International Journal of Pattern Recognition
and Artificial Intelligence
Vol. 32, No. 11 (2018) 1855020 (33 pages)

DOI: 10.1142/S0218001418550200

World Scientific
www.worldscientific.com

Interacting Hidden Markov Models for Video Understanding

Pradyumna Narayana*, J. Ross Beveridge† and Bruce A. Draper‡

Department of Computer Science
Colorado State University
Fort Collins, CO 80523, USA
**prady@cs.colostate.edu*
†*ross@cs.colostate.edu*
‡*draper@cs.colostate.edu*

Received 17 August 2016
Accepted 16 April 2018
Published 5 June 2018

People, cars and other moving objects in videos generate time series data that can be labeled in many ways. For example, classifiers can label motion tracks according to the object type, the action being performed, or the trajectory of the motion. These labels can be generated for every frame as long as the object stays in view, so object tracks can be modeled as Markov processes with multiple noisy observation streams. A challenge in video recognition is to recover the true state of the track (i.e. its class, action and trajectory) using Markov models without (a) counterfactually assuming that the streams are independent or (b) creating a fully coupled Hidden Markov Model (FCHMM) with an infeasibly large state space. This paper introduces a new method for labeling sequences of hidden states. The method exploits external consistency constraints among streams without modeling complex joint distributions between them. For example, common sense semantics suggest that *trees cannot walk*. This is an example of an external constraint between an object label ("tree") and an action label ("walk"). The key to exploiting external constraints is a new variation of the Viterbi algorithm which we call the Viterbi–Segre (VS) algorithm. VS restricts the solution spaces of factorized HMMs to marginal distributions that are compatible with joint distributions satisfying sets of external constraints. Experiments on synthetic data show that VS does a better job of estimating true states with the given observations than the traditional Viterbi algorithm applied to (a) factorized HMMs, (b) FCHMMs, or (c) partially-coupled HMMs that model pairwise dependencies. We then show that VS outperforms factorized and pairwise HMMs on real video data sets for which FCHMMs cannot feasibly be trained.

Keywords: Hidden Markov models; video analysis; Segre variety.

1. Introduction

Hidden Markov Models (HMMs) model time series data. When there are multiple streams of evidence, one approach to modeling these streams is to assume that they are independent and model each stream with its own HMM.[22] This approach has the

advantage that the state spaces of the HMMs are relatively small, allowing the HMMs to be trained on modest amounts of data. Unfortunately, the independence assumption is usually not valid. An alternative is to create a single, fully coupled HMM (FCHMM), where the state space is the cartesian product of labels. FCHMMs can model arbitrary dependencies among labels, but unfortunately their state spaces are so large that it is often infeasible to train them without overfitting to the available data. In between these two extremes are hybrid models that represent some but not all possible dependencies.[12,37] These systems have state spaces that are larger than those of independent, factorized HMMs but smaller than those of FCHMMs.

The goal of this paper is to couple multiple HMMs without needing more training data by exploiting *a priori* knowledge about consistent and inconsistent combinations of states. The approach is to train separate HMMs for every label stream, thereby keeping the state spaces small and the need for training data low. At the same time, external information in the form of binary constraint tensors κ indicates which combinations of states can never co-occur. This new source of information is not learned from training samples, but is manually derived from external sources such as ontologies. As the constrained tensor κ integrates knowledge about consistent and inconsistent combinations of states into HMMs, the method requires that the state space is known *a priori* and the states have semantic meaning associated with them.

Systems that model independent HMMs (also known as factorized HMMs) learn marginal distributions, one for each label stream. In contrast, systems that model FCHMMs learn a single, joint distribution across all label streams. Our approach is based on the observation that the space of all joint distributions that can be factored into independent marginal distributions is a Segre variety.[21] Our system finds the distribution in the Segre variety that best describes the data while obeying the consistency constraints. Because this distribution is a Segre variety, it can be factored into independent marginal distributions. This allows us to model each stream independently while still exploiting consistency constraints among states.

The Viterbi–Segre (VS) algorithm infers the most likely sequence of states with the given sequences of observations $\mathcal{O}$, a tensor of constraints κ, and factorized HMMs modeling the individual streams. Conceptually, VS can be described as an extension of the well-known Viterbi algorithm. The Viterbi algorithm produces a vector A of probabilities at every timestep, where $A[i]$ represents the probability of the most likely sequence of states ending in state i. In the case of factorized HMMs, such as the HMMs for object class, action and trajectory considered here, there are multiple probability vectors (one for each label stream) that are concatenated to form A. The VS algorithm constrains A for factorized systems to be consistent with the constraints in κ, even though κ represents dependencies among streams. The details of VS are presented in Sec. 4.3, with the most significant contribution being the minimization problem defined in Eq. (8).

The proposed VS algorithm is evaluated first on synthetic data sets inspired by real-world scenarios, and then on real data from the VIRAT dataset[29] and a newer Classroom data set. Synthetic data allow us to systematically vary the level of label noise and the ratio of consistent to inconsistent combinations of states. It also allows us to measure variance by running multiple trials at fixed levels of noise and consistency. We compare VS to the standard Viterbi algorithm on factorized HMMs, FCHMMs, and a partially coupled HMM proposed by Brand.[13] VS outperforms all three alternatives on the synthetic data across all levels of noise and inconsistency. We then apply VS to the VIRAT video data set and a Classroom video data set and show that it also improves performance on less-constrained, real-world data. It should be noted that the goal of this paper is not to push the state-of-the-art on the VIRAT dataset; that would best be accomplished by improving the motion tracks and features extracted from the videos. Instead, we use previously published tracks and features from VIRAT to demonstrate that VS infers labels more accurately than factorized HMMs.

The rest of this paper is organized as follows. Section 2 provides a motivating example based on video interpretation problems in Computer Vision. Section 3 reviews the existing extensions of HMMs that integrate information from multiple sources. Section 4 introduces a new formulation of HMMs that takes binary constraint knowledge into account and defines the VS algorithm. A detailed description of the synthetic data, VIRAT video dataset, Classroom video dataset and the experimental results are discussed in Sec. 5. Finally, concluding remarks are presented in Sec. 6.

2. Motivating Example

Video interpretation often begins by extracting and tracking independently moving objects, as depicted in Fig. 1. Moving objects represent potential actors of interest, and analyzing their appearance and motion allows us to describe videos in terms of actors, actions, and relations between them. Ultimately, the goal of programs such as DARPA's *Mind's Eye* project is to automatically generate natural language descriptions of unlabeled videos.[18]

In this context, motion tracks are the primary target of analysis. Motion tracks are created when a moving object is detected, and continue for as long as the moving object can be tracked. For example, a person might be detected when they walk into the scene, and tracked until they become occluded, leave the field of view, or the tracker fails. For each frame in a track, a region of interest (ROI) specifies a small image chip that is centered and scaled on the moving object, as shown in the bottom row of Fig. 1. Each track can therefore be thought of as its own short, small movie, centered on a single actor and with relative positions to other tracks. The video as a whole is then analyzed in terms of these tracks.

Tracks can be labeled in many ways. The most common goal is to recognize the type of object being tracked, for example whether it is a person, a car, or just a tree

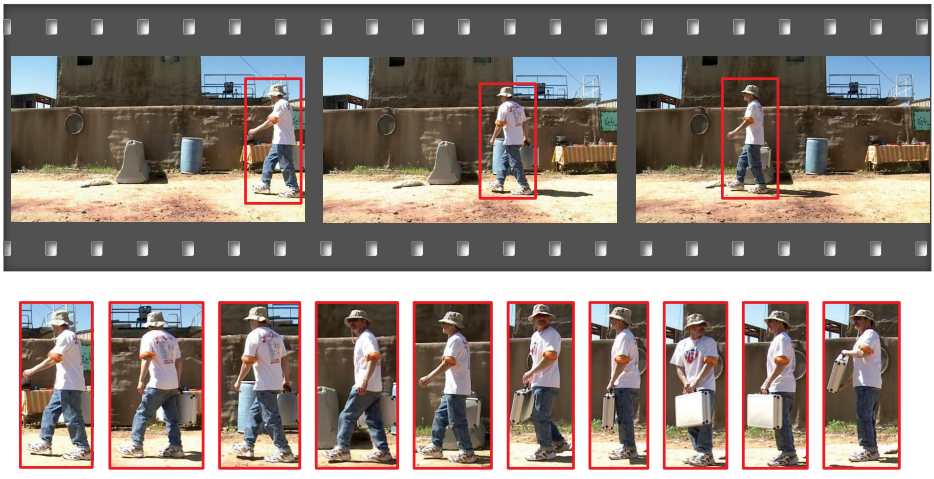

Fig. 1. The top row shows three frames from a video, with the ROIs corresponding to a moving object outlined in red. The bottom row shows the expanded versions of 10 ROIs from the same track. Object tracks are the primary target of analysis in video interpretation, with the goal of recognizing object classes (in this case person), actions (walking) and trajectories (to the left) (color online).

swaying in the wind. Many classifiers have been developed that analyze image chips and return the label of the most likely object class,[20,27] and these labels generally correspond to nouns, e.g. *person* or *vehicle*. Action recognition is another goal leading to another set of labels. Techniques have been developed to label still images[11,32] and brief video snippets[25,30,36,42] according to the actions they portray. In this case, the labels correspond to verbs rather than nouns, for example *walk*, *stand* or *sway*. Yet another goal is to label tracks according to their position or relative positions,[35] such as when a track reaches a doorway or when two tracks meet. In this case, the classifiers analyze the track trajectories and the labels correspond to prepositional phrases, such as *at X* or *toward Y*. There are, of course, still other ways to categorize and label object tracks, but we focus on these three types of labels — object labels, action labels and trajectory labels — since they provide nouns, verbs and prepositions that can be used to generate natural language descriptions of videos.

Object tracks are examples of time series data with multiple, noisy label streams. As already mentioned, one approach to managing noise in label streams is to assume that the labeling processes are independent of each other and use HMMs to smooth each label stream independently.[22] Unfortunately, the independence assumption is not valid. *The tree walks* is nonsense precisely because the choice of actor is not independent of the choice of action. Prepositions have similar dependencies; consider the nonsense sentence *the tree stood toward the door*. Another alternative is to model tracks using FCHMMs, where the state space is the cartesian product of labels. For example, the state space might be all possible combinations of object labels with action labels and trajectory labels. FCHMMs can model arbitrary dependencies

among labels, but unfortunately their state spaces are so large that it is generally infeasible to train them without overfitting to the available data.

In contrast, we train separate HMMs for every label stream, thereby keeping the state spaces small and the need for training data low. However, we assume a new source of information in the form of a binary constraint tensor κ that indicates which cartesian tuples are consistent, and which are not. For example, (*person, walk, left*) is consistent, but (*tree, walk, left*) is not. κ is not learned from training samples, but could be derived instead from external sources of knowledge, such as linguistic knowledge bases that indicate which combinations of nouns, verbs and prepositions are allowed.[2,19,38] In the context of this paper, κ was built manually.

3. Related Work

The work by Yu *et al.*[48] and Siddharth *et al.*[39] is the closest to ours in terms of goals and application domain. Like this paper, they aim to integrate natural language concepts with computer vision, although the role of language in the two systems is different. Siddharth *et al.* extend the simultaneous object detection, tracking and event recognition work by Barbu *et al.*[3] 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. 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. Unfortunately, in a real-world setting, the search space may be large, and the method relies on having a specific object detector for every possible object in the lexicon.

More relevant to the algorithm developed in this work is the extensive literature on HMMs. HMMs have a long history in both computer vision and natural language, but they have limitations when modeling interacting processes. Many real-world signals are generated by processes with multiple underlying variables. These processes generate multiple channels of data which may be dependent or independent. The resulting signals have structure in both time and space.

HMMs estimate probability distributions over the possible states of a single hidden variable, given time series data. When the data are generated by processes with multiple hidden variables, there are two common approaches. One is to combine the hidden variables into a single hidden variable, whose possible states are the cartesian product of the states of the source variables. For example, if the hidden variables correspond to the object, action and trajectory of a video track, states of the combined variable would be 3-tuples of objects, actions and trajectories. The resulting HMM is called an FCHMM. Unfortunately, the number of possible states is exponential in the number of chains. The complexity of FCHMMs is untenable, not only because the state space is large, but also because of the enormous amounts of data required to train such a system. The number of parameters to be estimated in

the state transition matrix alone is the square of the number of states. Since the number of states is exponential in the number of chains, the number of parameters to estimate becomes huge very quickly, with the result that there is often insufficient data, leading to undersampling.

An alternative approach is to model each variable with an independent HMM and couple their outputs. These models are called factorial HMMs (FHMMs).[23] This is the weakest form of coupling and is suited for modeling data from independent processes. The number of states in an FHMM is linear in the number of chains. Although FHMMs can model multiple independent processes, many applications have multiple channels of data that carry complementary information. Modeling such data with FHMMs is inappropriate as interactions among processes are modeled as noise.

Coupled HMMs (CHMMs) are another framework to model data from multiple dependent processes. CHMMs capture more parameters than FHMMs, but not as many as FCHMMs. CHMMs are appropriate to model processes that have their own internal dynamics but influence each other. In a CHMM, a state at time t depends on states at time $t-1$ of all chains. Different versions of CHMMs are proposed by Rezek *et al.*,[33,34] Brand *et al.*[12,13] and Zhong and Ghosh.[49] Each of these models captures different additional parameters. Brand's model,[13] which introduces additional parameters to represent pairwise dependencies, is used for comparison in this paper.

In recent years, HMMs have become more prominent in computer vision research as attention has shifted from still frames to video. HMMs are the most prominent probabilistic grammars used for action recognition in computer vision. Yamato *et al.* recognized the actions using discrete HMMs by representing sequences over a set of vector quantized silhouette features.[45] Bregler trained an HMM over a set of auto-regressive models, each approximating linear motions of blobs in the video frame for recognizing human dynamics in video sequence.[14] Xia *et al.* recognized the actions using histograms of three-dimensional (3D) joints and discrete HMMs.[44]

HMMs are also commonly used for gesture recognition. Starner and Pentland use HMMs for real-time American sign language recognition.[40] Presti *et al.* model the gestures by using a set of HMMs, one for each gesture, trained using discriminative approach that allows different HMMs to share the same state space.[31] Zhong *et al.* use HMMs to recognize complex single hand gestures.[46] Wang *et al.* also use HMMs for dynamic gesture trajectory modeling and recognition.[43] HMMs are also used to detect activities involving multiple actors and actions. Zhou *et al.* use Multi-Observation HMM to model trajectories and detect unusual events in crowded scenes.[50] Karaman *et al.* use hierarchical HMM to detect activities of daily living, by the fusion of multi-modal visual and audio features.[26] As already discussed, Yu *et al.*[48] and Siddarth *et al.*[39] also use HMMs to detect activities in video. Belgacem *et al.* use a hybrid Conditional Random Field (CRF)/HMM for one-shot gesture learning.[9] CRFs are Markov models known for their discriminative ability and HMMs are generative models used for modeling. CRFs and HMMs have complementary advantages and

disadvantages and this hybrid model combines the advantages of both models. Yu and Deng proposed a similar hybrid model in speech recognition with CRF replaced by a deep neural network for classification.[47] However, none of these works tackle to problem of fusing information across multiple interacting label streams.

4. Methodology

To explain the VS algorithm as clearly as possible, we describe it in three subsections. The first presents the underlying concept without the math, while the second introduces the notation and the third presents the algebraic description.

4.1. *Concept*

The Viterbi algorithm estimates the probability distributions. To be precise, it calculates the probability of the most likely path from a start state to a state s at time t for all s, and normalizes these probabilities across the states to sum to 1. From this, it is possible to calculate the most likely state sequence with the given sequence of observations. The VS algorithm has the same goal, except that there are multiple hidden variables and the models are FHMMs (because FCHMMs are too expensive to train). As an additional source of information, however, the VS algorithm has access to constraints that specify which combinations of values are allowed (e.g. *person*, *walk*, *left*) and which are incompatible (e.g. *tree*, *walk*, *left*).

The key to the VS algorithm lies in understanding the mapping between factorial and fully coupled state spaces, as depicted in Fig. 2. States in a factorial system represent the assignment of values to variables, and estimated probabilities over these states represent marginal probability distributions, since the value of one

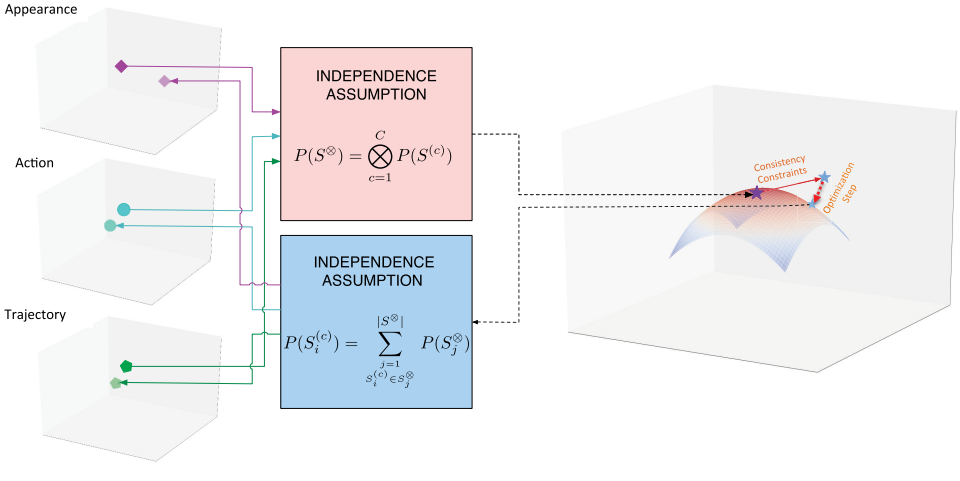

Fig. 2. A geometric illustration of the VS algorithm illustrating the mapping between marginal probabilities (on the left) and joint distribution (on the right). The space of joint distribution that can be factored into marginal distribution is a Segre Variety.

variable is independent of the value of another. States in a fully coupled system represent the assignments of tuples of values to tuples of variables, and estimated probabilities assigned to these states represent joint distributions. Given a joint distribution, a simple linear projection computes the corresponding marginal distribution. In the other direction, the mapping is ambiguous. Every marginal distribution maps to at least one, and in general, many, possible joint distributions.

The constraints embodied in κ veto certain n-tuples of values and thereby restrict the set of possible joint distributions to those with zero probabilities for the inconsistent states. This matters even though we do not have a FCHMM to assign joint distributions, because not all marginal distributions map to a joint distribution with zero probabilities for the inconsistent states. In particular, the combination of marginal distributions that map to an allowed joint distribution forms a Segre variety.[10,21] The VS algorithm restricts the estimated probabilities in the factorized models to lie on this Segre variety. To be more precise, at every step, the probabilities estimated on the basis of factorial models are replaced by the most similar probability distribution in the range of the allowed joint distributions.

4.2. *Terminology*

The following terminology is important for formally specifying the VS algorithm. Data generated by a process are considered a chain in this paper. Examples of chains include object, action and trajectory labels. The mutual information between the chains is lost when they are interpreted independently. The VS algorithm presented below exploits external knowledge of consistent and inconsistent combinations of states to capture some of the mutual information among chains. The following terminology is applicable to HMMs that fuse information from C chains:

- C: Number of chains.
- Set of states: $S = \cup_{c=1}^{C} S^{(c)}$, where $S^{(c)}$ is the set of states for cth chain. The total number of states $|S| = \sum_{c=1}^{C} |S^{(c)}|$.
- Cartesian product states: $S^{\otimes} = \otimes_{c=1}^{C} S^{(c)}$. 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.
- $\bar{A}$ is the vector of transitional probability matrices: $\bar{A} = \{A^{(1)}, A^{(2)}, \ldots, A^{(C)}\}$.
- $\bar{B}$ is the vector of observation probability matrices: $\bar{B} = \{B^{(1)}, B^{(2)}, \ldots, B^{(C)}\}$.
- $\bar{\pi}$ is the vector of prior probability vectors: $\bar{\pi} = \{\pi^{(1)}, \pi^{(2)}, \ldots, \pi^{(C)}\}$.
- κ is an unrolled binary constraint knowledge tensor 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 $\lambda_\kappa = (\bar{A}, \bar{B}, \bar{\pi}, \kappa)_C$.

4.3. *VS algorithm*

The Viterbi algorithm is a dynamic programming algorithm that finds the most likely sequence of states without explicitly searching the space of all possible state sequences. The VS algorithm extends the standard Viterbi algorithm to integrate data from multiple channels with not only just the HMMs λ, but also an unrolled tensor of binary consistency constraints κ. The VS algorithm runs 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 probability along a single path in a chain c ending at a state $S_i^{(c)}$, accounting for the first t observations. The probability is represented by $\delta_t^{(c)}(i)$. At a given timestep, this value can be calculated for every state $s \in S$. Let Δ_t be the vector of the probabilities for all states $s \in S$ at time t as shown in Eq. (1). Let Ω_t be the probability vector for all states if the chains are coupled as a FCHMM as shown in Eq. (2). Δ_t can be considered as a point in $|S|$ space (factorial state space). Similarly, Ω_t can be considered as a point in $|S^\otimes|$ space (cartesian state space):

$$\Delta_t = \{\delta_t^{(c)}(i)\}, \quad \forall S_i^{(c)} \in S, \tag{1}$$

$$\Omega_t = \{\delta_t(i)\}, \quad \forall S_i \in S^\otimes. \tag{2}$$

Unfortunately, the HMMs are trained and run in factorial state space and Δ_t does not reflect the inconsistencies in κ, since inconsistencies are a form of dependency among chains. Assuming that the chains are independent, one can generate the probability values in cartesian state space Ω_t as shown in Eq. (3). This probability vector in cartesian state space lies on a Segre variety,[10,21] hence the algorithm name. Mathematically, the Segre morphism defined as $\sigma_{n,m} : \mathbb{P}^n \times \mathbb{P}^m \to \mathbb{P}^{(n+1)(m+1)-1}$ takes a pair of points $([x], [y])$ to their product $([x_0 : x_1 : \cdots : x_n], [y_0 : y_1 : \cdots : y_m]) \mapsto [x_0 y_0 : x_1 y_0 : \cdots : x_n y_m]$ and the image of the Segre map is a Segre variety and is defined as $\sum_{n,m} = \sigma_{n,m}(\mathbb{P}^n \times \mathbb{P}^m)$[24]:

$$P(S^\otimes) = \bigotimes_{c=1}^{C} P(S^{(c)}). \tag{3}$$

The fully coupled probability vectors can be projected back to the factorial state space. 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. Mathematically, the probability of a state in factorial state space is calculated from the probabilities in cartesian state space by Eq. (4). If the probability vector in the cartesian state space lies on the Segre variety, the probability vector can be

re-divided into C-independent vectors by the following equation:

$$P(S_i^{(c)}) = \sum_{\substack{j=1 \\ S_i^{(c)} \in S_j^{\otimes}}}^{|S^{\otimes}|} P(S_j^{\otimes}), \tag{4}$$

where $S_i^{(c)} \in S_j^{\otimes}$ shows that the state $S_i^{(c)}$ in factorial state space is in the C-tuple of a state in cartesian state space $S_j^{\otimes}$.

Equation (4) can be written as a linear combination of cartesian state probabilities 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 states $S^{\otimes}$. The values of the ρ matrix are filled with binary values based on Eq. (4) as shown in the following equation:

$$\rho[i][j] = \begin{cases} 1 & \text{if } S_i \in S_j^{\otimes}, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Having defined Δ, Ω and ρ, the probability vector Ω in the cartesian state space can be projected to the probability vector Δ in the factorial state space as

$$\Delta = \rho\Omega. \tag{6}$$

As the probabilities in factorial state space do not reflect the inconsistencies in κ, they can be projected on to the Segre variety in cartesian state space assuming independence. When the probability of all the inconsistent states is set to zero, as shown in Eq. (7), the probability vector is pushed away from the Segre variety. The resulting probability vector cannot be re-divided into C-independent vectors.

$$P(S^{\otimes}) = \kappa \odot \bigotimes_{c=1}^{C} P(S^{(c)}). \tag{7}$$

The VS algorithm finds probability vectors over the factorial state space that are consistent with at least one joint distribution over the cartesian state space that satisfies the consistency constraints. Since many distributions are the projections of conforming joint distributions, the VS algorithm sets up a minimization problem to find the closest such distribution to the original estimate of the factorial model.

The probability vector Ω is the vector that abides by the consistency constraints κ and minimizes the Frobenius 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

$$\begin{gathered} f(\Delta, \rho, \kappa) = \text{Min } ||\Delta - \rho\Omega||_F, \\ \text{such that} \quad 1 \cdot \Omega = 1 \quad \text{and} \quad \Omega \geq 0 \quad \text{and} \quad 1 \cdot ((1 - \kappa) \odot \Omega) = 0. \end{gathered} \tag{8}$$

Algorithm 4.1. VS 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: $O = \{O^{(1)}, O^{(2)}, \ldots, O^{(C)}\}$, 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 timestep. The Q array holds the indices of most likely states that generates the observation sequence O.

1: $\rho, SEQSCORE, BACKPTR = \text{INITIALIZE}(S, S^{\otimes}, \bar{\pi})$
2: $\Delta = SEQSCORE[:, 1]$
3: $\Omega = \text{OPTIMIZE}(\Delta, \rho, \kappa)$
4: $SEQSCORE[:, 1] = \rho\Omega$
5: **for** $t = 2$ to T **do**
6: $\quad SEQSCORE, BACKPTR$
$\quad = \text{VITERBI\_STEP}(SEQSCORE, BACKPTR, \lambda_\kappa, t, O_t)$
7: $\quad \Delta = SEQSCORE[:, t]$
8: $\quad \Omega = \text{OPTIMIZE}(\Delta, \rho, \kappa)$
9: $\quad SEQSCORE[:, t] = \rho\Omega$
10: **end for**
11: $Q = \text{BACKTRACK}(SEQSCORE, BACKPTR, C, S)$
12: **return** Q

The constraint $1 \cdot \Omega = 1$ makes sure that the probabilities of Ω sum to 1. The constraints $\Omega \geq 0$ and $1 \cdot \Omega = 1$ constrain the probability values to be between 0 and 1 inclusive. $1 \cdot ((1 - \kappa) \odot \Omega) = 0$ imposes the consistency constraints in Ω. Equation (8) can be solved by any convex optimization solver; we use CVXPY.[17]

Algorithm 4.1 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 timestep, the optimization function defined in Eq. (8) is applied, and the resultant vector is projected back as shown in Eq. (6). The steps represented by red color in the algorithm shows our contribution to enforce the consistency constraints κ. The optimization method used in lines three and eight of the algorithm is any linear optimization method that can solve the optimization problem laid out in Eq. (8). Other steps (Initialization, Viterbi Step, and Backtracking) are the standard Viterbi algorithm steps, except that the steps are applied to multiple chains. These methods are presented in appendix.

4.3.1. *Complexity analysis*

The VS algorithm is described above and is shown in Algorithm 4.1. The time complexity of applying VS to multiple chains is $O(T(|S^{\max}|^2 C + |S|^{3.5} \ln(\epsilon^{-1})))$,

where T is the sequence length, $|S^{\max}|$ is the maximum number of states for a chain ($|S^{\max}| = \max_{c=1}^{C} |S^{(c)}|$), C is the number of chains, $|S|$ is the total number of states in all chains ($|S| = \sum_{c=1}^{C} |S^{(c)}|$), and ϵ is the optimization error. The complexity of calculating best score probability for paths ending at different states at a timestep t is $O(|S^{\max}|^2 C + |S|^{3.5} \ln(\epsilon^{-1}))$. This involves running Viterbi step ($O(|S^{\max}|^2)$ for all channels resulting in $O(|S^{\max}|^2 C)$ complexity. An optimization step is then performed to integrate consistency constraints and the complexity of solving conic quadratic problems is $O(|S|^{3.5} \ln(\epsilon^{-1}))$.[1]

The complexity of solving conic quadratic problems being $O(|S|^{3.5} \ln(\epsilon^{-1}))$ is a theoretical result. In practice, the complexity is less as the optimization is usually solved in 10 to 100 iterations.[1] On an average, it takes around 0.04 s to run the Viterbi algorithm on a sequence of length 100 (on a FHMM with 3 chains of 5, 5, and 3 states), whereas it takes around 0.8 s to run VS on sequences of length 100 on a Macbook Pro with 2.2 GHz Intel Core i7 processor, running python with no parallelization or other code optimizations.

The space complexity of VS is $O(|S^{\otimes}| + |S|T)$. The Viterbi algorithm calculates the best score probabilities for paths ending at different states across all timesteps requiring $O(|S|)$ space per timestep. In addition, VS requires a binary knowledge tensor κ of length $|S^{\otimes}|$, where $|S^{\otimes}| = \prod_{c=1}^{C} |S^c|$ and the optimization step involves projecting the probabilities in factorial space to the Segre variety in cartesian state space at every timestep. So, the total space complexity of VS is $O(|S^{\otimes}| + |S|T)$.

4.4. *Baum–Welch–Segre algorithm*

The Baum–Welch algorithm[5,6,8,28] is used to train HMMs to maximize the probability of a given observation sequence using the Expectation Maximization method.[16] 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. $P(O|\bar{\lambda}) \geq P(O|\lambda)$.[7,8] The initial model is crucial for training, as the Baum–Welch algorithm finds a local optima.

Similar to the VS algorithm, the Baum–Welch algorithm can be extended to train HMMs by integrating data from multiple channels along with the consistency constraints. The extended Baum–Welch (Baum–Welch–Segre) algorithm learns the same parameters as an FHMM, but the consistency constraints are weakly embedded within the parameters. The consistency constraints thus embedded do not need any additional parameters.

We explored two variants of the Baum–Welch–Segre algorithm. The first variant integrates the consistency constraints only at the beginning of training. Once the HMM models for each chains are initialized as in the standard Baum–Welch algorithm, the Baum–Welch–Segre algorithm enforces consistency constraints across the initial models by posing it as an optimization problem similar to the one in Eq. (8). Then, the models are trained as FHMMs using the Expectation Maximization

algorithm. Integrating the consistency constraints after the models are initialized results in a better starting state for the *EM* algorithm.

The second variant of the Baum–Welch–Segre algorithm integrates the consistency constraints multiple times during the training of the HMMs. This variant of the Baum–Welch–Segre algorithm starts as the above algorithm. But, once the models converge, the consistency constraints are integrated and the chains are trained once again. This process is repeated until the observation probability converges between two runs.

Although the models trained by the Baum–Welch–Segre algorithm produce marginally better results compared to the models trained by standard the Baum–Welch algorithm, the results are not statistically significant. So, neither method is recommended nor explained in great detail in this paper.

5. Experiments

The performance of VS is first evaluated on synthetic data and then on two real-world datasets namely, the VIRAT dataset and the Classroom dataset. The VS algorithm is run on models trained as FHMM. In FHMMs, the probability of the observations is conditioned on the states in all chains. Figure 3 shows an FHMM with two chains, S and S'. The probability of observation at time t is conditioned on the states from two chains at time t, $P(O_t|S_t, S'_t)$.

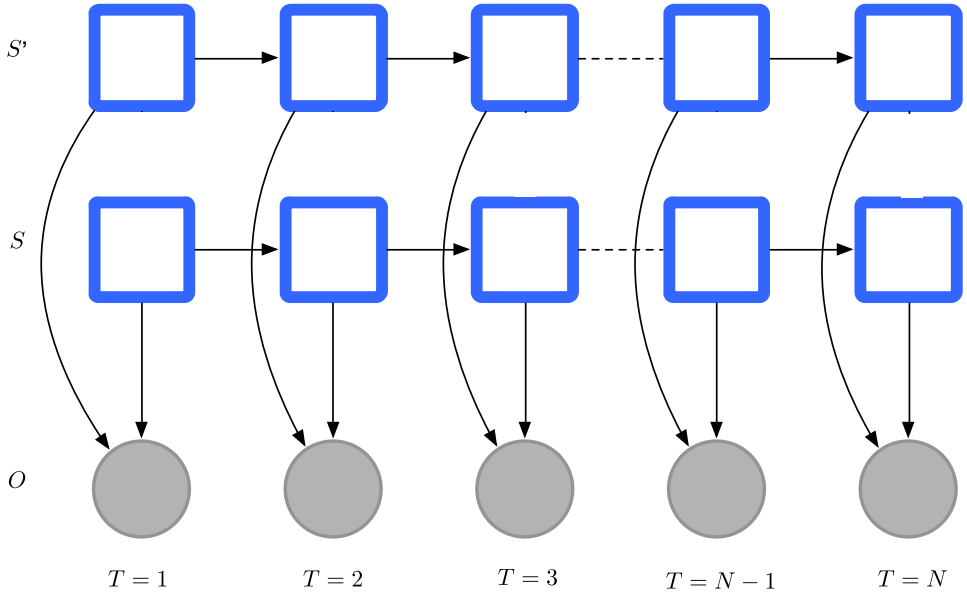

Fig. 3. FHMM with two chains. The observation probabilities of FHMM are conditioned on the states in both chains.

The experiments on synthetic data are discussed in Sec. 5.1. The performance of VS with respect to factorial Viterbi is evaluated on the real-world VIRAT video dataset in Sec. 5.2 and Classroom dataset in Sec. 5.3.

5.1. *Synthetic data*

Synthetic data allow us to compare the VS algorithm to related algorithms under a variety of controlled conditions. Synthetic data are generated in a way so as to mimic labels from a computer vision system that generates natural language descriptions of videos. Three sets of observations are generated for each class: object labels, action labels and trajectory labels. Five object labels, five action labels and three trajectory labels are included in the synthetic data, and these labels form the states and observation symbols for factorized HMMs. Note that the number of labels per chain was kept small to make it feasible to train and run FCHMMs on the model.

5.1.1. *Data generation*

An FCHMM is used to generate synthetic observation data that mimics data from dependent processes. The three chains above lead to an FCHMM with 75 states, where each state is a 3-tuple of object, action and trajectory. The prior probability table π is filled by assuming that each state is equally likely to be the start state. The transition probability matrix for the FCHMM is generated by Kronecker product of the three individual transition probability matrices. The decoupled transition probability matrices have a special structure inspired by the real world. In general, the probability of any label changing from one timestep to the next is small, since videos are captured at roughly 30 frames per second. Therefore the transition probability matrices are diagonally dominant. To be more specific, the object within a track does not change very often at all. This only happens, for example, when someone gets into a car and then drives off, so that the track that initially followed a person then follows a car. Actions change somewhat more often, although most actions have a duration of at least a few seconds. At 30 frames per second, the likelihood of action labels changing from one frame to another is low, although higher than the likelihood of an appearance state change. Trajectory labels are the most dynamic; imagine someone swaying back and forth. So, the diagonal entries of the object, action and trajectory transition probability matrices are set to 0.8, 0.7 and 0.6, respectively. The other values are uniformly distributed such that the sum of the probabilities of each row is 1. The Kronecker product of these three transition probability matrices gives the FCHMM transition probability matrix.

In an ideal world where the classifiers were perfect, the classifier labels would always correctly match the underlying states. In such an ideal world, the observation probability matrix of the FCHMM would be an identity matrix. Unfortunately, real-world classifiers often produce incorrect labels. Despite this noise, we still assume that the observation probability matrix is diagonally dominant. Observation error, i.e. the measure of how often a classifier assigns an incorrect label, is added to the

observation probability matrix. This is done by setting the diagonal values as $1 - \frac{\text{error}}{100}$ and uniformly setting the off diagonal values such that the sum of individual rows is 1.

Some combinations of object, action and trajectory labels do not occur in real world. For example, *trees cannot walk*. As these states do not occur, the respective prior probabilities, the respective rows and columns in the transition table, and the respective rows in observation table, are zeroed out. In addition, Gaussian noise is added to the transition and observation probability matrices to introduce dependencies among chains.

The synthetic data, both train and test, are created by running the FCHMM to generate both a sequence of observations and a ground truth set of underlying states. There are two variables in synthetic data generation — the number of inconsistent states and the observation error. The percent of inconsistent states is varied from 0 to 80 in increments of 20. 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. Fifteen sets of synthetic data are generated for each combination of these two variables, yielding 300 different datasets. The 640 sequences each of length 100 are generated for each dataset. These sequences are divided into 5 sets of 128 sequences. A five-fold cross-validation is performed by training on 128 sequences and testing on 512 sequences.

5.1.2. *Experimental performance of VS on synthetic data*

This section evaluates the performance of the proposed VS algorithm on synthetic data. As VS integrates consistency constraints only at run-time, a trained HMM is required to run the VS algorithm. The HMM is trained as an FHMM on 128 sequences of training data generated as discussed above using the standard Baum–Welch algorithm. Then, the VS algorithm is run using the trained models as FHMMs. Moreover, the standard Viterbi algorithm is run on individual chains to get the results of FHMM. For comparison, FCHMM and Brand's version of a coupled HMM (BCHMM) are also trained (on 128 sequences) and tested (on 512 sequences). The FCHMM is trained using the standard Baum–Welch algorithm with the state space being the cartesian product of state spaces from individual chains. The standard Viterbi algorithm is then run using the trained model and the cartesian state space. BCHMM is trained and run based on the algorithms in Refs. 12 and 13. All HMMs were trained on 128 sequences of length 100 and run on 512 sequences of length 100. Figure 4 shows the results of the experiment.

Figure 4 shows four subplots, one for each level of observation error (20%, 40%, 60% and 80%). Each subplot shows the percentage of states correctly predicted (accuracy) as a function of the percent of inconsistent states. A five-fold cross-validation is done on 15 sets of synthetic data generated for every combination of observation error and inconsistent state percentage; the error bars show the standard deviation among the trials. The plots show that the performance of the VS algorithm

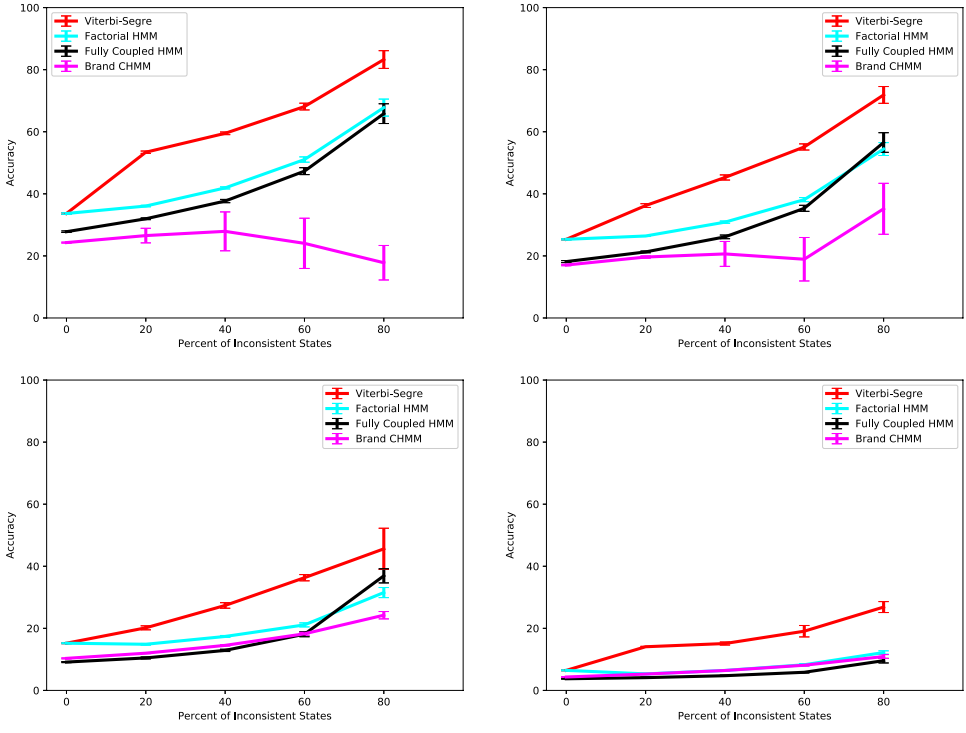

Fig. 4. Performance of the VS algorithm, factorial Viterbi, fully coupled Viterbi and Brand's algorithm. The plots show accuracy as a function of the percent of inconsistent states at different levels of observation noise.

is always better than the other models. Moreover, the relative performance gain increases as the observation error grows. Apparently, the less reliable any given observation is, the more important the consistency constraints become.

For a given level of observation error, the benefit of using the VS algorithm increases as the number of inconsistent states increase. This increase in the performance gain is due to the additional knowledge that can be integrated when large numbers of states are inconsistent. Unfortunately, as the number of inconsistent states increases, the standard deviation of the performance also increases. However, the same behavior can be seen with the other HMMs as well. As the number of inconsistent states increases, the search space becomes less smooth and develops more local optima, resulting in a higher standard deviation across runs.

We can conclude that the VS algorithm is the highest performer in accuracy on synthetic data, followed by FHMM, FCHMM and BCHMM. The poor performance of BCHMM can be attributed to the way the task was defined. BCHMM trains in cartesian state space. But after every iteration, the model is factored to a factorial space and projected back onto a cartesian state space. The algorithm captures the dependencies between two chains in the factoring step. However, the synthetic data

have dependencies among three chains that cannot be captured as two-way dependencies. Moreover, training in cartesian space requires more training data and may lead to overfitting. As the BCHMM is being trained in cartesian state space, and as it cannot capture the dependencies that exist between multiple chains, it performs worse than other HMMs. Although FCHMM performs better than BCHMM, VS and FHMM outperform FCHMM. It is interesting to note that although an FCHMM was used to generate the synthetic data, FCHMM is not the best performer. As already discussed, FCHMMs train in cartesian state space and require enormous amounts of training data. However, the HMMs in this experiment are trained on 128 sequences of length 100. These data might be insufficient to train a 75-state FCHMM resulting in the poor performance of FCHMM on synthetic data. In contrast, FHMMs have sufficient training data making them the second best model on synthetic data. VS has sufficient training data and access to additional consistency constraints, so VS performs the best on the synthetic data.

5.1.3. *Performance of VS with partial knowledge of inconsistent states*

The previous section showed that the proposed VS algorithm outperforms FHMM, FCHMM and BCHMM on synthetic data by integrating consistency constraints during inference. This section evaluates the performance of VS when only some of the inconsistent states are known. The HMM is trained as an FHMM on 128 sequences of training data as discussed in the previous section, and VS is run using the trained FHMM models. VS requires a binary constraint knowledge tensor κ that specifies the consistent and inconsistent states. The consistent states are denoted by 1 and the inconsistent ones are denoted by 0 in the binary tensor κ. Partial knowledge of inconsistent state configuration is given to the VS algorithm by marking a subset of inconsistent states as consistent. The knowledge percentage signifies the percentage of inconsistent states given to the VS algorithm. 0% knowledge specifies that no knowledge of inconsistent states is provided to VS (all states are marked consistent), and 100% knowledge specifies that the complete information of consistent and inconsistent states (no inconsistent state is marked consistent) is given to VS.

We ran VS on 240 out of the 300 synthetic datasets generated in Sec. 5.1 (excluding 60 datasets that have 0% inconsistencies or no inconsistent labels) with varying percentage of knowledge about inconsistent states. The knowledge percentage is varied between 0 and 100 in increments of 20. The synthetic data are generated with 75 states and the percentage of inconsistent states in data generation specifies the percentage of states that are inconsistent. The knowledge percentage further specifies the percentage of these inconsistent states given to VS. For example, the synthetic data generated with 40% inconsistent states have 30 inconsistent states, and providing 40% knowledge on this synthetic data signifies randomly selecting 60% (or 18 out of 30) of inconsistent states and marking them as consistent, thereby giving information about only 12 inconsistent states.

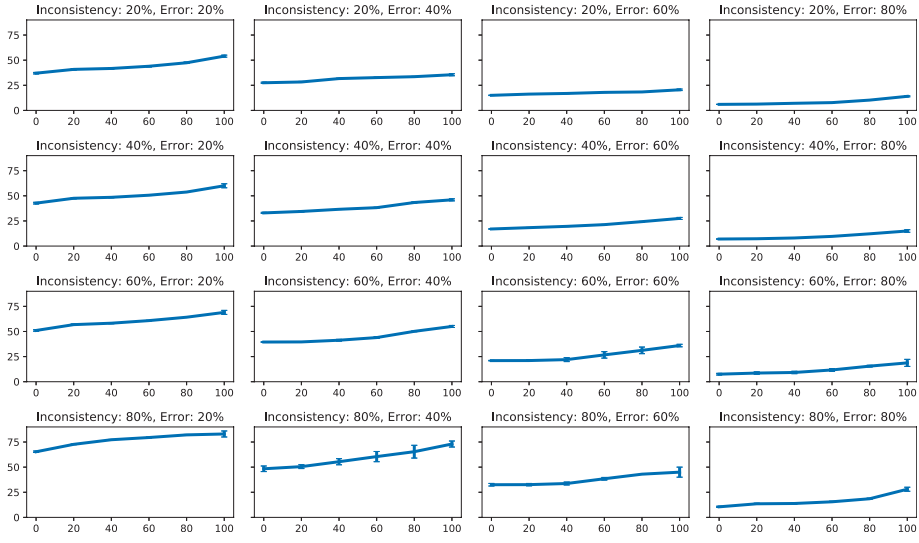

Fig. 5. Performance of the VS algorithm with different percentages of knowledge of inconsistent states given *a priori*. Each plot corresponds to a different combination of observation error and percentage of inconsistent states, with observation error increasing to the right and percentage of inconsistent states increasing down the columns. Each plot shows accuracy as a function of the percentage of inconsistent states knowledge given to VS.

Figure 5 shows 16 subplots, one for each combination of inconsistent states (20%, 40%, 60% and 80%) and observation error (20%, 40%, 60% and 80%). Each subplot shows the percentage of states correctly predicted (accuracy) as a function of the percent of knowledge of inconsistent states given to VS. For each combination of inconsistent states and observation error, 15 sets of synthetic data are generated and a five-fold cross-validation is performed on each of these 15 sets, resulting in 75 train and test datasets, as discussed in Sec. 5.1. For each dataset, we randomly select some inconsistent states based on the percentage of knowledge and make them consistent. This process is repeated 15 times; the error bars signify the standard deviation over 1125 runs (15 sets of synthetic data $\times$ 5 cross-validation $\times$ 15 trials). The plots show that as the knowledge of inconsistent states given to VS increases, the accuracy increases. VS performs best when it is provided with all the information of consistent and inconsistent states. When no knowledge (0%) of inconsistent states is provided (all states are consistent), the performance of VS is equal to the performance of FHMM.

5.2. *VIRAT dataset*

Having seen that VS outperforms other HMMs on synthetic data, we evaluate its performance on real-world data from the VIRAT video dataset.[29] VIRAT contains 329 outdoor videos with naturally occurring events. The events involve interactions

Fig. 6. Different scenes from VIRAT dataset.

between multiple actors, vehicles and facilities. Snapshots of different scenes from the VIRAT dataset are shown in Fig. 6. The dataset includes object tracks with ground truth annotations that label objects and actions, but not trajectories. There are annotations for five objects: *person*, *car*, *vehicle*, *object* and *bike*. There are also annotations for 11 events: *load*, *unload*, *open*, *close*, *get-in*, *get-out*, *gesture*, *carry*, *run*, *enter* and *exit*. Unfortunately, not all tracks are annotated with actions, forcing us to infer ground truth action labels as described below.

5.2.1. *Training protocol*

As the VIRAT dataset comes with object tracks with ground truth annotations, we use the provided object tracks to keep the experimental evaluation clean. To apply VS to VIRAT, we need classifiers to assign object and action labels (observations) to frames of VIRAT object tracks. We train two separate classifiers, one for objects and one for actions. To avoid testing on training data, we adopt the five-fold cross-validation protocol recommended for VIRAT.[41] On each trial, we train a support vector machine (SVM) classifier to assign object labels to track frames. The input features are concatenations of scale-invariant feature transform (SIFT) and local binary pattern (LBP) features extracted from the track frame, and the output is one of the five object labels.

The action classifier is a little more complicated. Although appearance labels are provided for all tracks in the training data, only a handful of tracks have action labels, and then only at specific moments. In other words, ground truth action labels

Table 1. Method for inferring ground truth action labels from ground truth appearance labels and trajectories, when no ground truth label is otherwise provided.

	Person	Car	Vehicles	Object	Bike
Still	Still	Still	Still	Still	Still
Motion	Walk	Drive	Drive	Carry	Ride

are not provided for every frame of every track. Therefore, additional ground truth action labels are generated automatically based on appearance labels and trajectories, as shown in Table 1. This results in more training data and four additional action labels: *walk*, *drive*, *carry* and *ride*. Actions are classified using an approximate nearest neighbor classifier.[30] The classifier models time windows as points on a Grassmann manifold, and measures geodesic distances between video snippets on the manifold.

Trajectory labels are also required for VS. Simple trajectory labels (*still*, *move* and *translate*) are calculated based on the displacement of the track over 48 frames ($\sim$1.5 s). Trajectory label *still* refers to a track that does not move at all. *Move* label is assigned to tracks that move back and forth. Such tracks have absolute motion, but their relative motion is still zero. Tracks that have both absolute motion and relative motion are labeled with trajectory label *translate*. As ground truth data for trajectory labels are not available, the quality of these labels is not evaluated below.

Once appearance, action and trajectory classifiers are trained, FHMMs are trained on the appearance, action and trajectory chains using the Baum–Welch algorithm. Consistency constraints κ for the VS algorithm were inferred once manually from the label semantics. For example, people walk but cars drive.

5.2.2. *Experimental performance*

On every trial, FHMMs are trained as described above, and tested (1) as independent HMMs using the Viterbi algorithm and (2) as consistent HMMs using the VS algorithm. FCHMMs are not used for evaluation as the cartesian state space of VIRAT dataset is large (5 appearance labels $*$ 15 action labels $*$ 3 trajectory labels $=$ 225 states) and requires large amounts of training data to perform comparatively to VS and FHMM. Figure 7 shows the label accuracy of both methods on VIRAT appearance labels. In general, VS is more accurate than Viterbi. Although the difference is small ($92.16\% \pm 0.5\%$ versus $88.64\% \pm 0.9\%$), VS is more accurate than Viterbi for all five object classes. As the label *bike* is part of many inconsistent states, it shows the greatest performance improvement. Although the label *vehicle* is also highly constrained, it shares many of the same consistent labels with cars, so its improvement is less.

The performance of VS and Viterbi on action labels is shown in Fig. 8. Once again, VS outperforms Viterbi overall ($84.02\% \pm 2.1\%$ to $74.05\% \pm 3.6\%$) and is better than or equal to Viterbi on every action label. In general, the performance increase is

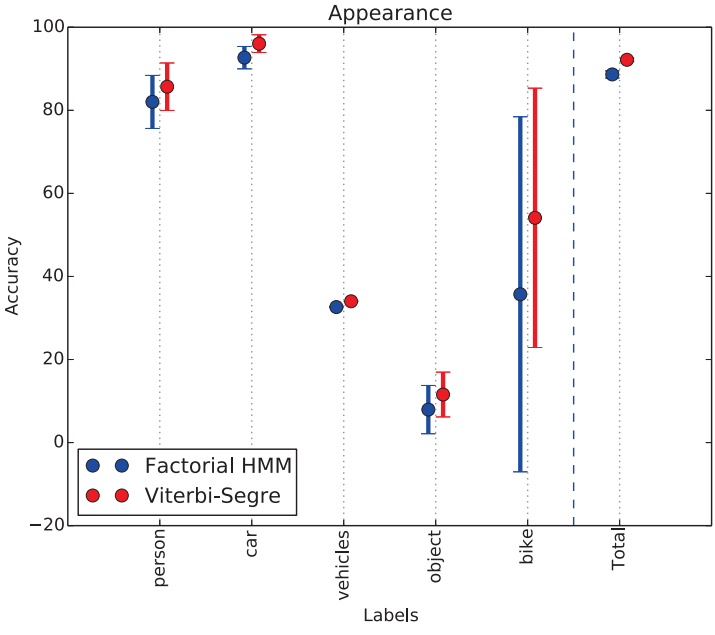

Fig. 7. Performance of the VS algorithm and factorial Viterbi algorithm on appearance labels of VIRAT dataset.

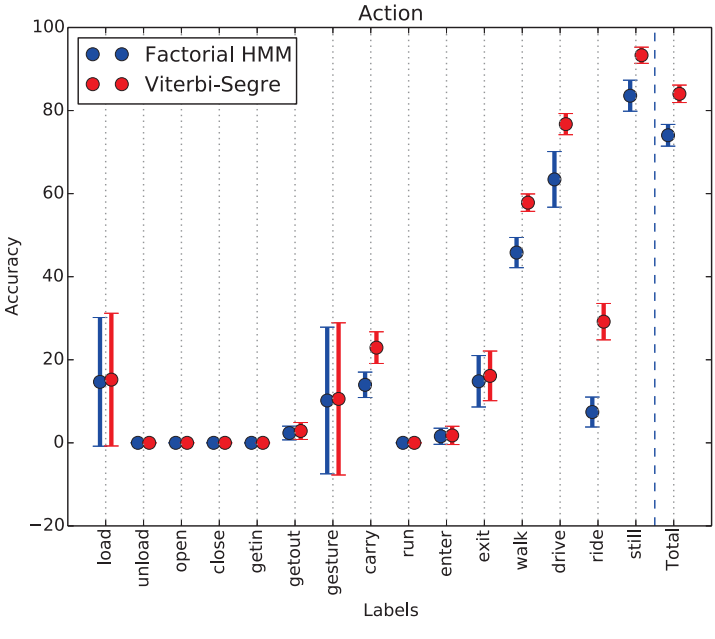

Fig. 8. Performance of the VS algorithm and factorial Viterbi algorithm on action labels of VIRAT dataset.

better for actions than for objects, in part because the observation noise is greater and in part because some actions are highly constrained in terms of actions and/or trajectories. For example, *ride* is only consistent with (*bike*, *translate*) while *drive* is only consistent with (*car*, *translate*) and (*vehicles*, *translate*). As a result, these labels see the most improvement. Although the labels *gesture*, *enter* and *exit* are only consistent with *person*, they are consistent with all trajectory labels, and *person* is consistent with many other actions, so the gain is less.

Although VS improves accuracy for both object and action labels, its goal is to improve the accuracy of tuples. In other words, it tries to maximize the number of frames for which both the object and action labels are correct. Figure 9 shows the performance of VS over five significantly improved tuples, and for tuples overall. The figure shows that the tuple (*bike*, *ride*) becomes much more accurate, which stems from the consistency constraints *bike* and *ride* participate in. This improvement is also reflected in the performance gain of *bike* in Fig. 7 and *ride* in Fig. 8. VS improved the performance of tuple (*object*, *still*) as well, although the accuracy gain for the label *object* is not large in Fig. 7. We conclude that VS improves the action label *still* for objects with the help of trajectory labels (motion information). Overall, the accuracy of tuples improves from $67.84\% \pm 3.47\%$ to $78.42\% \pm 1.7\%$.

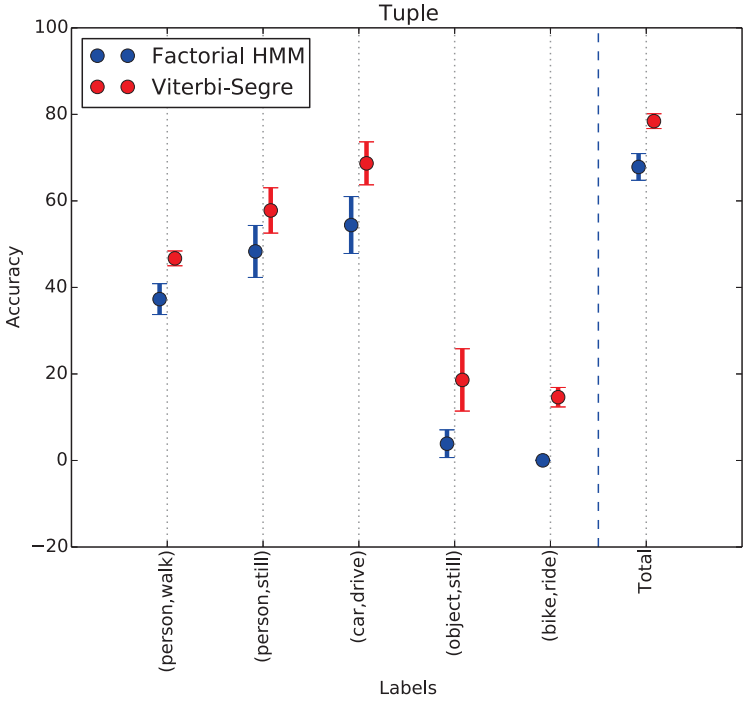

Fig. 9. Performance of the VS algorithm and factorial Viterbi algorithm on tuple labels of VIRAT dataset.

5.3. *Classroom dataset*

The VS algorithm outperformed FHMM on the VIRAT dataset. However, the object tracks for the VIRAT dataset were already provided, thus the classifiers predicted the object and action tracks with good accuracy. VS algorithm then improved the results further. To evaluate the performance of VS algorithm when there is high noise in the input label stream, we run the VS algorithm on a Classroom dataset that does not have manually annotated object tracks.

The Classroom dataset contains video data collected in a single day in one-fourth grade classroom during the Fall of 2013. The videos were recorded mid-week during a typical school day when no special events were taking place. The classroom was participating in the USDA-funded Fuel For Fun study, which is a school and family-based obesity prevention effort that utilizes experiential cooking and tasting curricula as well as active recess lessons within the school environment.[15] Video data were collected during the baseline assessment of the Fuel for Fun study, and thus no intervention components had been delivered prior to video data collection. Concurrent with video data collection, children wore GENEActiv devices, and accelerometer data were recorded at a sampling frequency of 75 Hz. A video camera (GoPro, San Mateo, CA) was mounted on the ceiling in the corner of the classroom prior to students' arrival for the day. At the start of the school day, the teacher explained the presence of and reasoning for the camera in the classroom, and asked that children carryon with normal classroom activities. Continuous video recording of the classroom space took place from 8:15 AM to 2:05 PM. During this time, children spent approximately 3 h and 40 min in the classroom space.

The main goal of the dataset is to monitor the physical activity of the kids in the classroom and validate the data provided by body-worn sensors. This will be reflected in the appearance and action labels below. There are inherent challenges in the dataset such as changes in lighting, occlusions from other kids and furniture, and kids staying still for an extended period of time. Figure 10 shows different frames from the Classroom dataset.

5.3.1. *Training protocol*

Unlike the VIRAT dataset, the Classroom dataset does not come with object tracks. Object tracks are extracted from the classroom dataset by running ViBe.[4] ViBe classifies pixels of each frame in a video either as foreground or background pixels. A connected component algorithm is then run on each frame to get sets of connected foreground pixels. Connected components below a particular size are discarded. The remaining foreground connected components are then overlapped across time to form an object track. Object tracks below a length of 48 frames are discarded as the action classifier requires a minimum track length of 48 frames. The tracks are then manually annotated with their respective object and action labels. However, manually annotating the tracks with object and action labels require a lot of human

Fig. 10. Different frames from classroom dataset.

effort. We annotated the first 10 min of the video for this experiment, accounting to 18,000 frames.

As already mentioned, each track is manually annotated with appearance, action and trajectory labels. The appearance labels appeared in the first 10 min of classroom video in alphabetical order are *arm*, *chair*, *feet*, *head*, *leg*, *other*, *people*, *person_floor*, *person_sit*, *person_stand*. Appearance labels *arm*, *chair*, *feet*, *head*, *leg* are self-explanatory. The label *other* refers to tracks that appear on walls, books, roof, etc. that may not contribute any important information to the activity levels of children. Appearance label *person* refers to two or more persons. The other three labels classifies a single person according to the pose (sitting on floor, sitting on chair, standing). These labels reflect the goal of monitoring activity levels in children. The annotated action labels for the tracks are > *one activity*, *active*, *quiet*, *running*, *twisting*, *unknown*, *walking*. The label > *one activity* refers to action labels for people tracks. As people tracks has more than one person, they are always assigned > *one activity* label. The action label *quiet* is assigned to idle tracks and *active* is assigned to tracts that are not idle. *Twisting* label is assigned to roof hangings (appearance label — *other*) that twist with air. *Unknown* label is assigned to tracks whose action labels are hard to comprehend for human annotators. Moreover, each track is labeled with trajectory labels *still*, *move* and *translate* similar to VIRAT dataset.

Similar to the VIRAT dataset, we need classifiers to assign object and action labels (observations) to frames of Classroom object tracks. Although simple trajectory labels (still, move, and translate) are calculated based on the displacement of the track over 48 frames (1.5 s), we do not evaluate the quality of these labels.

However, trajectory labels are used in VS to improve the performance. We employ the same training protocol used for VIRAT dataset to assign object and action labels. The only difference is that we employ three fold cross-validation as the video we are working on has only 18,000 frames.

Once appearance, action and trajectory classifiers are trained, (separate) FHMMs are trained on the appearance, action and trajectory chains using the Baum–Welch algorithm. Consistency constraints κ for the dataset were automatically inferred for the appearance, action tuples from the ground truth. Then, these constraints are extended manually to accommodate trajectory labels.

5.3.2. *Experimental performance*

On every fold of three-fold cross-validation, FHMMs are trained as described above, and tested (1) as independent HMMs using the Viterbi algorithm and (2) as consistent HMMs using the VS algorithm. The accuracy of both methods on appearance labels is shown in Fig. 11. The performance of VS algorithm is either equal to or better than the Viterbi algorithm on all appearance labels. The appearance label that has the highest performance gain by using the VS algorithm is *person_stand*. Actions such as *walking*, *running* are consistent only with *person_stand* appearance label. They contributed to an accuracy increase to $36.6\% \pm 16.4\%$ from $28.6\% \pm 14.7\%$.

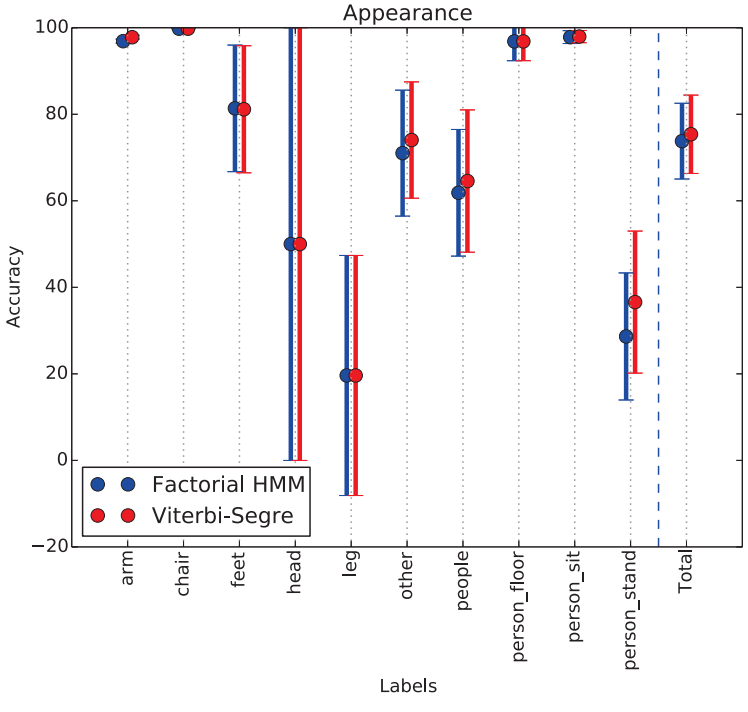

Fig. 11. Performance of the VS algorithm and factorial Viterbi algorithm on appearance labels of classroom dataset.

Other appearance labels that have performance gains from using VS are *arm*, *other*, *people*. The performance gain of the appearance label *people* can be attributed to it being consistent only with action label > *one activity*. Similar observations can be inferred for other labels too. Although appearance labels such as *chair*, *feet*, *head*, *leg* are consistent with only one action label *quiet*, VS algorithm did not improve their accuracy as action label *quiet* is consistent with all appearance labels except *people*. The overall accuracy of VS algorithm on appearance labels is $75.4\% \pm 9\%$ compared to the accuracy of $73.8\% \pm 8.8\%$ for Viterbi algorithm.

The performance improvement of using VS algorithm when labels are highly constrained is evident by its performance on action labels as shown in Fig. 12. The overall accuracy of Viterbi algorithm on action labels is $59.7\% \pm 3.4\%$ whereas the accuracy of VS algorithm is $81.5\% \pm 5.7\%$. Action label *active* has the highest gain by using the VS algorithm ($4.5\% \pm 6.3\%$ to $85.2\% \pm 11.6\%$). Action classifier misclassified most of the *active* labels as *quiet*. However, trajectory labels can easily distinguish between these two. Action label *active* is only consistent with trajectory label *move* whereas action label *quiet* is only consistent with trajectory label *still* and vice versa. So, trajectory labels can easily distinguish between action labels *active* and *quiet*. The same arguments hold in the case of action labels *running* and *walking*.

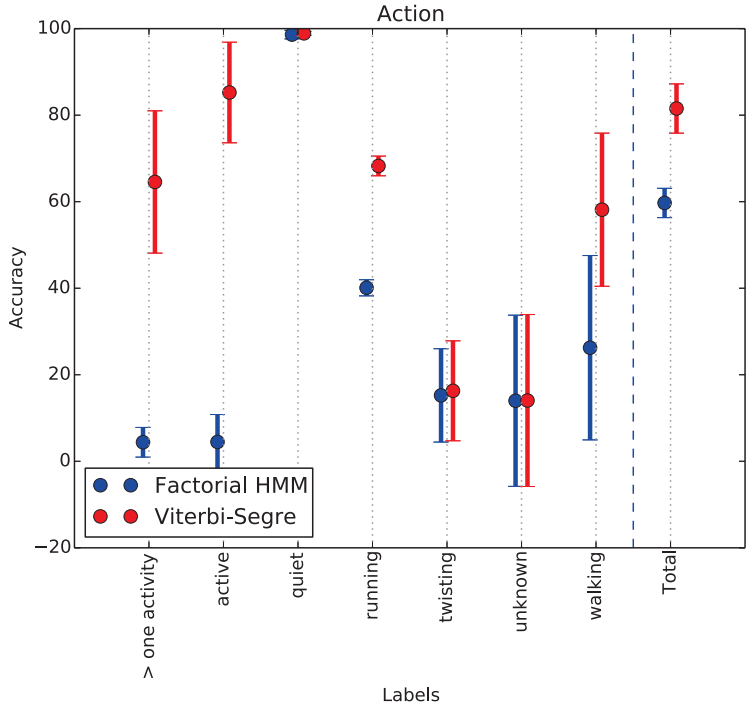

Fig. 12. Performance of the VS algorithm and factorial Viterbi algorithm on action labels of Classroom dataset.

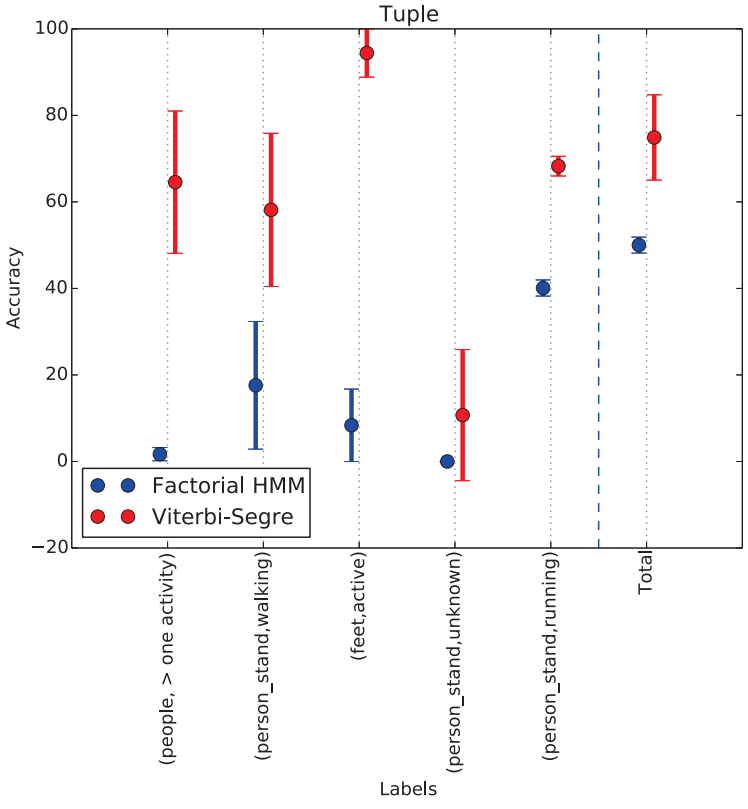

Fig. 13. Performance of the VS algorithm and factorial Viterbi algorithm on tuple labels of classroom dataset.

The second highest performance improvement is in $>$ *one activity* label ($4.4\% \pm 3.4\%$ to $64.6\% \pm 16.4\%$). The reason for this increase can be attributed to the label being consistent only with appearance label *people* and vice versa and the high accuracy of appearance classifier on label *people* (61.9%).

As VS algorithm optimizes the tuple information by applying consistency constraints, the tuple labels benefit the most from VS algorithm as shown in Fig. 13. The performance of the tuple constraints increase from $50.02\% \pm 1.8\%$ to $74.9\% \pm 9.8\%$ when VS algorithm is used instead of Viterbi. The tuple (*feet*, *active*) gained the highest performance gain from VS algorithm ($8.4\% \pm 8.4\%$ to $94.4\% \pm 5.6\%$). The original labels had highly accurate appearance labels but highly inaccurate action labels. Most of the action labels were misclassified as being *quiet*. However, the trajectory label is *move* that is only consistent with action label *active*. So, appearance and trajectory labels together correct the misclassified action label. Similarly, other tuple accuracies increased due to the interplay between appearance, action, trajectory labels and consistency constraints.

6. Conclusion

The task of describing unlabeled videos in natural language creates a new variation of an old problem. The old problem is information fusion across multiple, dependent sources in time series data. The new variation is the availability of consistency constraints that veto certain combinations of states (e.g. *trees cannot walk*). This led us to re-examine the venerable Viterbi algorithm within the relatively new framework of Bayesian networks as Segre varieties. The result is a new algorithm, the VS algorithm, that incorporates consistency constraints between sources within an FHMM model.

Experiments are performed on both synthetic and real data. Synthetic data are generated by varying the percentage of inconsistent states and the percentage of observation error. The VS algorithm is compared to other models, namely FHMM, FCHMM and BCHMM. The VS algorithm performs better on synthetic data than all the other models, followed by FHMM, FCHMM and BCHMM. The VS algorithm is then compared to its closest competitor, FHMM, on the VIRAT and classroom datasets. This experiment suggests that the VS algorithm performs better than FHMM on real-world data, too. The VS algorithm is able to outperform the competition because it incorporates language-based consistency constraints that other algorithms cannot. Conveniently, VS can integrate consistency constraints at run time that were not available during training.

Acknowledgment

This research has been supported by funding (W911NF10-2-0066) from the DARPA Mind's Eye program.

Appendix 1

Algorithm. 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: $BACKPTR$;

1: Initialize a ρ array of size $|S| \times |S^{\otimes}|$
2: **for** $i = 1$ to S **do**

```
    for j = 1 to |S^⊗| do
      if S_i ∈ S_j^⊗ then
        ρ[i][j] = 1
      else
        ρ[i][j] = 0
      end if
    end for
  end for
  Initialize a SEQSCORE array of size |S| × T
  Initialize another BACKPTR array of size |S| × T
  for c = 1 to C do
    for i = 1 to |S^(c)| do
      j = decoupledStateIndex(S_i^(c))
      SEQSCORE[j, 1] = π_i^(c)
      BACKPTR[j, 1] = 0
    end for
  end for
  return ρ, SEQSCORE, BACKPTR
```

Appendix 2

Algorithm. Viterbi Step

Input: $SEQSCORE$; $BACKPTR$; $\lambda_\kappa; t; O_t$

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
```

Appendix 3

Algorithm. Backtracking

Input: $SEQSCORE$; $BACKPTR$; C; S

Output: An array with most likely state sequence indices for each chain Q.

```
Initialize Q array of length C × 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^(c)_Q[t+1]]
      Q[c,t] = BACKPTR[k,t+1]
   end for
end for
return Q
```

References

1. E. D. Andersen, Complexity of solving conic quadratic problems http://erlingdandersen.blogspot.com/2013/11/complexity-of-solving-conic-quad ratic.html.
2. C. F. Baker, C. J. Fillmore and J. B. Lowe, The berkeley framenet project, in *Proc. 36th Annual Meeting of the Association for Computational Linguistics and 17th Int. Conf. Computational Linguistics-Volume 1* (Association for Computational Linguistics, Montreal, Quebec, Canada, 1998), pp. 86–90.
3. A. Barbu, A. Michaux, S. Narayanaswamy and J. M. Siskind, Simultaneous object detection, tracking, and event recognition, *Adv. Cogn. Syst.* **2** (2012) 203–220.
4. O. Barnich and M. Van Droogenbroeck, Vibe: A universal background subtraction algorithm for video sequences, *IEEE Trans. Image Process.* **20**(6) (2011) 1709–1724.
5. L. E. Baum, An equality and associated maximization technique in statistical estimation for probabilistic functions of markov processes, *Inequalities* **3** (1972) 1–8.
6. L. E. Baum and J. Eagon, An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology, *Bull. Am. Math. Soc.* **73**(3) (1967) 360–363.
7. L. E. Baum, T. Petrie, G. Soules and N. Weiss, A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains, *Ann. Math. Stat.* **41**(1) (1970) 164–171.
8. L. E. Baum and G. R. Sell, Growth transformations for functions on manifolds, *Pacific J. Math* **27**(2) (1968) 211–227.
9. S. Belgacem, C. Chatelain and T. Paquet, A hybrid crf/hmm for one-shot gesture learning, in *Adaptive Biometric Systems* (Springer, Cham, 2015), pp. 51–72.

10. E. Bertini, Einführung in die projective geometrie mehrdimensionaler räume, *Bull. Am. Math. Soc.* **31**(8) (1925) 463–464, doi: http://dx.doi.org/10.1090/S0002-9904-1925-04102-6PII (1925) 0002–9904.
11. L. Bourdev, S. Maji and J. Malik, Describing people: A poselet-based approach to attribute classification, in *Proc. IEEE Int. Conf. Computer Vision* (IEEE, Barcelona, Spain, 2011), pp. 1543–1550.
12. M. Brand, Coupled hidden markov models for modeling interacting processes, MIT Media Lab Vision and Modeling Technical Report #405 (1996).
13. M. Brand, N. Oliver and A. Pentland, Coupled hidden markov models for complex action recognition, in *1997 Proc. IEEE Computer Society Conf. Computer Vision and Pattern Recognition* (IEEE Computer Society, Washington, DC, USA, 1997), pp. 994–999.
14. C. Bregler, Learning and recognizing human dynamics in video sequences, in *1997 Proc. IEEE Computer Society Conf. Computer Vision and Pattern Recognition* (IEEE, San Juan, Puerto Rico, 1997), pp. 568–574.
15. L. Cunningham-Sabo, B. Lohse, S. Baker and L. Bellows, Cooking with kids 2.0: Plus parents and play, *J. Nutr. Educ. Behav.* **45**(4) (2013) S80.
16. A. P. Dempster, N. M. Laird, D. B. Rubin *et al.*, Maximum likelihood from incomplete data via the em algorithm, *J. Royal Stat. Soc.* **39**(1) (1977) 1–38.
17. S. Diamond, E. Chu and S. Boyd, CVXPY: A Python-embedded modeling language for convex optimization, *The Journal of Machine Learning Research,* version 0.2 http://cvxpy.org/(2014).
18. J. Donlan, Darpa mind's eye program: Broad agency announcement, Technical Report, DARPA-BAA-10-53 (2010).
19. C. Fellbaum, *WordNet* (Wiley Online Library, New York, 1999).
20. J. Gall and V. Lempitsky, Class-specific hough forests for object detection, in *Decision Forests for Computer Vision and Medical Image Analysis* (Springer, London, 2013), pp. 143–157.
21. L. D. Garcia, M. Stillman and B. Sturmfels, Algebraic geometry of bayesian networks, *J. Symb. Comput.* **39**(3) (2005) 331–355.
22. Z. Ghahramani, An introduction to hidden markov models and bayesian networks, *Int. J. Pattern Recogn. Artif. Intell.* **15**(1) (2001) 9–42.
23. Z. Ghahramani and M. I. Jordan, Factorial hidden markov models, *Mach. Learn.* **29**(2–3) (1997) 245–273.
24. J. W. P. Hirschfeld and J. A. Thas, *General Galois Geometries* (Springer, 1991).
25. R. Hou, A. R. Zamir, R. Sukthankar and M. Shah, Damn-discriminative and mutually nearest: Exploiting pairwise category proximity for video action recognition, in *2014 European Conf. Computer Vision (ECCV)* (Springer, Cham, 2014), pp. 721–736.
26. S. Karaman, J. Benois-Pineau, V. Dovgalecs, R. Mégret, J. Pinquier, R. André-Obrecht, Y. Gaëstel and J.-F. Dartigues, Hierarchical hidden markov model in detecting activities of daily living in wearable videos for studies of dementia, *Multimed. Tools Appl.* **69**(3) (2014) 743–771.
27. A. Krizhevsky, I. Sutskever and G. E. Hinton, ImageNet classification with deep convolutional neural networks, *Adv. Neural Inf. Process. Syst.* (2012) 1–9.
28. D. J. MacKay, Ensemble learning for hidden markov models, Technical Report, Cavendish Laboratory, University of Cambridge (1997).
29. S. Oh, A. Hoogs, A. Perera, N. Cuntoor, C.-C. Chen, J. T. Lee, S. Mukherjee, J. Aggarwal, H. Lee, L. Davis *et al.*, A large-scale benchmark dataset for event recognition in surveillance video, in *2011 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2011), pp. 3153–3160.

30. S. O'Hara and B. A. Draper, Scalable action recognition with a subspace forest, in *2012 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2012), pp. 1210–1217.
31. L. L. Presti, M. La Cascia, S. Sclaroff and O. Camps, Gesture modeling by hanklet-based hidden markov model, in *Asian Conf. Computer Vision — ACCV 2014* (Springer, Cham, 2015), pp. 529–546.
32. M. Raptis and L. Simon, Poselet key-framing: A model for human activity recognition, in *2013 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2013), pp. 2650–2657.
33. I. Rezek, M. Gibbs and S. J. Roberts, Maximum *a posteriori* estimation of coupled hidden markov models, *J. VLSI Signal Process. Syst. Signal Image Video Technol.* **32**(1–2) (2002) 55–66.
34. I. Rezek, P. Sykacek and S. J. Roberts, Learning interaction dynamics with coupled hidden markov models, *IEE Proc. Sci. Meas. Technol.* **147**(6) (2000) 345–350.
35. R. Romdhane, B. Boulay, F. Bremond and M. Thonnat, Probabilistic recognition of complex events, in *Int. Conf. Vision Systems (ICVS)* (Springer, Berlin, 2011), pp. 122–131.
36. S. Sadanand and J. J. Corso, Action bank: A high-level representation of activity in video, in *Proc. IEEE Computer Society Conf. Computer Vision and Pattern Recognition* (IEEE, Providence, RI, 2012), pp. 1234–1241.
37. L. K. Saul and M. I. Jordan, Boltzmann chains and hidden markov models, *Adv. Neural Inf. Process. Syst.* **7** (1995) 435–442.
38. K. K. Schuler, Verbnet: A broad-coverage, comprehensive verb lexicon Ph.D. thesis, Univ. of Pennsylvania, Philadelphia, PA, USA (2005).
39. N. Siddharth, A. Barbu and J. M. Siskind, Seeing what you're told: Sentence-guided activity recognition in video, in *2014 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)* (IEEE, Columbus, OH, USA, 2014), pp. 732–739.
40. T. Starner and A. Pentland, Real-time american sign language recognition from video using hidden markov models, in *Motion-Based Recognition* (Springer, Dordrecht, 1997), pp. 227–243.
41. VIRAT, Virat video dataset http://www.viratdata.org/(2012).
42. H. Wang and C. Schmid, Action recognition with improved trajectories, in *IEEE Int. Conf. Computer Vision ICCV 2013* (IEEE, Sydney, 2013), pp. 3551–3558.
43. X. Wang, M. Xia, H. Cai, Y. Gao and C. Cattani, Hidden-markov-models-based dynamic hand gesture recognition, *Math. Probl. Eng.* **2012** (2012).
44. L. Xia, C.-C. Chen and J. Aggarwal, View invariant human action recognition using histograms of 3D joints, in *2012 IEEE Computer Society Conf. Computer Vision and Pattern Recognition Workshops (CVPRW)* (IEEE, Providence, RI, 2012), pp. 20–27.
45. J. Yamato, J. Ohya and K. Ishii, Recognizing human action in time-sequential images using hidden markov model, in *1992 Proc. IEEE Computer Society Conf. Computer Vision and Pattern Recognition* (IEEE, Champaign, IL, USA, 1992), pp. 379–385.
46. Z. Yang, Y. Li, W. Chen and Y. Zheng, Dynamic hand gesture recognition using hidden markov models, in *2012 7th Int. Conf. Computer Science & Education (ICCSE)* (IEEE, Melbourne, VIC, Australia, 2012), pp. 360–365.
47. D. Yu and L. Deng, Deep neural network-hidden markov model hybrid systems, in *Automatic Speech Recognition* (Springer, London, 2015), pp. 99–116.
48. H. Yu, N. Siddharth, A. Barbu and J. M. Siskind, A compositional framework for grounding language inference, generation, and acquisition in video, *J. Artif. Intell. Res.* **52** (2015) 601–713.

49. S. Zhong and J. Ghosh, A new formulation of coupled hidden markov models, Technical Report, Department of Electronic and Computer Engineering, University of Texas, Austin, USA (2001).
50. S. Zhou, W. Shen, D. Zeng and Z. Zhang, Unusual event detection in crowded scenes by trajectory analysis, in *2015 IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, Brisbane, QLD, Australia, 2015), pp. 1300–1304.

Pradyumna Narayana received his BTech in Information Technology from Jawaharlal Nehru Technological University, Hyderabad, India in 2011 and MS from the Department of Computer Science, Colorado State University in 2014. He is expecting to finish his PhD degree from Colorado State University in August 2018. He is a computer vision researcher with eight publications. His research interests include gesture recognition, action recognition and image recognition using deep learning techniques.

J. Ross Beveridge received his BS degree in Applied Mechanics and Engineering Science from the University of California at San Diego in 1980 and his MS and PhD degrees in Computer Science from the University of Massachusetts in 1987 and 1993, respectively. He has been in the Computer Science Department at Colorado State University since 1993, where he was an Assistant Professor from 1993 to 2000, an Associate Professor from 2000 to 2010, and where he is currently a Full Professor. He is a member of the IEEE Computer Society as well as the ACM. He has served as an Associate Editor for the IEEE Transactions on Pattern Recognition and Machine Intelligence (PAMI), Pattern Recognition and Image and Vision Computing. He is General Co-Chair for the The 14th IEEE International Conference on Automatic Face and Gesture Recognition to be held in Lille France in May 2019. He served as General Co-Chair for The International Joint Conference on Biometrics held in Denver, Colorado in October 2017. He was a Program Co-Chair for the 2015 IEEE Seventh International Conference on Biometrics: Theory, Applications and Systems. He was a Program Co-Chair for the 1999 IEEE Conference on Computer Vision and Pattern Recognition and frequently serves on numerous workshop and conference Program Committees. He is an author of over 150 publications in computer vision and related fields. He oversees open-source software distributions dating back to 2002 which have been downloaded by over 30,000 users world wide.

Bruce Draper received his BS from Yale University in 1984, and his MS and PhD from the University of Massachusetts (Amherst) in 1987 and 1993, respectively. He has been on the faculty at Colorado State University since 1996, and a Full Professor since 2011. He is a Computer Vision Researcher with over 100 publications and has been actively involved in the research community, including serving as General Co-chair for CVPR in 1999.