Learning Complex and Sparse Events in Long Sequences

Marco Botta[1] and **Ugo Galassi** [2]and **Attilio Giordana** [3]

Abstract. *The Hierarchical Hidden Markov Model (HHMM) is a well formalized tool suitable to model complex patterns in long temporal or spatial sequences. Even if effective algorithms are available to estimate HHMM parameters from sequences, little has been done in order to automatize the construction of the model architecture. The primary focus of this paper is on a multi-strategy algorithm for inferring the HHMM structure from a set of sequences, where the events to capture are present in a relevant portion of them. The algorithm follows a bottom-up strategy, in which elementary facts in the sequences are progressively grouped, thus building the abstraction hierarchy of a HHMM, layer after layer. In this process, clustering algorithms and sequence alignment algorithms, widely used in domains like molecular biology, are exploited. The induction strategy has been designed in order to deal with events characterized by a sparse structure, where gaps filled by irrelevant facts can be intermixed with the relevant ones. Irrelevant facts are modeled by "gaps", i.e., HMMs of the noise. Gaps are hypothesized when there is no significant statistical evidence for hypothesizing the existence of a specific episode. Moreover, gaps can be replaced in a second time by a episode model, after new facts have been acquired. The method is extensively evaluated on artificial datasets.*

1 Introduction

Discovering patterns hidden in long temporal or spatial sequences is a fundamental task in many real word domains, and it is attracting increasing attention in the literature. However, the task is not easy and the difficulty increases along with the length of the sequences to be searched and the complexity of the patterns to be discovered. In this paper we propose a method for discovering the occurrence of complex events in long sequences. We assume that a complex event is a partially ordered set of short chains (episodes) of elementary events (instants), interleaved with gaps where irrelevant facts may occur. Moreover, we assume the presence of noise, which can make episodes hard to recognize. Episodes are represented as strings of symbols, being a symbol the label assigned to an instant. Then, noise may be modeled as insertion, deletion and substitution errors according to a common practice followed in Pattern Recognition. Typical applications where such kind of events are found are molecular biology and computer security. An approach to deal with such type of patterns, which reported impressive records of successes in speech recognition [8] and DNA analysis [1], is based on Hidden Markov Model (HMM) [9]). However, developing applications based on HMM does not reduce to running a learning algorithm but it may be a very costly process. In general, complex applications require to construct an ad hoc system, where several partial HMMs are developed and integrated with procedural knowledge obtained from experts of the domain. A more formal proposal to design and train complex HMMs is represented by the Hierarchical Hidden Markov Model (HHMM) [2]. The problem of estimation HMM and HHMM parameters has been widely investigated while little has been done in order to learn their structure. a fiew proposals can be found in the literature in order to learn the structure of HMM [11], or of a variant of it called logical HMM [5], but almost nothing has been done with respect to HHMM. This paper proposes a multi-strategy method for automating the construction of HHMM structure by induction from a set of sequences.

The learning method proceeds bottom-up by constructing step by step a hierarchy of stochastic automata, and is guided by the statistical evidence of episodes and of co-occurrence of episodes. One of the novelties in this paper is represented by a special construct called *gap*, used to model the presence of gaps in sequences.

In the framework of the main learning algorithm, classical algorithms for string alignment [1, 4], statistical clustering and HMM synthesis ($http : //hmmer.wustl.edu/$) are used.

The method has been evaluated on a suite of artificial datasets constructed in order to encode a progression of induction tasks of increasing difficulty.

[1] Dipartimento di Informatica, Universitá di Torino, C.so Svizzera 185, 10149 Torino, Italy email: botta@di.unito.it

[2] Dipartimento di Informatica, Universitá Amedeo Avogadro, Spalto Marengo 33, 15100 Alessandria, Italy email: galassi@mfn.unipmn.it

[3] Dipartimento di Informatica, Universitá Amedeo Avogadro, Spalto Marengo 33, 15100 Alessandria, Italy email: attilio.giordana@mfn.unipmn.it

2 The Approach to Learning HHMM

A Hierarchical Hidden Markov Model is a generalization of the Hidden Markov Model, which is a stochastic finite state automaton [9] defined by a tuple $\langle S, O, A, B, \pi \rangle$, where:

- S is a set of states, and O is a set of atomic events (observations),
- A is a probability distribution governing the transitions from one state to another. Specifically, any member $a_{i,j}$ of A defines the probability of the transition from state s_i to state s_j, given s_i.
- B is a probability distribution governing the emission of observable events depending on the state. Specifically, an item $b_{i,j}$ belonging to B defines the probability of producing event O_j when the automaton is in state s_i.
- π is a distribution on S defining, for every $q_i \in S$, the probability that s_i is the initial state of the automaton.

A first problem that arises, with a HMM defined in this way, is that, when the set of states S grows large, the number of parameters to estimate (A and B) rapidly becomes intractable.

A second problem is that the probability of a sequence being generated by a given HMM decreases exponentially with its length. Then, complex and sparse events become difficult to discover.

The HHMM proposed by Fine, Singer and Tishby [2] is an answer to both problems. On one hand, the number of parameters to estimate is strongly reduced by assigning a null probability to many transitions in distribution A, and to many observations in distribution B. On the other hand, it allows a possibly long chain of elementary events to be abstracted into a single event, which can be handled as a single item. This is obtained by exploiting the regular languages property of being closed under substitution, which allows a large finite state automaton to be transformed into a hierarchy of simpler ones.

More specifically, numbering the hierarchy levels with ordinals increasing from the highest towards the lowest level, observations generated in a state ${s_i}^k$ by a stochastic automaton al level k are sequences generated by an automaton at level $k+1$. Moreover, no direct transition may occur between the states of different automata in the hierarchy. An example of HHMM is given in Figure 1.

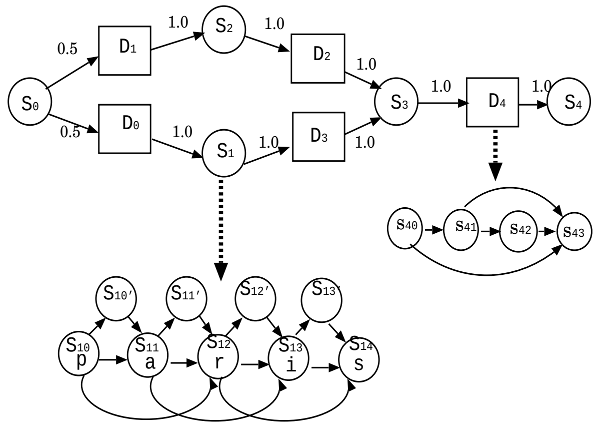

Figure 1. Example of Hierarchical Hidden Markov Model. Circles denotes states with observable emission, whereas rectangles denote *gaps*.

However, it is worth noticing that the hierarchy defined in HHMM is not essential, but it can always be flattened by replacing a state ${s_i}^k$ in an automaton at level k with the graph of the automaton at level $k+1$, which computes the observation generated in ${s_i}^k$ (substitution property for regular languages). Nevertheless, the hierarchical structure, as defined by [2], may help very much the inference of the entire structure of the automata by part of an induction algorithm.

Up to now the research efforts about HHMM concentrate on the algorithms for estimating the probabilities governing the emissions and the transition from state to state. In the seminal paper by Fine et al. [2], the classical Baum-Welch algorithm is extended to the HHMM. Then, in a more recent work, Murphy and Paskin [7] derive a linear (approximated) algorithm by mapping a HHMM into a Dynamic Bayesian Network. Finally, a recent paper by Xie et. al. [12] proposes a multi-strategy method for incrementally adjusting the structure of an HHMM by adding and deleting states on line. However, before starting the incremental learning procedure, an initial structure has still to be defined.

In this paper we will follow another approach inspired to a practice widely used to develop complex systems in pattern recognition applications. The basic algorithm is bottom-up and constructs the HHMM hierarchy starting from the lowest level. The first step consists in searching for possible episodes, i.e., short chains of consecutive symbols that appear frequently in the learning sequences, and building a HMM for each one of them. As episodes are considered independently one from another, this phase tends to produce also models for spurious episodes, which in a second time should be discarded. At the same time, it may happen that relevant episodes be disregarded just because their frequency is not high enough. However, such kind of errors will be fixed at a second time. The HMMs learned so far, are then used as feature constructors. Each HMM is labeled with a different *name* and the original sequences are rewritten into the new alphabet defined by the set of names given to the models. Every subsequence, which can be attributed to a specific HMM, is replaced by the corresponding name. We will discuss in Section 5 how possible conflicts are solved.

The subsequences between two episodes, not attributed to any model, are considered gaps and will be handled by means of special construct called *gap*. We will call this last operation *sequence abstraction*. After this basic cycle has been completed, an analogous procedure is repeated on the abstracted sequences. Models are now built for sequences of *episodes*, searching for long range regularities among co-occurrent episodes. In this process, spurious episodes not showing significant regularities can be discarded. The major difference, with respect to the first learning step, is that the models built from the abstract sequences, are now observable markov models. This makes the task easier and decreases the computational complexity. As explained in the next section, in this step, models (*gaps*) are built also for the long intervals falling between episodes.

In principle, we can think to apply again an abstraction step to the learning sequences and repeat the cycle, building a third level of the hierarchy, and so on. However, up to now, we considered only problems where two levels are sufficient.

After building the HHMM structure in this way, it can be refined using standard training algorithms like the ones proposed in [2, 7]. However, we propose other two refinement methods.

The first method concerns the recovery of episodes that have been lost in the primary learning phase because they did not have a sufficient statistical evidence. As said above, this missed information has actually been modeled by *gaps*. A nice property of the HHMM is that sub-models in the hierarchy have a loose interaction with one another, and so their structure can be reshaped without destroying the global structure. This means that the model of a gap can be turned into the model of an episode later on, when further data will be available. This is actually done on demand: all sub-sequences attributed to a given *gap* are collected, creating a new learning set where the learning process is repeated. If there is now evidence for an episode, a model is built up and replaced to the *gap*.

The second method consists in repeating the entire learning cycle using as learning set only the portion of the sequences where the instance of the previously learned HHMM has been found with sufficient evidence. Repeating the procedure allows more precise models to be learned for episodes, because false episodes will no longer participate to the learning procedure. In Section 6 we will give an example of the practical benefits due to this kind of refinement.

3 Dealing with gaps

A problem to solve in order to apply HMM to sparse events in very long sequences is the presence of long gaps filled by noise. In fact, the entire sequence including the gaps has to be explicitly modeled [1]. Moreover, the presence of long gaps tends to decrease the probability of events containing long gaps in favor of events containing short or no gaps. This may be in contradiction with many real world phenomena, where temporal intervals are required in order to let physical processes be completed.

We will exploit HHMM architecture to specifically model gaps. *Gaps* are HMMs designed (or inferred from data), allowing the distribution of gap durations to be tailored according to the position in

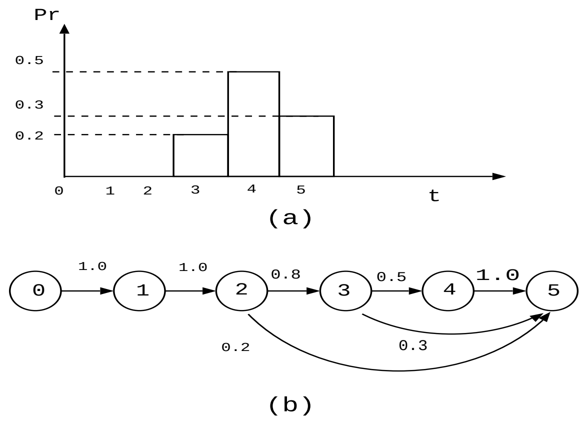

Figure 2. Hidden Markov Model for gaps. (a) Example of a probability distribution over the duration of a gap due to a physical process. (b) The left-right automaton that correctly models distribution (a).

the sequence. *Gaps* can be explicitly bound by supplying temporal constrains.

In many applications gap models have been used [1]. Typically they correspond to states with a self-loop where the emission is random, or no emission exists, depending on whether the gap contains noise or is it an empty interval. The drawback of this model is that the distribution probability on the gap length decays exponentially. In fact, this may be unrealistic when the gap duration reflects the time required in order to complete some physical process. In this case, the distribution tends to have a bell shape as, for instance, in Figure 2-(a). It is immediate to verify that left-to-right HMMs of the form described in Figure 2-(b) are able to model distributions of this type. In the following, we will consider two variants build on this structure. In the first variant, the emission in a state is a symbol randomly selected in O. This variant still has the drawback that the probability distribution of the gap length depends on a term that decreases exponentially with the duration. Nevertheless, the advantage is that this model can be immediately integrated in the HHMM and handled by the existing training algorithms. In the second variant, the emission is represented by the symbol '*' that means 'any' symbol in O. In other words, the probability of the emission '*' is 1 in all states, so that the probability distribution will completely be determined by the probability values of matrix A. This model can exactly capture a distribution like the one in Figure 2-(a), but it has the disadvantage of requiring special care during the training phase. In fact, increasing the length of a gap will implicitly increase the probability of the entire event, so that gap tend to be dominant. The solution that we adopt in this case consists in preventing Baum-Welch algorithm from training gaps, which will be estimated separately in a second time.

The distribution encoded by a HMM of the type described in Figure 2-(b) has inelastic bounds. In some cases, elastic bounds may be preferable. This can be obtained by adding self-loops to states.

4 Setting Temporal Constraints

Domain knowledge in the form of temporal constraints can be used to bias the induction process. More specifically, *inelastic* or *elastic* constraints on the duration of an episode, or of a gap, can be stated. Constraints may be generic, i.e, they apply to all episodes (or gaps), or specific, i.e., they apply to a specified episode (or gap). The different types of constraints are described in Table 1. Generic constraints

Table 1. Temporal constraints. Square brackets denote hard constraints; angle brackets denote soft constraints.

Hard form	Soft form	Comment
L(*) = $[T_1, T_2]$	L(*) = $\langle T_1, T_2 \rangle$	Episodes must/should have length between T_1 and T_2
D(*,*) = $[T_1, T_2]$	D(*) = $\langle T_1, T_2 \rangle$	Gaps between two episodes must/should have a duration between T_1 and T_2
L(E) = $[T_1, T_2]$	L(E) = $\langle T_1, T_2 \rangle$	The length of Episode E musts/should be between T_1 and T_2
D(E_1,E_2) = $[T_1, T_2]$	L(E_1,E_2) = $\langle T_1, T_2 \rangle$	The duration of gap between E_1 and E_2 must/should be between T_1 and T_2

are used to discard episodes that do not meet the assigned constraints on episode and gap duration.

Specific constraints can be used, on demand, to bias the structure of the HMM of an episode or of a gap. As discussed in the previous section, using left-to-right HMMs, the duration of an event (episode) is determined by the number of states that can be chained in process. Then, a given constraint can be satisfied by properly shaping the structure of the HMM. Soft constraints are satisfied by adding self loops to states.

5 The Learning Algorithm

As described in Section 2, the first step of the learning algorithm consists in searching for frequent episodes, and then building a HMM for each one of them. This basic step is followed by one or more steps in which the higher levels of a HHMM are constructed.

In this process local alignment algorithms [10] based on Levenstein's edit distance [6] are extensively used in order to discover episodes.

Let L be a set of learning sequences. The basic learning step is as follows:

1. For every different pair of sequence (l_1, l_2) in L find all local alignments between l_1 and l_2, having a sufficient statistical evidence, and collect them into a set S. These will be the potential episodes.
2. Apply a clustering algorithm to subsequences S using the Levenstein distance as distance measure. Clusters having a cardinality below a given threshold t_c are discarded.
3. For every retained cluster C_i construct a model M_i of the subsequences contained in it. To every model M_i give a symbolic name $\mathbf{M}_i$.
4. Construct an abstract alphabet Σ containing all names $\mathbf{M}_i$ given to the automata constructed in the previous step.
5. Abstract every sequences $l_i \in L$ using the alphabet Σ.

HMMs of the episodes are constructed following a procedure widely used in molecular biology: Let C_i be a cluster of subsequence generated by a local alignment algorithm:

1. Construct the multiple alignment MA$_i$ among all subsequences in C_i
2. Convert MA$_i$ into a left-to-right hidden markov model HMM$_i$. Depending on the given constraints a different model type may be chosen.
3. Estimate the parameters λ of HMM$_i$ on the sequences in C_i.

The algorithm for the abstraction step is as follows. Firstly, every sequence in L is processed searching for subsequences corresponding to instances of the HMMs constructed in the previous step. Everywhere an instance of model HMM_i is found, a hypothesis h_i = ($\mathbf{M}_i$, b, e, p) is emitted, being $\mathbf{M}_i$ the symbol in Σ associated to HMM$_i$, b the instant where h_i begins in the original sequence, and e

the instance where it ends; p is the probability for h_i being generated by HMM$_i$. In this way, for every sequence l a lattice of hypotheses is obtained. Afterward, lattices are processed extracting from each one the sequence, which includes the most likely hypotheses and is compatible with the given constraints. The default constraint is that hypotheses must not overlap. Finally, every sequence is transformed again in a string of symbols in order to be able to process it with standard local alignment algorithms in the cycle that will follow. Gaps between consecutive items in the sequences are represented using special symbols. In order to account for gap length avoiding to explicitly introduce numerical information, gaps are subdivided into broad categories depending on their length. Every category is then labeled with a different symbol. The edit distance measure is defined in order to account for the numeric distance among the different categories.

The further steps in order to construct the higher levels in the hierarchy resemble to the basic one, except for the construction of the model from abstract representation of the sequences. In this case, as previously noticed, the model is an observable Markov model. Again, a left-to-right automaton is constructed, which may contain self loops, or not, depending on the constraints. The transition probabilities of the automaton are estimated from the sequences. The procedures currently used for building the multiple alignment and for constructing the model are standard ones taken from [4, 1].

The procedure for constructing the automaton also takes care of gap models. Firstly, an automaton M containing only emission states is constructed (see Figure 1). Then, for every pair of consecutive states (s_i, s_j) in M, the gaps occurring in the sequences between the emission of s_i and s_j are collected. In this way the distribution histogram of the gap duration is obtained and the corresponding *gap* HMM is constructed. Finally, the automaton M is extended by inserting the states corresponding to gaps.

6 Evaluation

The algorithm has been evaluated using artificial data generated according to a specific test procedure in order to monitor its ability at discovering "known patterns" depending on the difficulty of the problem. A problem is represented by a dataset containing sequences of symbols, where a complex event, generated by an hand-crafted HHMM[4], is hidden. Random noise and spurious episodes have been added to the all sequences filling the gaps between consecutive episodes.

6.1 Discovering Chains of Episodes

The HHMMs used in a first group of experiments generates sequences of names of towns, in a predefined order. The HHMM also models noise in the data, in form of insertion, deletion and substitution errors. The gaps between the names are filled by symbols randomly chosen in the alphabet defined by the union of the letters contained in the names. Moreover, random subsequences, up to 15 characters long, have been added at the beginning and the end of each sequence. The global length of the sequences ranges from 60 to 120 characters. The difficulty of the task has been controlled by varying the degree of noise. Two sets of experiments have been designed in this framework. In the first one, a sequence of problems has been generated varying the number of words ($5 \leq w \leq 8$), the word length ($5 \leq L \leq 8$) and the noise level ($N \in \{0\%, 5\%, 10\%, 15\%\}$. For

[4] HHMMs with a two level structure have only been used in this evaluation.

Table 2. Performances obtained in the first set of experiments. The sequence length ranges from 60 to 140 characters. The CPU time, for solving a problem, ranges from 42 to 83 seconds on a Pentium IV 2.4Ghz.

		Noise Level			
w	L	0%	5%	10 %	15%
5	5	0.00	0.02	0.02	0.02
5	6	0.06	0.12	0.12	0.09
5	7	0.00	0.01	0.01	0.05
5	8	0.00	0.03	0.02	0.04
6	5	0.06	0.01	0.04	0.04
6	6	0.02	0.10	0.06	0.19
6	7	0.00	0.03	0.02	0.05
6	8	0.00	0.04	0.05	0.05
7	5	0.02	0.05	0.11	0.07
7	6	0.01	0.10	0.05	0.14
7	7	0.00	0.06	0.02	0.05
7	8	0.01	0.06	0.09	0.09
8	5	0.00	0.00	0.01	0.10
8	6	0.03	0.08	0.10	0.14
8	7	0.00	0.01	0.01	0.08
8	8	0.01	0.03	0.08	0.09

every triple $< w, L, N >$, 10 different datasets has been generated for a total of 640 learning problems.

The second set of experiments aimed at monitoring the ability of the algorithm at discovering hidden events, when relevant sequences are mixed with irrelevant ones. Starting from the datasets generated for the first experiment, new detasets are obtained by replacing a percentage (from 10% to 50%) of sequences with others not containing instances of the target event.

The most important results are summarized in Table 2. The error rate is evaluated as the edit distance (i.e. the minimum number of corrections) between the maximum likelihood sequence (maximum consensus) generated by the Viterbi algorithm [3] from the original HHMM and the one generated from the learned HHMM. When, an entire word is missed, the corresponding error is set equal to the its length. Experiments in table 2, reporting an error rate much higher than the others, have missed words. In all cases, the learning cycle has been iterated twice, as explained in Section 5. The average error rate after the second swept decrease of about 50% with respect to the first one.

Table 3. Performances obtained in the second set of experiments. The noise percentage on single sequence is 10%, whreas the percentage of irrelevant sequences range from 10% to 50% The values have been averaged on the word length L.

	Irrelevant Sequences (%)				
w	10%	20%	30%	40%	50%
5	0.24	0.15	0.22	0.06	0.14
6	0.17	0.10	0.11	0.08	0.16
7	0.15	0.18	0.20	0.12	0.19
8	0.14	0.14	0.17	0.21	0.16

From Table 2, it appears that the model extracted from the data without noise is almost error free. Moreover, the method seems to be little sensitive with respect to the sequence length while the error rate roughly increases proportionally to the noise in the original model (the 15% of noise corresponds to an average error rate of about 19%).

The experimentation related to the second group of experiments is reported in Table 3. It appears that the algorithm is quite insensitive to the presence of irrelevant sequences, provided that its proportion does not dominate the relevant ones.

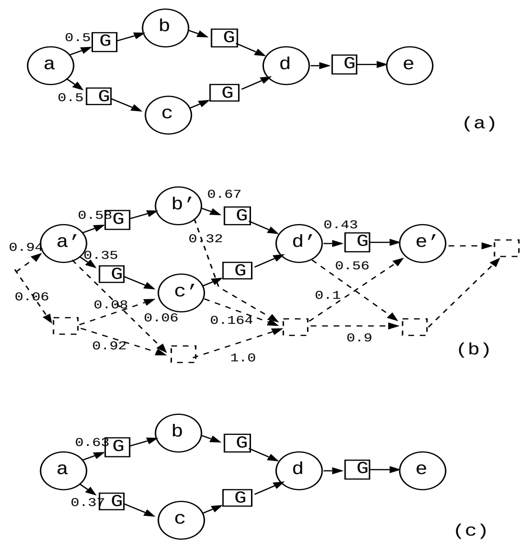

Figure 3. Example of HHMM inferred by the algorithm. (a) The higher level of the target HHMM. (b) The model inferred at the first sweep. (c) The final automaton after refinement.

6.2 Discovering a Partially Ordered Set of Episodes

A third group of experiments is designed to check the ability of the method at discovering partially ordered sets of episodes (i.e. learning a higher level automata defined by a graph).

The experimentation has been run with many different patterns. For the sake of space we will restrict to describe an example among the most significant ones. Every generated dataset contains 330 sequences. The 90% of the sequences contain an instance of a target HHMM that should be discovered by the learning program, whereas the 10% contain sequences of spurious episodes non generated by the target HHMM. The sequence length ranges from 80 to 120 characters.

The target HHMM used to generate the datasets has a structure described in Figure 1. In order to make the task misleading, the gaps have been filled with subsequences containing random noise. To this purpose *gap* models with random emission in the states have been used. Both the HMMs of episodes, in the target HHMM, and the model of the spurious episodes have been constructed in order to have a measurable distance among them. In this way the difficulty of the task can be evaluated and predetermined.

The distance measure between two HMMs is the one described by Rabiner in [9]

$$D(\lambda_1, \lambda_2) = \frac{1}{T|X|} \sum_{x \in X} [log(p_2(x) - log(p_1(x))] \qquad (1)$$

In other words the distance $D(\lambda_1, \lambda_2)$ between two HMMs is given by the average of the ratio between the probability in λ_2 and the probability in λ_1 for sequences x generated in λ_1. T is a parameter that has been set to the value of the length of sequence x. Notice that the distance is not symmetric.

To generate a HMM λ_2 having a predefined distance from another HMM λ_1 the following procedure has been used. Let D be a predefined distance value. Let moreover $\hat{X}$ a stochastic sampling of the set X of observations that can be generated by λ_1. Model, λ_2 is modified by performing the gradient ascent/descent in the parameter space of λ_2 until the desired value D is obtained.

In the test reported in Figure 3, six different models have been used, generated by the above algorithm starting from slightly different structures. Models A, B, C, D, E are used to build up the target HHMM. Model F is used to generate pseudo episodes in misleading sequences.

The distance matrix among the HMMs is reported in Table 4

Table 4. distance matrix among the HMMs used to generate episodes. A, B, C, D, E model the episodes in the corresponding states a, b, c, d, e in Figure 3. F is the model of the episods in pseudo sequences.

	A	B	C	D	E	F
A	0.0	∞	2	2	0.47	0.7
B	∞	0.0	0.94	1.16	1.80	2.48
C	∞	0.50	0.0	0.14	2	2.9
D	∞	0.23	∞	0.0	1.43	2.6
E	0.04	∞	1.8	1.58	0.0	0.6
F	0.8	∞	2	2.1	0.75	0.0

As it appears from Figure 3, the structure of the hidden model already emerged after the first sweep, and has been reconstructed with a reasonable approximation after a refinement step. The CPU time ranges from 395 to 1370 seconds on a Pentium IV 2.0 GHz depending on the problem instance.

7 Conclusion

We have proposed a method for inferring, from data and domain knowledge, a HHMM for complex events. The learning algorithm is multi-strategy, and is guided by statistical evidence and regularities discovered in sequences, and is easy to integrate with existing algorithm for training HHMM parameters. In preliminary tests on artificial datasets, the method succeeded in reconstructing two level HHMMs without the help of domain knowledge.

REFERENCES

[1] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis*, Cambridge University Press, 1998.

[2] S. Fine, Y Singer, and N. Tishby, 'The hierarchical hidden markov model: Analysis and applications', *Machine Learning*, **32**, 41–62, (1998).

[3] G. D. Forney, 'The viterbi algorithm', *Proceedings of IEEE*, **61**, 268–278, (1973).

[4] D. Gussfield, *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, 1997.

[5] K. Kersting, T. Raiko, and L. De Raedt, 'A structural gem for learning logical hidden markov models', in *Working Notes of the Second KDD-Workshop on Multi-Relational Data Mining (MRDM-03)*, Washington, DC, USA, (August 2003).

[6] V.I. Levenstein, 'Binary codes capable of correcting insertions and reversals', *Soviet. Phys. Dokl.*, **10**, 707–717, (1966).

[7] K. Murphy and M. Paskin, 'Linear time inference in hierarchical hmms', in *Advances in Neural Information Processing Systems (NIPS-01)*, volume 14, (2001).

[8] L. Rabiner and B. Juang, *Fundamentals of Speech Recognition*, Prentice Hall, Englewood Cliffs, NY, 1993.

[9] L.R. Rabiner, 'A tutorial on hidden markov models and selected applications in speech recognition', *Proceedings of IEEE*, **77(2)**, 257–286, (1989).

[10] T.F. Smith and M.S. Waterman, 'Identification of common molecular subsequences', *Journal of Molecular Biology*, **147**, 195–292, (1981).

[11] A. Stolcke and S. Omohundro, 'Hidden markov model induction by bayesian model merging', *Advances in Neural Information Processing Systems*, **5**, 11–18, (1993).

[12] L. Xie, S. Chang, A. Divakaran, and H. Sun, *Learning hierarchical hidden Markov models for video structure discovery*, volume Tech. Rep. 2002-006, ADVENT Group, Columbia University, December 2002.