Taming the Zoo of Discrete HMM Subspecies & Some of their Relatives

Henning CHRISTIANSEN, Christian Theil HAVE, Ole Torp LASSEN, Matthieu PETIT

PLIS-CBIT, Roskilde University, Denmark

Abstract. Hidden Markov Models, or HMMs, are a family of probabilistic models used for describing and analyzing sequential phenomena such as written and spoken text, biological sequences and sensor data from monitoring of hospital patients and industrial plants. An inherent characteristic of all HMM subspecies is their control by some sort of probabilistic, finite state machine, but which may differ in the detailed structure and specific sorts of conditional probabilities. In the literature, however, the different HMM subspecies tend to be described as separate kingdoms with their entrails and inference methods defined from scratch in each particular case. Here we suggest a unified characterization using a generic, probabilistic-logic framework and generic inference methods, which also promote experiments with new hybrids and mutations. This may even involve context dependencies that traditionally are considered beyond reach of HMMs.

Keywords. Probabilistic models; sequence analysis, Hidden Markov Models, context dependencies.

Introduction

Discrete[1] Hidden Markov Models (HMMs) have become a very popular tool used for the modeling and analysis of a variety of sequential phenomena, e.g., biological sequence data, shallow text analysis and speech recognition; see, e.g., [2,3] for overview and background. A HMM is a probabilistic finite state machine that produces sequences of symbols from some finite alphabet.

In a typical application, the state machine is intended as a (simplified) reconstruction of some sort of system, whose internal details typically cannot be inspected, and the sequence of emission symbols represents the observable behaviour of that system. For DNA analysis, for example, we may consider an actual genome sequence as being produced by an informed typist who, in a probabilistic way, decides when to enter states that corresponds to typing, say promoter sub-sequences, operons, genes etc., or intergenic subsequences whose contents is debated.

One of the advantages of HMMs is that analysis of a sequence can take place in linear time as a function of the sequence length. The price, however, is a lack of sophistication, as the underlying sequence languages are regular [4]. Basically, the only

[1]We ignore continuous HMMs completely as they do not fit into the probabilistic-logic framework that we rely on in this paper; see, e.g., [1] for a definition.

memory available at a given time in the run of a state machine is knowledge about the current state; in its simplest form, a HMM has a probability table for each state to govern transition to the next state and a table for the emission. Subspecies of HMMs may differ by having the probabilities conditioned by one or more previously passed states and-or emissions; in this case we talk about higher-order HMMs. Other distinctive features can be different ways of structuring the state machine, e.g., by product or hierarchical structures of separate machines and other ways that are considered below.

The following three ways of reasoning with HMMs are essential for practical applications:

- Analysis of a sequence S, also called ***prediction***, means to identify the most probable path of states that the machine may pass through in order to produce S; prediction is typically made using some adaptation of the Viterbi algorithm [5], which is a dynamic programming algorithm that reads the sequence symbol by symbol and keeps track of one best path up to each possible state. Its time complexity is known to be $\mathcal{O}(n\,b^2)$ where n is the length of the sequence and b the number of machine states.
- The probabilities which, thus, determine which path is preferred, can be set manually but more interestingly produced by ***training*** from known sequences using machine learning techniques, e.g., the EM algorithm [6], that we shall not describe further.
- Finally, ***sampling*** is a process of using the HMM to produce sequences that are representative for the distribution defined by the HMM.

In the present paper, we propose a uniform treatment of different subspecies of discrete HMMs, showing how they can be defined in very concise ways using state-of-the-art probabilistic-logic programming as represented by the PRISM system [7,8]. In addition, we obtain for free, implementations of prediction, training, and sampling, relying on PRISM's generic built-in facilities. We will also show examples of models that go beyond what can be considered as HMMs in a strict sense, such as probabilistic context-free grammars and definite clause grammars as well as context-sensitive extensions to HMMs, can be treated in similar ways. Our reasons for writing this paper can be summarized as follows.

- It can be a bit of a nuisance trying to get an overview of the literature in the field, as there seems to be a trend that each paper defines from scratch its own probability calculations, adaptations of Viterbi and learning algorithms, and implementation principles – and it is left to the reader to figure out that all these anyhow are instances of a common pattern. We hope that the present exposition may contribute to a better overview.
- We want to relieve the researcher who wants to apply, design or experiment with alternative sequential models from wasting valuable time on tiresome programming in imperative languages and on learning the idiosyncrasies of different, specialized software, anyhow doing more or less the same thing.
- We want to emphasize the advantages of using probabilistic-logic programming in teaching, as a uniform theoretical treatment and implementation framework for these inherently related models.

As an example of how this unification can be exploited, it is fairly straightforward to set up a testbed for comparing how well different models perform with respect to precision and recall for the same collections of training and validation as done, e.g., by [9].

We should emphasize that the principle of implementing plain HMMs in PRISM is not our invention but has been used as an example, e.g., in the PRISM User's manual [8]. Our formulation below may be a bit more elegant for the plain HMM case, as this has been a particular goal for us, and most of the variations that we unfold have not been investigated systematically before in PRISM.

We expect a basic knowledge of probability theory and elementary Prolog programming. For reasons of space, we do give all details of all our models, but only highlight the essential fragments; the website [10] lists all models in full text and explains how they can be executed efficiently.

Below, in section 1, we give as background a standard definition of HMMs and their probabilities. Section 2 provides a compact introduction to PRISM and shows how a plain HMM can be defined in a very concise way together with minor extensions with silent states and duration. The following sections compose a stroll through the HMM Zoo Garden, showing different subspecies properly kept in order by definitions in PRISM; sec. 3 shows how transitions and emissions may be conditioned by events in the past, so-called higher-order HMMs; sec. 4 shows HMMs that concern several sequences at a time, e.g., for alignment, and different ways that multiple HMMs can form symbioses; sec. 5 goes a step further showing how context-dependencies beyond regular and even context-free languages can be characterized as direct extensions of plain HMMs. Finally, section 6 mentions other sequence models that can be described in PRISM, but not treated in details here, such as probabilistic versions of context-free and definite clause grammars; sec. 7 mentions a few related works, and sec. 8 gives a brief summary and conclusion.

1. Definition of a Common Ancestor: 1st Order HMMs

We define here the most primitive subspecies of Hidden Markov models, in which transition and emissions probabilities are conditioned by the current state only. For simplicity of the definition, we assume one fixed initial state.

Definition 1 *A* Hidden Markov Model *(HMM) is a quadruple* $\langle \Sigma, A, T, E\rangle$*, where*

- Σ *is a finite set of* states*, one of which is distinguished as* initial *and one or more as* final *states, and* A *a finite set of symbols called the* emission alphabet*;*
- T *is a set of* transition probabilities $\{p(s_i; s_j)\}$ *for pairs of states, with* s_i *not final, such that for each* s_i, $\sum_{s_j} p(s_i; s_j) = 1$*;*
- E *is a set of* emission probabilities $\{p(s_i; e_j)\}$ *for pairs of state, with* s_i *not final, and letters* $e_j \in A$ *such that for each* s_i, $\sum_{e_j} p(s_i; e_j) = 1$.

A run *for a given HMM is defined is as a pair of sequences of states* $s_0 \cdots s_{n+1}$ *and letters* $e_0 \cdots e_n$ *where* s_0 *(*s_{n+1}*) is an initial (final) state,* $p(s_i; s_{i+1}) > 0$ *and* $p(s_i; e_i) > 0$*;* $s_0 \cdots s_{n+1}$ *is called a* path *for* $e_0 \cdots e_n$.

The probability *of a given run* R *is defined as follows, using the notation above.*

$$P(R) = \prod_{i=0}^{n} p(s_i; s_{i+1}) \prod_{i=0}^{n} p(s_i; e_i)$$

Notice that the set of non-zero transition probabilities defines the underlying finite state machine of a specific HMM. For an overview of inference methods for such HMMs, including prediction and training, see, e.g., [2].

In the next section we introduce the generic probabilistic-logic framework PRISM and show how a definition of standard HMMs in PRISM is a realization of the semantics given by def. 1. For the remaining part of this paper, we will take a specification in the PRISM language as *the* definition of a subspecies.

2. Probabilistic-Logic Modelling in PRISM

The PRISM language [7,8] extends Prolog with so-called multi-valued switches: a call `msw(`*name*`, X)` represents a probabilistic choice of value to be assigned to `X`. A switch is introduced by a declaration of the form

`values(`*name*`, [`$\cdots$ *outcomes* $\cdots$`])`

and defines a family of random variables, one for each execution of `msw(`*name*`,` $\cdots$`)` in a program run. The name can be parametric — as we will show in the example below — in a way that makes it possible to define conditional probabilities.

The probability table associated with an `msw` can be defined by an explicit declaration or be produced by learning as we show below. The overall semantics of a PRISM program is given as a probabilistic Herbrand model, in which any logical atom has a probability of being true, given as the product of the probabilities for the `msw`s applied in a proof tree for that atom. For this semantics to be well-defined, any choice point in the program must be governed by an `msw`.

We use the example below both to show the implementation of a HMM in accordance with def. 1 and to explain further details about the PRISM system.

2.1. Lesson 1: First-order HMMs in PRISM

Let us consider a play of heads-and-tails in which the cashier has two coins, an honest and a dishonest one. At each time step, he will throw the current coin and decide which next step to take, i.e., to pick one of the coins or close the game. This may be described as a HMM whose states are {`honest`, `dishonest`, `close`} where the initial one is `honest` and the final one `close`; the emissions are {`head`, `tail`}. We can encode this HMM in the following `msw` declarations and probability settings, written as they may appear in a PRISM source file.

```
values(trans(_CurrentState),[honest,dishonest,close]).

values(emit(_CurrentState), [head,tail]).

:- set_sw(trans(honest), [0.4,0.5,0.1]),
   set_sw(trans(dishonest), [0.2,0.7,0.1]),
   set_sw(emit(honest),[0.5,0.5]),
   set_sw(emit(dishonest),[0.7,0.3]).
```

Notice in the values declarations, that the names are parameterized by a variable (starting with an underscore) meaning that for whatever term is substituted for that variable, there exists an msw. The set_sw directives set up distributions for the instances of trans(_) that appear in program below, i.e., trans(honest) and trans(dishonest), and emission probabilities for the two non-final states. Initial and final states can be defined in terms of Prolog predicates as follows.

```
initial(honest).
final(close).
```

As it may appear, the format shown so far can be applied to encode any individual HMM according to def. 1. The following lines of PRISM code provide a general definition of a HMM encoded in this way; i.e., if you want to work with another HMM, you just need to change the values and set_sw declarations. It is tail recursive and it can be understood as a generative specification of all possible runs and their probabilities.

```
hmm(Sequence,Path):- initial(S0), hmm(Sequence,S0,Path).

hmm([],Final,[Final]):- final(Final).

hmm([A|As],S,[S|Ss]):-
   \+final(S),
   msw(emit(S),A),
   msw(trans(S),Snext),
   hmm(As,Snext,Ss).
```

The top predicate hmm/2 defines a probability distribution for the runs of this HMM, so for example $P($[head,head], [honest,dishonest,close]$) = 0.5 \times 0.5 \times 0.1 \times 0.7 = 0.0175$, formed as the product of msw probabilities used for generating this particular run. In fact, [honest,dishonest,close] is the Viterbi path for [head,head].

The PRISM system provides the necessary tools for inference.

Sampling is the simplest; the program is executed as a plain Prolog program with the outcome of each msw determined by pseudorandom numbers. Time complexity is linear in the length of the generated sample.

Prediction can be made using one of PRISM's built-in Viterbi algorithms. The call viterbig(hmm([head,head],Path)) will instantiate the logical variable Path to the path that provides the maximum probability for the call. Time complexity depends on the details of the model; HMMs can run in linear time.[2]

Training: PRISM comes with several built-in algorithms for supervised and unsupervised learning; supervised learning may, e.g., be performed from a large file of ground atoms such as hmm([head,head], [honest,dishonest, close]) and this will replace the explicit set_sw directives shown above; we refer to the PRISM manual for details [8]. Time complexity depends in a non-trivial way on the complexity of the program and the size of the training data.

[2]A program transformation technique described in [11] is needed to have this Viterbi computation for a HMM run in linear time; this is explained at our website [10] that contains all program examples shown in the present paper. A forthcoming release of PRISM is expected to incorporate this technique, cf. [12], so we ignore this issue for the remainder of this paper.

All examples shown in the present paper can be trained in reasonable time from realistic training data.

The main advantage of using a system such as PRISM is that one and the same specification, such as the few program lines shown above, implements the different reasonings in one go.

2.2. *Minor variations: Silent states, duration modeling*

Here we indicate a few local varieties of the basic HMM species that are often encountered in the literature. A *silent state* is one that does not produce any emission symbol. Let us change the heads-and-tails model a little, so it also takes into account that the cashier, when he is using the honest coin may decide to ignore an outcome (that does not fit his interests). This is done as follows, using an additional outcome of the `msw` for `emit(honest)` and modify the code a bit so the "interpretation" of `silent` is to add nothing to the emission sequence.[3] We have used here Prolog's conditional expression *test* -> *if-true-action* ; *if-false-action*.

```
values(emit(dishonest), [head,tail]).
values(emit(honest), [head,tail,silent]).
...
hmm(As1,S,[S|Ss]):-
   \+ final(S),
   msw(emit(S),A),
   msw(trans(S),Snext),
   (A=silent -> As1=As ; As1 = [A|As]),
   hmm(As,Snext,Ss).
```

Duration in a HMM means that it can stay for a number of steps in a given state before going to a next state, and with a certain distribution for this number. It is well-known that basic HMMs have problems modeling duration[4] and the literature is rich in contortions to the layout of the state machine to compensate for this using duplicated states, etc.; see, e.g., [2]. In our framework, we can model duration in the most natural way, namely by selecting a number in a probabilistic way, and iterate an emission from the given state that number of times. For the heads-and-tails example, we may like to model that the cashier uses a chosen coin about 6 times in a row with a little variation, as follows. Notice that we need to prevent transition from a state to itself after the loop.

```
values(duration,[4,5,6,7,8]).
values(trans(honest),[dishonest,close]).
values(trans(dishonest),[honest,close]).
...
:- ... set_sw(duration,[0.15, 0.2, 0.3, 0.2, 0.15]).
...
```

[3]The current version of PRISM loops when using the silent state programs for prediction due to the existence of infinite paths. A new, generalized Viterbi algorithm for PRISM is under development which avoids this problem. The website for this paper [10] provides a minor modification of the program that works also under the present system.

[4]Using a possible transition from a state to itself in a plain HMM gives a geometric distribution of the possible durations.

```
hmm(As,S,Ss):-
   \+final(S),
   msw(duration,T),
   iterateHmm(T,As,S,Ss).

iterateHmm(0,As,S,Ss):-
   msw(trans(S), Snext),
   hmm(As,Snext,Ss).

iterateHmm(T,[A|As],S,[S|Ss]):-
   T > 0, T1 is T-1,
   msw(emit(S),A),
   iterateHmm(T1,As,S,Ss).
```

More sophisticated patterns can be defined with different length distributions for different states.

3. Single Sequence HMMs with dependencies on a portion of the past

The term *higher-order HMM* refers traditionally to different varieties of a subspecies that extends the conditioning of the transition and-or emission probabilities with information about the past.

The basic HMM definition shown in section 2.1 can easily mutate into higher-order, adding extra arguments to the `hmm` predicate and extra degrees of freedom in the `msw` definitions. We illustrate this for a so-called 2nd order HMM in which the transition probabilities are now conditioned, not only by current state, but also the previous; the emissions are unchanged in this example but can also be conditioned in a similar way. The symbol `border` imitates a dummy state before the initial state.

```
hmm(Sequence,Path):- initial(S0), hmm(Sequence,S0,border,Path).

hmm([],Final,_,[Final]):- final(Final).

hmm([A|As],S,Sprevious,[S|Ss]):-
   \+final(S),
   msw(emit(S),A),
   msw(trans(S,Sprevious),Snext),
   hmm(As,Snext,S,Ss).
```

When, e.g., the HMM goes through a path $s_0, s_1, \ldots$, the sequence of calls to the recursive `hmm` predicate can be sketched as follows.

$$\texttt{hmm(}\cdots s_0\texttt{,border}\cdots\texttt{),hmm(}\cdots s_1\texttt{,}s_0\cdots\texttt{),hmm(}\cdots s_2\texttt{,}s_1\cdots\texttt{),}\ \cdots$$

While the 1st order is suited to be trained to learn combinations of two emissions letters in a row, an nth order is suited to learn patterns of $n + 1$ letters.

We expect the general principle to be clear by now, so that the reader may experiment with his or her own varieties of higher order HMMs that utilizes different portions of the past in various ways.

4. Multi-sequence and combined HMMs

4.1. HMMs for Sequence Alignment

A HMM can be used to align two or more sequences. This application of HMMs have been particularly successful in computational biology, where an alignment of biological sequences can be used to draw conclusions about the evolutionary process. In the following we sketch a pair HMM [13] (see [2] for a presentation which is easier to compare with other literature in the field), which aligns two sequences, A and B, by simultaneously emitting them. Besides the silent `end` state, the pair HMM has three states; the `match` state aligns the next two symbols from each sequence to each other, the `insert` state aligns the current symbol of sequence A to a gap in sequence B and the `delete` state aligns the current symbol of sequence B to a gap in sequence A. The model is not fully connected as it is not possible to visit the `delete` state immediately after the `insert` state and vice versa.

```
values(trans(match),[match,delete,insert,end]).
values(trans(delete),[match,delete,end]).
values(trans(insert),[match,insert,end]).
initial(match).
final(end).
```

We consider here emissions over the alphabet $\{\mathtt{a},\mathtt{c},\mathtt{g},\mathtt{t}\}$. For simplicity, we characterize emissions in a uniform way as pairs, with "–" representing a missing character, e.i., `pair(a,a)` is a typical emission from a match state and `pair(a,-)` a typical emission from an `insert` state. In the `set_sw` directives, we set the probability of `emit`(x,y), $x \neq y$, in the `match` state to almost zero, and the probability of `emit`(x,y), $y \neq$ "–" to exact zero for `insert`, and analogously for the `delete` state. We simplify the main predicate using an auxiliary programmed in Prolog.

```
first(-,Ls,Ls):- !.
first(L,Ls,[L|Ls]).
```

The recursive specification of the pair HMM is now straightforward as follows.

```
hmm(Seq1,Seq2,Path):- initial(S0), hmm(Seq1,Seq2,S0,Path).

hmm([],[],Final,[Final]):- final(Final).

hmm(As1,Bs1,S,[S|Ss]):-
   \+ final(S),
   msw(emit(S),pair(A,B)),
   first(A,As,As1), first(B,Bs,Bs1),
   msw(trans(S),Snext),
   hmm(As,Bs,Snext,Ss).
```

4.2. Hierarchical HMMs

A hierarchical HMM [14] is similar to a 1st order HMM, but instead of emitting probabilistically a letter from each state, a sub-model is selected. Each sub-model is an ordinary HMM, which produces a sequence of letters until the control is given back to the top level. Thus a string produced by a hierarchical HMM is a concatenation of strings produced by different sub-models. The following structure assumes that the states in the different sub-models form disjoint subgraphs; in the choice among sub-models, their initial states serve also the purpose of identifying them. Notice that the `subHmm` predicate carries a state for the top level HMM which is "jumped to" when a sub-model reaches its final state.

```
hmm(Sequence,Path):- initial(S0), hmm(Sequence,S0,Path).

hmm([],Final,[Final]):- top_final(Final).

hmm(As,TopS,[TopS|Ss]):-
   \+ top_final(TopS),
   msw(submodel(TopS),SubInitS),
   msw(trans(TopS), TopSnext),
   subHmm(TopSnext,As,SubInitS,Ss).

subHmm(TopS,As,S,Ss):-
   sub_final(S),
   hmm(As,TopS,Ss).

subHmm(TopS,[A|As],S,[S|Ss]):-
   \+ sub_final(S),
   msw(emit(S),A),
   msw(trans(S),Snext),
   subHmm(TopS,As,Snext,Ss).
```

The `subHMM` predicate is similar to the basic HMM definition of in section 2.1 with the only difference that when it reaches its final state, it makes a recursive call to the `topHMM` state referred to by the variable `TopS`, rather than stopping.

A nontrivial application of hierarchical HMMs defined in PRISM for testing gene-finders has been made by [15].

4.3. Factorial HMMs

This term, introduced by [16], refers to a subspecies whose finite state machine is defined as the product of two or more state machines. A factorial HMM can be mapped into a plain HMM (whose state set is the product of the individual state sets), but it has the advantages that there are fewer transitions and probabilities that need to be specified: if two sub-machines has n, resp., m states, the factorial HMM includes at most $n^2 + m^2$ different transition probabilities, compared with the corresponding plain HMM that needs up to $n^2 \times m^2$ transition probabilities. The emissions are conditioned by the states of both sub-machines. This is straightforward to incorporate into a PRISM specification, we just need to pass two state variables around and determine the new state for both sub-machines in each step.

```
hmm(Sequence,Path):-
  initial1(S10), initial2(S20), hmm(Sequence,S10,S20,Path).

hmm([],Final1,Final2,[(Final1,Final2)]):-
    final1(Final1) ; final2(Final2).
```

```
hmm([A|As],S1,S2,[(S1,S2)|Ss]):-
   \+final1(S1), \+final2(S2),
   msw(emit(S1,S2),A),
   msw(trans1(S1),S1next),
   msw(trans2(S2),S2next),
   hmm(As,S1next,S2next,Ss).
```

4.4. Bayesian Coupled HMMs

Recent work on sequence analysis using PRISM for biological sequence analysis [17] shows how PRISM models can be put together in a Bayesian network, such that the resulting analysis from one model is used for conditioning the probabilities of other models. This applies also to the special sort of PRISM models that are HMMs and gives rise to yet another highly specialized HMM subspecies (symbiosis may be a better term) called Bayesian Coupled HMMs. As an example, we consider a system for weather analysis, put together as a network with only two nodes. We assume sequences of observations being either `sun`, `rain`, or `snow`. This model composes two HMMs in the following way.

- `hmm1` is a plain HMM whose states represent temperature plus an end state {`minus`, `aboutzero`, `plus`, `end`};
- `hmm2` has states that represent wind speed plus an end state {`quiet`, `breeze`, `storm`, `end`}; it extends the HMMs that we have seen so far by having the transitions conditioned also by the states produced by another HMM, in this example `hmm1`.

The PRISM code for such a conditioned HMM is as follows.

```
hmm2(Sequence,Condition,Path):-
    initial2(S0),
    hmm2(Sequence,Condition,S0,Path).

hmm2([],_,Final,[Final]):- final2(Final).

hmm2([A|As],[C|Cs],S,[S|Ss]):-
   \+final2(S),
   msw(emit2(S),A),
   msw(trans2(S,C),Snext),
   hmm2(As,Cs,Snext,Ss).
```

The implementation described by [17], which is built on top of PRISM, applies a principle for prediction that runs prediction with PRISM's Viterbi algorithm for each model at time, fixing the path produced before sending it on to the subsequent models. For the example above, we can illustrate this by the following query.

```
?- Seq=[snow,sun], viterbig(hmm1(Seq,P1)),
                   viterbig(hmm2(Seq,P1,P2)).
...
P1 = [aboutzero,aboutzero,end]
P2 = [breeze,quiet,end]
```

The first model, `hmm1`, predicts a path given as `P1`, which then is given to the prediction with `hmm2`; notice that `hmm2` only makes sense as a probabilistic model when this argument is given as a ground list.

Putting different PRISM models together as Bayesian networks in this way yields both a way of decomposing complex models and a way to reduce computational complexity. Bayesian Coupled HMMs can be formed from any number of sub-models and conditioned in a variety of ways, including taking in signals produced by external, not necessarily probabilistic analyses; see the referenced paper for details.

5. Adding context-sensitive information

The use of Prolog as a backbone to tie together the different probabilistic choices in our models makes it fairly straightforward to express also some context-sensitive constraints: at any point in a sequence, information can be collected, passed further on via predicate arguments to any other point in the string and applied to restrict, or condition probabilistically, what can occur at that second point. We show here two examples.

5.1. Copying substrings: Pseudoknots

A pseudoknot is a phenomenon that may be observed in the tertiary structure of an RNA molecule. It can be explained syntactically as a pattern in which a certain subsequence, that we call the *glue zone*, appears later with inverted letters, but in the same order. Inversion means to interchange any ‘`a`’ with a ‘`t`’ and any ‘`c`’ with a ‘`g`’ and vice versa. The following is an example of such a pattern; the interesting subsequence and its repeated, inverted version are indicated by boxes.

$$\texttt{aaaaaaaaaaaaaa}\boxed{\texttt{agata}}\texttt{aaaaaa}\boxed{\texttt{tctat}}\texttt{aaaaa}$$

As is well-known from formal language theory, this language exceeds both regular and context-free languages; it is a context-sensitive language. We can describe this as a combination of HMMs and a deterministic predicate that inserts the copy string. In order to keep the predicate arguments simple, we use separate predicates for each indicated substring; the model is put together by the following predicates that are applied in the given order.

```
hmm1, hmmGlue, hmm2, copyGlue, hmm3
```

The first predicate is like a usual 1st order HMM, with the exceptions that when it gets to its final states, it continues with `hmmGlue` instead of stopping.

```
hmm1(As,SFinal,Path):-
   final(SFinal),
   initialGlue(SGlue0),
   hmmGlue(As,SGlue0,Path,[]).

hmm1([A|As],S,[S|Path]):- \+ final(S), ...
```

The second, `hmmGlue`, is structured in the same way, except that it builds a separate list of the letters generated (in inverted form), as to have them ready for the copying later.

```
hmmGlue(As,SGlueFinal,Path,Glue):-
   finalGlue(SGlueFinal),
   initial(S0),
   hmm2(As,S0,Path,Glue).

hmmGlue([A|As],S,[S|Path],Glue):-
   \+ finalGlue(S),
   msw(emitGlue(S),A),
   msw(transGlue(S),Snext),
   addInvertedLetter(Glue,A,Glue1),
   hmmGlue(As,Snext,Path,Glue1).
```

The `addInvertedLetter` predicate adds the indicated letter in inverted form at the end of the currently collected copy string; it can be defined in Prolog as follows.

```
addInvertedLetter(Glue,A,Glue1):-
   invert(A,Ainvert),
   append(Glue,[Ainvert],Glue1).

invert(a,t). invert(t,a). invert(c,g). invert(g,c).
```

There is no reason to show the predicate `hmm2` as it is a mere replicate of `hmm1` except that it continues to `copyGlue`, defined as follows, when it has done its job.

```
 copyGlue(As,Path,[]):-
   initial(S0),
   hmm3(As,S0,Path).

copyGlue([A|As],[copy|Path],[A|Glue]):-
    copyGlue(As,Path,Glue).
```

Note that there are no `msw` calls here as the predicate is deterministic, with the looping controlled by the last arguments that contains the string to be inserted. Finally `hmm3` is a standard HMM. A more sophisticated version of this model could allow mutations in the inserted copy, using a model similar to the pair HMM used for alignment in section 4.1.

The website [10] gives the full text of this extended HMM as well as an alternative, and more elegant, version based on difference lists. We refrain from bringing the latter here, as we do not expect our average Zoo guest be familiar with the programming technique of difference lists. An earlier version of this pseudoknot program was given in [18].

5.2. Constrained HMMs

The subspecies of Constrained HMMs (CHMMs) are defined as ordinary HMMs with side-constraints on the runs. Such constraints are easily defined in PRISM; let `con(`*sequence*`,`*path*`)` be a predicate considered as constraint which accepts some runs and fails on others. When `hmm` represents an HMM, the following PRISM definition defines a CHMM.

```
chmm(Sequence,Path):- hmm(Sequence,Path), con(Sequence,Path).
```

CHMMs have the interesting property that the sum of probabilities of the runs it produce may be < 1, and the remaining probability mass considered as garbage probability [19].

The disadvantage of the definition just shown is that prediction may be very slow, as a breaking of the constraint is not seen before all `msws` have been instantiated; furthermore, due to subtle technical reasons, PRISM's Viterbi algorithms cannot handle this sort of specification in a correct way in all cases. Constraints that can be checked incrementally are better suited for prediction as shown in the following example, and will actually work in the current PRISM system. We extend the head-and-tail HMM with the constraint that the dishonest coin may be used at most three times in a game; the new last argument of the recursive predicate counts the number of dishonest states encountered so far.

```
hmm(Sequence,Path):- initial(S0), hmm(Sequence,S0,Path,0).
...
hmm([A|As],S,[S|Ss], Count):-
   \+final(S),
   msw(emit(S),A),
   msw(trans(S),Snext),
   check_count(Snext,Count,Count1),
   hmm(As,Snext,Ss,Count1).

check_count(honest,Count,Count).
check_count(dishonest,3,_):- !, fail.
check_count(dishonest,Count,Count1):- Count1 is Count+1.
check_count(close,_,_).
```

The `check_count` written in plain Prolog takes care of the counting and fails when the count would exceed 3. Constrained HMMs are treated in depth in [20].

6. Other related species

Probabilistic context-free grammars are another known model for sequences that also can be described in PRISM; it is shown as an example in the PRISM User's Guide [8]. A probabilistic version of popular Prolog based Definite Clause Grammars based on PRISM has been demonstrated by [21]. Such models describe a given sequence using a tree-structured recursion, rather than the tail recursive style we have used here, and it is natural to use the technique of difference lists. In this respect, they are not natural descendants of HMMs but should be considered as their own species.

Notice that PRISM's language, being an extension of Prolog, is Turing-complete, which means that PRISM can describe probabilistic versions of any recursively enumerable language (nothing said here about the efficiency of reasoning).

7. Related work

Dynamic Bayesian Networks (DBNs) [22] are directed graphical models for modeling sequential data. DBNs contain nodes for each time-slice, i.e., discrete time steps, and edges between such nodes signify conditional probabilities. DBNs contain individual variables for the nodes at each time step as opposed to the inductive definition of HMMs where the same random variables are reused. DBNs cannot handle context-dependencies as those we model in section 5.

There are various programming libraries available which support development of HMMs, e.g., [23]. Such libraries suffice for a variety of tasks, but are limited to express the kinds of models they were intended to.

PRISM is not the only language available that allows specification of HMMs. Several probabilistic-logic programming languages, which have identical power to PRISM, exist; see [24] for an overview. We have chosen PRISM because it is especially convenient for characterization of recursively defined structures (such as sequences) due to the Prolog backbone and its switches which are perfect for defining conditional probabilities of discrete (interior) events. PRISM provides a good balance between the ease by which HMMs can be expressed and the flexibility to model powerful HMM subspecies and contains generic inference algorithms for working with these models. A probabilistic version of regular expressions is described in [25] which also uses PRISM for its implementation.

8. Conclusion

There are countless subspecies of Hidden Markov Models and in this paper we have presented a few of them through probabilistic logic programs expressed in PRISM. PRISM provides a language which caters for experimentation with HMM species and the like and offers powerful inference algorithms that generalize to any model expressed in the language. A general theme should be clear from the examples; the different subspecies of HMMs can be represented with only minor variations of the program code.

Acknowledgement This work is supported by the project "Logic-statistic modelling and analysis of biological sequence data" funded by the NABIIT program under the Danish Strategic Research Council.

References

[1] Rabiner, L.: A tutorial on Hidden Markov Models and selected applications in speech recognition. Proceedings of the IEEE **77**(2) (February 1989) 257–286

[2] Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological Sequence Analysis. Cambridge University Press (1998)

[3] Jurafsky, D., Martin, J.H.: Speech and Language Processing (2nd Edition) (Prentice Hall Series in Artificial Intelligence). 2 edn. Prentice Hall (2008)

[4] Chomsky, N.: On certain formal properties of grammars. Information and Control **2**(2) (1959) 137–167

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

[6] Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. Journal of the Royal Statistical Society. Series B (Methodological) **39**(1) (1977) 1–38

[7] Sato, T., Kameya, Y.: PRISM: A language for symbolic-statistical modeling. In: IJCAI 97, Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence. (1997) 1330–1339

[8] Sato, T.: PRISM, PRogramming in Statistical Modeling (Link checked March 2011) Website; `http://sato-www.cs.titech.ac.jp/prism/`.

[9] Mørk, S., Holmes, I.: Probabilistic logic models of protein coding potential (2011) In preparation.

[10] Christiansen, H.: Taming the Zoo of Discrete HMM Subspecies & Some of their Relatives; example programs and guidelines (Established March 2011) Website; `http://www.ruc.dk/~henning/hmmzoo`.

[11] Christiansen, H., Gallagher, J.P.: Non-discriminating arguments and their uses. In Hill, P.M., Warren, D.S., eds.: ICLP. Volume 5649 of Lecture Notes in Computer Science., Springer (2009) 55–69

[12] Zhou, N.F., Kameya, Y., Sato, T.: Mode-directed tabling for dynamic programming, machine learning, and constraint solving. In: ICTAI (2), IEEE Computer Society (2010) 213–218

[13] Thorne, J.L., Kishino, H., Felsenstein, J.: An evolutionary model for maximum likelihood alignment of DNA sequences. Journal or Molecular Evolution (1991) 114–124

[14] Fine, S., Singer, Y., Tishby, N.: The hierarchical Hidden Markov Model: Analysis and applications. Machine Learning **32**(1) (1998) 41–62

[15] Christiansen, H., Dahmcke, C.M.: A machine learning approach to test data generation: A case study in evaluation of gene finders. In Perner, P., ed.: MLDM. Volume 4571 of Lecture Notes in Computer Science., Springer (2007) 742–755

[16] Ghahramani, Z., Jordan, M.I.: Factorial Hidden Markov Models. Machine Learning **29**(2-3) (1997) 245–273

[17] Christiansen, H., Have, C.T., Lassen, O.T., Petit, M.: Bayesian annotation networks for complex sequence analysis (2011) Submitted to an international conference.

[18] Christiansen, H.: Logic-statistic modeling and analysis of biological sequence data: a research agenda. In: Pre-Proceedings of the 2007 International Workshop on Abduction and Induction in Artificial Intelligence (AIAI'07). (2007) 42–49

[19] Christiansen, H.: Implementing probabilistic abductive logic programming with constraint handling rules. In Schrijvers, T., Frühwirth, T.W., eds.: Constraint Handling Rules. Volume 5388 of Lecture Notes in Computer Science. Springer (2008) 85–118

[20] Christiansen, H., Have, C.T., Lassen, O.T., Petit, M.: Inference with constrained Hidden Markov Models in PRISM. Theory and Practice of Logic Programming **10**(4-6) (2010) 449–464

[21] Have, C.T.: Stochastic definite clause grammars. In: RANLP 2009, International Conference: Recent Advances in Natural Language Processing (proceedings), INCOMA Ltd (2009) 139–144

[22] Murphy, K.: Dynamic Bayesian Networks: Representation, Inference and Learning. PhD thesis, UC Berkeley, Computer Science Division, Berkeley, USA (July 2002)

[23] Schliep, A., Rungsarityotin, W., Schönhuth, A., Georgi, B.: The general Hidden Markov Model library: Analyzing systems with unobservable states. In: Proceedings of the Heinz-Billing-Price. (2004) 121–136

[24] De Raedt, L., Demoen, B., Fierens, D., Gutmann, B., Janssens, G., Kimmig, A., Landwehr, N., Mantadelis, T., Meert, W., Rocha, R., Santos Costa, V., Thon, I., Vennekens, J.: Towards digesting the alphabet-soup of statistical relational learning. In Roy, D., Winn, J., McAllester, D., Mansinghka, V., Tenenbaum, J., eds.: Proceedings of the 1st Workshop on Probabilistic Programming: Universal Languages, Systems and Applications, Whistler, Canada (December 2008)

[25] Have, C.T., Christiansen, H.: Modeling repeats in DNA using extended probabilistic regular expressions (2011) Proceedings of 1st International Work-Conference on Linguistics, Biology and Computer Science: Interplays; Tarragona, Spain (these proceedings).