them as shown in figure 2.6. Saul and Jordan proposed a $\mathrm{O}\left(TN^3\right)$ algorithm to train two Boltzmann chains using decimation, a statistical mechanics method to obtain correlation on a connected pair of nodes. The coupling we propose in this thesis is similar to LHMMs except that the consistency constraints are imposed between hidden states of various chains instead of correlation.

Hidden Markov Decision Trees (HMDT) were proposed by Jordan and Ghahramani [21] to represent hierarchical structure in a signal. They model the constraints imposed by a master process on a slave process as shown in figure 2.7.

Another framework to model data from multiple dependent processes is Coupled HMM (CHMM). CHMMs are appropriate to model processes that have their own internal dynamics but influence each other as shown in figure 2.8. For example, in tennis, both players play according to their own dynamics. But, a player going to the net will drive the opponent back, weak serves will pull him forward. Such models can be modeled well with CHMMs. CHMMs are used for comparison in this thesis and are discussed in detail in the next section.

2.3.2. Coupled Hidden Markov Models

A fully coupled HMM can best model multiple chains of data. But, it requires enormous amounts of training data and has exponential complexity. So, CHMMs are proposed as

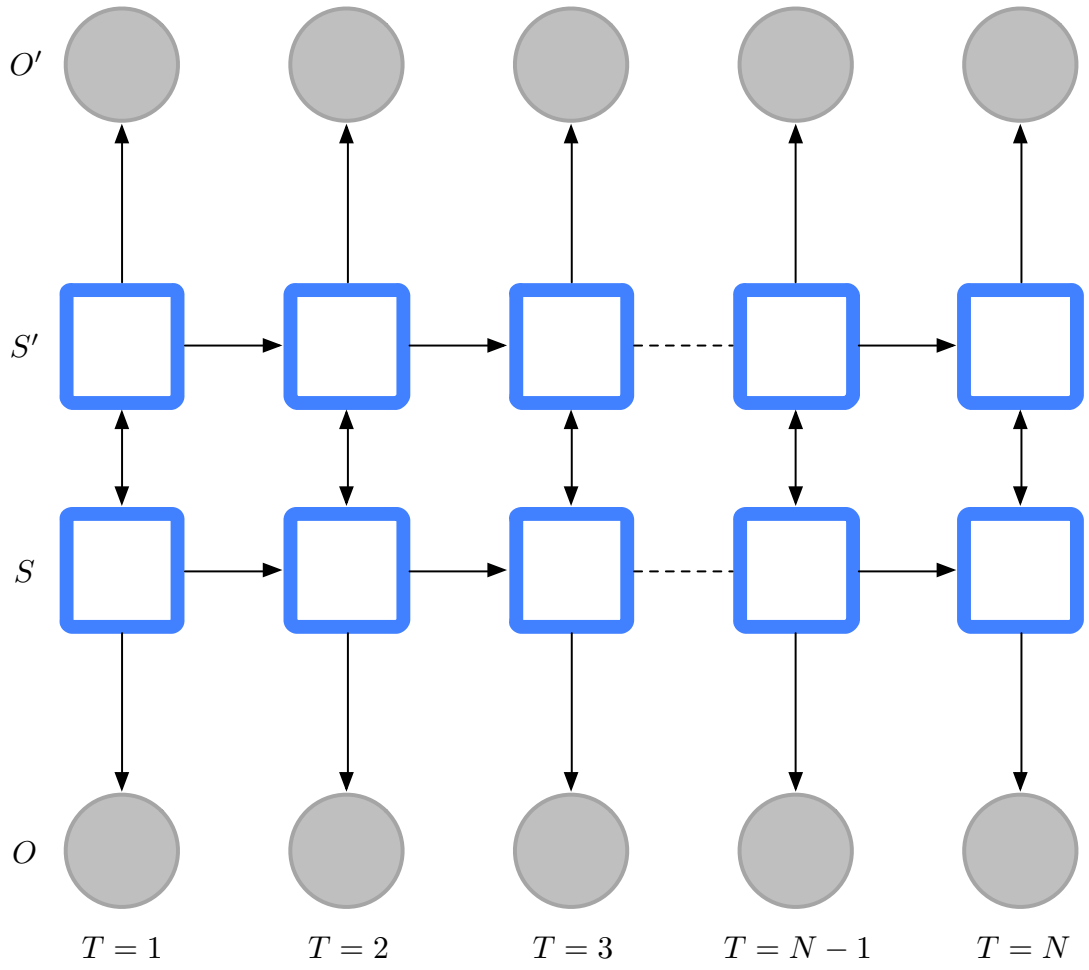

FIGURE 2.6. Linked Hidden Markov Model

an extension of standard HMMs to model multiple chains of data from multiple dependent processes. In a CHMM, a state at time t depends on states at time t-1 of all chains as shown in figure 2.8. Different variations of CHMMs are proposed by Rezek *et al.* [35, 36], Brand *et al.* [9, 10], Zhong and Ghosh [46].

For a two chain CHMM, Rezek *et al.* proposed the likelihood function that encodes the HMM model as

$$P(S, O|\lambda) = P(S_1)P(S'_1)\prod_{t=1}^{T} P(O_t|S_t)P(O'_t|S'_t)P(S_{t+1}|S_t, S'_t)P(S'_{t+1}|S_t, S'_t) \tag{2.24}$$

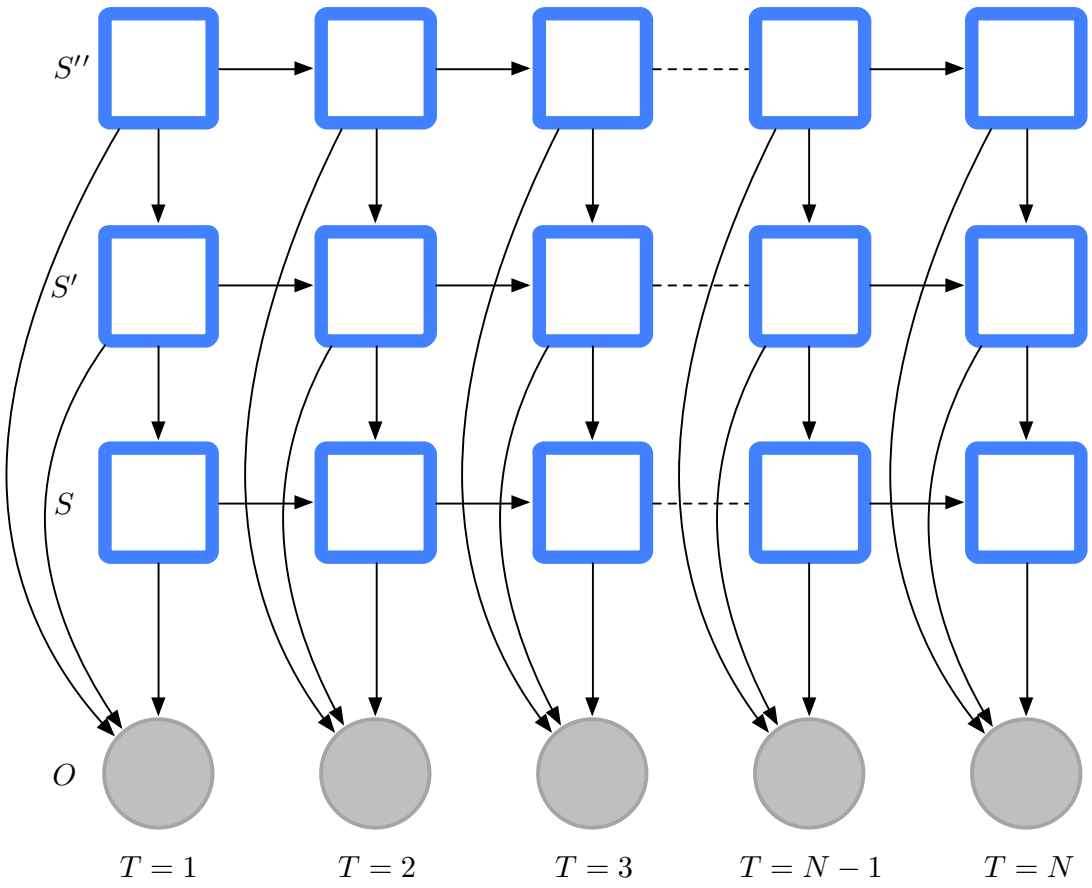

FIGURE 2.7. Hidden Markov Decision Tree

where,

- S_t denotes the state at time t of the first chain at time t
- S'_t denotes the state of the second chain at time t
- O_t is the observation of first chain at time t
- O'_t is the observation of second chain at time t
- $P(S_{t+1}|S_t, S'_t)$ are the state transition probabilities of the first chain
- $P(S'_{t+1}|S_t, S'_t)$ are the state transition probabilities of the second chain
- $P(O_t|S_t), P(O'_t|S'_t)$ are the observation densities of the first and second chains respectively
- $P(S_1), P(S'_1)$ are the prior probabilities of first and second chains respectively

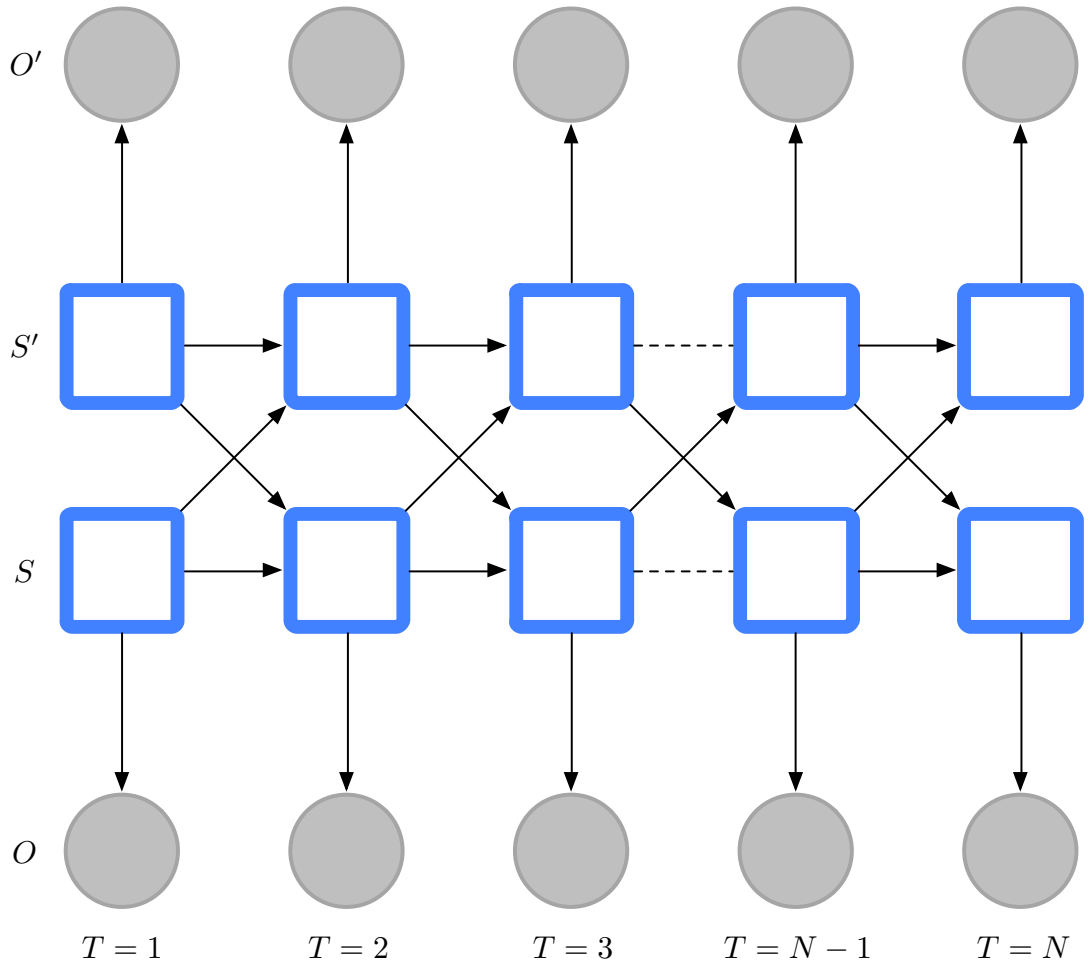

FIGURE 2.8. Coupled Hidden Markov Model

They maximized the likelihood function in equation (2.24) using maximum likelihood estimation [35] or Maximum a Posteriori Estimation [36]. Irrespective of the methods to maximize the likelihood function, the CHMM model proposed by Rezek *et al.* requires to estimate a transition table of size $NC \times N^C$. Although the number of parameters in the proposed transition table is less than that of a FCHMM, the complexity is still exponential.

Brand *et al.* proposed a *factor and couple* HMM to couple HMMs with causal influences [10]. This HMM is based on projections between standard HMMs and a FCHMM. HMMs are coupled by introducing coupling parameters between hidden state variables of different chains. In a factor and couple HMM, training is done in the fully coupled state

space via standard HMM methods. Then a subspace manifold is embedded within the fully coupled state space that represents all possible parameterizations of a much smaller system of coupled HMMs. For a factor and couple HMM of C chains, each with N states, the transition table dimensions are $NC \times NC$. Although the parameters to learn are not exponential in the number of chains, the training of a factor and couple HMM still takes place in cartesian state space. The factor and couple HMM has the same time complexity as a FCHMM.

Brand proposed a deterministic N-heads dynamic programming approach to train CHMM in $\mathrm{O}\left(T(CN)^2\right)$ time [9] via Expectation Maximization. Inference on a CHMM of C chains requires to visit all paths in the joint state trellis that is N^C states wide. An N-heads dynamic programming algorithm relaxes the assumption that all state transitions must be visited. The posterior probability mass of a HMM is not evenly distributed among all the state sequences. It is concentrated in the state sequences that are close to the MAP state sequence. The sequences with low probability carry relatively little information. So, the N-heads dynamic programming problem samples a subset of paths that have the highest probability paths to closely approximate the full combinatoric result. N-heads dynamic programming performs slightly better than factor and couple HMM in terms of log likelihood of the probability of observation sequences. N-heads dynamic programming was also used by Oliver *et al.* for modeling human interactions[28].

Zhong and Ghosh proposed a new formulation of CHMM in which the joint transition probability of a CHMM is modeled as a linear combination of transition probabilities and coupling probabilities [46]. The weights of the linear combination is a factor of the influence of one chain on the other. The formulation of Zhong and Ghosh has C^2 additional parameters to learn than that of Brand. These additional parameters capture the influence of one chain on the other. The presence of coupling parameters in the joint transition probability

equation makes it no longer in pure product form. So, the dynamic programming tricks can't be applied here as done by Brand. The complexity of the training of the proposed CHMM is equal to that of FCHMM. Zhong and Ghosh approximated the learning using Lagrangian multipliers to make the computation practical. Unfortunately, although the log likelihood of the approximated inference is better than that of exact inference, the classification/recognition accuracy of the approximated inference algorithm is not very close to the exact inference method.

2.4. Constrained HMM

Constrained HMMs build some topology into the hidden state representation [37]. Constrained HMM constrains the transition parameters of an HMM to make sure that the hidden states evolve in a structured way. Left-to-right HMMs can be considered as a special case of Constrained HMMs. In left-to-right HMMs, the transition matrix of an HMM is constrained to be an upper diagonal matrix. In a similar way, Constrained HMMs' transition table is initialized with random values. Based on the topology, the values of the impossible transitions are set to zero. Then, the Constrained HMM has same inference procedures as a standard HMM. The learning procedure for a Constrained HMM is much easier in some cases, as the transition probabilities that are constrained by the topology need not be learned. The similarity of Constrained HMM to the proposed HMM lies in constraining the transition and observation probabilities based on the consistency constraints. There are other works by Christiansen *et al.* by the name Constrained HMM [13, 14]. But, they constrain the observations emitted by a state. Such models are not relevant to the applications proposed in this thesis.

2.5. English language Ontological Databases

This thesis explores a way to couple HMMs to model appearance, actions and relations without increasing the number of samples required to train the HMMs. The information that links objects, actions and relations comes from natural language databases. The knowledge corpi of natural language has information about words, along with their parts of speech and the knowledge of what words occur together. WordNet, VerbNet, and FrameNet are such knowledge corpi for the English language. In this thesis, the constraints are generated by hand, but in principle they could be generated from these sources. The knowledge databases are discussed in the following subsections.

2.5.1. WordNet

WordNet is a lexical database for the English language. It contains information about some 110,000 nouns, 11,000 verbs, 22,000 adjectives, and 4,500 adverbs. WordNet groups words based on their meanings by interlinking words with a semantic relation. As a result, words that are found in close proximity to one another in the network are semantically disambiguated [17].

The relations supported by WordNet are polysemy, synonymy, hyponymy, and meronymy. Synonymy is the main relation among words in WordNet. It groups words with similar meaning together, for example, the words car and automobile. A group of synonyms is called a synset. Polysemy represents one-to-many relation where one word has many different meanings. Hyponymy can be considered as an *IS-A* relation. It links more general words such as a vehicle to a more specific word such as car. Meronymy is a bidirectional relation that represents a *part-whole* relation. It holds between words such as car and wheel.

The mapping from appearances to nouns is many to many, whereas the mapping from appearances to synsets is many to one. Although, WordNet groups words based on different relations, it doesn't have the linkages between nouns and verbs that we are looking for. For example, using WordNet, we can know that car is a vehicle. But, there is no relation which states that vehicles can't walk. To find such mutual information between different parts of speech, we will look into another lexical database called VerbNet in the next subsection.

2.5.2. VerbNet

WordNet doesn't provide a comprehensive account of all possible syntactic frames and predicate argument structures associated with a verb. VerbNet is developed to account for this shortcoming of WordNet. VerbNet is a verb lexicon compatible with WordNet, but with explicitly stated syntactic and semantic information. VerbNet denotes the semantic relationship between predicate and argument [40]. As this information represents the relation between verbs and nouns, we can exploit this information from VerbNet and WordNet. Moreover, VerbNet is also compatible with FrameNet which is discussed next.

2.5.3. FrameNet

FrameNet is a lexical resource for English, based on frame semantics and supported by corpus evidence. FrameNet contains a wide range of semantic and syntactic combinatory possibilities for each word in each of its senses. Word senses are grouped into conceptual structures called frames that represent cognitive concepts. A frame is a schematic representation of a situation involving various participants, propositions, and other conceptual roles. Thus, FrameNet can recognize the relationship between different parts of speech[1, 38]. For example, walk is described by the "self_motion" frame in FrameNet. The definition of

this frame is "The self_mover, a living being, moves under its own direction along a path". WordNet hypernymy relation, links walk with self-propelled motion, and person with a living being. We can link WordNet and FrameNet and say that cars can't walk, and walk is not possible without motion. Thus, WordNet, VerbNet, and FrameNet can be used to find the mutual information between the appearance, action, and trajectory labels. In this thesis, the constraints are generated by hand, but in principle they could be generated from these sources.

2.6. HMMs in Computer Vision

There are numerous works in Computer Vision that uses HMMs. Out of all those works, the one by Siddharth *et al.*[41] is the closest one to ours. They integrate natural language concepts with computer vision as we do. But, the way language is used is different. The work by Siddharth *et al.* is an extension to the simultaneous object detection, tracking, and event recognition work by Barbu *et al.* [2]. Barbu *et al.* used an object detector for each of the objects to be recognized. They over generate the detections to alleviate the problem of false negatives. From the over generated set of detections, they select the detections in multiple frames that abide to the optical flow in the detections. Once, the detections are tracked over multiple frames, the tracks that conform to the event model are selected. Tracking in multiple frames is accomplished by optical flow, and event recognition is done using an HMM, based on the detections that depend on the event model. Thus, Barbu *et al.* simultaneously perform object detection, tracking and event recognition. However, the events that can be recognized by this system is confined to a unary predicate.

Siddharth *et al.* extended the simultaneous object detection, tracking, and event recognition system to recognize complex events that have multiple predicates by exploiting the

similarities in the compositional structure of language and events. Given a sentence, a video, and a lexicon, their system detects the activity given by the sentence in the video. This is done by parsing the sentence based on lexicon to detect the nouns, adjectives, verbs, adverbs, and prepositions. An object detector is used for each of the nouns in the sentence. The detections in multiple frames that adhere to the optical flow are considered a track. The tracks that adhere to the verbs, adverbs, adjectives, and prepositions in the sentence are selected based on the rules in the lexicon. Thus, sentence-guided activity recognition in video is accomplished.

If the activity in the video is unknown, the algorithm systematically searches the space of all possible sentences that can be generated by a context-free grammar and finds the sentence that has the maximum score. In a real world setting, the search space is huge, possibly infinite, and searching in this space is intractable. But, the authors suggest that beam search can give an approximate answer, while being tractable. However, the beam search method reduces the complexity that arises from the recursion of grammar to generate sentences. But, real world has a large number of nouns, verbs, adverbs, prepositions, and adjectives. The grammar quickly grows even if we restrict to simple grammar and the method quickly becomes intractable. Moreover, the method is specific to the lexicon, and can't be easily extended. One reason is that the method uses object detectors, and it is not possible to have an object detector for all possible objects in the world. Another reason is that it is not possible to write rules for all verbs, adverbs, adjectives, and prepositions. For example, one can't write a rule for actions like clap, or wave unless there is a hand detector.

Although Siddharth *et al.* tries to integrate vision and language as we do, the way the language is used is different. They use language to parse sentences to find different parts of speech. Based on the parsed sentence, and pre-compiled rules, the activity recognition is

done along with object detection, and tracking simultaneously. In contrast, we exploit the consistency constraints from language and integrate it into HMMs to account for the lost mutual information. Our method is simple and more general and can be applied to real world without exploding the complexity. Moreover, the HMMs need not be retrained when some constraints are added or changed.

Dragon and Van Gool estimated the ground plane from a monocular camera using a Hidden Markov Model [16]. Assuming that the motion of the camera is orthogonal to the ground plane normal, they formulated the problem as a continuous HMM. They jointly solve the problem of estimating the motion of the camera along with ground plane estimation by formulating the unknowns as hidden states i.e., the camera motion vector and ground plane normal. The hidden states are estimated by decomposing homographies. They use blocked Gibbs sampling to refine the solution proposed by their HMM. The proposed approach works robustly in a large variety of sequences, including tilted cameras and wobbling images. The suggested approach can be easily extended to track other structures like vanishing points over time.

Peng *et al.* used HMMs for full body gesture recognition. They use two uncalibrated cameras to make the gesture recognition view invariant. They extract silhoutte images from two cameras and do a multilinear projection on them to get view invariant pose and orientation vectors. Then, an HMM is applied to the pose vectors for gesture recognition [30]. Rajko and Qian proposed reduced parameter HMM models for gesture recognition. The proposed models have $\mathrm{O}(n)$ and $\mathrm{O}(1)$ complexity, and their performance is comparable to standard HMMs for gesture recognition [34]. Peng and Qian used the reduced HMM parameter proposed by Rajko and Qian to make their view invariant full body gesture recognition to work in real time [29].

Tang *et al.* proposed a conditional variant of a variable duration HMM to learn latent temporal structure of complex events in Internet videos [43]. Unlike, standard HMMs, where a state emits only one observation, variable duration HMM states can emit an observation sequence. The conditional variant proposed by Tang *et al.* is similar to a conditional random field [32]. The proposed HMM achieves competitive accuracies on event detection and activity recognition tasks while being simple and fast.

2.7. Summary

In this chapter, we reviewed the basics of an HMM along with the algorithms to train, and run them. Then, we looked into various extensions of HMMs to model multiple sequences of data. Having learned that CHMM is the appropriate extension to model multiple streams of data from dependent processes, we reviewed the previous works on CHMMs. In addition, we looked into Constrained HMMs as we impose constraints on some combination of states in a similar way. As we are exploiting the information from natural language databases, we reviewed WordNet, VerbNet, and FrameNet. Finally, we reviewed some latest applications of HMMs in the field of Computer Vision.

CHAPTER 3

Consistent Hidden Markov Models

Hidden Markov Models (HMMs) are popular probabilistic models to integrate information over time. Given a set of observation sequences, HMM can be trained using the Baum-Welch algorithm as discussed in section 2.1.5. Given an HMM model, and an observation sequence, the Viterbi algorithm discussed in section 2.1.6 is used to find the most likely state sequence. A standard HMM is represented as $\lambda = (A, B, \pi)$ and the related terminology is explained in section 2.1.1

Traditionally HMMs are used to model data generated from a single process. However, in the context of computer vision applications, such as activity recognition, data generated from multiple processes must be integrated. As traditional HMMs don't integrate data from multiple processes, HMMs need to be extended to model data from multiple processes. There are varieties of couplings to extend HMMs as discussed in section 2.3.1. Each coupling serves its own purpose. The contribution of this thesis is to propose a new formulation of an extended HMM that allows us to exploit the knowledge from natural language without requiring significantly more training data.

Section 3.1 explains the terminology of a consistent HMM. This section is essential to understand the following sections. The extended version of the Viterbi algorithm, to find the most likely state sequence, is explained in section 3.2. Finally, section 3.3 explains the extended Baum-Welch algorithm we propose to train a consistent HMM

3.1. Terminology related to Consistent HMM

The following terminology is important to explain the extended Baum-Welch and Viterbi algorithms that integrates knowledge from multiple processes. Data generated by each process is considered a chain in this thesis. The mutual information between chains is lost when they are interpreted independently. The algorithms presented below exploits knowledge from natural language to capture the mutual information between different chains. The following terminology is applicable to an HMM that should fuse information from different chains.

- C: Number of chains.
- Set of States: $S = \bigcup_{c=1}^{C} S^{(c)}$, where $S^{(c)}$ is the set of states for c^{th} chain. The total number of states $|S| = \sum_{c=1}^{C} |S^{(c)}|$. The state at time t for chain c is denoted by $q_t^{(c)} = S_i^{(c)} \in S^{(c)}$.
- Cartesian product states: $S^{\otimes} = \bigotimes_{c=1}^{C} S^{(c)}$. Cartesian product states are all the possible combinations of states from C chains. The total number of cartesian product states $|S^{\otimes}| = \prod_{c=1}^{C} |S^c|$ where each state is a C-tuple.
- Set of observation symbols: $V = \bigcup_{c=1}^{C} V^{(c)}$, where $V^{(c)}$ is the set of observation symbols for c^{th} chain. The total number of observation symbols are $|V| = \sum_{c=1}^{C} |V^{(c)}|$. The observation symbol at time t for chain c is denoted by $O_t^{(c)}$.
- Cartesian product observation symbols: $V^{\otimes} = \bigotimes_{c=1}^{C} V^{(c)}$. Cartesian product observation symbols are all the possible combinations of observation symbols from C chains. The total number of cartesian product observation symbols $|V^{\otimes}| = \prod_{c=1}^{C} |V^c|$ where each observation symbol is a C-tuple.
- $\bar{A}$ is the vector of transitional probability matrices: $\bar{A} = \{A^{(1)}, A^{(2)}, ..., A^{(C)}\}$.
- $\bar{B}$ is the vector of observation probability matrices: $\bar{B} = \{B^{(1)}, B^{(2)}, ..., B^{(C)}\}$.
- $\bar{\pi}$ is the vector of prior probability vectors: $\bar{\pi} = \{\pi^{(1)}, \pi^{(2)}, ..., \pi^{(C)}\}$.

- κ is the binary constraint knowledge extracted from language databases. κ is a vector of length $|S^{\otimes}|$ such that for each of the cartesian product states $s \in S^{\otimes}$, the corresponding κ value is 1 if the state s is consistent, 0 otherwise.

$$\kappa[i] = \begin{cases} 1 & S_i^{\otimes} \text{ is consistent} \\ 0 & \text{otherwise} \end{cases}$$

Having defined the above terminology, the proposed HMM is defined as $\lambda_\kappa = (\bar{A}, \bar{B}, \bar{\pi}, \kappa)_C$

3.2. Extended Viterbi Algorithm with Consistency Constraints

The power of an HMM lies in recovering the most likely state sequence that produced the given observation sequence O, given a model λ. While there might be multiple state sequences that generate a given observation sequence, the Viterbi algorithm finds the most likely sequence. The Viterbi algorithm for a standard HMM is discussed in subsection 2.1.6.

The goal of this section is to extend the standard Viterbi algorithm to integrate data from multiple channels so as to satisfy consistency constraints. Extended Viterbi algorithm is accomplished by running the standard Viterbi algorithm on each chain as though they are independent. However, at every timestep the algorithm enforces consistency constraints across chains.

The standard Viterbi algorithm calculates the best score probability along a single path in a chain c ending at a state $S_i^{(c)}$ accounting for the first t observations. The best score probability of state i is represented by $\delta_t^{(c)}(i)$ and is calculated by equation 2.20. At a given timestep, this value can be calculated for every state $s \in S$. Let Δ_t be the vector of the best score probabilities for all states $s \in S$ at time t as shown in equation 3.1. Let Ω_t be

the best score probability vector for all states if the chains are coupled as a fully coupled HMM as shown in equation 3.2. Δ_t can be considered as a point in $|S|$ dimensional space. Let us call this space a factorial state space. Similarly, Ω_t can be considered as a point in $|S^{\otimes}|$ dimensional space. Let us call this space a cartesian state space.

$$\Delta_t = \{\delta_t^{(c)}(i)\} \qquad \forall S_i^{(c)} \in S \tag{3.1}$$

$$\Omega_t = \{\delta_t(i)\} \qquad \forall S_i \in S^{\otimes} \tag{3.2}$$

The consistency constraints κ exploited from language are in cartesian state space. However, the HMMs are trained in the factorial state space i.e., HMMs are trained by assuming each chain as independent. So, the Viterbi algorithm runs in factorial state space, but at every timestep the algorithm projects the vector Δ into cartesian state space, applying the consistency constraints. Then the point in cartesian state space is projected back into the factorial state space.

Projecting a point from cartesian state space to factorial state space is straight forward. The probability of a state $S_i^{(c)}$ in factorial state space is the sum of probabilities of all states in cartesian state space $S^{\otimes}$ that has state $S_i^{(c)}$ in it. If a state in factorial state space $S_i^{(c)}$ is in the C-tuple of a state in cartesian state space $S_j^{\otimes}$, it is represented as $S_i^{(c)} \in S_j^{\otimes}$. Mathematically, the probability of a state in factorial state space is calculated from the probabilities in cartesian state space by the following equation.

$$P(S_i^{(c)}) = \sum_{\substack{j=1 \\ S_i^{(c)} \in S_j^{\otimes}}}^{|S^{\otimes}|} P(S_j^{\otimes}) \tag{3.3}$$

The probability of a state in factorial state space is the sum of probabilities of corresponding states in cartesian state space as shown in equation 3.3. This can be written as a linear combination of probabilities of all cartesian states where the corresponding weights are either 0 or 1. Therefore, there exists a matrix ρ that projects the probability vector from a cartesian state space to a factorial state space. The dimensions of ρ matrix are $|S| \times |S^{\otimes}|$. The rows of ρ matrix are indexed by the states S, whereas the columns are indexed by the cartesian product states $S^{\otimes}$. The values of the ρ matrix are filled with binary values based on equation 3.3 as shown in the following equation.

$$\rho[i][j] = \begin{cases} 1 & \text{if } S_i \in S_j^{\otimes} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

Having defined $\Delta, \Omega,$ and ρ, the probability vector Ω in the cartesian state space can be projected to the probability vector Δ in the factorial state space by the following equation.

$$\Delta = \rho\Omega \tag{3.5}$$

However, dependency information is lost when a point in a high dimensional cartesian state space is projected to a point in a low dimensional factorial state space. The information captured by $|S^{\otimes}|$ dimensions can't be captured by $|S|$ dimensions. Projecting back from factorial state space to cartesian state space doesn't recreate the original high dimensional vector as the information lost during projection onto factorial state space can't be recovered unless the participating chains are independent. If the participating chains are independent, except for the consistency constraints κ, we can integrate the consistency constraints κ into

the projection step as shown below.

$$P(S^{\otimes}) = \kappa \odot \bigotimes_{c=1}^{C} P(S^{(c)}) \tag{3.6}$$

However, usually there are other dependencies between the chains that we are not aware of. Moreover, if we project from a factorial state space to a cartesian state space by applying only consistency constraints as shown in equation 3.6, the projection is not "stable". That is, a different vector is generated everytime a projection is done from factorial state space to cartesian state space by equation 3.6, and the cartesian state vector is projected back by equation 3.5. This is because the dependency information is lost when a point is projected from a cartesian product space to a factorial state space. In other words, applying equation 3.6 followed by equation 3.5 does not produce the same probability vector you started with.

We need to integrate consistency constraints into the cartesian state space vector such that the consistency information κ is not lost in projections between two spaces, to the extent possible. Therefore we pose this problem as a linear optimization problem that finds the closest cartesian vector that follows the consistency constraints κ and can be best explained by a factorial model. This cartesian vector Ω is the vector that abides by the consistency constraints κ and minimizes the L_2 norm between Δ and the projection of Ω onto the factorial state space. Let $f(\Delta, \rho, \kappa)$ be a function that projects the probabilities from the factorial state space to a fully coupled state space by imposing binary consistency constraints. The optimization problem can be written as:

$$f(\Delta, \rho, \kappa) = \text{ Min } ||\Delta - \rho\Omega||_2$$

$$\text{s.t} \quad 1.\Omega = 1 \quad \text{and} \quad \Omega >= 0 \quad \text{and} \quad 1.((1-\kappa) \odot \Omega) = 0 \tag{3.7}$$

The constraint $1.\Omega = 1$ makes sure that the probabilities of Ω sum to 1. The constraint $\Omega >= 0$ along with $1.\Omega = 1$ constrains the probability values to be between 0 and 1 inclusive. $1.((1-\kappa) \odot \Omega) = 0$ imposes the consistency constraints in Ω.

In other words, we can assume that there exists a manifold in the cartesian state space where each point on the manifold can be mapped to the factorial state space without any loss of information. By definition, when a point from a factorial state space is projected onto the cartesian state space by assuming independence between the chains, the resultant point lies on this manifold in the cartesian state space. However, when consistency constraints are imposed, the resulting point is no longer on this manifold. The optimization step projects the knocked off point back onto the manifold such that the consistency constraints stay intact, and this new point, when projected back to the factorial state space, lies close to the original points. This is pictorially depicted in the figure 3.1.

Algorithm 3.1 gives the pseudo code for the proposed algorithm. Conceptually, the algorithm is similar to running the standard Viterbi algorithm in every chain. But, at every time step, the optimization function defined in equation 3.7 is applied, and the resultant vector is projected back as shown in equation 3.5. The steps represented by red color in the algorithm shows our contribution to enforce the consistency constraints κ. The optimization method used in lines 3 and 8 of the algorithm is any linear optimization method that can solve the optimization problem laid out in equation 3.7. Other steps are the standard Viterbi

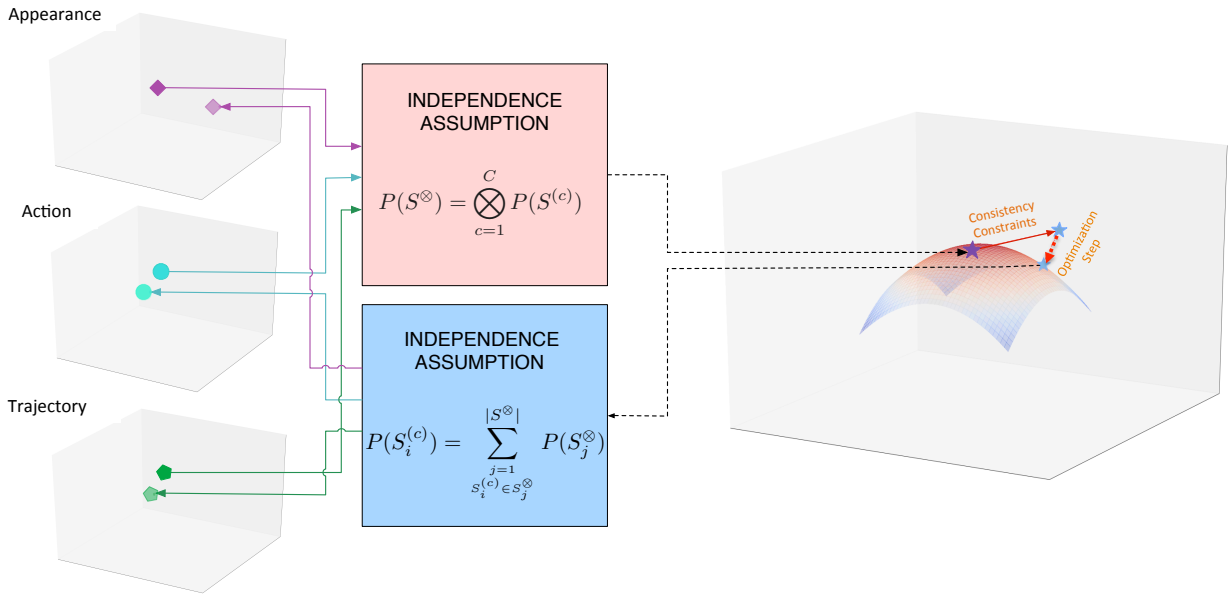

FIGURE 3.1. Projection between factorial and consistent state spaces

algorithm steps, except that the Viterbi algorithm is applied to multiple chains simultaneously. The extension of standard Viterbi algorithm steps to apply the Viterbi algorithm on multiple chains simultaneously is shown in the algorithms presented in appendix A.

The same algorithm is shown in figure 3.2. The green, brown, and blue rectangles depicts the best score probability vectors for each chain at a time step. The vector of all such probabilities is denoted by Δ. Optimization step is performed to project Δ to a cartesian state space with consistency constraints. The resulting vector Ω is represented by red rectangles in the figure. This vector is projected back to the factorial state space and the Viterbi step is run to calculate best score probabilities at next time step.

The algorithm presented in this section allows us to find the most likely state sequence that generates the given observation sequence by imposing consistency constraints. The advantage of this method is that constraints can be imposed during run time without retraining the HMMs. This algorithm will be evaluated in chapter 4. The next section discusses our extension of the Baum-Welch algorithm which is used to train the HMMs. The extended

Algorithm 3.1 Extended Viterbi Algorithm

Input: An HMM $\lambda_\kappa = (\bar{A}, \bar{B}, \bar{\pi}, \kappa)_C$;
Factorial states S;
Cartesian states $S^{\otimes}$;
A sequence of observed symbols: $O = \{O^{(1)}, O^{(2)}, ..., O^{(C)}\}$ where $O^{(c)} = O_1^{(c)} O_2^{(c)} O_3^{(c)} ... O_T^{(c)}$;

Output: An array Q of length $C \times T$, where each row is indexed by a chain, and each column is indexed by a time step. The Q array holds the indices of most likely states that generates the observation sequence O.

$\rho, SEQSCORE, BACKPTR =$ INITIALIZE$(S, S^{\otimes}, \bar{\pi})$
$\Delta = SEQSCORE[:, 1]$
$\Omega =$ OPTIMIZE(Δ, ρ, κ)
$SEQSCORE[:, 1] = \rho\Omega$
for $t = 2$ to T **do**
    $SEQSCORE, BACKPTR =$ VITERBI_STEP$(SEQSCORE, BACKPTR, \lambda_\kappa)$
    $\Delta = SEQSCORE[:, t]$
    $\Omega =$ OPTIMIZE(Δ, ρ, κ)
    $SEQSCORE[:, t] = \rho\Omega$
end for
$Q =$ BACKTRACK$(SEQSCORE, BACKPTR, C, S)$
return Q

Baum-Welch algorithm allows us to impose consistency constraints during training as shown in the next section.

3.3. Extended Baum-Welch Algorithm with Consistency Constraints

HMMs can recover the most likely state sequence that produced an observation sequence O by the Viterbi algorithm. But the Viterbi algorithm requires a model λ to recover the most likely sequence. The model λ is trained to maximize the likelihood of the observation sequence. The Baum-Welch algorithm to train HMMs based on the expectation maximization method is discussed in subsection 2.1.5.

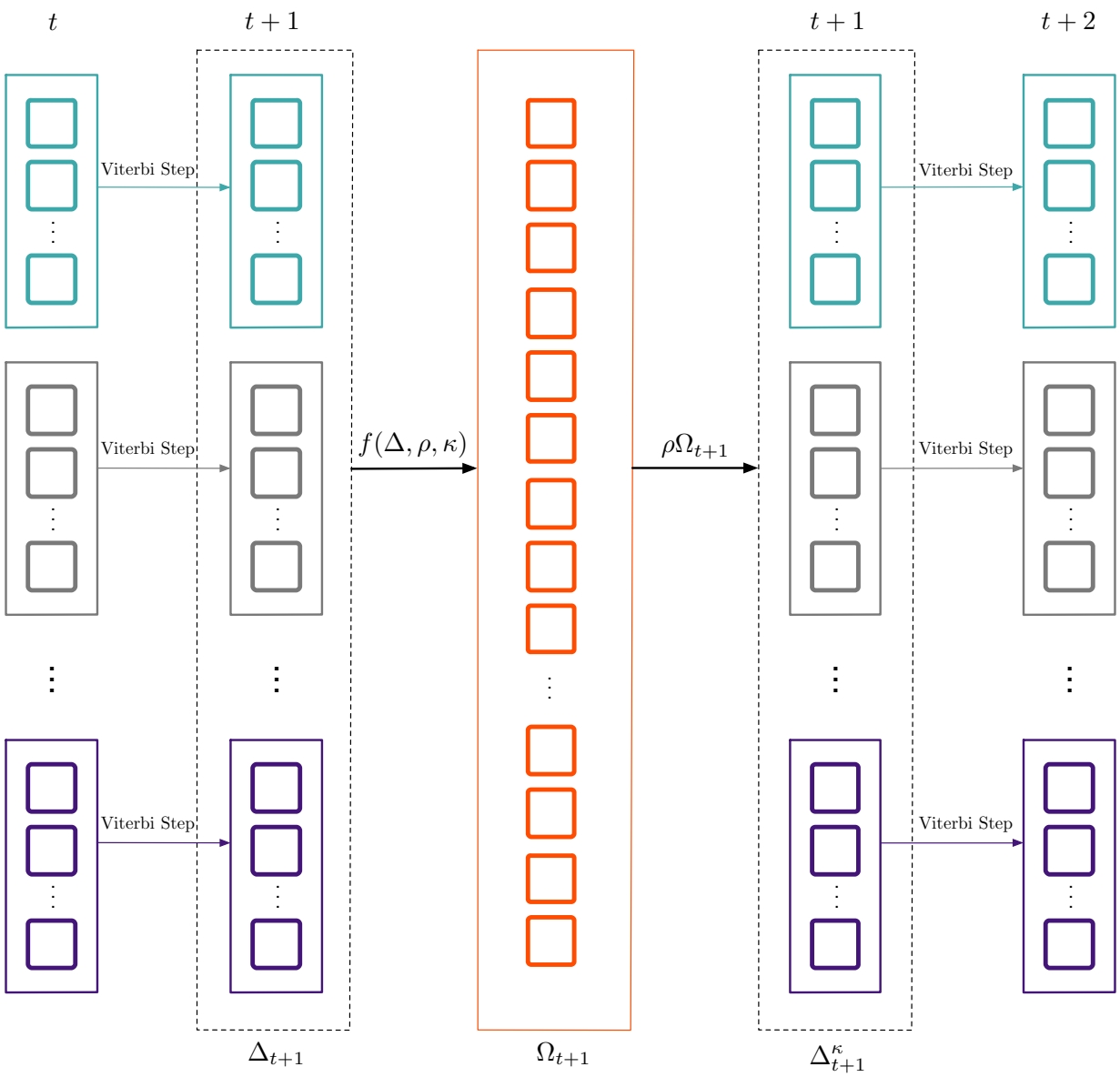

FIGURE 3.2. Extended Viterbi Algorithm with Consistency Constraints

This section extends the standard Baum-Welch algorithm to train HMMs by integrating data from multiple channels along with the consistency constraints. The extended Baum-Welch algorithm learns the same parameters as a factorial HMM, but the consistency constraints are weakly embedded within the parameters. The consistency constraints thus embedded do not need any additional parameters. The extended Baum-Welch algorithm is accomplished by first initializing the models for each chains as in the standard Baum-Welch

algorithm. However, once the models are initialized, the algorithm enforces consistency constraints across the initial models. Then each model is trained using the standard Baum-Welch algorithm as if they are independent. Integrating the consistency constraints after the models are initialized results in a better starting state for the standard Baum-Welch algorithm.

The standard Baum-Welch algorithm trains an HMM model for a chain c denoted by $\lambda^{(c)} = \{A^{(c)}, B^{(c)}, \pi^{(c)}\}$. There are different transition probability and observation probability matrices for each chain. There is a need to represent the transitions probability matrices from all the chains in a single matrix to integrate the consistency constraints in them. The same is the case with the observation probability matrices. As the dimensions of these matrices vary across the chains, the matrices can't be stacked either horizontally or vertically. To overcome this issue, the matrices can be stacked in a block diagonal fashion as show in figures 3.3 and 3.4. Moreover, it is easier to project a transition or an observation matrix from a cartesian state space to a factorial state space in the block diagonal format.

Let the block diagonal matrix for the transition probabilities be denoted by A^C. The rows and columns of this matrix are indexed by the set of states S and the dimension of the matrix is $|S| \times |S|$. If the row and the column labels corresponds to the states from the same chain, the corresponding entry is filled with the transition probability value from the respective chain. If the row and the column labels corresponds to different chains, the corresponding entry is set to zero. Figure 3.3 shows the block diagonal transitional probability matrix. Transition probability matrices from different chains are represented with different colors and the white spaces represent 0. Mathematically, it is represented as

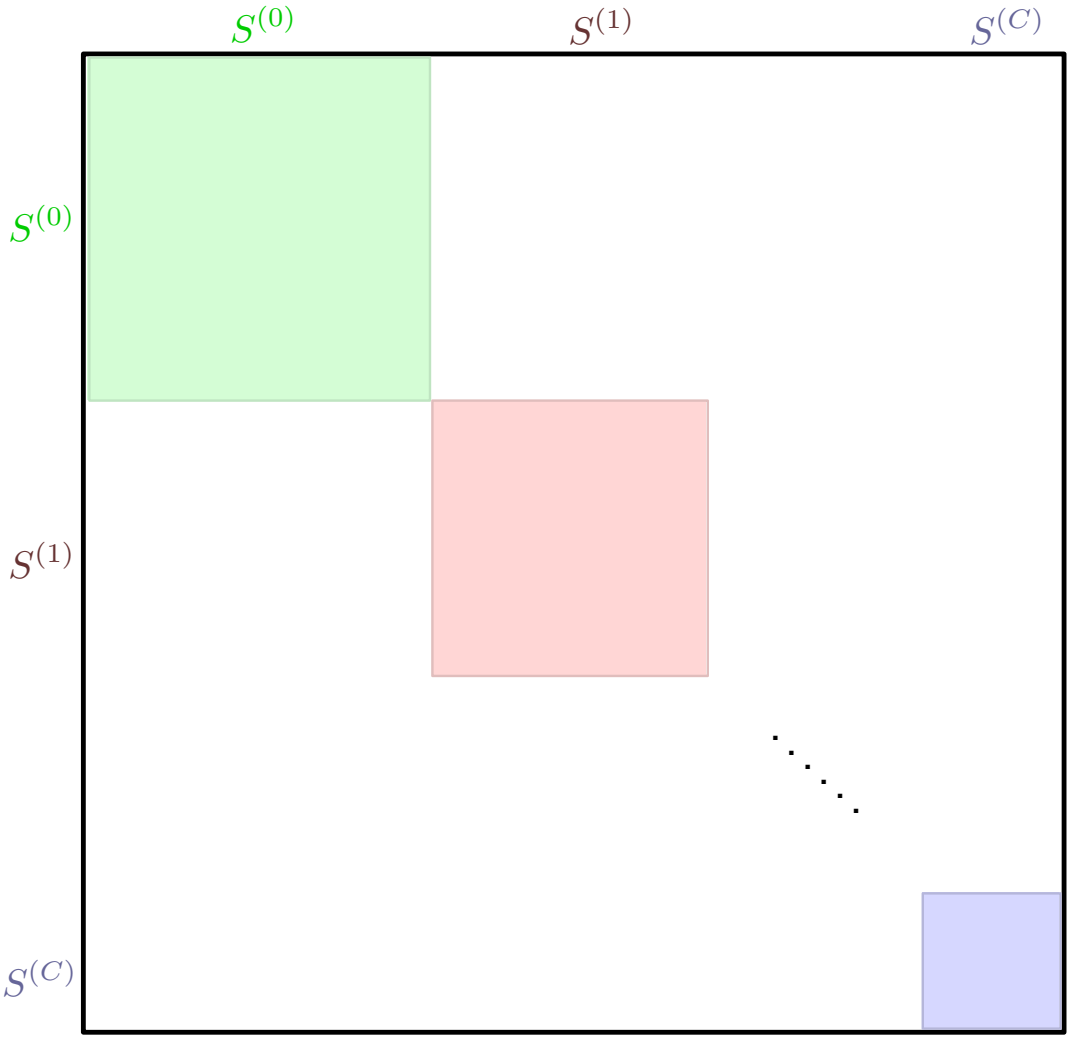

FIGURE 3.3. Transition matrices stacked in form of block diagonal matrices

$$A^C[i][j] = \begin{cases} A^{(c)}[k][l] & \exists c, k, l : S_i = S_k^{(c)} \text{ and } S_j = S_l^{(c)} \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

The block diagonal matrix for the observation probabilities is denoted by B^C. It is similar to A^C except that the columns are indexed by the set of observations V. The dimension of this matrix is $|S| \times |V|$. If the row and column labels corresponds to the same chain, the corresponding entry is filled with the observation probability value from the respective chain.

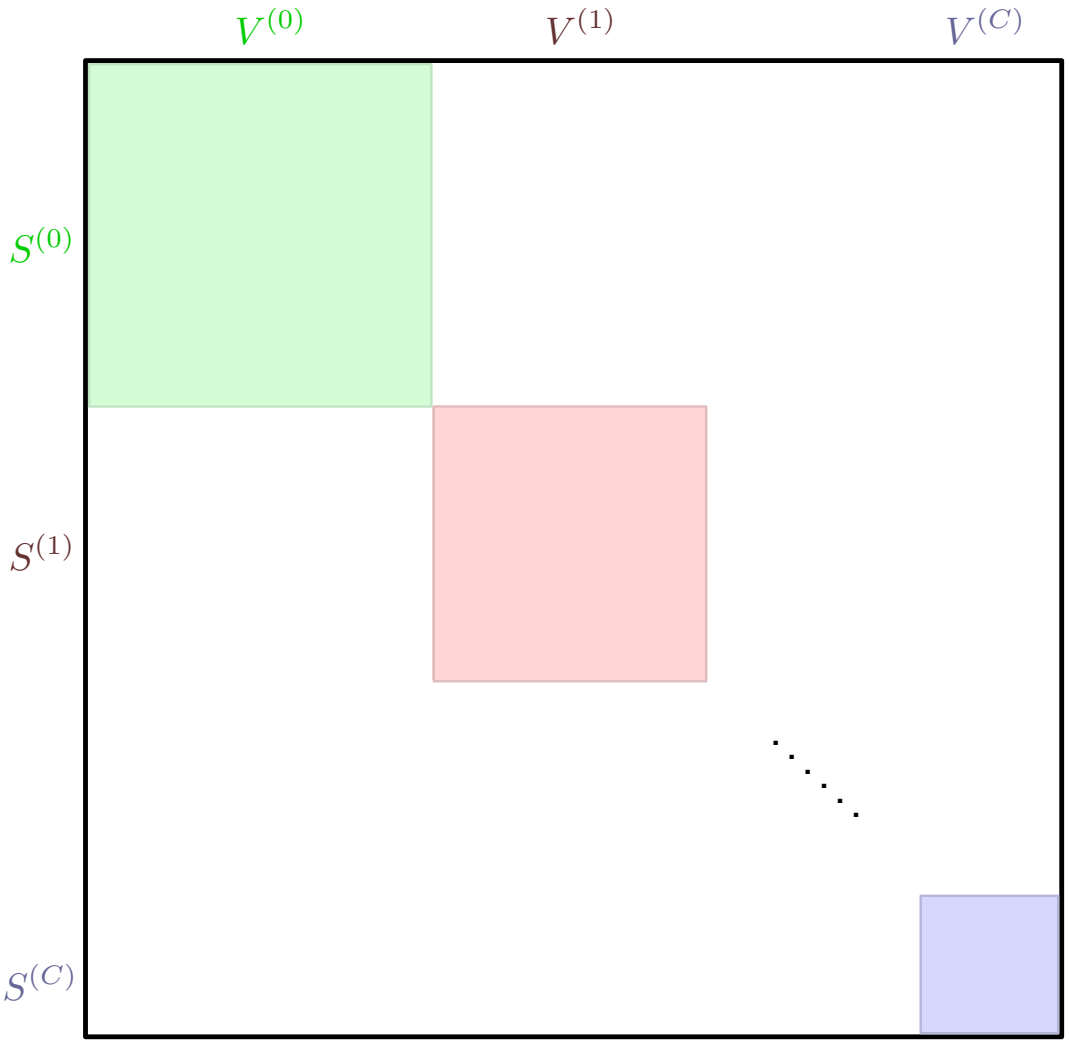

FIGURE 3.4. Observation Matrices stacked in the form of block diagonal matrices

Else, the value is set to zero. Figure 3.4 shows the block diagonal observation probability matrix where different colors represent the observation probability matrices from different chains and the white spaces represent 0. Mathematically, the matrix B^C is represented as

$$B^C[i][j] = \begin{cases} B^{(c)}[k][l] & \exists c, k, l : S_i = S_k^{(c)} \text{ and } S_j = V_l^{(c)} \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

Let η be the function that generates the block diagonal matrix from a vector of individual matrices. The inverse of this function extracts the individual matrices from the given block diagonal matrix. For the transition and the observation probability matrices, the function is represented as

$$\eta(\bar{A}) = A^C \tag{3.10}$$

$$\eta(\bar{B}) = B^C \tag{3.11}$$

$$\eta^{-1}(A^C) = \bar{A} \tag{3.12}$$

$$\eta^{-1}(B^C) = \bar{B} \tag{3.13}$$

Let $A^{\otimes}$ represent the transition probability matrix and $B^{\otimes}$ represent the observation probability matrix in the cartesian state space. Projecting these matrices from the cartesian state space to a factorial state space can be accomplished by summing over multiple values in the cartesian state space. The transition probability in the factorial state space between states i and j of chain c, $P[S_i^{(c)}|S_j^{(c)}]$, is the sum of transition probabilities between the states that has $S_i^{(c)}$ and $S_j^{(c)}$ in the cartesian state space $S^{\otimes}$. Mathematically, the transition probability between two states in the factorial state space is calculated from the transition probabilities in the cartesian state space by the following equation.

$$P(S_i^{(c)}|S_j^{(c)}) = \sum_{\substack{k=1 \\ S_i^{(c)} \in S_k^{\otimes}}}^{|S^{\otimes}|} \sum_{\substack{l=1 \\ S_j^{(c)} \in S_l^{\otimes}}}^{|S^{\otimes}|} P(S_k^{\otimes}|S_l^{\otimes}) \tag{3.14}$$

The transition probability between two states in the factorial state space is the sum of the corresponding transition probabilities in the cartesian state space as shown in equation 3.14. In other words, a transition probability value in the factorial state space can be calculated by summing over corresponding rows and columns in the cartesian state space transition probability matrix. This can be written as a linear combination of transition probabilities of all cartesian states where the corresponding weights are either 0 or 1. Therefore, there exists a matrix υ_S that sums over the rows of the cartesian space transition probability matrix. The transpose of this matrix sums over the required columns. The dimensions of υ_S matrix are $|S| \times |S^{\otimes}|$. The rows of υ_S matrix are indexed by the states S, whereas the columns are indexed by the cartesian product states $S^{\otimes}$. The values of the υ_S matrix are filled with binary values based on equation 3.14 as shown in the following equation.

$$\upsilon_S[i][j] = \begin{cases} 1 & S_i \in S_j^{\otimes} \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

Having defined the υ_S matrix, the transition probability matrix can be projected from the cartesian state space to a factorial state space by premultiplying the cartesian space transition probability matrix by υ_S and post multiplying the resultant with the transpose of υ_S as shown in the following equation.

$$A^S = \upsilon_S . A^{\otimes} . \upsilon_S^T \tag{3.16}$$

The resulting matrix A^S is in the block diagonal format. But the non diagonal blocks are not filled with zeros, unlike A^C. To zero out the non diagonal blocks, we define a binary matrix ζ that has ones in the diagonal blocks, and zeros elsewhere. Point wise

multiplication of matrix ζ with A^S converts the matrix to the required block diagonal format A^C. Mathematically, the matrix ζ is defined as

$$\zeta[i][j] = \begin{cases} 1 & \exists c : S_i \in S^{(c)} \text{ and } S_j \in S^{(c)} \\ 0 & \text{otherwise} \end{cases} \tag{3.17}$$

Pointwise multiplication of equation 3.16 with ζ as shown in equation 3.18 results in the transition probabilities in the factorial state space in the defined block diagonal format. Combining equation 3.18 with equation 3.12 gives the vector of transition probability matrices $\bar{A}$ in the factorial state space as shown in equation 3.19.

$$A^C = \zeta \odot (\upsilon_S . A^{\otimes} . \upsilon_S^T) \tag{3.18}$$

$$\bar{A} = \eta^{-1}(\zeta \odot (\upsilon_S . A^{\otimes} . \upsilon_S^T)) \tag{3.19}$$

Similarly, the observation probability matrix can be projected from the cartesian state space to a factorial state space except that the summation of columns is different for the observation probability matrix. Transition probability matrix is indexed by the same rows and columns. So, the transpose of υ_S was sufficient to sum over columns. But, the observation probability matrix rows are indexed by states $S^{\otimes}$, whereas the columns are indexed by observation symbols $V^{\otimes}$. So, the probability of observing an observation $V_i^{(c)}$ given a state $S_j^{(c)}$ in the factorial state space can be calculated by summing over all the probabilities of observing an observation that has $V_i^{(c)}$ in it, given the cartesian state space that has factorial state space $S_j^{(c)}$ in it. Mathematically, it can be expressed by the following equation.

$$P(V_i^{(c)}|S_j^{(c)}) = \sum_{\substack{k=1 \\ V_i^{(c)} \in V_k^{\otimes}}}^{|V^{\otimes}|} \sum_{\substack{l=1 \\ S_j^{(c)} \in S_l^{\otimes}}}^{|S^{\otimes}|} P(V_k^{\otimes}|S_l^{\otimes}) \tag{3.20}$$

Equation 3.20 is similar to the equation 3.14 except for the outer sum that sums over columns. So, we can reuse the υ_S matrix defined in equation 3.15 to sum over the rows. To sum over the columns, we define a matrix υ_V that sums over the columns of the cartesian space transition probability matrix. The dimensions of υ_V matrix are $|S^{\otimes}| \times |V|$. The rows of υ_V matrix are indexed by the cartesian states $S^{\otimes}$, whereas the columns are indexed by the observation symbols in the factorial state space V. The values of the υ_V matrix are filled with the binary values based on the equation 3.20 as shown in the following equation.

$$\upsilon_V[i][j] = \begin{cases} 1 & V_j \in V_i^{\otimes} \\ 0 & \text{otherwise} \end{cases} \tag{3.21}$$

Observation probability matrix is projected from the cartesian state space to a factorial state space by pre-multiplying $B^{\otimes}$ with υ_S, post-multiplying the resultant with υ_V, and point wise multiplying the result with ζ matrix. The block diagonal matrices are extracted by applying the inverse η function given by equation 3.13.

$$\bar{B} = \eta^{-1}(\zeta \odot (\upsilon_S.B^{\otimes}.\upsilon_V)) \tag{3.22}$$

As already discussed in section 3.2 with respect to the extended Viterbi algorithm, some dependency information is lost during projection from cartesian state space to factorial state space which can't be recovered unless the participating chains are independent. If the participating chains are independent except for the consistency constraints κ, the consistency

constraints κ can be integrated into the projection step as shown below.

$$A^{\otimes} = (\kappa.\kappa^T) \odot \bigotimes_{c=1}^{C} A^{(c)} \tag{3.23}$$

$$B^{\otimes} = (\kappa.1^T) \odot \bigotimes_{c=1}^{C} B^{(c)} \tag{3.24}$$

For the transition probability matrix, both the rows and columns corresponding to inconsistent states must be zeroed out. $k.k^T$ in equation 3.23 produces a binary matrix of shape $|S^{\otimes}| \times |S^{\otimes}|$ indexed by the cartesian states $S^{\otimes}$ where the inconsistent rows and columns are zeroed out. In contrast, the observation probability rows are indexed by the states, whereas the columns are indexed by the observation symbols. So, only the rows corresponding to inconsistent states must be zeroed out in the case of observation probability matrix. $\kappa.1^T$ accomplishes this by horizontally stacking the κ vector to create a matrix of dimensions $|S^{\otimes}| \times |V^{\otimes}|$.

However, there may be other dependencies between the chains we are not aware of as already discussed in the section 3.2. Moreover the projection between the factorial and cartesian state spaces is not stable due to the loss of mutual information in projection from the cartesian state space to a factorial state space. That is, a different set of transition probability matrices are generated everytime a projection is done from the factorial state space to a cartesian state space by equation 3.23, and the cartesian state transition probability matrix is projected back by equation 3.19. The same is the case with the observation probability matrix.

The projections between the state spaces must be stabilized to avoid a new set of matrices between multiple projections. Moreover, the projection must integrate consistency

constraints into the cartesian state space vector such that the consistency information κ is not lost in the projections between two spaces. This problem is also posed as linear optimization problem as done in section 3.2. The optimization problem finds the closest cartesian matrix with consistency constraints κ that can be best explained by the factorial model. This cartesian matrix abides by the consistency constraints κ and minimizes the L_2 norm between the actual block diagonal matrix in the factorial state space and the projection of the cartesian state space matrix onto the factorial state space. Figure 3.1 explains this optimization step for the extended Baum-Welch algorithm, similar to the extended Viterbi algorithm.

Let $g(A^C, \upsilon_S, \zeta, \kappa)$ be a function that projects the block diagonal transition probability matrix from the factorial state space to a fully coupled state space by imposing binary consistency constraints. The optimization problem discussed above is formulated as:

$$\begin{aligned} g(A^C, \upsilon_S, \zeta, \kappa) = \ & \text{Min } ||A^C - \zeta \odot (\upsilon_S.A^{\otimes}.\upsilon_S^T)||_2 \\ \text{s.t} \quad A^{\otimes}.1 = \kappa \quad & \text{and} \quad A^{\otimes} >= 0 \quad \text{and} \quad A^{\otimes}.(1-\kappa) = 0 \end{aligned} \tag{3.25}$$

The constraint $A^{\otimes}.1 = \kappa$ along with $A^{\otimes} >= 0$ makes sure that the rows sum to 1 if the states are consistent and the values of the rows corresponding to inconsistent states are set to zero. Also they constrain the probability values to be between 0 and 1 inclusive. $A^{\otimes}.(1-\kappa) = 0$ imposes the consistency constraints on columns of the cartesian state transition probability matrix.

Let $h(B^C, \upsilon_S, \upsilon_V, \zeta, \kappa)$ be a function that projects the block diagonal observation probability matrix from the factorial state space to a fully coupled state space by imposing

binary consistency constraints. The optimization problem can be written similar to the above optimization problem. The only difference is that the columns are not constrained in the observation probability matrix. The optimization problem for observation probability matrix is written as:

$$h(B^C, \upsilon_S, \upsilon_V, \zeta, \kappa) = \text{ Min } ||B^C - \zeta \odot (\upsilon_S.B^{\otimes}.\upsilon_V)||_2$$
$$\text{s.t} \quad B^{\otimes}.1 = \kappa \quad \text{and} \quad B^{\otimes} >= 0 \tag{3.26}$$

The pictorial representation of the extended Baum-Welch algorithm is shown in figure 3.5. The algorithm takes initialized models for each chain and generates a block diagonal transition probability matrix and a block diagonal observation probability matrix as shown in equations 3.8 and 3.9 respectively. The red square in the figure is the projection of block diagonal transition probability matrix into a cartesian state space given by the equation 3.25. The blue square is the optimization problem given by the equation 3.26 that projects the block diagonal observation probability matrix into a cartesian state space. The transition and observation probabilities in the cartesian state space are projected back to the factorial state space through red and blue rectangles. The back projection depicted as red and blue rectangles is done as shown in equations 3.19 and 3.22 respectively. These steps embed the consistency constraints in the factorial state space matrices. Once the consistency constraints are embedded, the individual chains are trained independent of each other using the Baum-Welch algorithm discussed in section 2.1.5.

The pseudo-code for the extended Baum-Welch algorithm is shown in algorithm 3.2. The pseudo-code in red shows our contribution on top of the standard Baum-Welch algorithm. Any linear optimization solver that can solve the optimization problems in equations 3.25

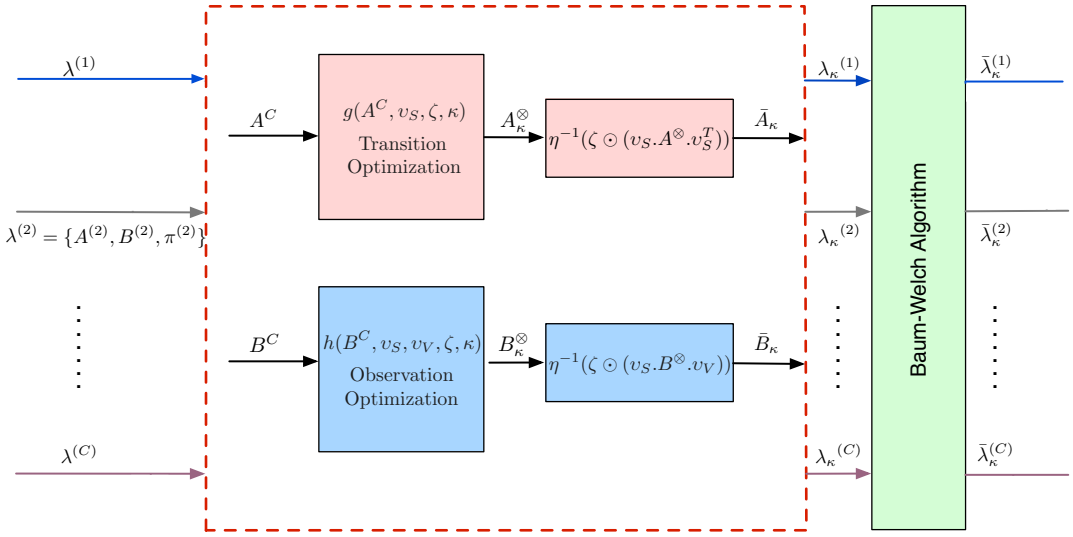

FIGURE 3.5. Extended Baum-Welch Algorithm

and 3.26 can be used in lines 4 and 5 respectively. The pseudo-code for the standard Baum-Welch algorithm used in line 9 is presented in appendix B.

Algorithm 3.2 Extended Baum-Welch Algorithm

Input: States $\bar{S} = S^{(0)}, S^{(1)}, ..., S^{(C)}$;
Observation Symbols $\bar{V} = V^{(0)}, V^{(1)}, ..., V^{(C)}$; A sequence of observed symbols: $O = \{O^{(1)}, O^{(2)}, ..., O^{(C)}\}$ where $O^{(c)} = O_1^{(c)} O_2^{(c)} O_3^{(c)} ... O_T^{(c)}$;
Output: HMM model: $\lambda_{\kappa} = (\bar{A}, \bar{B}, \bar{\pi}, \kappa)_C$

1: $\bar{\lambda} = \text{InitializeModel}(\bar{S}, \bar{V})$
2: $A^C = BlockDiagonalMatrix(\bar{A})$
3: $B^C = BlockDiagonalMatrix(\bar{B})$
4: $A^{\otimes}_{\kappa} = TransitionOptimization(A^C)$
5: $B^{\otimes}_{\kappa} = ObservationOptimization(B^C)$
6: $\bar{A}_{\kappa} = TransitionBackProjection(A^{\otimes}_{\kappa})$
7: $\bar{B}_{\kappa} = ObservationBackProjection(B^{\otimes}_{\kappa})$
8: **for** $c = 1$ to C **do**
9: $\quad \lambda^{(c)} = Baum - Welch(\lambda^{(c)})$
10: **end for**
11: **return** $\bar{\lambda}$

Another variant of the extended Baum-Welch algorithm is evaluated in this thesis where the consistency constraints are integrated multiple times, unlike the above algorithm where

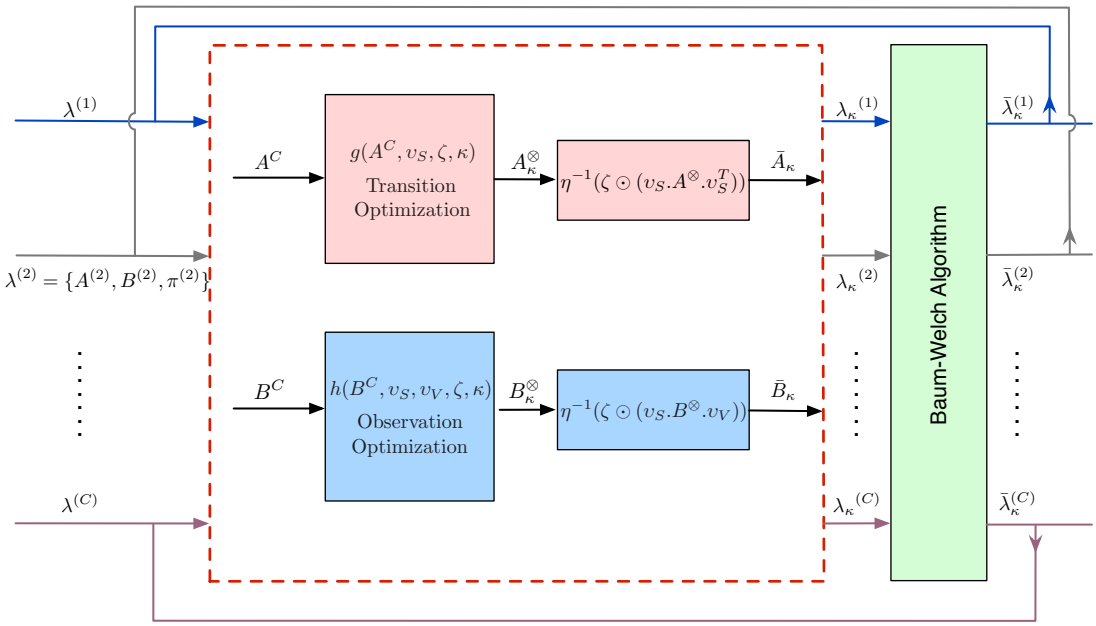

FIGURE 3.6. Extended Baum-Welch Algorithm with Consistency Constraints Integrated Multiple Times

the consistency constraints are integrated only at the beginning of the algorithm. This variant of the extended Baum-Welch algorithm starts as the above algorithm. But, once the individual models converge, this consistency constraints are integrated again and the chains are trained individually once again. This process is repeated until the observation probability of all the chains converge between two runs. This variant of the extended Baum-Welch algorithm is depicted in the figure 3.6. However, this variant of the extended Baum-Welch algorithm doesn't perform as well as the above algorithm. When the consistency constraints are applied multiple times, they push the transition and observation probability matrices to a space where most of the rows are identity. These results are further explained in chapter 4.

3.4. Summary

This chapter presented the extended Viterbi algorithm with consistency constraints to find the best state sequence, given an observation sequence. The advantage of this algorithm is that the consistency constraints can be included during run-time, although the models are not trained with the consistency constraints. This chapter also presented the extended Baum-Welch algorithm that trains the HMM by taking consistency constraints into account. The performance of these algorithms is evaluated on synthetic data in the next chapter.

CHAPTER 4

Experiments

Having proposed the algorithms to train and run a Consistent HMM in the previous chapter, we evaluate the performance of these algorithms. Synthetic data is generated to evaluate the performance of the Consistent HMM. Synthetic data gives the flexibility to evaluate the Consistent HMM on a variety of datasets with changing behavior, by changing the parameters used in the data generation. The synthetic data is explained in the next section along with the way it is generated. Then the extended Baum-Welch and extended Viterbi algorithms are evaluated individually and both in tandem. Finally, experiments are performed to evaluate the performance of Consistent HMM with respect to other models.

4.1. Synthetic Data

As already discussed, appearance, action and trajectory information of a track is essential for activity recognition. Section 2.2 discussed an example with respect to appearance labels. In addition to the appearance labels, we consider action and trajectory labels to generate synthetic data. Let us consider five appearance labels, five action labels, and three trajectory labels. These labels form the states and observation symbols for the respective HMMs.

The synthetic data is generated using an HMM model. A starting state is selected randomly based on the prior probabilities. Then the state transition probabilities will guide the next state changes, and the observation for a particular state is selected randomly based on the observation probabilities. The state sequence form the ground truth of the synthetic data, and the observation sequences can be considered as the data generated by the appearance labeling, action labeling, and trajectory labeling systems.

An HMM model is required to generate the synthetic data. In this thesis, we are trying to model the dependencies that occur between multiple interacting chains. So, the data should not be generated by independent HMM models, but a cartesian state HMM. The states and observation symbols in cartesian space are generated by the cartesian product of appearance, action, and trajectory labels. The cartesian product results in seventy five cartesian states, where each state is a 3-tuple.

We assume that it is equally likely to start at any state. So, the prior probability matrix for cartesian HMM is filled uniformly such that the prior probabilities sum to 1. In other words, the prior probability of each state is $\frac{1}{75}$.

The transition probability matrix for the cartesian HMM is generated by the Kronecker product of the individual transition probability matrices. The decoupled transition probability matrices have a special structure inspired by the real world. In the real world, objects tend to have the same appearance, do the same action, follow a specific trajectory more often than changing them. So, the decoupled transition probability matrices are diagonally dominant. To be more specific, the appearance of an object doesn't change often in the real world. So, the diagonal entry of appearance transition probability matrix is set to 0.8, and the other values are uniformly distributed such that the sum of the probabilities of each row is 1. Actions tend to change slightly more often, and trajectories change more often. So, diagonals of action and trajectory probability matrices are set to 0.7 and 0.6 respectively and the non-diagonal values are uniformly distributed. The Kronecker product of these three decoupled transition probability matrices gives the cartesian state transition probability matrix.

In an ideal world where the labeling systems are perfect, the labeling systems always output the states they see. In such cases, the observation probability matrix is an identity

matrix. But, the labeling systems often produce noisy labels. Even though the systems are noisy, we assume that the probability of labeling a correct state is more than the probability of labeling an incorrect state. This assumption makes the observation probability matrix diagonally dominant. Observation error is the measure of how often the labeling systems assign incorrect labels. Observation error can be added to the observation probability matrix by setting the diagonal values as $1 - \frac{error}{100}$ and uniformly setting the off diagonal values such that the sum of individual rows is 1.

Some combinations of appearance, action and trajectory labels do not occur in real world. In other words, some cartesian states are inconsistent. For example, trees can't walk. As these states don't co-occur, the respective rows and columns in the transition table, and the respective rows in observation table are zeroed out. Zeroing out some rows and columns can be viewed as adding dependencies between the processes. In addition to the consistency dependencies, there might be other dependencies that we are not aware of. Gaussian noise is added to the transition and observation probability matrices to account for such dependencies. Gaussian noise is generated with mean of zero and standard deviation as the quarter of the minimum value of the respective matrix excluding zeros. The probability values are very small, and adding a small amount of negative noise may make the probability value negative, which is against the basic probability rules. So, the absolute of the Gaussian noise is added to the transition and observation probability matrices, wherever the states are consistent. As the inconsistent states are already zeroed out, Gaussian noise is not added to the inconsistent states. After adding the inconsistencies and Gaussian noise, the matrices are row normalized.

There are two variables in synthetic data generation - the number of inconsistent states and the observation error. Different sets of synthetic data is generated by varying these

two variables. The percent of inconsistent states is varied from 0 to 80 in increments of 20 i.e., 0%, 20%, 40%, 60%, or 80%. The inconsistent states are randomly selected based on the percentage of inconsistent states. The observation error is varied between 20 and 80 in increments of 20. So, five values are chosen for inconsistent states and four values are chosen for observation error. Synthetic data is generated for each combination of these two variables. Fifteen models are generated for each combination to account for random inconsistent states and Gaussian noise. The combination of inconsistent states, observation error and fifteen models for each inconsistent states and observation error gives 300 different datasets.

Synthetic data is generated for each model by running an HMM as already discussed. For each model, 640 sequences are generated, each of length 100. These sequences are divided into a training set that comprises of 128 sequences, and a testing set that comprises of 512 sequences. The states that generated the observations are stored as ground truth data to evaluate the performance of HMMs.

A pilot study is performed in section 4.2 to show that a fully coupled HMM requires enormous amounts of training data. A different set of synthetic data is generated for this purpose. The percent of inconsistent states is set to zero and the observation error is chosen to be 20, 40, 60, or 80. For each observation error, 5 models are generated resulting in 20 different models. Each model is used to generate 2500 sequences each of length 100. These sequences are equally partitioned into train and test set.

The generated synthetic data is used to evaluate the performance of the proposed consistent HMM with respect to different HMMs in the following sections. Pilot study is performed by varying the training sample size from the pilot study synthetic data train set.

4.2. Pilot Study

Section 2.3 introduced various extensions to HMMs to model data from multiple interacting processes. A fully coupled HMM (FCHMM) was introduced as an extension of HMM to model data from multiple interacting processes. A FCHMM has states exponential in the number of chains. So, the state space is huge and enormous amounts of data are necessary to train it. As already stated, FCHMM is not practical even for a small number of chains with limited states. This section does a pilot study that supports the above claim.

Synthetic data generated for the pilot study with 1250 train sequences and 1250 test sequences is used for this study. A FCHMM is trained using Baum-Welch algorithm until convergence, with training set of size varying from 1 to 1024 in the powers of 2. Once the FCHMM is trained, Viterbi algorithm is used to estimate the most likely state sequences for the test set.

The performance of the FCHMM is evaluated by comparing the estimated state sequence with the ground truth data. The percentage of states correctly predicted is the accuracy of the FCHMM and is used as a performance measure. Figure 4.1 gives the results of the pilot study.

The above figure shows the plot of accuracy of the FCHMM versus training set size for different observation errors. The graph shows a trend of increase in the accuracy with train set size. The graph also shows a trend in the observation error. The less the error in the observation probability matrix, the better the accuracy is, for a given train set size. Even for a small dataset with 3 chains and 75 cartesian states, FCHMM achieved a maximum accuracy of only 32.39%, for a train set of size 1024 each of length 100. Moreover, as the observation error increased, the accuracy decreased achieving an accuracy of 4.34% for an observation error of 80%. As FCHMM achieved a maximum accuracy of 32.39% by looking

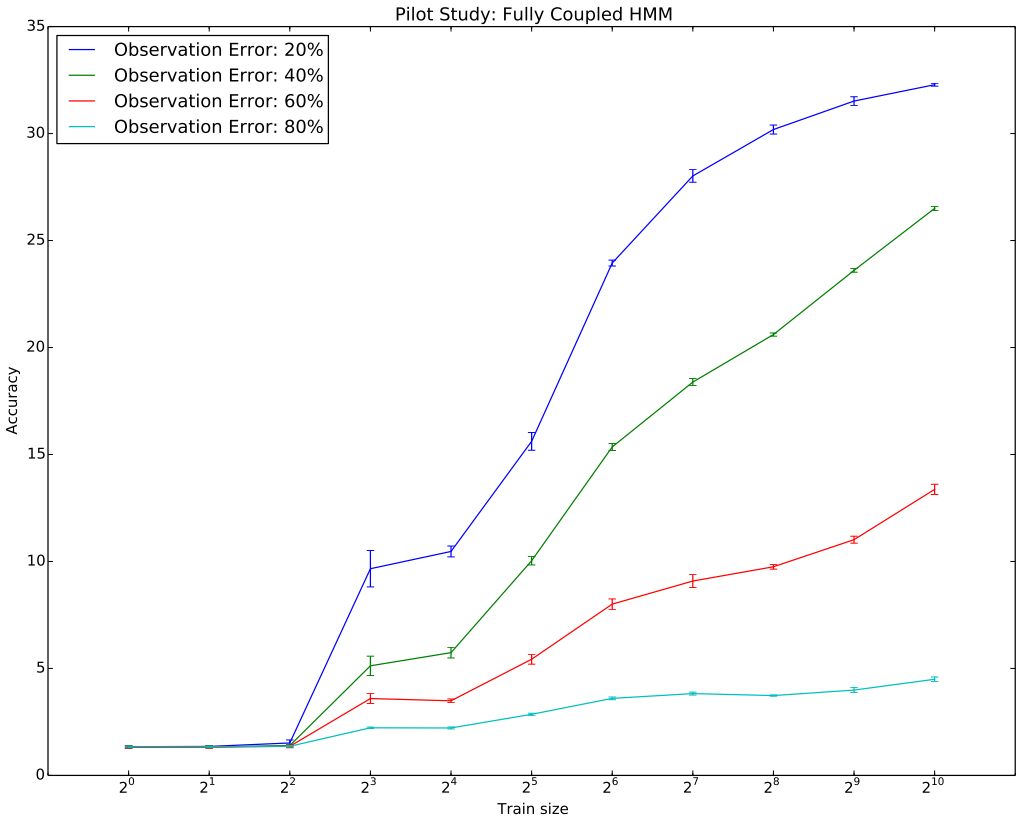

FIGURE 4.1. PilotStudy: Plot of train set size vs accuracy for different values of observation error of a FCHMM

at a million data points even for a small toy problem, it is not a practical extension to model data from multiple interacting processes.

4.3. Experiment 1: Performance of Extended Viterbi Algorithm without retraining

One of the contributions of this thesis is to propose an extended Viterbi algorithm that can integrate consistency constraints at run-time without the need of retraining the HMM. This section shows the results to support the contribution.

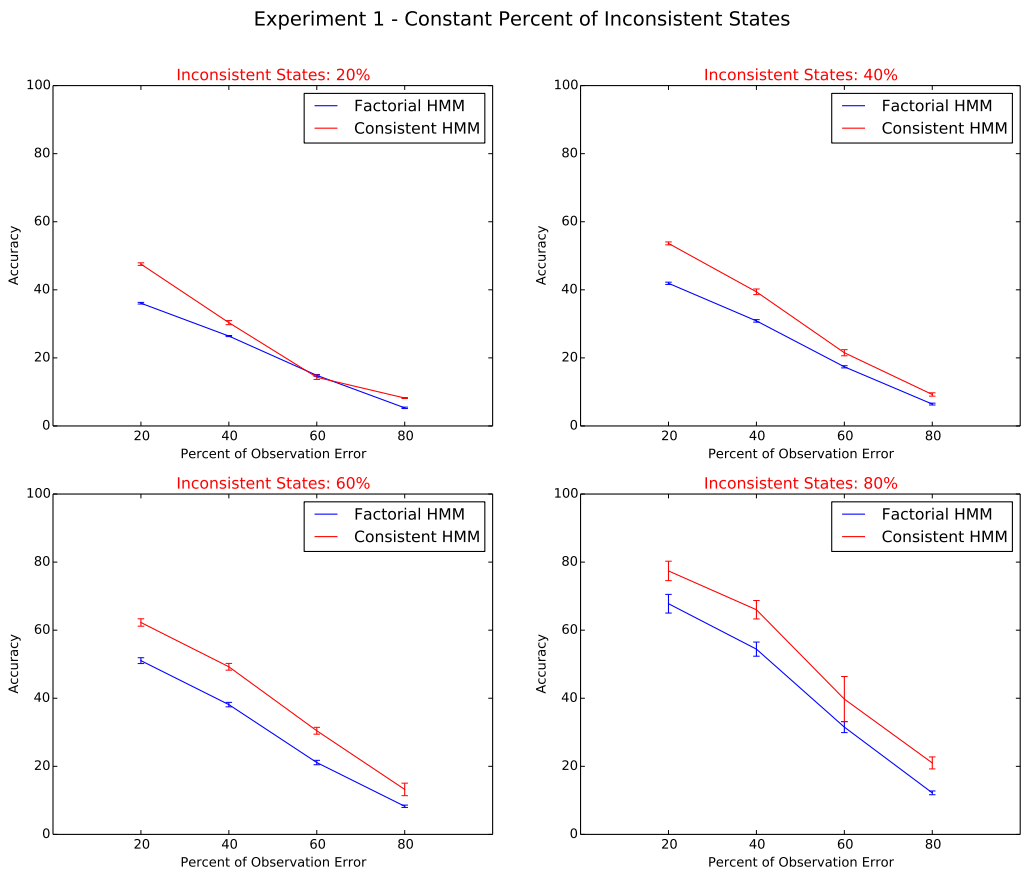

FIGURE 4.2. Experiment 1: Plots of Observation error percentage vs Accuracy for different percent of inconsistent states

As this experiment compares the performance of extended Viterbi algorithm without the need of retraining, a trained HMM must be provided. The HMM is trained as a factorial HMM using the standard Baum-Welch algorithm. Then, the extended Viterbi algorithm discussed in section 3.2 is run using the model trained as a FHMM. The standard Viterbi algorithm is also run to see the performance gain of the extended Viterbi algorithm. Figures 4.2 and 4.3 shows the results of this experiment.

Figure 4.2 shows four subplots, one for each percent of inconsistent states. Each subplot shows the change in percentage of states correctly predicted(accuracy) with the observation error. The plots show that the performance of the extended Viterbi algorithm is always better

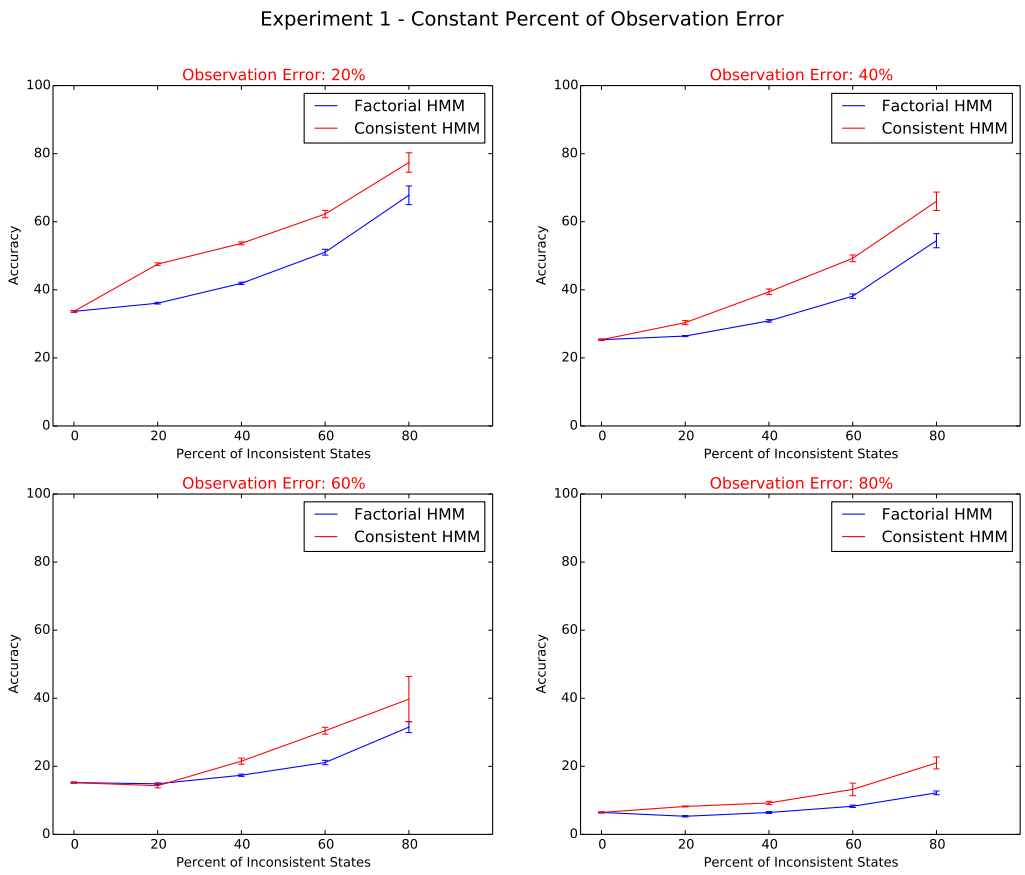

FIGURE 4.3. Experiment 1: Plots of Inconsistent state percentage vs Accuracy for different percent of observation error

than the standard Viterbi algorithm. The performance gain is more at the lower extremes of observation error. As the observation error increases the performance gain of running the extended Viterbi algorithm decreases. The reason for this behavior can be attributed to the model learned by the factorial HMM. As the observation error increases, the more noise in the data results in the noise model being learned. So, the gain in integrating the inconsistency constraints to the noise model is less compared to the gain in integrating the inconsistency constraints to the data model.

Figure 4.3 shows another view of the same data. Each subplot represents an observation error and each subplot shows the trend of change in accuracy with inconsistent states. As the

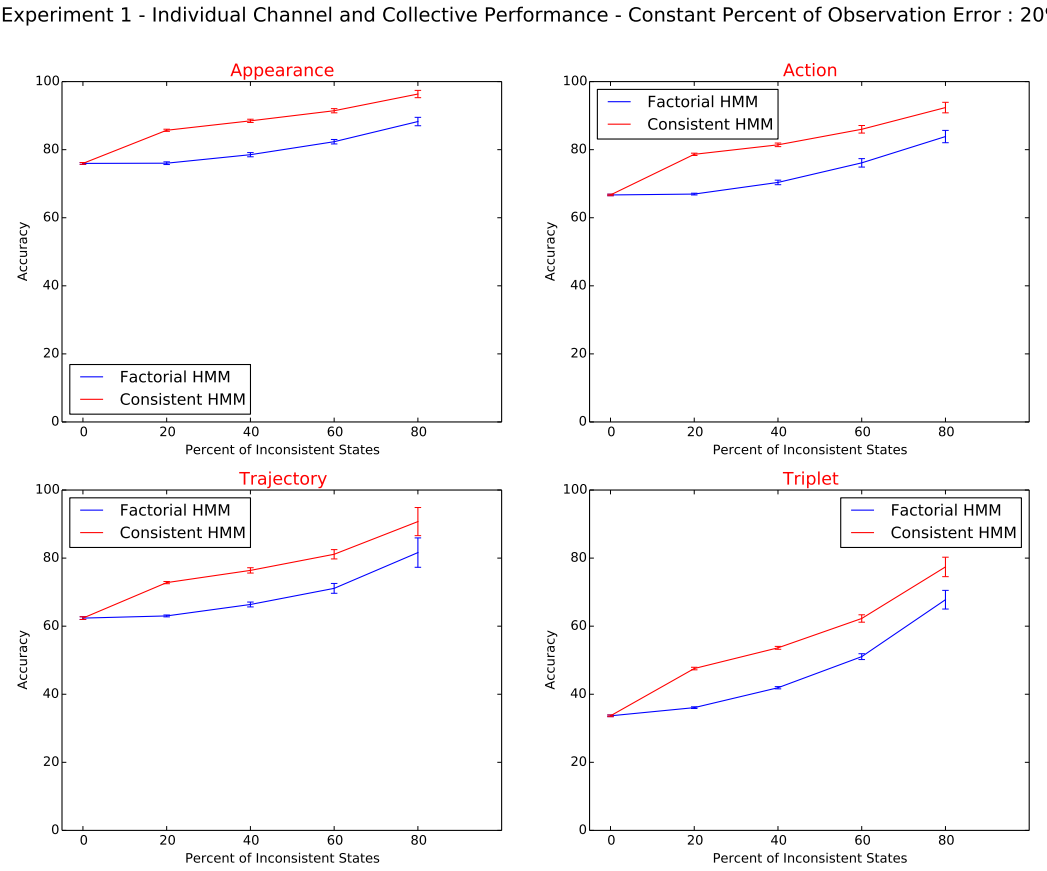

FIGURE 4.4. Experiment 1: Plots of Inconsistent states percentage vs Accuracy for appearance, action, trajectory chains, and triplet for an observation error of 20%

inconsistent states increase, the gain of using the extended Viterbi algorithm increases. The increase in the performance gain is due to the additional knowledge that can be integrated when a large number of states are inconsistent. Moreover, as the inconsistent states increase, the standard deviation also increases. Although the error bars of the extended Viterbi algorithm is higher, an identical behavior can be seen with standard Viterbi algorithm as well. As the number of inconsistent states increase, the search space is no longer smooth and has many local optimas resulting in high standard deviation.

Figures 4.4 and 4.5 show the plots of inconsistent states versus accuracy for appearance, action, trajectory streams individually and the whole triplet for observation error of 20% and

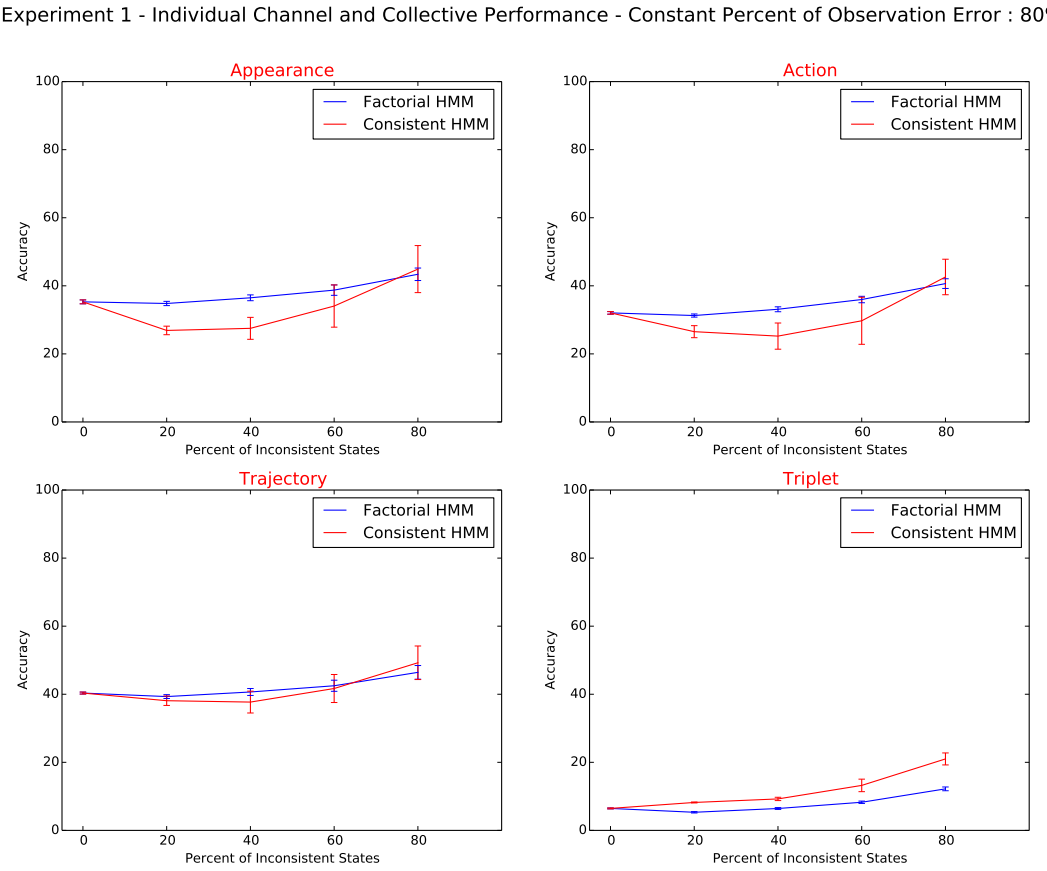

FIGURE 4.5. Experiment 1: Plots of Inconsistent states percentage vs accuracy for appearance, action, trajectory chains, and triplet for an observation error of 80%

80% respectively. As can be seen in figure 4.4, for small observation error, the performance of the extended Viterbi algorithm is better than the standard HMM for individual chains as well as the triplet collectively. Although the accuracy of individual chains is less for high observation error, as can be seen in figure 4.5, the triplet accuracy is still high. The extended Viterbi algorithm is designed to maximize the accuracy of the state combinations. So, it is compromising on the accuracy of individual chains to increase the accuracy of the combination.

Another interesting observation that can be made from the figure 4.5 is that the performance of the extended Viterbi algorithm is less in appearance chain, followed by action

chain, and trajectory chain compared to the standard HMM algorithm. The synthetic data is generated such that the appearance chain changes state less often, followed by the action chain, and the trajectory chain that changes states more often. Moreover, the trajectory chain has fewer states compared to the other two. By the above observations, we infer that the performance gain in individual chains is more if they change states more often. When the state transitions are fewer, the model learned by Baum-Welch training is good, and adding inconsistencies during run time hampers the performance. However, the accuracy of the extended Viterbi algorithm for the state combination triplet is always high compared to the standard Viterbi algorithm.

Additional plots of accuracy versus observation error and accuracy versus inconsistent states for individual chains and combinations are shown in appendix C. They exhibit the same observations outlined above.

4.4. Experiment 2: Performance of Extended Baum-Welch Algorithm

Experiment 2 evaluates the performance gain of integrating the consistency constraints during the training phase. Factorial HMM is used as a comparison model. Consistent HMM is trained by the extended Baum-Welch algorithm by integrating the consistency constraints in the training phase as outlined in section 3.3. A factorial HMM is also trained using the standard Baum-Welch algorithm. Standard Viterbi algorithm is run on both the trained models to evaluate the performance gain of training the HMMs using the extended Baum-Welch algorithm. The results of this experiment are shown in the figures 4.6 and 4.7.

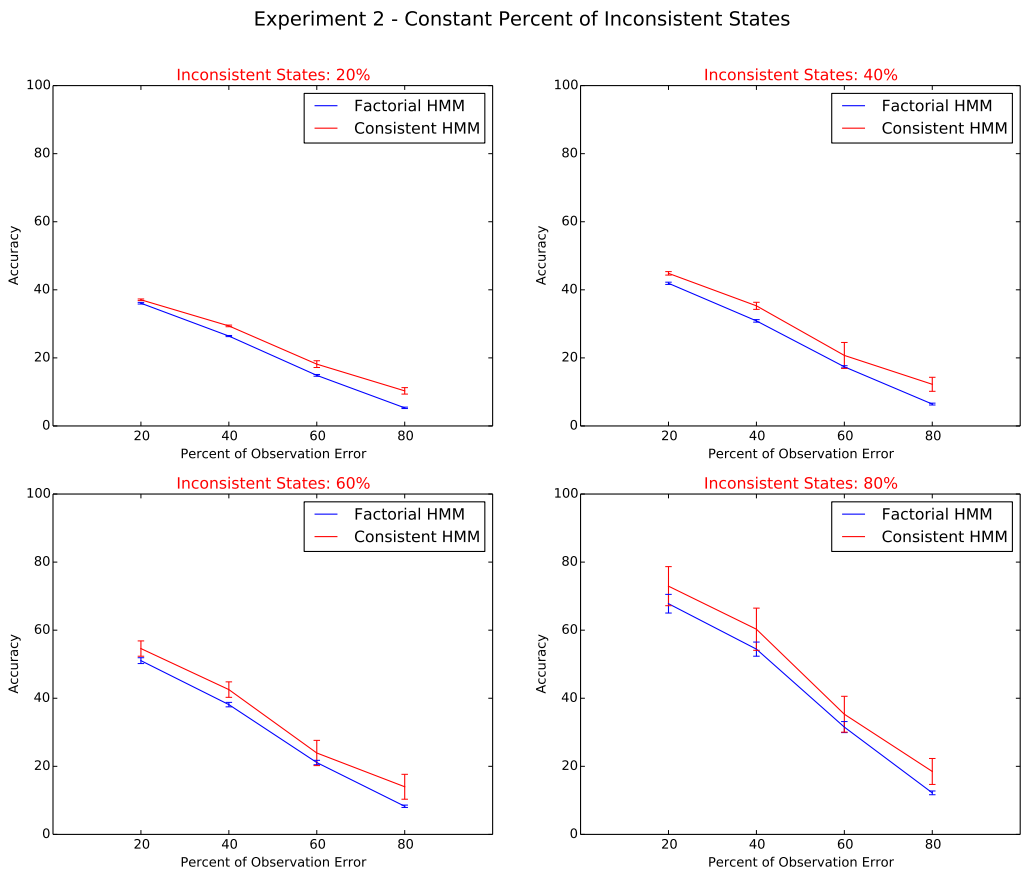

FIGURE 4.6. Experiment 2: Plots of Observation error percentage vs Accuracy for different percent of inconsistent states

Figure 4.6 shows the plots of accuracy versus observation error for different percentages of inconsistent states. As can be seen from the figure, the accuracy of the model trained by the extended Baum-Welch algorithm is better than that of the model trained by the standard Baum-Welch algorithm. For a given percentage of inconsistent states, the difference in the viterbi accuracy between the extended and standard Baum-Welch algorithms increases with the observation error. For low observation error and less inconsistent states, the model learned by both algorithms is almost the same. So, the gain in using the extended Baum-Welch algorithm is less in this case. As the inconsistent states and observation error increase,

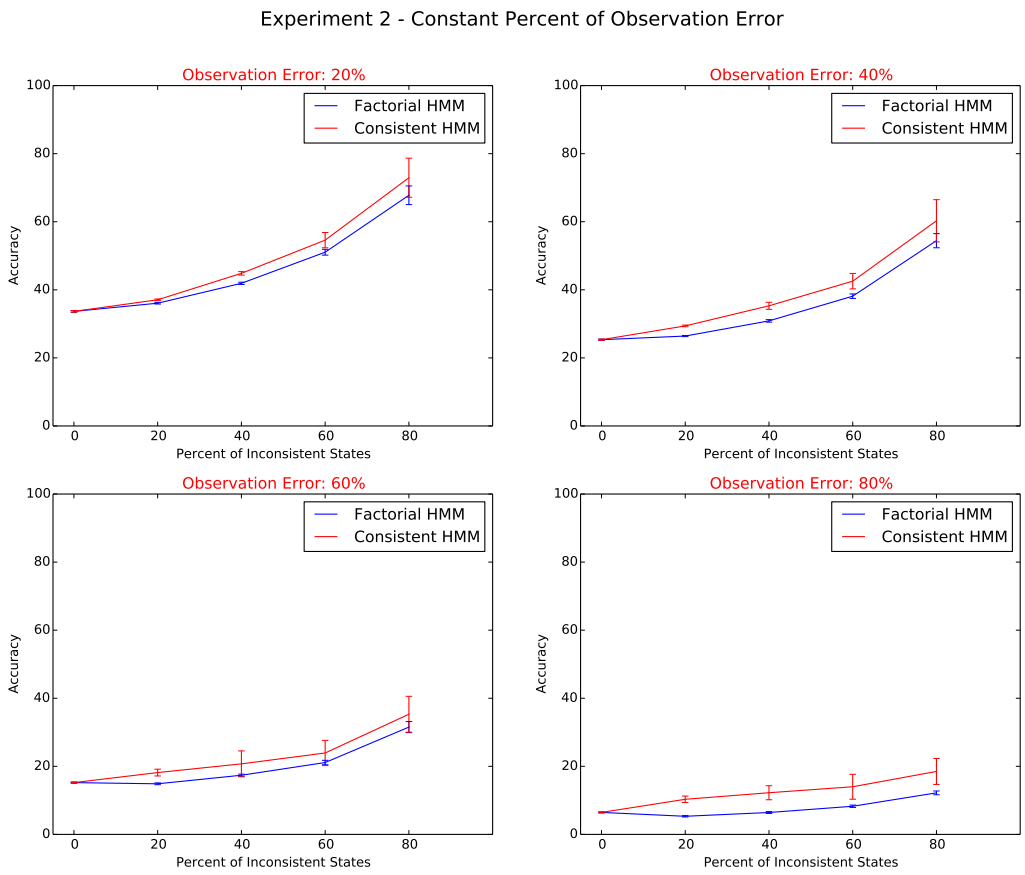

FIGURE 4.7. Experiment 2: Plots of Inconsistent state percentage vs Accuracy for different percent of observation error

the models learned by both algorithms are different. This is evident from the performance of the extended Baum-Welch algorithm.

Figure 4.7 shows another view of the data shown in figure 4.6. The figure shows the plots of accuracy versus inconsistent states for different observation errors. The figure shows the increase of accuracy of the extended Baum-Welch algorithm compared to the standard Baum-Welch algorithm with the increase in the number of inconsistent states. As the number of inconsistent states increase, more knowledge can be integrated in the training phase that leads to an increase in performance of the extended Baum-Welch algorithm.

Although the extended Baum-Welch algorithm performs better than the standard Baum-Welch algorithm, the improvement in the performance is not large. One possible reason for this behavior is the less expressive power of the factorial model. Although the consistency constraints are integrated in the consistent state space and the closest point is found in the factorial space, the projection in the factorial space has no constraints. When a standard Viterbi algorithm is run on the models in factorial state space, the consistency constraints are not considered and the performance gain is not high. Another reason is that the consistency constraints are integrated into the models in factorial state space at the beginning of the Baum-Welch algorithm. Except for this step, the extended and the standard Baum-Welch algorithms are the same. In other words, the extended Baum-Welch algorithm starts as a better initial model compared to the standard Baum-Welch algorithm based on the consistency constraints. Starting as a better initial model may place the extended Baum-Welch algorithm in a better local optima compared to the standard Baum-Welch algorithm, but in the vicinity of the standard Baum-Welch's local optima. So, the performance of extended Baum-Welch algorithm is only slightly better than the standard one.

An experiment is done to see if it is advisable to integrate the consistency constraints multiple times. This is done by running the Baum-Welch algorithm on all the chains until convergence, then integrating the consistency constraints by the optimization step, and repeating these steps until the chains converge between two iterations. This version of extended Baum-Welch algorithm resulted in low accuracy compared to that of the standard Baum-Welch algorithm, as can be seen in figures 4.8 and 4.9. Moreover, the standard deviation of the accuracy is large. Running the optimization step multiple times seemed to make the models get stuck in optima where most of the rows in the transition and observation probability matrices are identity. Once the models get stuck in this optima, they never to

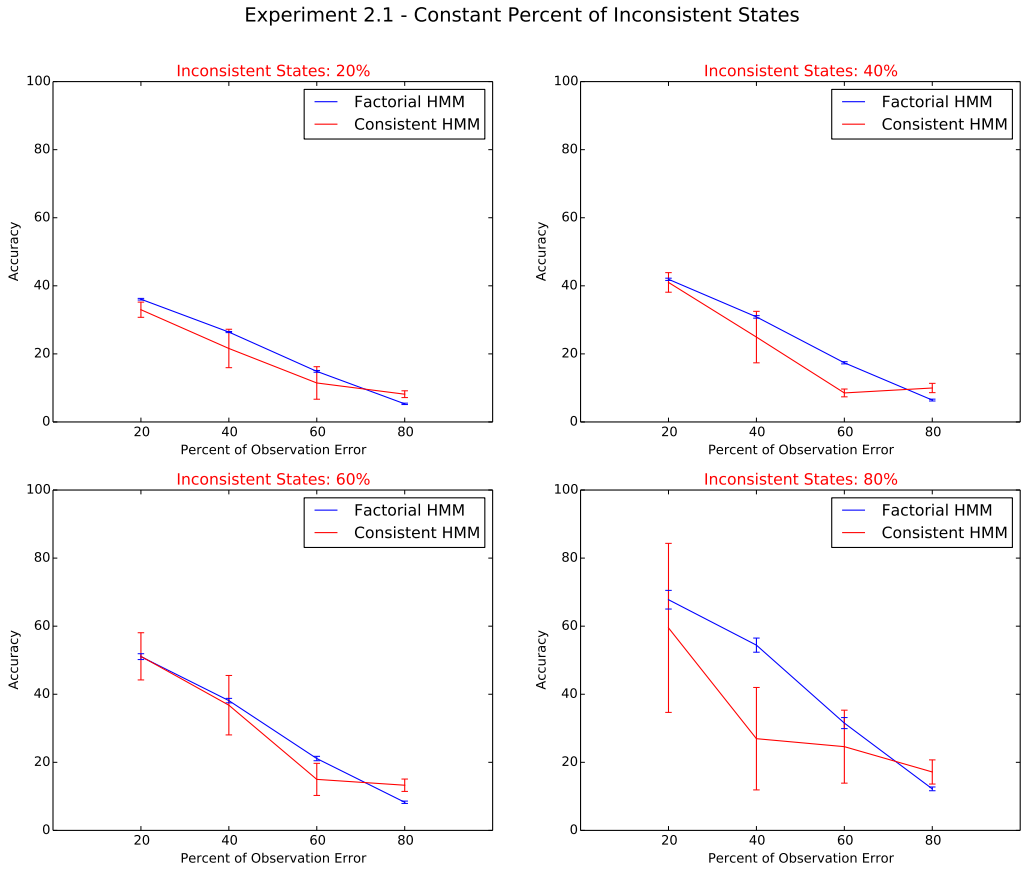

FIGURE 4.8. Experiment 2.1: Plots of Observation error percentage vs Accuracy for different percent of inconsistent states

move out of it, and the accuracy of the resulting model is less than that of the standard Baum-Welch algorithm.

Additional plots for specific observation error and percent of inconsistent states for individual chains and combinations are shown in appendix D. The plots are for the version of extended Baum-Welch algorithm without iterations. Additional plots show the same behavior as outlined above.

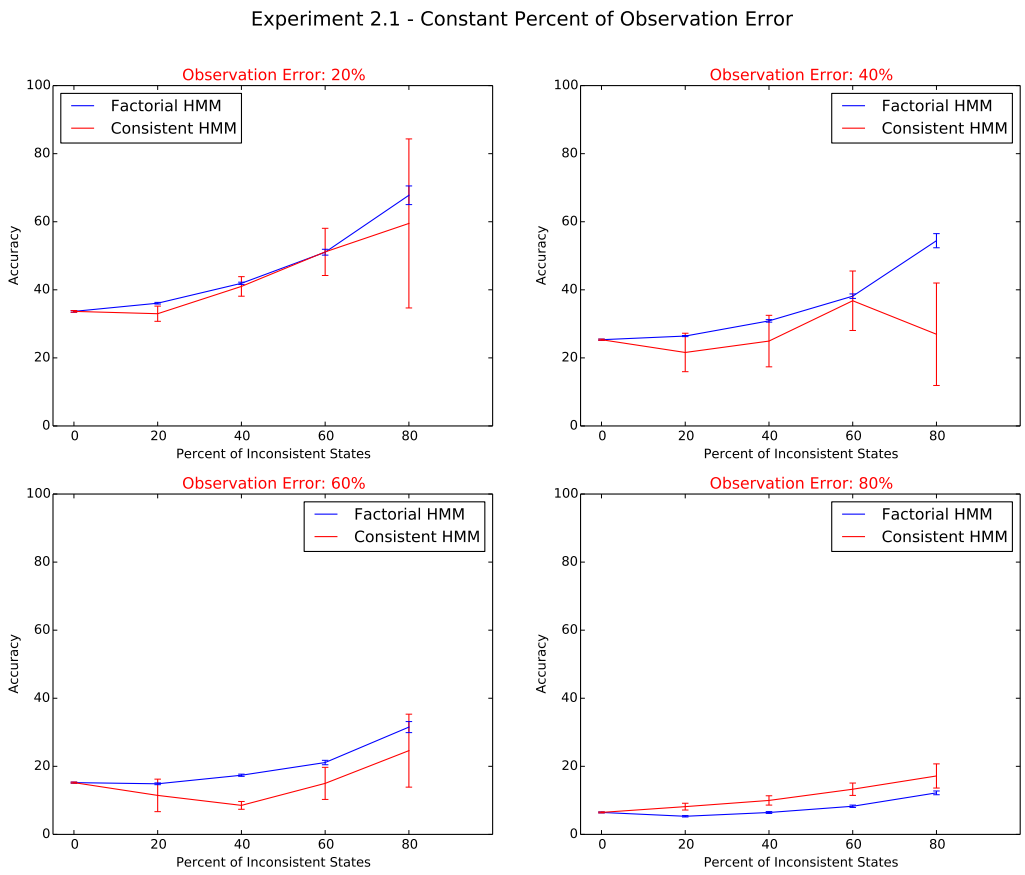

FIGURE 4.9. Experiment 2.1: Plots of Inconsistent state percentage vs Accuracy for different percent of observation error

4.5. Experiment 3: Performance of Consistent HMM trained using extended Baum-Welch algorithm and run by extended Viterbi algorithm

This experiment evaluates the performance of integrating the consistency constraints both during training and run-time. This is done by training and running the Consistent HMM using the proposed extended Baum-Welch algorithm and extended Viterbi algorithms respectively. Figures 4.10 and 4.11 show the plots for this experiment. The results of

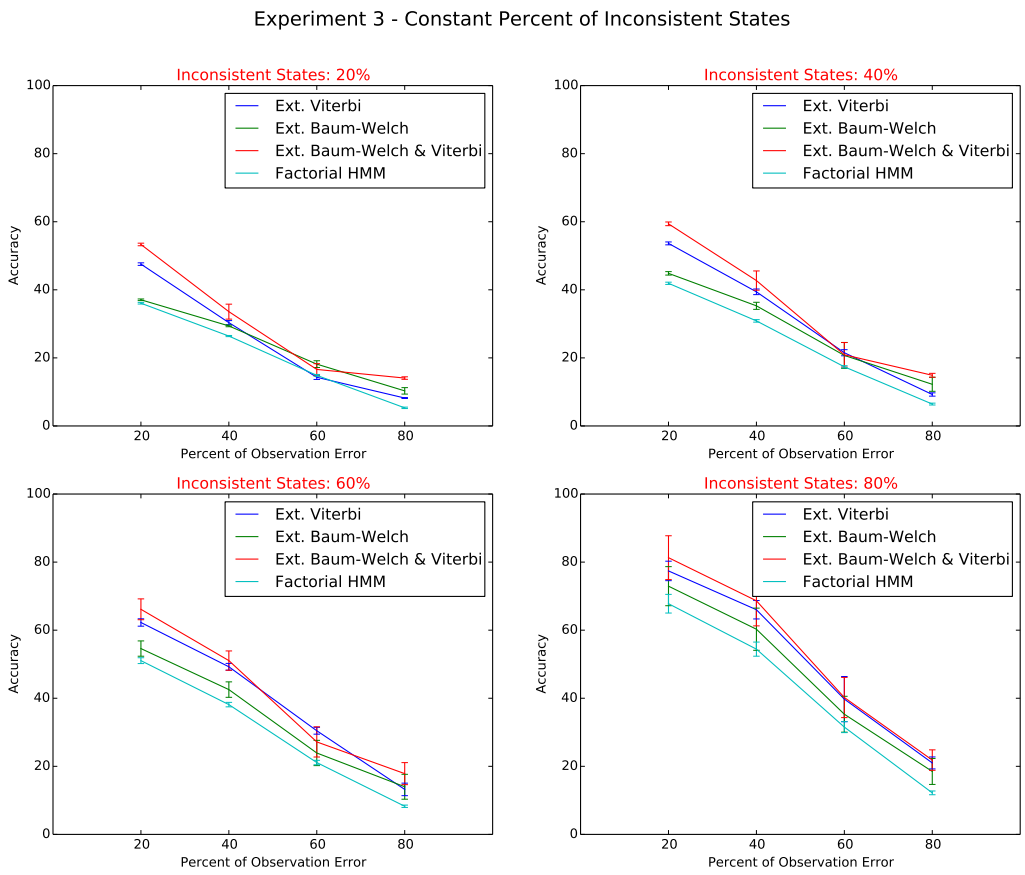

FIGURE 4.10. Experiment 3: Plots of Observation error percentage vs Accuracy for different percent of inconsistent states

experiments 1 and 2 are also shown to see if the actual performance gain occurs during training, running, or a combination of both.

Figure 4.10 shows the plots of accuracy versus observation error for different percentages of inconsistent states. The plots show results for models trained and run as factorial HMMs, models trained as consistent HMMs and run as factorial HMMs, models trained as factorial HMMs and run as consistent HMMs, and models trained and run as consistent HMMs. Although the model trained and run as a consistent HMM performs better than the other models every time except for an observation error of 60%, we can see that most of the performance gain comes from the extended Viterbi algorithm. Training the model as a

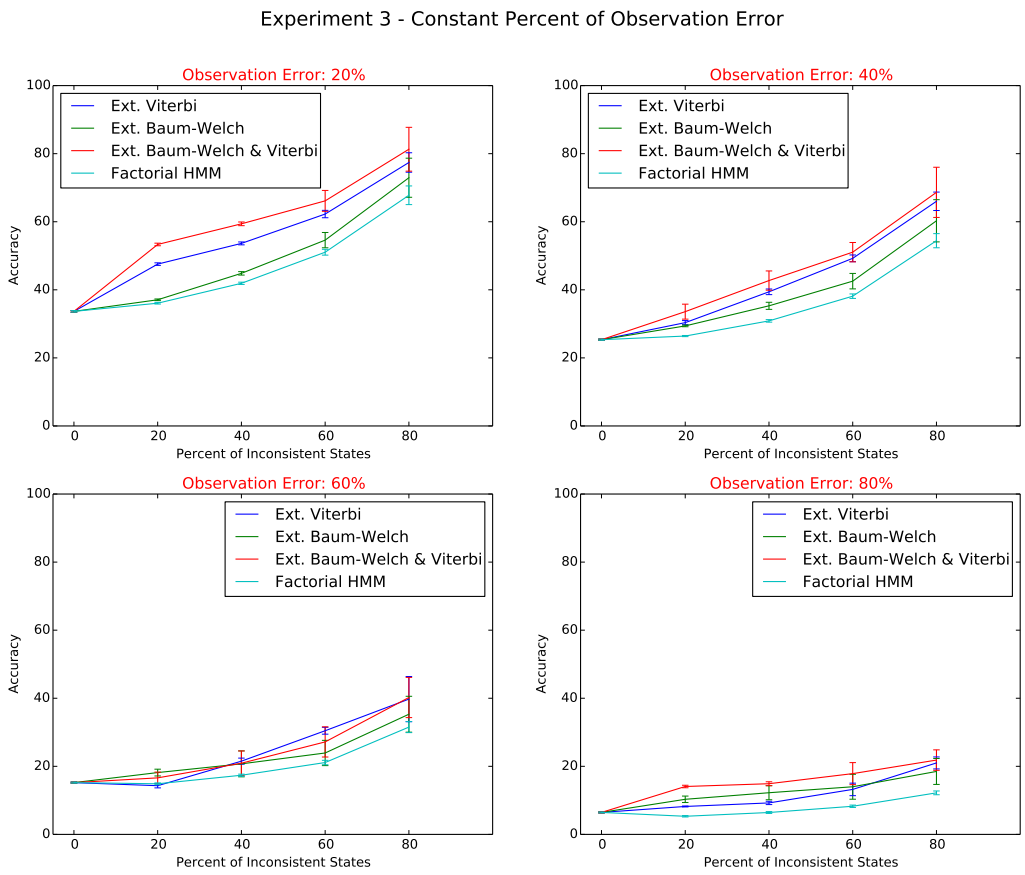

FIGURE 4.11. Experiment 3: Plots of Inconsistent state percentage vs Accuracy for different percent of observation error

Consistent HMM gives a slight advantage over training as a factorial HMM. When the extended Viterbi algorithm is run on the slightly better model from a consistent HMM, the performance increases than the model trained as a factorial HMM and run as a consistent HMM.

Figure 4.11 shows another view of the data shown in figure 4.10. As can be seen in the figure, training the model using the extended Baum-Welch algorithm is useful in the case of high observation error. When the observation error is less, the extended Viterbi algorithm alone is enough. Training the model using the extended Baum-Welch algorithm

and running it using the extended Viterbi algorithm doesn't hurt the performance, although the performance gain in doing so is not significant.

Additional plots for specific observation error and percent of inconsistent states for individual chains and combinations are shown in appendix E. The plots exhibit the behavior discussed above.

4.6. Experiment 4: Performance of Consistent HMM with respect to FHMM, FCHMM and BHMM

A final experiment is performed to compare the performance of consistent HMMs with factorial HMMs (FHMMs), fully coupled HMMs (FCHMMs) and Brand version of coupled HMMs (BCHMMs). Each model is trained and run with their respective Baum-Welch and Viterbi algorithms. The results of this experiment are shown in figures 4.12 and 4.13.

Figure 4.12 shows the plots of accuracy versus observation error for different percentages of inconsistent states. As can be seen from the figure, consistent HMM performs better than other models. A trend can be observed on the performance of different models. Consistent HMM performs better followed by FHMM, FCHMM and BCHMM. FCHMM is performing worse than the consistent and factorial HMMs as the data provided (128 sequences each of length 100) is insufficient to train the cartesian space of fully coupled HMM. This was predicted by the pilot study. BCHMM is performing the worst of all the models. The performance can be attributed to the way it is defined. BCHMM trains in cartesian state space. But after every iteration, the model is factored into a factorial space and projected back onto a cartesian state space. Moreover, the algorithm captures the dependencies between two chains in the factoring step. However, the synthetic data has dependencies in three

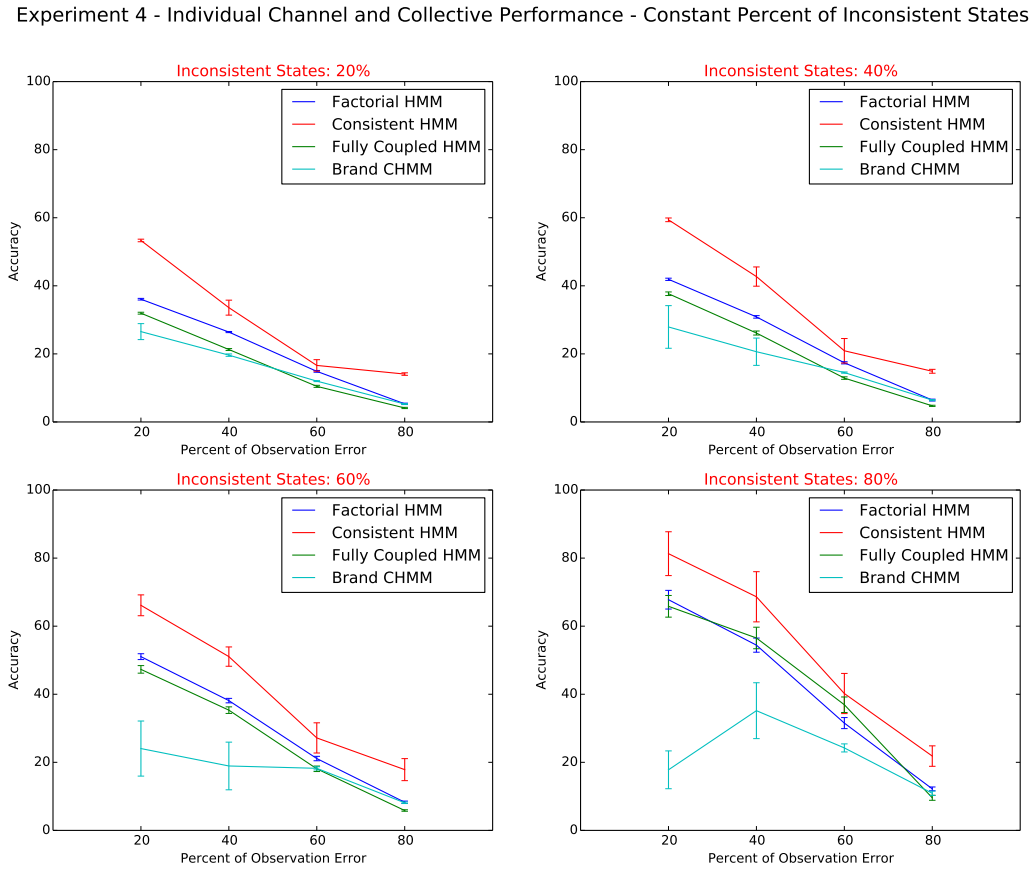

FIGURE 4.12. Experiment 4: Plots of Observation error percentage vs Accuracy for different percent of inconsistent states

chains that can't be captured as dependencies between two chains. As the BCHMM is being trained in cartesian state space, and as it can't capture the dependencies that exists between multiple chains, it is performing worse than the fully coupled HMM.

Figure 4.13 shows another view of the data in figure 4.12. We can see that the consistent HMM is the clear winner among all models. As the observation error increases, the performance of FHMM, FCHMM, BCHMM is converging, but the consistent HMM is performing better than the other models at all times.

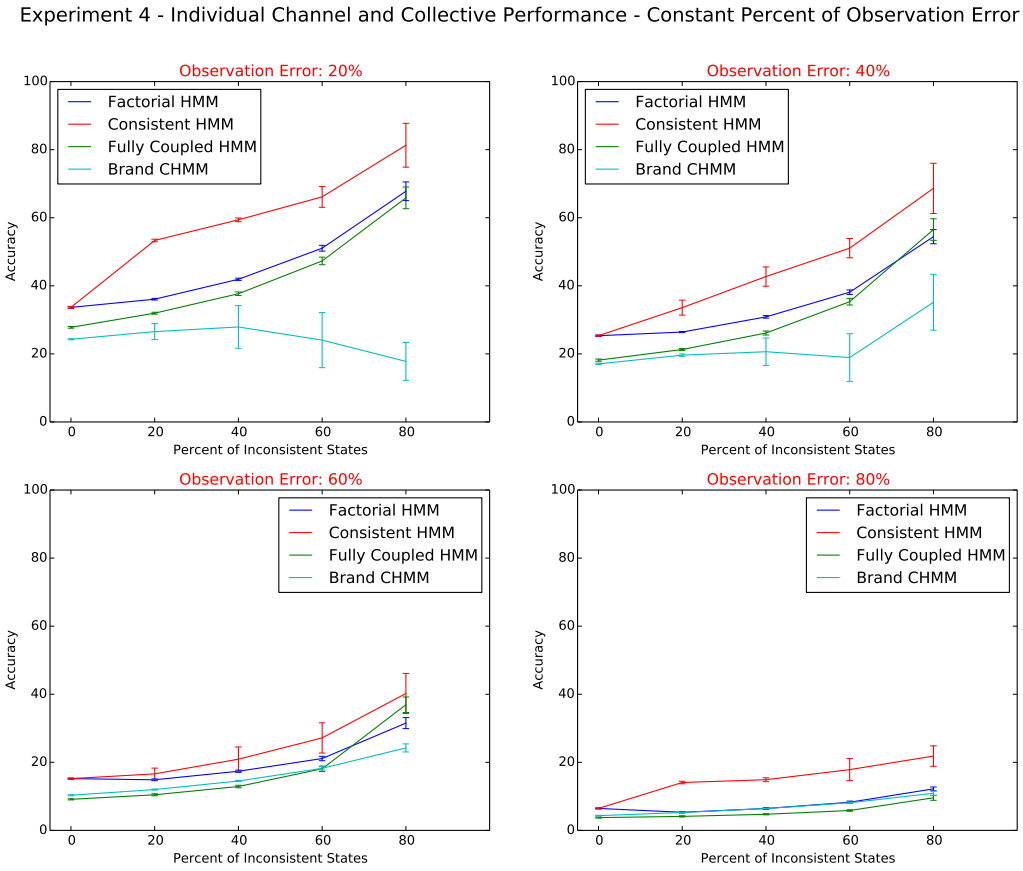

FIGURE 4.13. Experiment 4: Plots of Inconsistent state percentage vs Accuracy for different percent of observation error

Additional plots for specific observation error and percent of inconsistent states for individual chains and combinations are shown in appendix F. There are figures for fixed observation error and variable percent of inconsisten states and vice versa. All the additional plots exhibit the behavior discussed above.

4.7. Summary

This chapter started with the discussion of the Synthetic Data used in the thesis and the way it is generated. A pilot study is performed to show why a fully coupled HMM is not a practical model, although it is the way to model dependencies between multiple chains of

data. Then the performance of the extended Viterbi algorithm is evaluated by running the standard and extended Viterbi algorithms on a model trained as a factorial HMM. Then the performance of the extended Baum-Welch algorithm is evaluated in isolation by training two different models, one using the standard Baum-Welch algorithm and the other one using the extended Baum-Welch algorithm. The standard Viterbi algorithm is run on both the models to isolate the performance gain in training the model as a Consistent HMM. Another experiment is performed where a model is trained and run as a Consistent HMM. A final experiment is performed where a consistent HMM is compared to a FHMM, FCHMM and BCHMM.

The pilot study supported our claim that although the FCHMM is a best model to model dependencies between multiple chains, it requires enormous amounts of training data. FCHMM achieved a best accuracy of 32.39% when the training data comprised of 1024 sequences each of length 100. Other experiments proved that the proposed Consistent HMM is the best model compared to FHMM, FCHMM, BCHMM. The main performance gain of Consistent HMM comes when the consistency constraints are integrated at run-time (extended Viterbi algorithm).

CHAPTER 5

Conclusion and Future Work

5.1. Conclusion

Action recognition in a scene requires the appearance, action, and trajectory information of the important objects in the scene. Although there exists many methods to estimate the appearance, action, and trajectory labels of a tracked object, these labels are often noisy. Moreover, as the methods that give information about an object are independent of each other, mutual information between them is lost. Hidden markov models (HMM) can be used to smooth the noisy stream of labels over time. However, a simple HMM can't model data from multiple dependent processes.

A fully coupled HMM is an ideal model for modeling data from multiple dependent processes. Although FCHMM is the ideal model, it is untenable, as the state space grows exponentially in the number of chains. As a result, enormous amounts of training data are required to train an FCHMM. The pilot study in section 4.2 explains this behavior. Even for small synthetic dataset as discussed in this thesis, FCHMM achieves a maximum accuracy of 32.39% given 102,400 observations. Although the FCHMM is an ideal model to model data from multiple dependent processes, it is not practical.

On the other extreme, a factorial HMM is the simplest model to smooth noisy stream of labels over multiple chains. A factorial HMM is an HMM where individual HMMs are used to model separate chains. Although factorial HMM is the simplest model to smooth data over multiple chains, it doesn't capture mutual information between chains. As the factorial HMM doesn't capture the mutual information between chains and smoothes the chains independently, the estimated labels may be inconsistent. For example, the predicted

labels may be tree, walk, and fast, even though, we know that trees can't walk or move fast. However, the knowledge that states that trees can't walk or move fast is available in natural language ontological databases.

This thesis proposed a new formulation of an HMM called Consistent HMM. Consistent HMM integrates the knowledge of what state combinations are valid and what combinations are invalid into the factorial model. The extended Baum-Welch algorithm and extended Viterbi algorithm are proposed to integrate the knowledge of consistent states during training and run-time respectively. The extended Baum-Welch algorithm discussed in section 3.3 and the extended Viterbi algorithm discussed in section 3.2 integrates the consistency constraints in the cartesian state space by finding a point on cartesian state space with consistency constraints that can be explained by factorial models.

Experiments are performed on synthetic data generated by varying the percentage of inconsistent states and the percentage of error in the observation table. To assess the performance improvement of applying the consistency constraints during run time, an experiment is performed by training the HMM as a factorial HMM, and the consistency constraints were integrated at run-time by the extended Viterbi algorithm. This experiment revealed that integrating the consistency constraints during run-time yields better accuracy. Although the knowledge about the consistency constraints is not available during the training, the consistency constraints can still be included at run-time.

The performance improvement of including the consistency constraints during training was evaluated by training the Consistent HMM using the extended Baum-Welch algorithm. The performance improvement of integrating the consistency constraints during training was not as high as integrating them during run-time. One primary reason for this is that the factorial models' expressive power is far less compared to that of the cartesian models.

Moreover, the optimization step of integrating the consistency constraints was performed at the beginning of training phase, and the models are then trained as factorial HMM. So, the optimization step resulted in a better starting state for the training, from which the local search was performed without any consistency constraints.

There is obviously performance gain of integrating the consistency constraints both during training and run-time. But, most of the actual performance gain is from the extended Viterbi algorithm. The performance of integrating the consistency constraints both during training and run-time can be visualized as the sum of the performance gain of integrating the consistency constraints during training and run-time individually.

When the consistency constraints are applied only during run-time, there are cases where the performance of individual chains fell below the factorial HMMs. However, when the performance of the three chains were looked in tandem, it was always better than that of the factorial HMM. The proposed algorithm maximizes the accuracy of all the chains together, not individual ones. So, it sacrifies the performance of individual chains in order to increase the combined accuracy.

The performance of consistent HMMs is compared to other models, namely factorial HMM (FHMM), fully coupled HMM (FCHMM), and Brand's version of coupled HMM (BCHMM). On the synthetic data, consistent HMMs performed better than all other models followed by FHMM, FCHMM and BCHMM. Although FCHMM is an ideal model for the data depicted in the synthetic data, it did not perform as well as consistent HMM or FHMM due to a small number of training samples. BCHMM trains in fully coupled state space and tries to capture the knowledge in a state space that is less than the FCHMM but more than the FHMM. It captures binary dependencies in addition to the parameters learned by

FHMM. However, the dependencies in the synthetic data are ternary and BCHMM fails in capturing these dependencies as binary dependencies.

As can be seen from the experiments, although a FCHMM is an ideal model, it requires enormous amounts of training data. A FHMM can't model dependencies and a BCHMM can only model binary dependencies. Modeling real world involves a large number of labels and dependent processes. None of the FCHMM, FHMM and BCHMM are the ideal models to use in real world due to their inherent disadvantages as already discussed. Consistent HMM is an ideal HMM to model the processes in real world, as it can model dependencies without exploding the state space. All it requires is the binary knowledge of the consistent state combinations.

5.2. Future Work

The experiments in this thesis were performed on synthetic data. Although the consistent HMM performed better than the other models on the synthetic data, the performance should be evaluated on the real data. Real data has more states and interesting dependencies that are hard to integrate in synthetic data. It is interesting to see how consistent HMM fares with other models on real data. VIRAT video dataset [26] is being considered to compare the performance of consistent HMM with other models. Compared to existing action recognition datasets, VIRAT video dataset is more natural, diverse and challenging.

The algorithms proposed in this thesis requires knowledge of what states are consistent and what states are not. We assumed that this knowledge is already available. This knowledge can also be built manually based on the human knowledge of the consistent and inconsistent states. However, this knowledge is available in various natural language ontological databases discussed in section 2.5. But the knowledge in this databases is not in

the binary form as required by the proposed algorithms. So, the method to extract the knowledge from these ontological databases in the binary form must be investigated.

The proposed extended Baum-Welch algorithm and extended Viterbi algorithms integrate the consistency constraints through an optimization step. However, this optimization step is performed once the models are initiated in Baum-Welch algorithm, and after each step in Viterbi algorithm. The optimization step is not integrated right in the heart of these algorithms. The way to integrate the optimization step into the Baum-Welch and Viterbi algorithms is another interesting thing to explore.

The consistent HMM projects a point in cartesian state space, where no chains are independent, to factorial state space, where all chains are independent. In addition to projecting a point to a fully independent factorial state space, some strong dependencies between chains can be captured as done in Semi-Bayes classifier. Designing a semi-consistent HMM in lieu with semi-Bayes classifier is another area to explore.

The consistent HMM integrates the consistency constraints in the cartesian states and makes a projection into the factorial state space by minimizing the information loss. However, it is not possible to integrate the information in the cartesian state space into the small factorial state space. So, it is noteworthy to see if there exists a model between these two extremes that can capture the information in the cartesian state space, while not making the training step untenable.

The optimization step used in the proposed algorithms minimizes the L2 norm between a point in factorial state space and the projection of a point in cartesian state space with consistency constraints. However, Juang and Rabiner proposed that the entropy difference between two probabilistic models is an effective distance measure than L2 norm [22]. So, the

optimization step can be improved by minimizing the Kullback-Liebler divergence between two probabilistic models rather than minimizing the L2 norm.

Bibliography

[1] Collin F Baker, Charles J Fillmore, and John B Lowe. The berkeley framenet project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics, 1998. 5, 28

[2] Andrei Barbu, Aaron Michaux, Siddharth Narayanaswamy, and Jeffrey Mark Siskind. Simultaneous object detection, tracking, and event recognition. *Advances in Cognitive Systems*, 2:203–220, 2012. 29

[3] Leonard E Baum. An equality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8, 1972. 13

[4] Leonard E Baum and JA Eagon. An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bull. Amer. Math. Soc*, 73(3):360–363, 1967. 12, 13

[5] Leonard E. Baum and George R. Sell. Growth transformations for functions on manifolds. *Pacific Journal of Mathematics*, 27(2):211–227, 1968. URL http://projecteuclid.org/euclid.pjm/1102983899. 15

[6] Leonard E Baum and George R Sell. Growth transformations for functions on manifolds. *Pacific J. Math*, 27(2):211–227, 1968. 12, 13

[7] Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, pages 164–171, 1970. 15

[8] David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. Visual object tracking using adaptive correlation filters. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2544–2550. IEEE, 2010. 2

[9] Matthew Brand. Coupled hidden markov models for modeling interacting processes, 1997. 7, 22, 25

[10] Matthew Brand, Nuria Oliver, and Alex Pentland. Coupled hidden markov models for complex action recognition. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 994–999. IEEE, 1997. 22, 24

[11] Thomas Brox and Jitendra Malik. Object segmentation by long term analysis of point trajectories. In *Computer Vision–ECCV 2010*, pages 282–295. Springer, 2010. 1

[12] Olivier Cappé, Eric Moulines, and Tobias Rydén. *Inference in hidden Markov models*, volume 6. Springer, 2005. 8

[13] Henning Christiansen, Christian Theil Have, Ole Torp Lassen, and Matthieu Petit. A constraint model for constrained hidden markov models: a first biological application. In *Proceedings of WCB09: Workshop on Constraint Based Methods for Bioinformatics*, 2009. 26

[14] Henning Christiansen, Christian Theil Have, Ole Torp Lassen, and Matthieu Petit. Inference with constrained hidden markov models in prism. *TPLP*, 10(4-6):449–464, 2010. 26

[15] Arthur P Dempster, Nan M Laird, Donald B Rubin, et al. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal statistical Society*, 39(1): 1–38, 1977. 13

[16] Ralf Dragon and Luc Van Gool. Ground plane estimation using a hidden markov model. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE, 2014. 31

[17] Christiane Fellbaum. *WordNet*. Wiley Online Library, 1999. 5, 27

[18] G David Forney Jr. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973. 15

[19] Zoubin Ghahramani. An introduction to hidden markov models and bayesian networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(01):9–42, 2001. 7, 8

[20] Zoubin Ghahramani and Michael I Jordan. Factorial hidden markov models. *Machine learning*, 29(2-3):245–273, 1997. 20

[21] Michael I Jordany, Zoubin Ghahramaniz, and Lawrence K Sauly. Hidden markov decision trees. *Advances in neural information processing systems*, pages 501–507, 1997. 21

[22] B-H Juang and Lawrence R Rabiner. A probabilistic distance measure for hidden markov models. *AT&T technical journal*, 64(2):391–408, 1985. 82

[23] David JC MacKay. Ensemble learning for hidden markov models. *Technical report, Cavendish Laboratory, University of Cambridge*, 1997. 13

[24] Quanyi Mo and Bruce A Draper. Semi-nonnegative matrix factorization for motion segmentation with missing data. In *Computer Vision–ECCV 2012*, pages 402–415. Springer, 2012. 1

[25] Kevin P Murphy. Dynamic bayesian networks. *Probabilistic Graphical Models, M. Jordan*, 2002. 7

[26] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, JK Aggarwal, Hyungtae Lee, Larry Davis, et al. A large-scale benchmark dataset for event recognition in surveillance video. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3153–3160. IEEE, 2011. 81

[27] Stephen O'Hara and Bruce A Draper. Scalable action recognition with a subspace forest. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1210–1217. IEEE, 2012. 4

[28] Nuria M Oliver, Barbara Rosario, and Alex P Pentland. A bayesian computer vision system for modeling human interactions. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):831–843, 2000. 25

[29] Bo Peng and Gang Qian. Online gesture spotting from visual hull data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(6):1175–1188, 2011. 31

[30] Bo Peng, Gang Qian, and Stjepan Rajko. View-invariant full-body gesture recognition from video. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–5. IEEE, 2008. 31

[31] Harry Plantinga and Charles R Dyer. Visibility, occlusion, and the aspect graph. *International Journal of Computer Vision*, 5(2):137–160, 1990. 3

[32] Ariadna Quattoni, Sybor Wang, Louis-Phillipe Morency, Michael Collins, Trevor Darrell, and Mit Csail. Hidden-state conditional random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(10):1848–1852, 2007. 32

[33] Lawrence Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989. 7

[34] Stjepan Rajko and Gang Qian. Hmm parameter reduction for practical gesture recognition. In *Automatic Face & Gesture Recognition, 2008. FG'08. 8th IEEE International Conference on*, pages 1–6. IEEE, 2008. 31

[35] Iead Rezek, Peter Sykacek, and Stephen J Roberts. Learning interaction dynamics with coupled hidden markov models. *IEE Proceedings-Science, Measurement and Technology*, 147(6):345–350, 2000. 22, 24

[36] Iead Rezek, Michael Gibbs, and Stephen J Roberts. Maximum a posteriori estimation of coupled hidden markov models. *Journal of VLSI signal processing systems for signal, image and video technology*, 32(1-2):55–66, 2002. 22, 24

[37] Sam T Roweis. Constrained hidden markov models. In *NIPS*, pages 782–788, 1999. 26

[38] Josef Ruppenhofer, Michael Ellsworth, Miriam RL Petruck, Christopher R Johnson, and Jan Scheffczyk. Framenet ii: Extended theory and practice, 2006. 28

[39] Lawrence K Saul and Michael I Jordan. Boltzmann chains and hidden markov models. *Advances in neural information processing systems*, pages 435–442, 1995. 21

[40] Karin Kipper Schuler. Verbnet: A broad-coverage, comprehensive verb lexicon. 2005. 5, 28

[41] N Siddharth, Andrei Barbu, and Jeffrey Mark Siskind. Seeing what you're told: Sentence-guided activity recognition in video. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE, 2014. 29

[42] Chris Stauffer and W Eric L Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2. IEEE, 1999. 1

[43] Kevin Tang, Li Fei-Fei, and Daphne Koller. Learning latent temporal structure for complex event detection. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1250–1257. IEEE, 2012. 32

[44] Andrew J Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269, 1967. 15

[45] Maggie Wigness, Bruce A Draper, and J Ross Beveridge. Selectively guiding visual concept discovery. 2014. 3

[46] Shi Zhong and Joydeep Ghosh. A new formulation of coupled hidden markov models. Technical report, Tech. Report, Dept. of Electronic and Computer Engineering, U. of Texas at Austin, USA, 2001. 22, 25

APPENDIX A

Extended Viterbi Algorithm

Algorithm A.1 Viterbi Algorithm Initialization

Input: Decoupled states: S;
Cartesian states: $S^{\otimes}$;
Vector of Prior Probabilities $\bar{\pi}$;

Output: An array that projects a vector from cartesian state space to decoupled state space: ρ;
An array to maintain the max probability path score to end at very state: $SEQSCORE$;
An array to recover the path of most likely state sequence: $BACKPTR$;

Initialize a ρ array of size $|S| \times |S^{\otimes}|$
for $i = 1$ to S **do**
  for $j = 1$ to $|S^{\otimes}|$ **do**
    if $S_i \in S_j^{\otimes}$ **then**
      $\rho[i][j] = 1$
    else
      $\rho[i][j] = 0$
    end if
  end for
end for
Initialize a $SEQSCORE$ array of size $|S| \times T$
Initialize another $BACKPTR$ array of size $|S| \times T$
for $c = 1$ to C **do**
  for $i = 1$ to $|S^{(c)}|$ **do**
    j = decoupledStateIndex($S_i^{(c)}$)
    $SEQSCORE[j, 1] = \pi_i^{(c)}$
    $BACKPTR[j, 1] = 0$
  end for
end for
return ρ, $SEQSCORE, BACKPTR$

Algorithm A.2 Viterbi Step

Input: $SEQSCORE$; $BACKPTR$; λ_κ
Output: $SEQSCORE, BACKPTR$

for $c = 1$ to C **do**
  for $i = 1$ to $|S^{(c)}|$ **do**
    m = decoupledStateIndex($S_i^{(c)}$)
    $SEQSCORE[m, t] = 0$
    for $j = 1$ to $|S^{(c)}|$ **do**
      k = decoupledStateIndex($S_j^{(c)}$)
      l = observationIndex($O_t^{(c)}$)
      $score = SEQSCORE[k, t-1] * A^{(c)}[j, i] * B^{(c)}[i, l]$
      if SEQSCORE[m,t]≤score **then**
        SEQSCORE[m,t] = score
        BACKPTR[m,t] = i
      end if
    end for
  end for
end for
return $SEQSCORE, BACKPTR$

Algorithm A.3 Backtracking Step

Input: $SEQSCORE$; $BACKPTR$; C; S
Output: An array with most likely state sequence indices for each chain Q.

Initialize Q array of length $C \times T$, to hold the most likely sequence.
for $c = 1$ to C **do**
  Q[c,T] = 0
  maxScore = 0
  for $i = 1$ to $|S^{(c)}|$ **do**
    j = decoupledStateIndex($S_i^{(c)}$)
    if SEQSCORE[j,T] > maxscore **then**
      Q[c,T] = i
      maxScore = SEQSCORE[j,T]
    end if
  end for
  for $t = T - 1$ to 1 **do**
    k = decoupledStateIndex[$S_{Q[t+1]}^{(c)}$]
    Q[c,t] = BACKPTR[k,t+1]
  end for
end for
return Q

APPENDIX B

Extended Baum-Welch Algorithm

Algorithm B.1 Baum-Welch

Input: $Model \quad \lambda$
$ObservationSequenceList$
Output: $TrainedModel \quad \bar{\lambda}$

while Model λ not converged **do**
  Initialize $\Xi[i][j][t]$, a matrix of length $|S| \times |S| \times T$ to hold transition probabilities of all observation sequences
  Initialize $\Gamma[i][t]$, a matrix of length $|S| \times T$ to hold probabilities of starting at a state of all observation sequences
  for ObservationSequence O in ObservationSequenceList **do**
    α = ForwardProbability(λ,O)
    β = BackwardProbability(λ,O)
    $P(O|\lambda) = \sum_{i=1}^{|S|} \alpha[i][T]$
    Initialize $\xi[i][j][t]$, a matrix of length $|S| \times |S| \times T$ to hold transition probabilities for each time step
    for $i = 1$ to $|S|$ **do**
      for $j = 1$ to $|S|$ **do**
        for $t = 1$ to T **do**
          $\xi[i][j][t] = \dfrac{\alpha[i][t] * A[i][j] * B[j][O_{t+1}] * \beta[j][t+1]}{P(O|\lambda)}$
        end for
      end for
    end for
    Initialize $\gamma[i][t]$, a matrix of length $|S| \times T$ to hold probabilities of starting at a state for each time step
    for $i = 1$ to $|S|$ **do**
      for $t = 1$ to T **do**
        $\gamma[i][t] = \sum_{j=1}^{|S|} \xi[i][j][t]$
      end for
    end for
    $\Xi = \Xi + \xi$
    $\Gamma = \Gamma + \gamma$
  end for
  $\bar{\pi} = \Gamma[:][1]$
  $\bar{A} = \dfrac{\sum_{t=1}^{T-1} \Xi}{\sum_{t=1}^{T-1} \Gamma}$
  $\bar{B} = \dfrac{\sum_{s.t.O_t=v_k}^{T} \Gamma}{\sum_{t=1}^{T} \Gamma}$
end while
return $\bar{\lambda}$

Algorithm B.2 Forward Probability

Input: $Model \quad \lambda$
$ObservationSequence \quad O$
Output: $ForwardProbabilityMatrix \quad \alpha$

Initialize α array of length $|S| \times T$, to hold the forward probability of each state at every time step.
$\alpha[:, 1] = \pi \odot B[:, O_1]$
for $t = 2$ to T **do**
  for $i = 1$ to $|S|$ **do**
    $\alpha[i][t] = \left[\sum_{j=1}^{|S|} \alpha[j][t-1] * A[i][j]\right] * B[i][O_t]$
  end for
end for
return α

Algorithm B.3 Backward Probability

Input: $Model \quad \lambda$
$ObservationSequence \quad O$
Output: $BackwardProbabilityMatrix \quad \beta$

Initialize β array of length $|S| \times T$, to hold the backward probability of each state at every time step.
$\beta[:, T] = 1$
for $t = T - 1$ to 1 **do**
  for $i = 1$ to $|S|$ **do**
    $\beta[i][t] = \sum_{j=1}^{|S|} \beta[j][t+1] * A[i][j] * B[j][O_{t+1}]$
  end for
end for
return β

APPENDIX C

Experiment 1: Additional Plots of Performance of Integrating the Consistency Constraints only during Run-Time

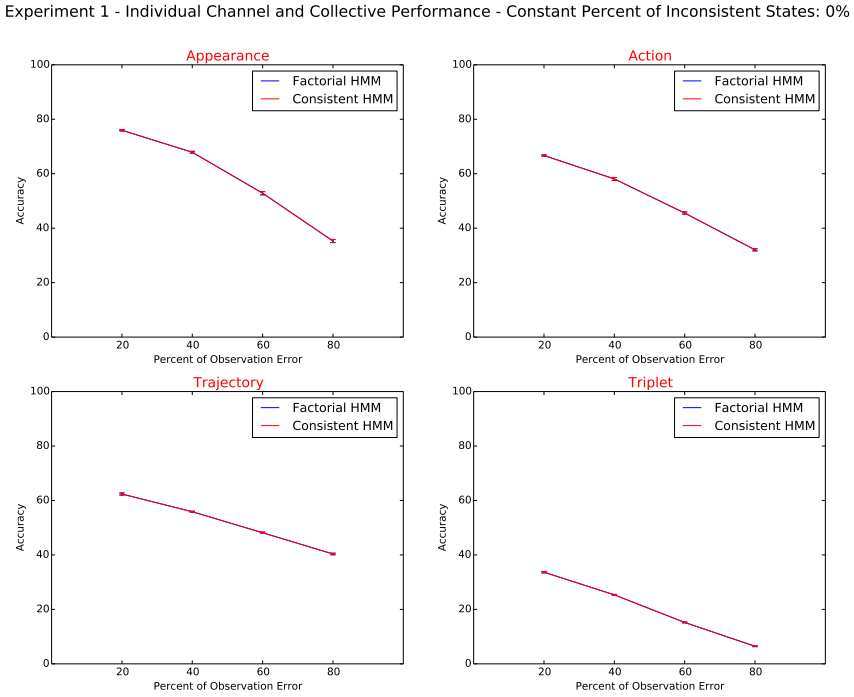

FIGURE C.1. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 0%

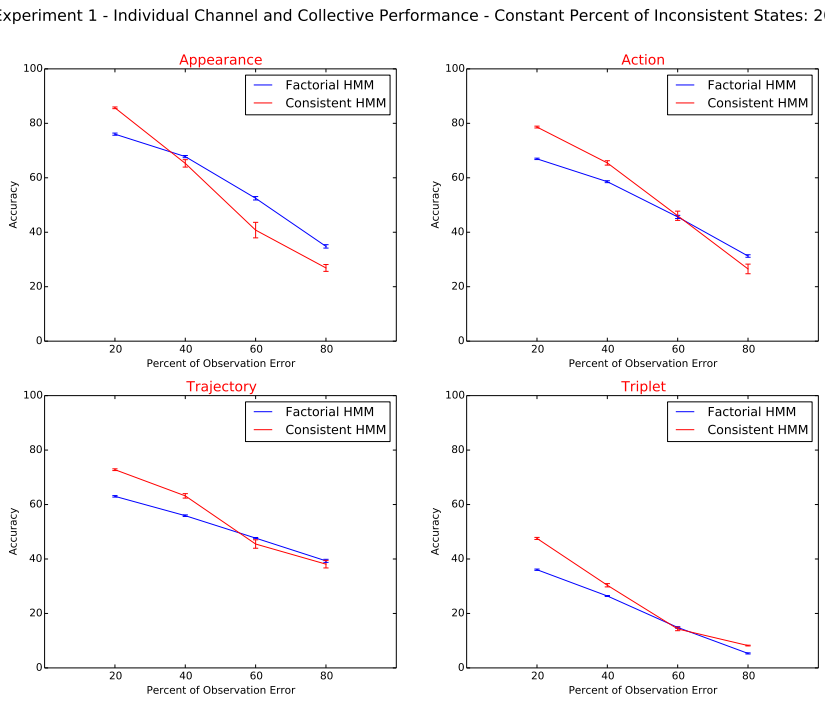

FIGURE C.2. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 20%

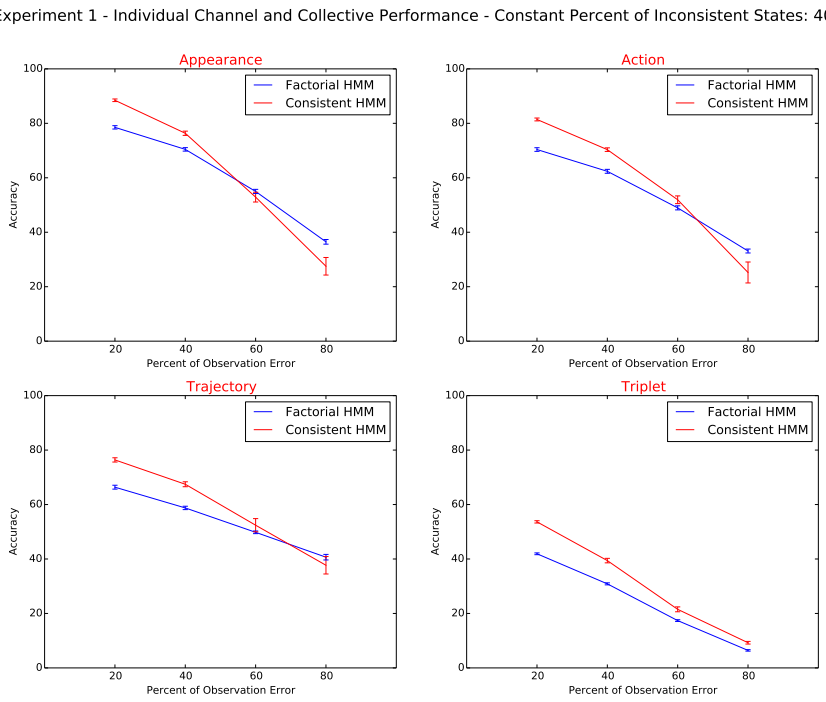

FIGURE C.3. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 40%

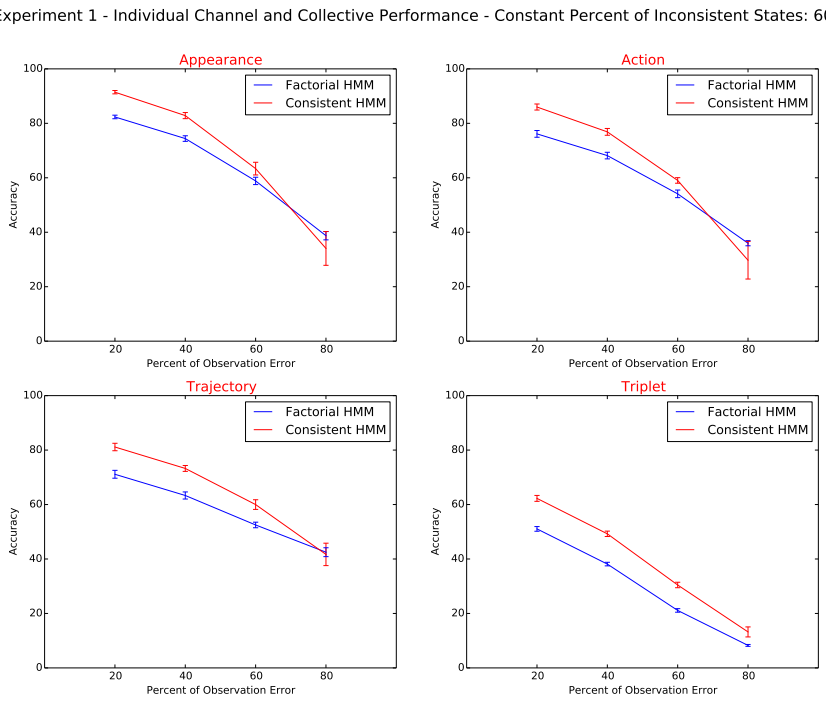

FIGURE C.4. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 60%

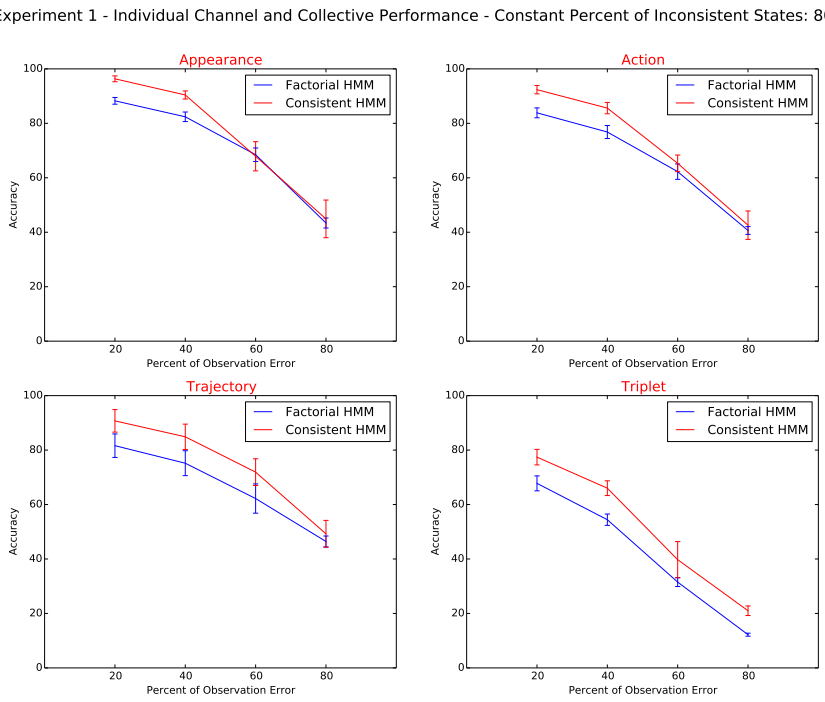

FIGURE C.5. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 80%

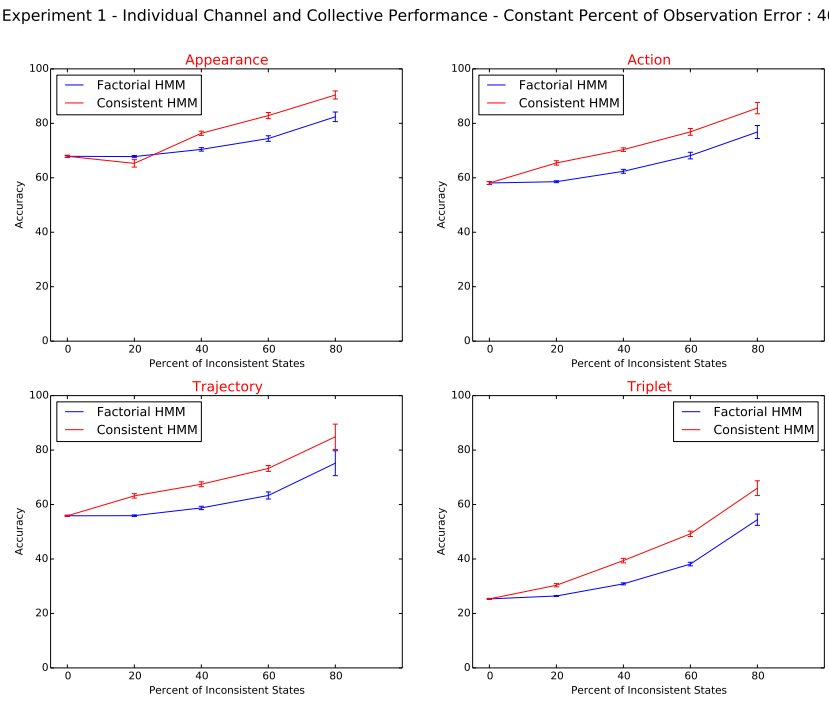

FIGURE C.6. Plots of Inconsistent states percentage vs accuracy for appearance, action, trajectory, and triplet for an observation error of 40%

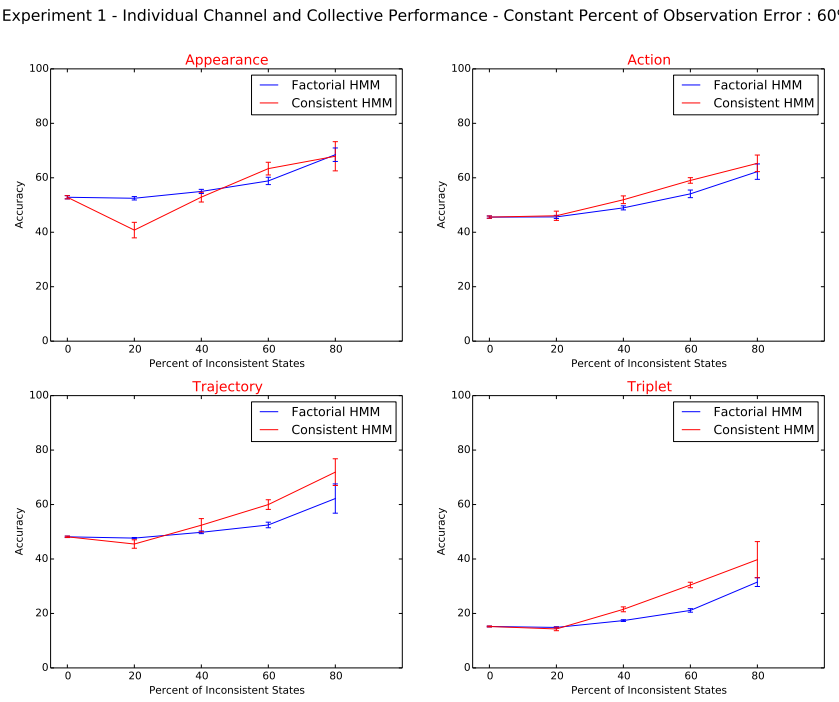

FIGURE C.7. Plots of Inconsistent states percentage vs accuracy for appearance, action, trajectory, and triplet for an observation error of 60%

APPENDIX D

Experiment 2: Additional Plots of Performance of Integrating Consistency Constraints only during Training

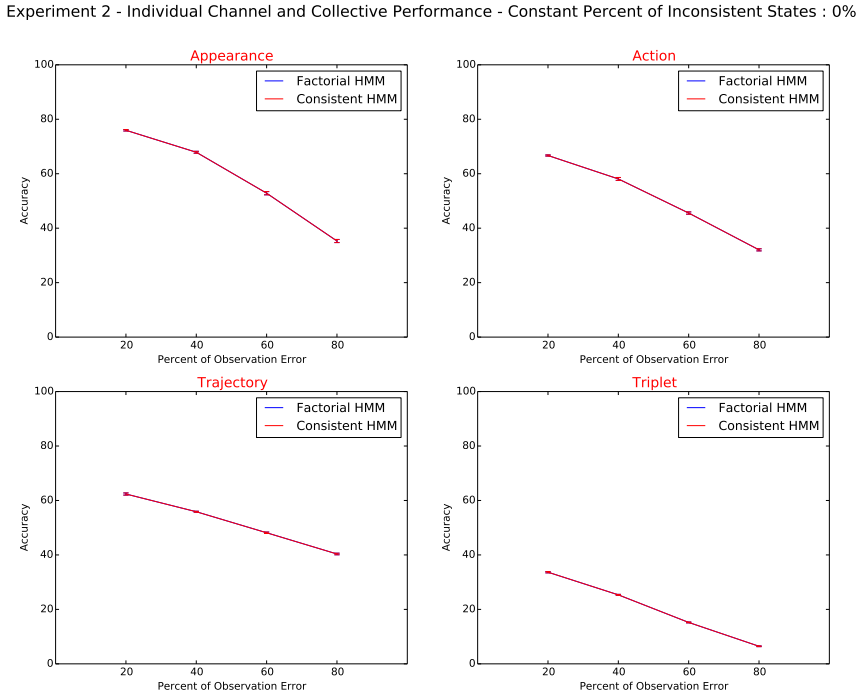

FIGURE D.1. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 0%

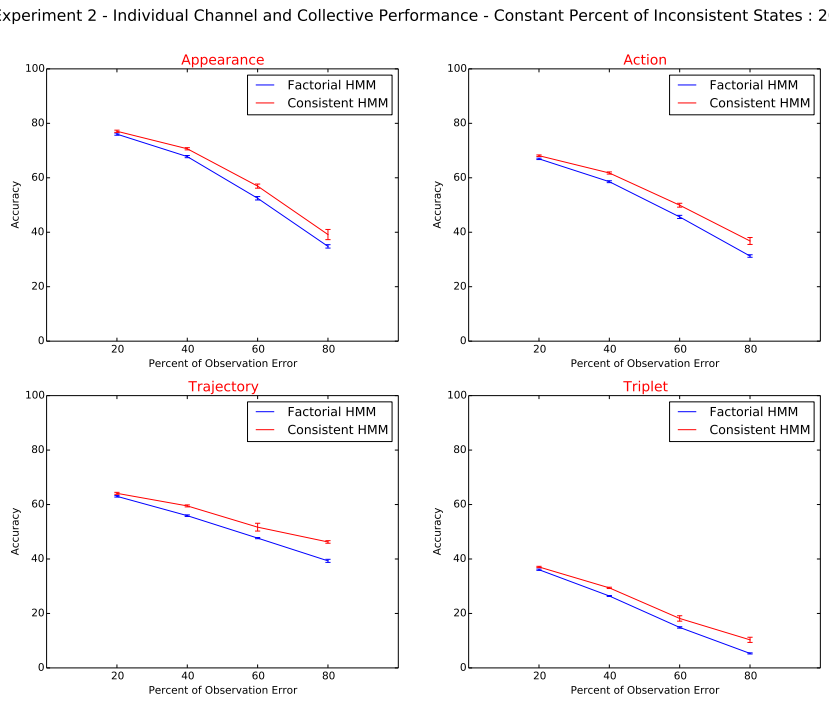

FIGURE D.2. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 20%

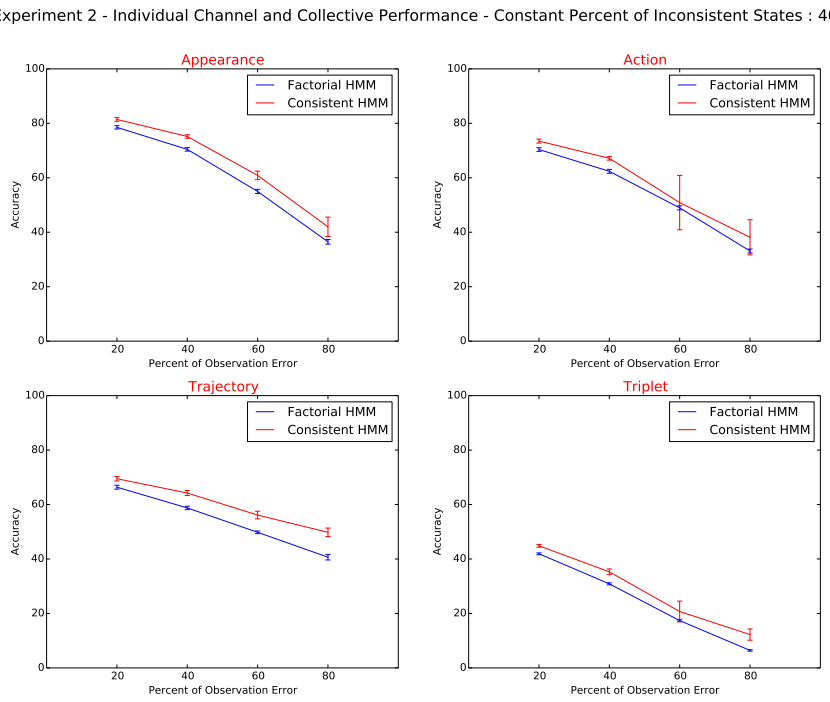

FIGURE D.3. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 40%

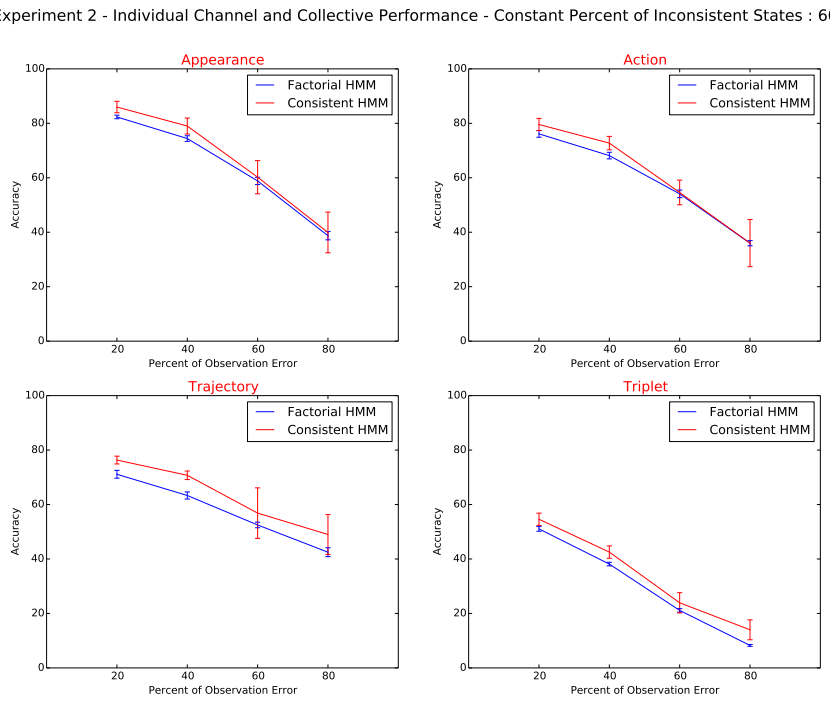

FIGURE D.4. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 60%

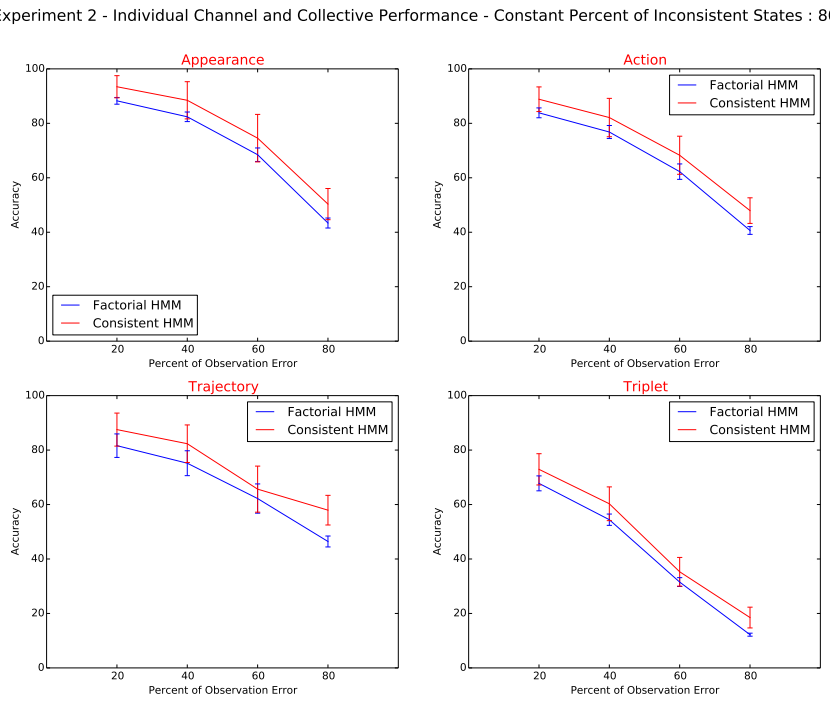

FIGURE D.5. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 80%

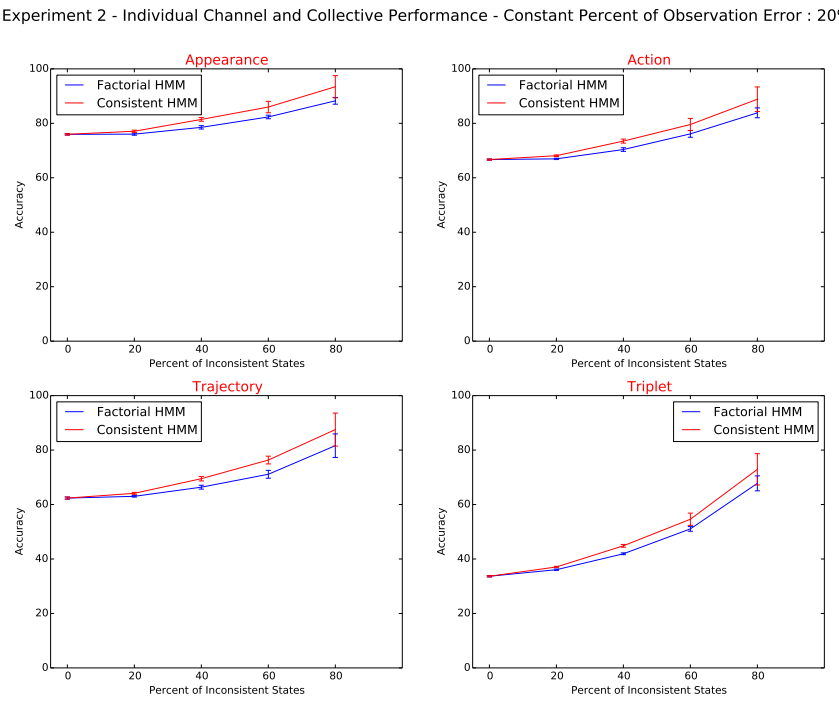

FIGURE D.6. Plots of Inconsistent states percentage vs accuracy for appearance, action, trajectory, and triplet for an observation error of 20%

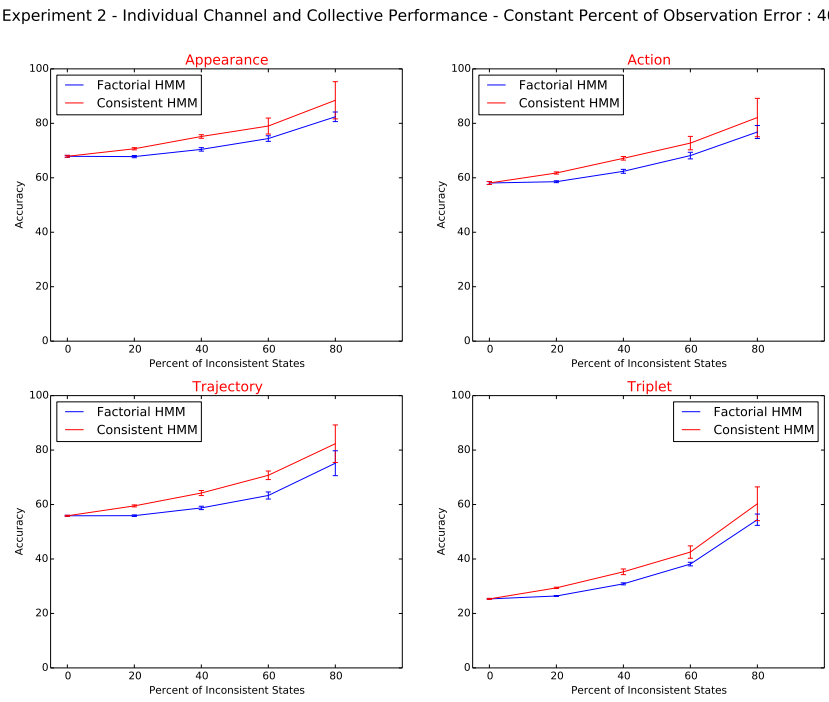

FIGURE D.7. Plots of Inconsistent states percentage vs accuracy for appearance, action, trajectory, and triplet for an observation error of 40%

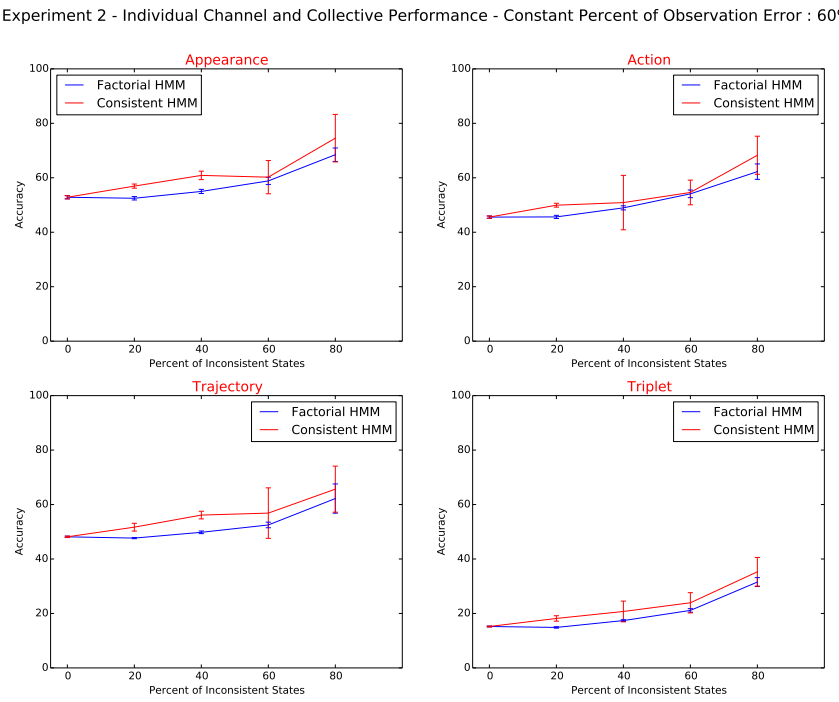

FIGURE D.8. Plots of Inconsistent states percentage vs accuracy for appearance, action, trajectory, and triplet for an observation error of 60%

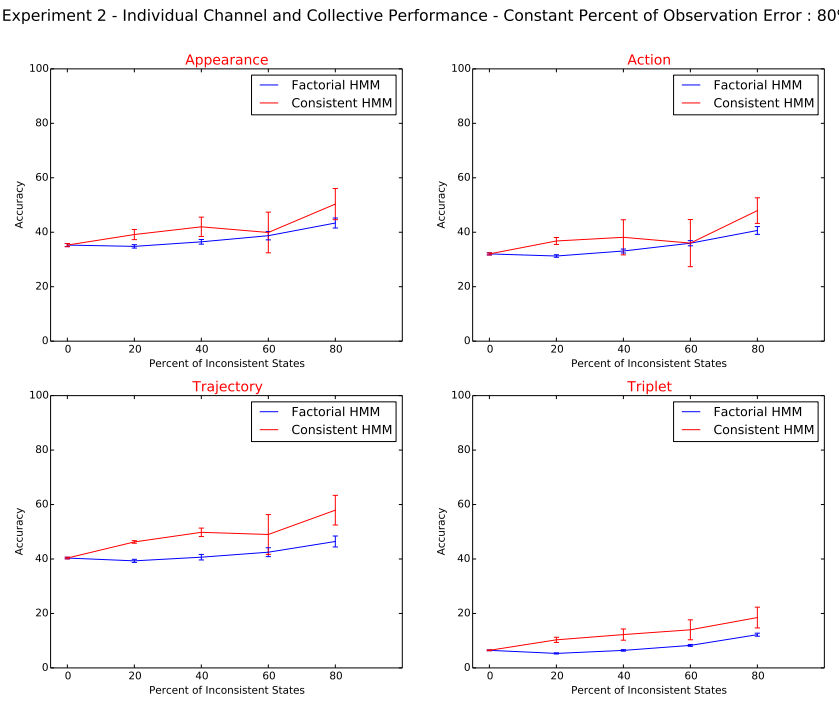

FIGURE D.9. Plots of Inconsistent states percentage vs accuracy for appearance, action, trajectory, and triplet for an observation error of 80%

APPENDIX E

Experiment 3 : Additional Plots of Performance of Integrating Consistency Constraints both during training and run-time

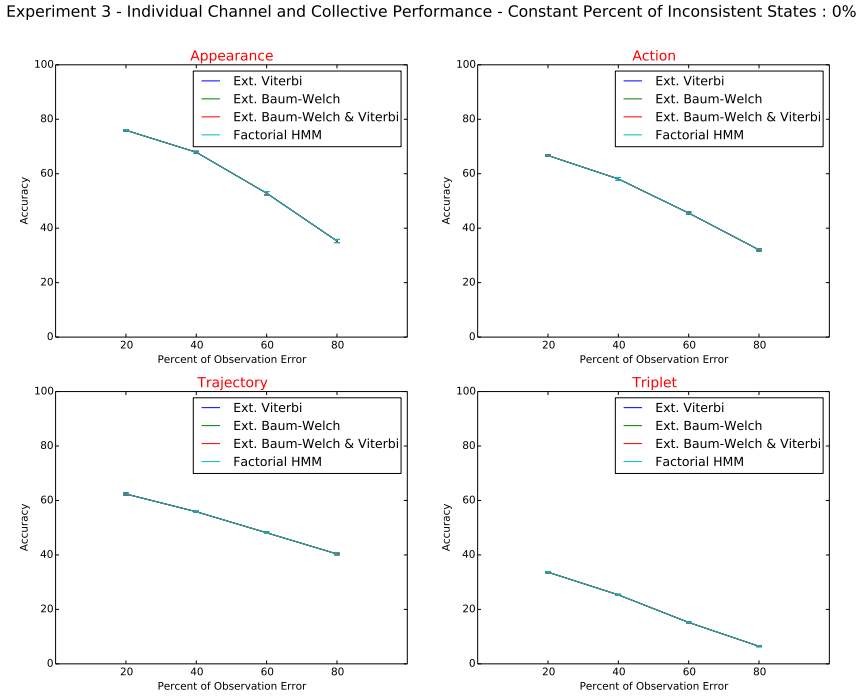

Figure E.1. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 0%

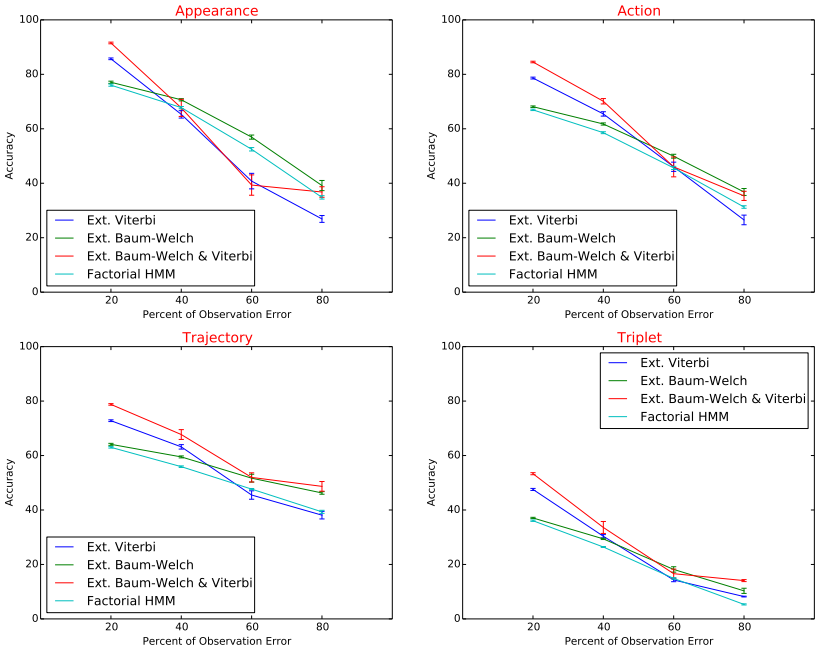

FIGURE E.2. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 20%

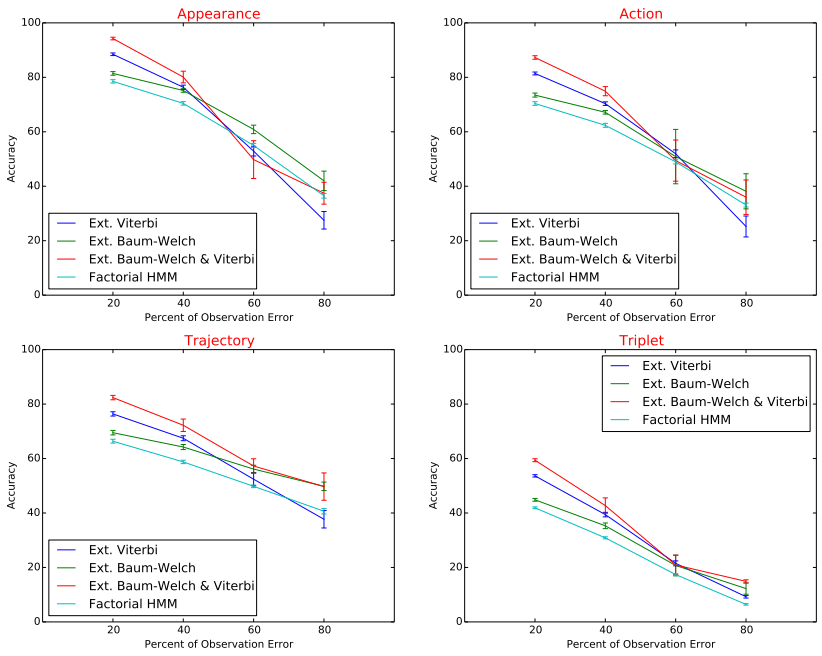

FIGURE E.3. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 40%

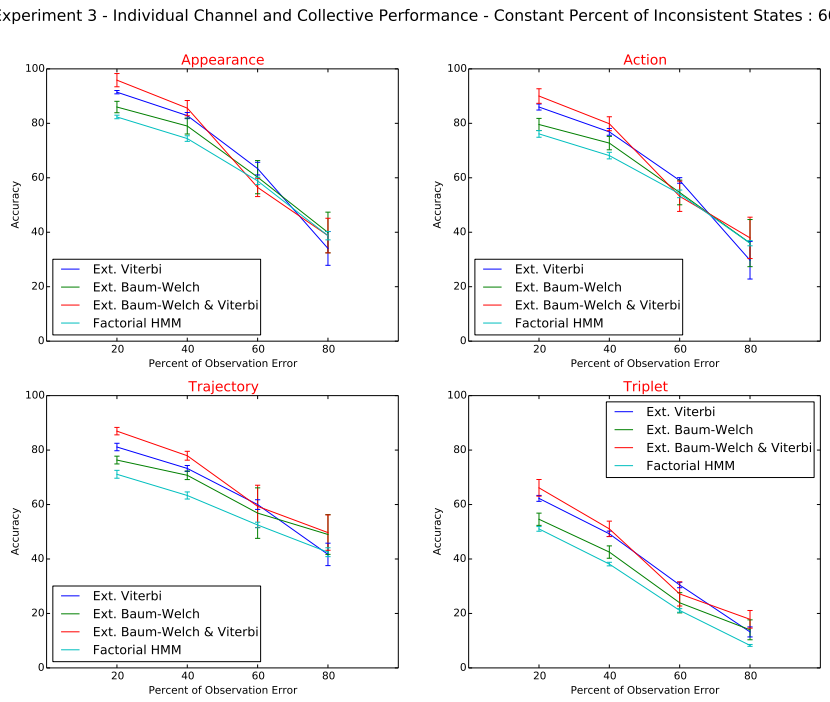

FIGURE E.4. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 60%

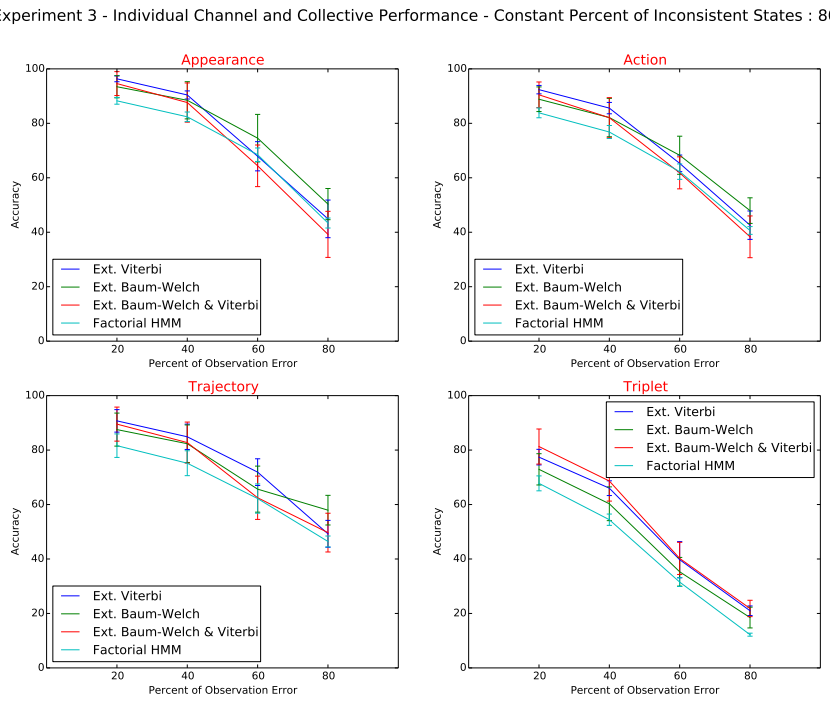

FIGURE E.5. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 80%

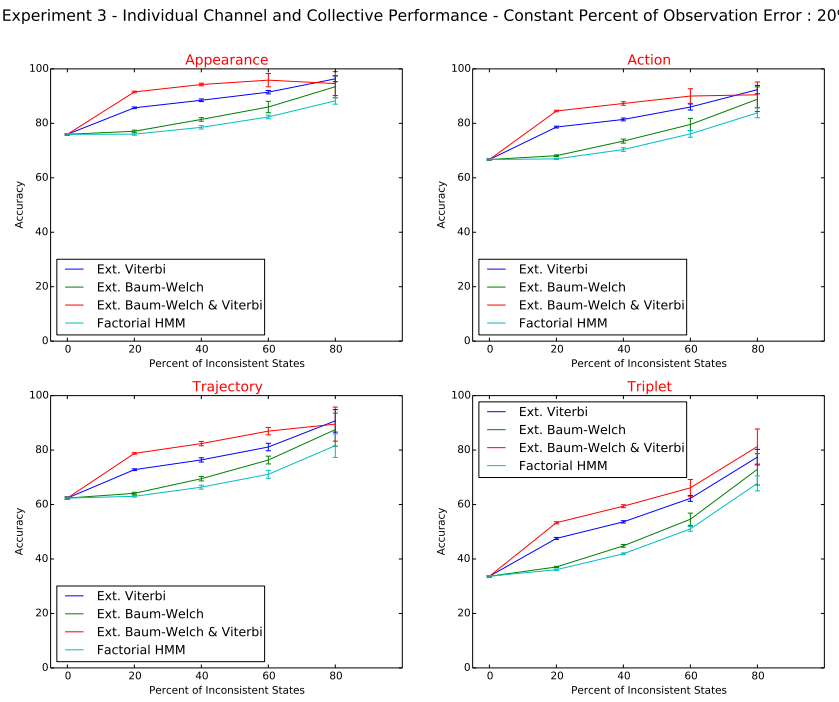

FIGURE E.6. Plots of Inconsistent states percentage vs accuracy for appearance, action, trajectory, and triplet for an observation error of 20%

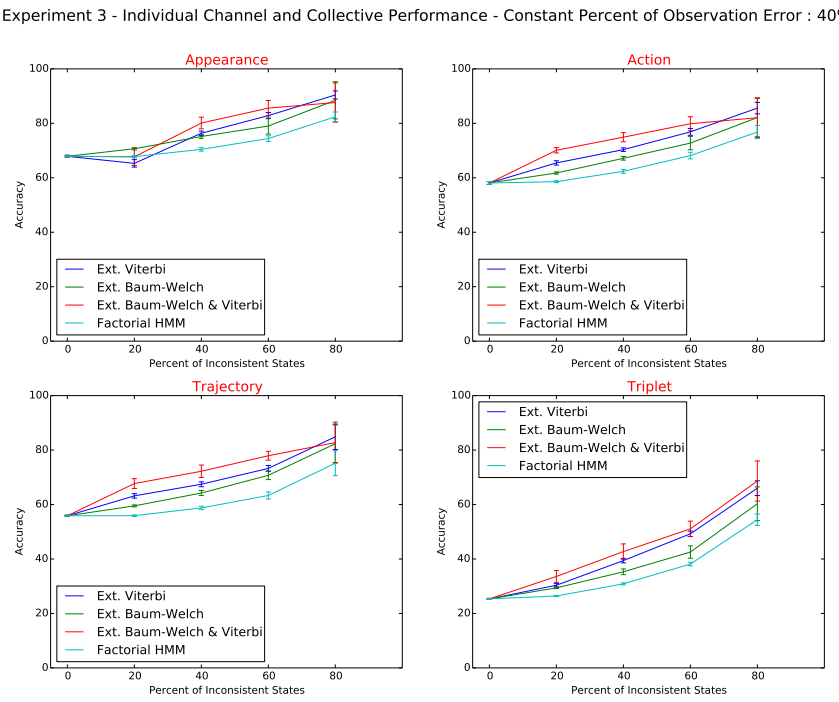

FIGURE E.7. Plots of Inconsistent states percentage vs accuracy for appearance, action, trajectory, and triplet for an observation error of 40%

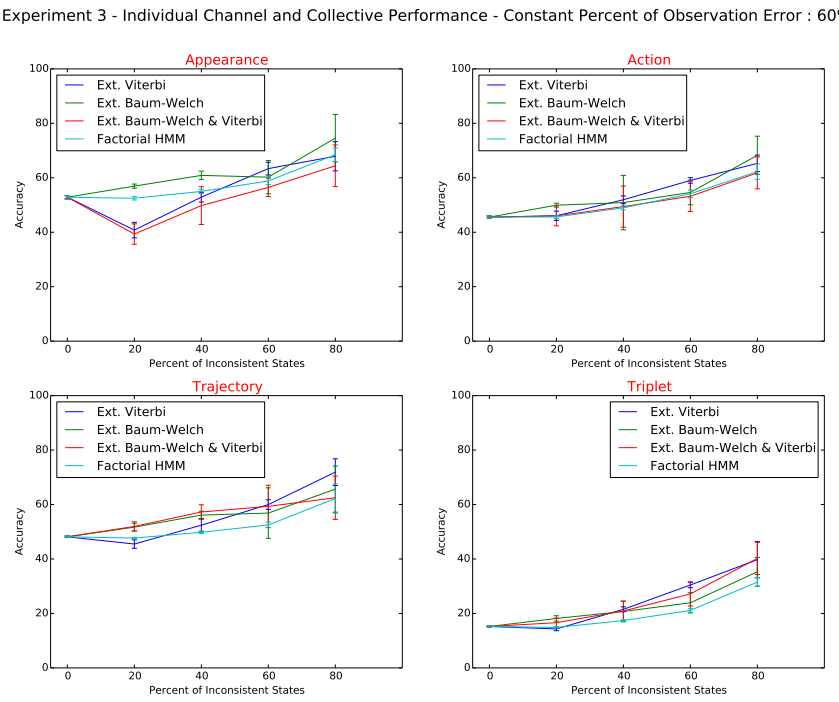

FIGURE E.8. Plots of Inconsistent states percentage vs accuracy for appearance, action, trajectory, and triplet for an observation error of 60%

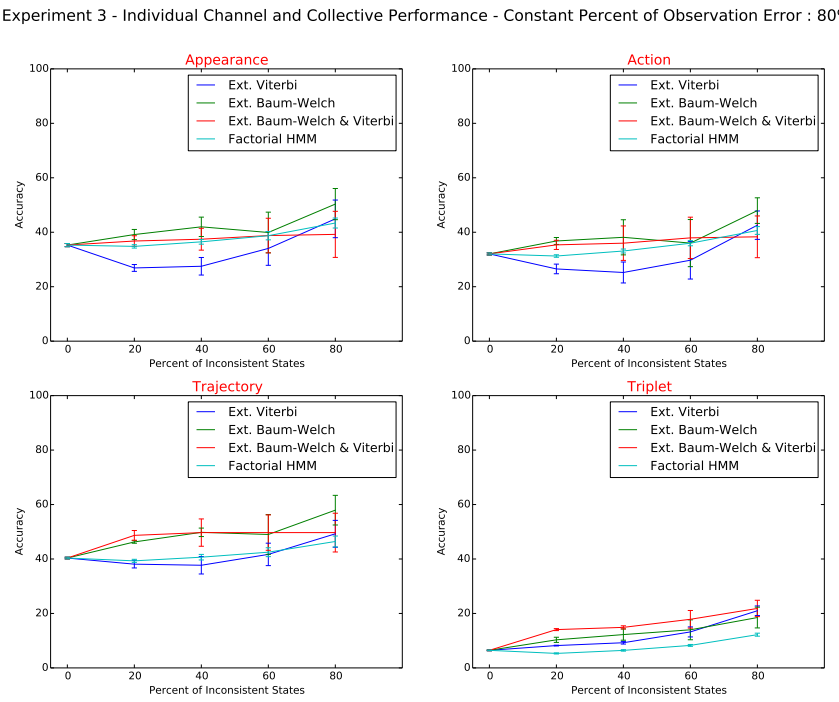

FIGURE E.9. Plots of Inconsistent states percentage vs accuracy for appearance, action, trajectory, and triplet for an observation error of 80%

APPENDIX F

Experiment 4: Additional Plots of Performance of Consistent HMM with respect to FHMM, FCHMM and BCHMM

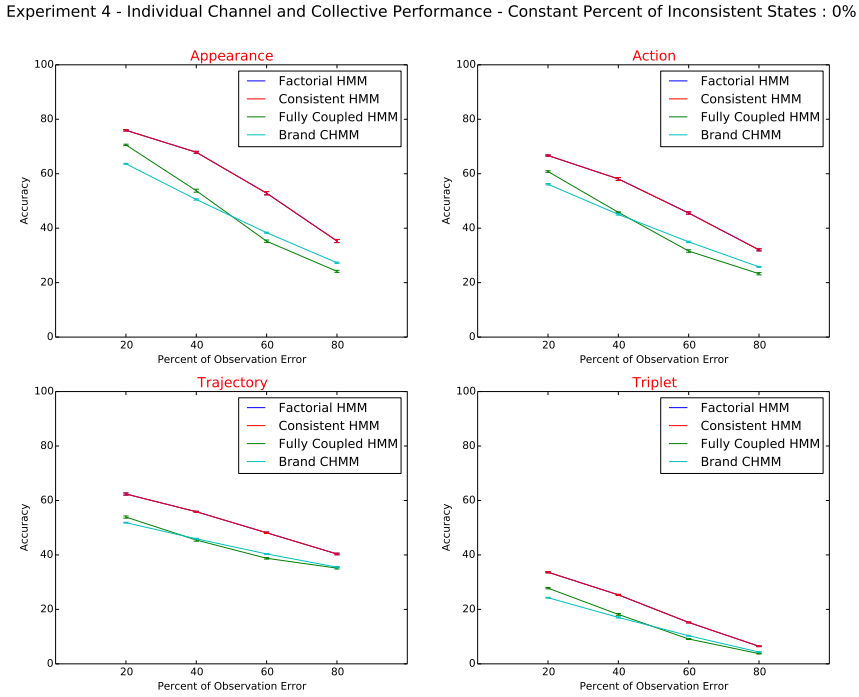

Figure F.1. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 0%

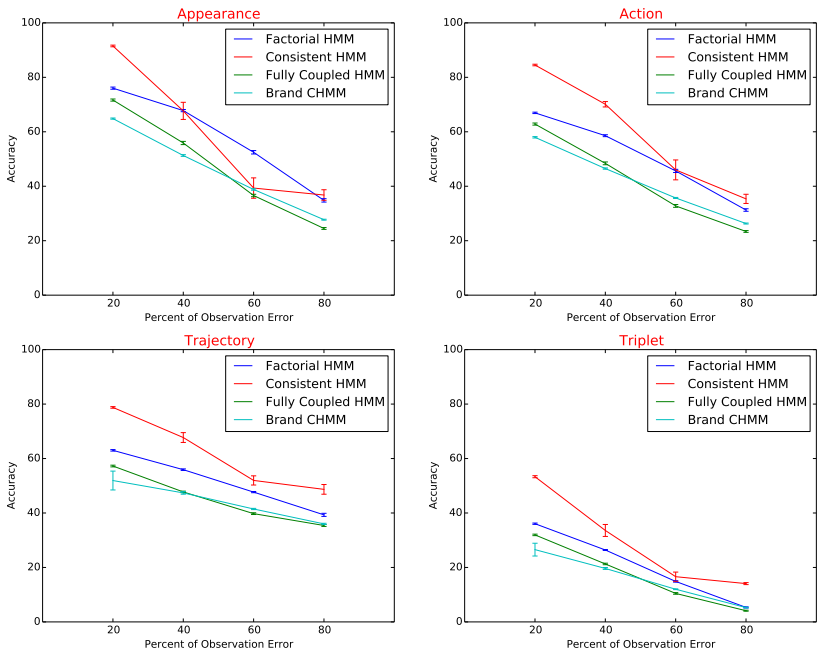

FIGURE F.2. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 20%

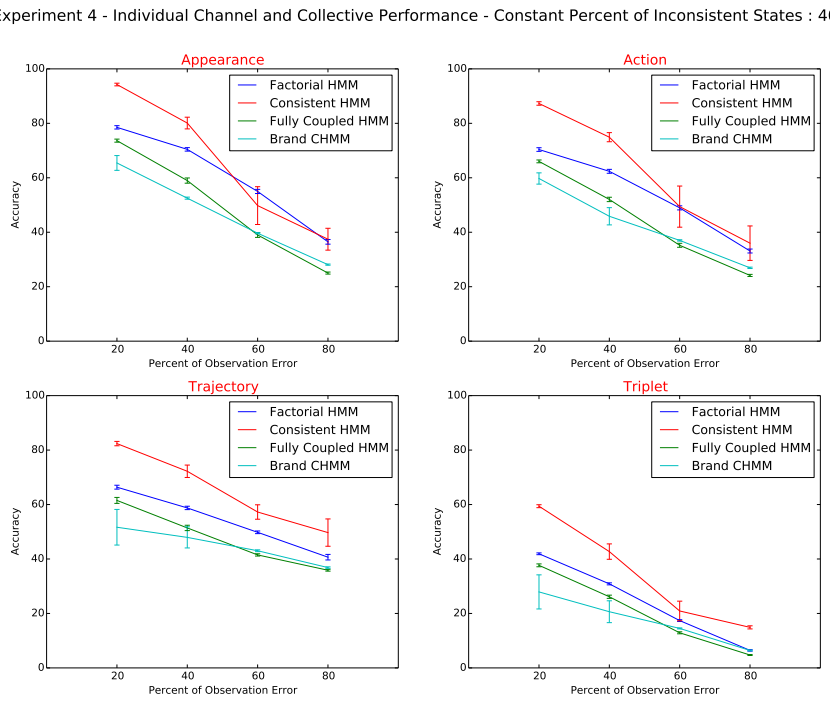

FIGURE F.3. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 40%

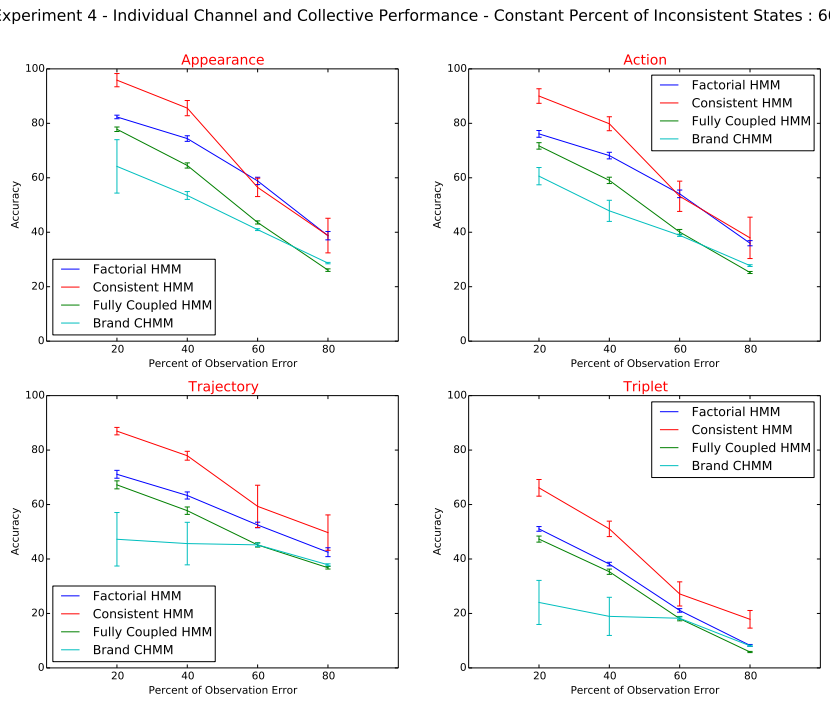

FIGURE F.4. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 60%

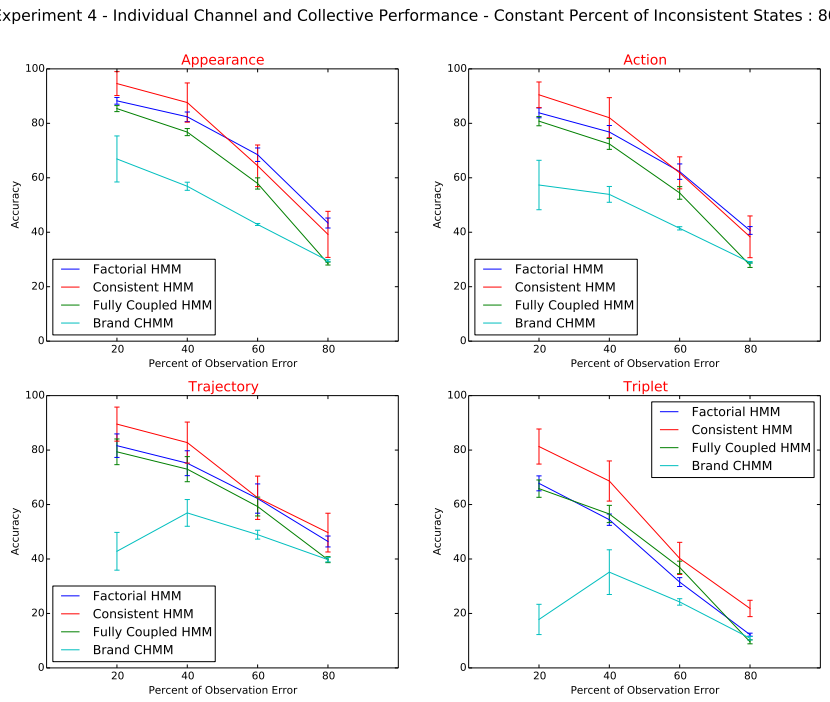

FIGURE F.5. Plots of Observation error percentage vs accuracy for appearance, action, trajectory, and triplet for an inconsistent states of 80%

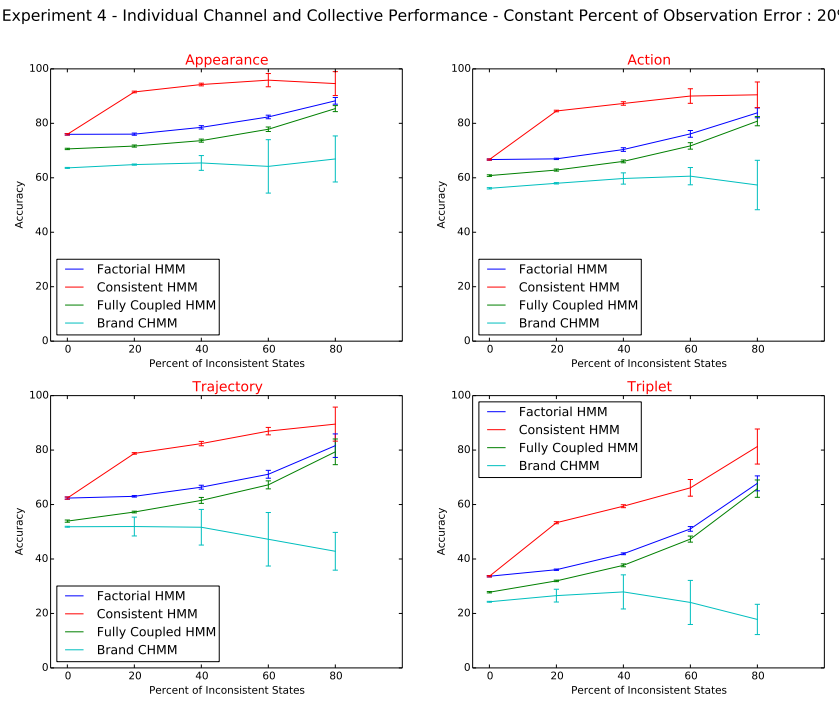

FIGURE F.6. Plots of Inconsistent states percentage vs accuracy for appearance, action, trajectory, and triplet for an observation error of 20%

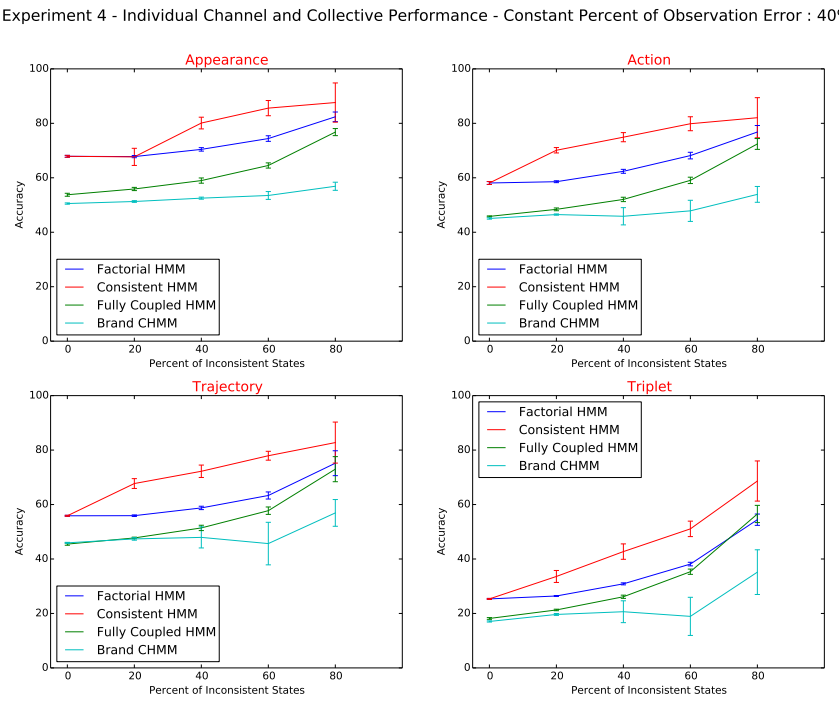

FIGURE F.7. Plots of Inconsistent states percentage vs accuracy for appearance, action, trajectory, and triplet for an observation error of 40%

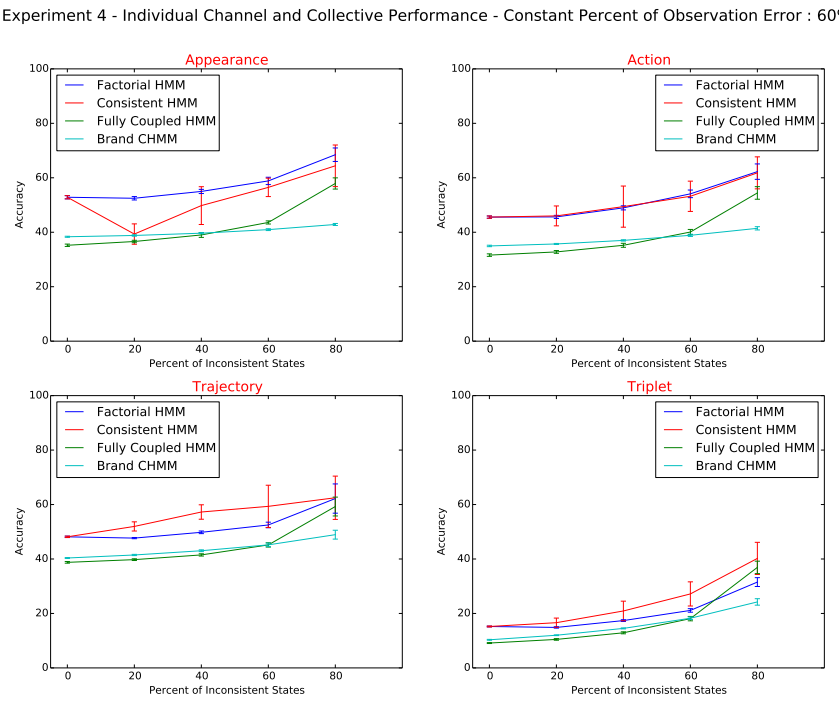

FIGURE F.8. Plots of Inconsistent states percentage vs accuracy for appearance, action, trajectory, and triplet for an observation error of 60%

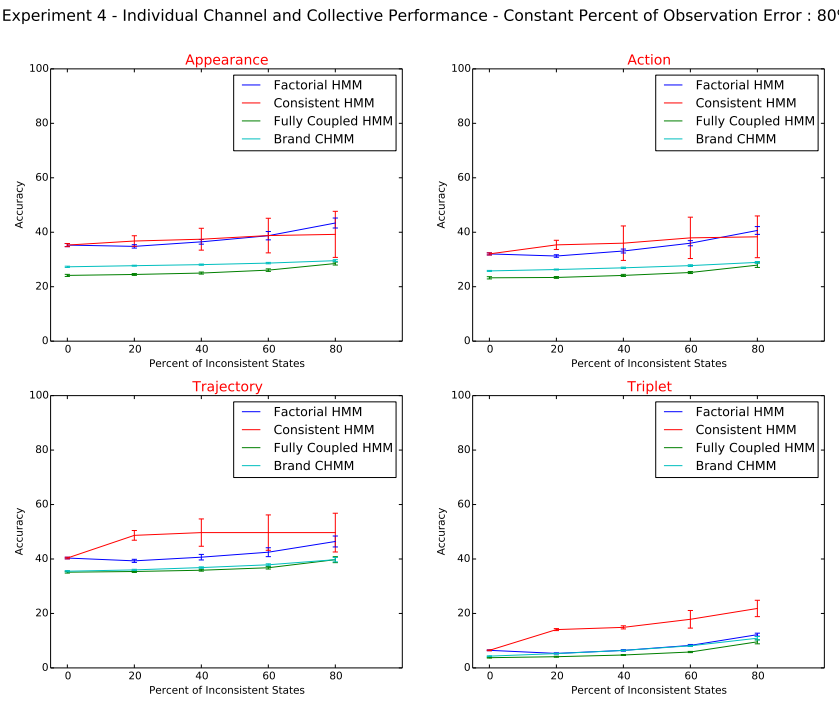

FIGURE F.9. Plots of Inconsistent states percentage vs accuracy for appearance, action, trajectory, and triplet for an observation error of 80%