Published online 8 September 2009

Nucleic Acids Research, 2009, Vol. 37, No. 21 ***e139***
doi:10.1093/nar/gkp662

HMMCONVERTER 1.0: a toolbox for hidden Markov models

Tin Yin Lam and Irmtraud M. Meyer*

Centre for High-Throughput Biology, Department of Computer Science and Department of Medical Genetics, University of British Columbia, Vancouver V6T 1Z4, Canada

Received June 24, 2009; Revised July 22, 2009; Accepted July 24, 2009

ABSTRACT

Hidden Markov models (HMMs) and their variants are widely used in Bioinformatics applications that analyze and compare biological sequences. Designing a novel application requires the insight of a human expert to define the model's architecture. The implementation of prediction algorithms and algorithms to train the model's parameters, however, can be a time-consuming and error-prone task. We here present HMMCONVERTER, a software package for setting up probabilistic HMMs, pair-HMMs as well as generalized HMMs and pair-HMMs. The user defines the model itself and the algorithms to be used via an XML file which is then directly translated into efficient C++ code. The software package provides linear-memory prediction algorithms, such as the Hirschberg algorithm, banding and the integration of prior probabilities and is the first to present computationally efficient linear-memory algorithms for automatic parameter training. Users of HMMCONVERTER can thus set up complex applications with a minimum of effort and also perform parameter training and data analyses for large data sets.

INTRODUCTION

Hidden Markov models (HMMs) are widely used in Bioinformatics for analyzing and comparing biological sequences, such as genomes, transcripts and proteins. Applications range widely from comparative gene prediction (1,2,3) to methods for identifying protein domains (4) and predicting protein interfaces (5) and inference models for genome-wide association studies (6). Comparative applications employing pair-HMMs and applications employing probabilistic models have become increasingly important as they can significantly improve the ability to detect functional features and to interpret the model's predictions.

As the states of a model and their connections capture the most important features of the biological systems being studied, model design is best done by a human expert. The implementation of prediction algorithms, however, can be time-consuming and error-prone. The key idea of our software package HMMCONVERTER is thus to separate the design of a particular model from the implementation of the algorithms.

Earlier efforts in this direction comprise the compilers GENLANG (7) and DYNAMITE (8). GenLang allows users to specify HMMs (as well as context-free grammars) using the language Prolog, whereas DYNAMITE (8) is a compiler that produces efficient C code for pair-HMMs to be used in conjunction with the Viterbi algorithm (9). A more recent development is the compiler by Steffen and Giegerich (10), which allows users to specify HMMs, pair-HMMs (and context-free grammars) using algebraic dynamic programming. HMMER3 (11) is a software package for automatically deriving profile HMMs from given multiple-sequence alignments such as those of the protein family data base PFAM (4). HMMOC (12) and MAMOT (13) are the two most recent developments and the first to explicitly cater for probabilistic models. MAMOT is a compiler that can be used to set up simple HMMs, but not generalized HMMs and pair-HMMs, i.e. those types of HMMs that are typically employed by Bioinformatics applications today. The compiler HMMOC is a more general framework that can be used to set up probabilistic HMMs for any number of input sequences as well as generalized and phylogenetic HMMs. Both MAMOT and HMMOC, however, incorporate only basic algorithms and do not provide computationally efficient linear-memory algorithms for generating predictions and for training the model's parameters.

Most existing Bioinformatics applications, however, are still based on hand-coded source code rather than high-level toolkits as many Bioinformatics applications today require computational efficiency and features that these software packages do not provide.

We here present a new HMM compiler, called HMMCONVERTER, which significantly extends the functionality of MAMOT and HMMOC and which is the

*To whom correspondence should be addressed. Tel: +001 604 8274232; Fax: +001 604 822 5485; Email: irmtraud.meyer@cantab.net

first to provide computationally efficient linear-memory algorithms for generating predictions and for training parameters. The package can thus be used to swiftly design powerful applications employing complex models and to analyze realistic data sets.

MATERIALS AND METHODS

Model architectures and state features

HMMCONVERTER takes as input an XML-file which specifies the model's architecture, the initial values of transition and emission probabilities, the names of the algorithms to be used for generating predictions and for training the model's parameters and the names of the input and output files. Figure 1 shows an example of a model definition. HMMCONVERTER supports generalized or semi-Markov HMMs, i.e. HMMs and pair-HMMs whose states emit or read an arbitrary number of symbols from the input sequence(s) at a time. [As we expect the models defined by HMMCONVERTER to be primarily used for analyzing given input sequences rather than for generating sequences, we will in the following say that a state reads (rather than emits) symbols.] In HMMCONVERTER it is, for example, possible to define a state which reads one letter from the first input sequence (e.g. an amino acid) and three letters from the second input sequence (e.g. a triple of nucleotides corresponding to a codon from a genomic DNA sequence). States can also be silent provided they do not form a cyclic path in the model's architecture. The XML-file is directly translated into efficient C++ code, providing checks of the XML file against DTD files as well as numerous consistency checks at compile and run-time.

```
<model>
        <Model_Type name="dishonest casino"/>
        <Annotation_Labels>
                <Annotation_Label name="Label" score="1">
                        <label id="Label.0" name="F"/>
                        <label id="Label.1" name="L" />
                </Annotation_Label>
        </Annotation_Labels>
        <Alphabets cases="1" set="123456"/>
        <Emission_Probs id="EP" size="2" file="emission_dishonest_hmm2.txt"/>
        <States>
                <State id="S.0" name="Start" xdim="0" />
                <State id="S.1" name="Fair" xdim="1" >
                        <Label><label idref="Label.0" /></Label>
                </State>
                <State id="S.2" name="Loaded" xdim="1" >
                        <Label><label idref="Label.1" /></Label>
                </State>
                <State id="S.3" name="End" xdim="0" />
        </States>
        <Transitions train="All">
                <from idref="S.0">
                        <to idref="All" exp="0.5"/>
                </from>
                <from idref="S.1">
                        <to idref="S.1" exp="0.79067"/>
                        <to idref="S.2" exp="0.20932"/>
                        <to idref="S.3" exp="0.00001"/>
                </from>
                <from idref="S.2">
                        <to idref="S.1" exp="0.46968"/>
                        <to idref="S.2" exp="0.53031"/>
                        <to idref="S.3" exp="0.00001"/>
                </from>
                <from idref="S.3"/>
        </Transitions>
</model>
```

Figure 1. Example of an XML model definition. This example shows how the model of the dishonest casino (19), its states and transitions are defined in HMMCONVERTER using XML. The emission probabilities are listed in the included flat-text file emission_dishonest_hmm2.txt. This is one of the examples included in the HMMCONVERTER software package. The HMMCONVERTER manual explains in detail how to define a variety of models, how to invoke parameter training algorithms and how to define different types of sequence analysis.

Prediction algorithms

For any given model, predictions can be generated with the Hirschberg algorithm (14), i.e. the linear-memory version of the well-known Viterbi algorithm (15). This is a unique feature of HMMCONVERTER and allows complex pair-HMMs to be used to analyze long input sequences. The Hirschberg algorithm is implemented in a way which allows users to directly specify the maximum amount of memory available on the respective computer. It is thus possible to optimize the time requirement of the Hirschberg algorithm by keeping the number of iterations that the algorithm has to perform to a minimum. Unlike banding, which is discussed below, the Hirschberg algorithm is a computationally more efficient variant of the Viterbi algorithm that is guaranteed to find the optimal Viterbi solution.

Banding

For comparative applications involving pair-HMMs, both the Viterbi and the Hirschberg algorithm can be heuristically accelerated by restricting the search space along the sequence dimensions in a 'band' or 'tube'. This sub-space of the search space can either be specified explicitly by the user or derived by HMMCONVERTER from BLAST matches (16) (Figure 2). The latter is done by a dedicated dynamic programming procedure that takes a set of local BLAST matches as input and derives the highest scoring sub-set of mutually compatible matches, i.e. a set of matches which can be simultaneously ordered along both sequence dimensions. If the 'band' or 'tube' is to be derived from BLAST matches, the user can specify the radius of the tube around the selected matches.

Banding can be combined with the Viterbi and Hirschberg algorithms, thus allowing the design of computationally extremely efficient applications. This feature is implemented in a very memory-efficient way using dedicated data structures, so that only memory inside the specified sub-space of the dynamic programming matrix is allocated. When used correctly, i.e. when the tube contains the optimal and most near-optimal alignments of the two input sequences, banding can lead to near-linear time and memory requirements for algorithms employed with pair-HMMs without incurring a loss in prediction performance, e.g. for comparative gene prediction (1). HMMCONVERTER explicitly checks that tubes specified by users are well-defined, but does not aim to adjust poorly chosen tubes.

Incorporating prior information

Most biological sequence data today can be easily assigned some form of prior annotation, e.g. matches to homologous sequence data or partial and incomplete information as shown in Figure 3. It thus makes a lot of sense to try to integrate this prior information as well as information on the corresponding confidence levels into the prediction and parameter training algorithms in order to improve both.

HMMCONVERTER can integrate prior knowledge about the annotation of the input sequence(s) probabilistically into the prediction algorithms as well as the parameter training algorithms. To use this feature, the user has to specify prior probabilities for different annotation labels for the desired positions along the input sequences. This information is integrated in a probabilistic way into the algorithms by biasing the nominal emission probabilities with the relevant position-specific prior probabilities. For an HMM and the input sequence $X = (x_1, \ldots, x_{L_x})$ of length L_x, the annotation label set $S = \{\text{Exon, Intron, Intergenic}\}$ and the prior information shown in Figure 3, for example, the nominal emission probability $e_s(x_i)$ of a state s for reading nucleotide x_i at a sequence position $i \in [90,370]$, would be replaced by $e_s(x_i)\ p_{l(s)}(i)$, where $l(s) \in S$ is the annotation label of state s and $p_{l(s)}(i)$ the prior probability of sequence position i for having annotation label $l(s)$.

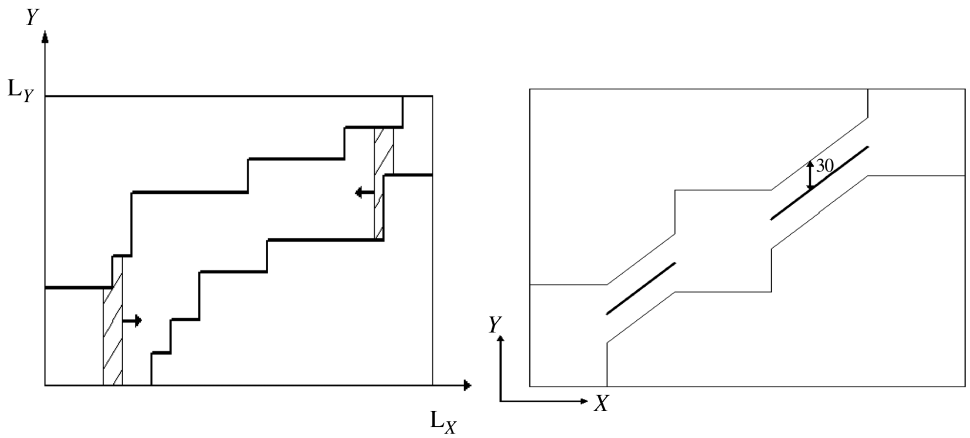

Figure 2. Banding. Projection of the three-dimensional search space for a pair-HMM onto the plane spanned by the two input sequences. The tube inside the large rectangle significantly reduces the three-dimensional search space. The two narrow vertical strips in the left figure correspond to the amount of memory allocated by the first iteration of the Hirschberg algorithm. The tube can be either user defined (left) or derived from BLAST matches (right). The two thick lines in the right figure correspond to the set of matches selected by the dynamic programming routine as the highest scoring sub-set of mutually compatible BLAST matches (the discarded BLAST matches are not shown here). In this example, the radius is specified as 30 by the user.

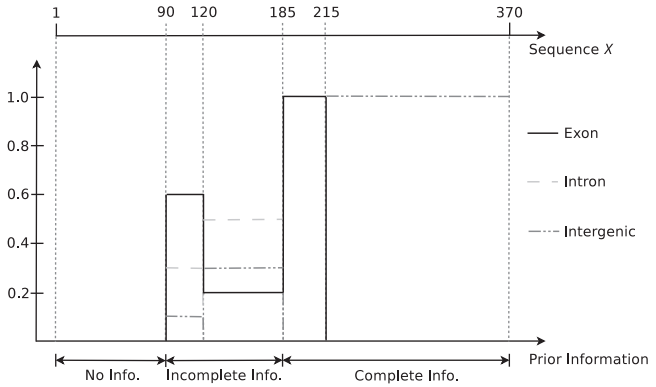

Figure 3. Prior information. Example of prior information for input sequence X of length L_x for an annotation label set $S = \{\text{Exon,Intron,Intergenic}\}$. For sequence interval [1,89], no prior information on the annotation of the sequence is available. This stretch of the sequence would thus be analyzed with an model whose nominal emission probabilities are not biased by any prior probabilities. For sequence interval [90,184], there is prior information concerning the likelihood of different sequence positions for being Exon, Intron and Intergenic. For the rest of the input sequence, i.e. for sequence interval [185,370], we know with certainty that the sequence positions in [185,214] are exonic and that the remainder of the sequence is intergenic. Note that the prior probabilities add up to 1 for every sequence position for which any prior information is supplied, reflecting the fact that in this case the three labels Exon, Intron and Intergenic in this annotation label set are mutually exclusive and that each sequence position has to fall into exactly one of these three categories.

It is also possible to specify priors for multiple, mutually compatible sets of annotation labels, e.g. information on the annotation and on the alignment when dealing with a pair-HMM.

HMMCONVERTER allows users to combine the use prior probabilities with all prediction and parameter training algorithms.

Being able to probabilistically incorporate prior information on the input sequences into the prediction algorithms allows users to design applications that make the best use of the already available annotation and to improve the prediction performance with respect to applications that cannot take prior information about input sequence into account. Furthermore, this feature allows for explicit testing of different hypotheses as competing annotations for the same input sequence can result in different predicted annotations. It is thus possible to design annotation applications which combine the positive aspects of manual human expert annotation with the advantages of automatic and readily reproducible predictions.

Also parameter training can be significantly improved by taking prior information on the input sequences probabilistically into account. None of the parameter training algorithms are guaranteed to find a set of parameters that optimize the global prediction performance. By incorporating prior information on the known or likely annotation of the input sequences, we can increase the chances of improving the resulting prediction performance. Using this feature it is, for example, possible to train the parameters of a pair-HMM given pairs of well-annotated input sequences with unknown global alignment.

It is thus possible to combine the Hirschberg algorithm or any training algorithm with a 'tube' and prior probabilities in order to create and train powerful applications and to analyze long sequences efficiently.

Training algorithms

It is typically easy to define the states of a model and its transitions, but can be very difficult to assign probabilities to the numerous transition and emission parameters in the model. As the prediction performance crucially depends on these parameter values, it is thus very important to be able to maximize the performance as a function of the parameter values. Unless the model has very few free parameters, this is best done automatically by employing a parameter training algorithm.

HMMCONVERTER is the first software package to provide an implementation of the new linear-memory algorithm (T. Lam and I. Meyer, Submitted for publication; 17) for Viterbi training (18) as well as the linear-memory algorithm (19,20) for Baum–Welch training (21). In addition, it incorporates a third, new method called posterior sampling training (T. Lam and I. Meyer, Submitted for publication; 17) which derives new parameter values from state paths that are sampled from the posterior distribution. Posterior sampling training can outperform Baum–Welch training, both in terms of performance and in terms of memory and time efficiency, and provides a significantly more robust way for parameter training than Viterbi training (T. Lam and I. Meyer, Submitted for publication; 17). We have also implemented a linear-memory algorithm (T. Lam and I. Meyer, Submitted for publication; 17) for this type of training into HMMCONVERTER.

All three training methods work in an iterative way and require termination criteria, the user can either specify a maximum number of iterations or alternatively, for Baum–Welch training, a minimum log-likelihood change. All three training algorithms can be combined with prior information on the training sequences and with banding, thereby further improving the chances of successful parameter training and rendering training even for complex models and long sequences feasible.

Table 1 summarizes the memory and time requirements for all prediction and parameter training algorithms provided by HMMCONVERTER.

Model parameterization, pseudo-counts and efficiency

Many of the models employed by Bioinformatics applications today have a large number of free parameters. In order to reduce the number of free parameters and to improve the performance outcome of parameter training, HMMCONVERTER allows users to parameterize the model's probabilities by providing arithmetic formulae in the XML file. Transition probabilities can be defined as functions of free transition parameters using the operators '+', '−', '·' and '/' and brackets. Emission probabilities can be parameterized using combinations of the two functions 'SumOver' and 'Product'. Using these

Table 1. Computational requirements

Feature	Implemented algorithm	Time requirement	Memory requirement	Reference
Viterbi algorithm	Viterbi algorithm	$\mathcal{O}(MT_{\max}L)$	$\mathcal{O}(ML)$	(9)
Hirschberg algorithm	Hirschberg algorithm	$\mathcal{O}(MT_{\max}L)$	$\mathcal{O}(M)$	(14)
Viterbi training	Lam–Meyer algorithm	$\mathcal{O}(MT_{\max}L)$	$\mathcal{O}(M)$	(17,18)
Baum–Welch training	Miklós–Meyer algorithm	$\mathcal{O}(MT_{\max}L)$	$\mathcal{O}(M)$	(20)
Posterior sampling training	Lam–Meyer algorithm	$\mathcal{O}(MT_{\max}LK)$	$\mathcal{O}(M + MT_{\max} + MK)$	(17,18)

Overview of the time and memory requirements for the different prediction and parameter training algorithms in HMMCONVERTER. The requirements are given for an HMM with M states and a connectivity of $T_{\max}$, where L is the length of the input sequence and K the number of state paths sampled in each iteration for every training sequence using the posterior sampling algorithm. Note that the requirements for the training algorithms are the requirements for each iteration.

functions, one can, for example, derive lower dimensional emission probabilities from higher dimensional ones [for example, the emission probabilities of an emit state that reads three symbols from sequence X at a time can be derived from those of a match state that reads six symbols at a time, three from input sequence X and three from Y using $P_{\text{emit}_x}(x_1, x_2, x_3) = \sum_{(y_1, y_2, y_3)} P_{\text{match}}(x_1, x_2, x_3, y_1, y_2, y_3)$] and one can express higher dimensional emission probabilities as a function of several lower dimensional ones [for example, $P_{\text{triple}}(x_1, x_2, x_3) = P_{\text{single}}(x_1) \cdot P_{\text{single}}(x_2) \cdot P_{\text{single}}(x_3)$]. As the functions 'SumOver' and 'Product' can be combined, the emission probabilities for most applications can be easily parameterized.

As this parameterization is respected by the training algorithms, the number of free parameters in complex models can be significantly reduced, thereby reducing the risk of over-fitting and increasing the chances of successful training. In addition, the user can specify arbitrary subsets of the free parameters to be trained simultaneously and can set pseudo-counts to prevent null probabilities and over-fitting. In order to address the numerical underflow problems that easily arise when probabilistic models are used to analyze long input sequences, all prediction and training algorithms operate internally in log-space.

Documentation

The HMMCONVERTER package comprises a detailed manual as well as several examples which illustrate the full functionality of the package, e.g. the dishonest casino, an HMM for CpG-island prediction (18) and a p53-transcription site prediction model (13).

Availability

The HMMCONVERTER package including detailed documentation and multiple examples is freely available under the GNU General Public License version 3 (GPLv3) and can be downloaded from http://people.cs.ubc.ca/~irmtraud/hmmconverter.

RESULTS AND DISCUSSION

We introduce a new software package, HMMCONVERTER, which can be used to design powerful applications that employ HMMs or pair-HMMs with a minimum of effort. The package requires the user to specify the model and the names of the algorithms to be used via an XML file which is directly translated into efficient C++ code. The user is thus shielded from the implementation of the underlying algorithms and can focus on what humans do best, namely model design. HMMCONVERTER is the only software package to provide computationally efficient linear-memory algorithms for generating predictions and for training the model's parameters (Figure 1). The package provides the linear-memory Hirschberg algorithm for generating predictions as well as three linear-memory training algorithms for Viterbi training, Baum–Welch training and a new, promising type of training called posterior sampling training. In addition, it supports probabilistic models and provides sophisticated features, such as banding and the integration of prior probabilities which can be combined with any of the prediction and training algorithms.

HMMCONVERTER thus allows the design of complex Bioinformatics applications that can be trained efficiently and used for sophisticated data analyses.

FUNDING

Discovery Grant of the Natural Sciences and Engineering Research Council, Canada; Leaders Opportunity Fund of the Canada Foundation for Innovation (to I.M.M.) Funding for open access charge: Natural Sciences and Engineering Research Council, Canada.

Conflict of interest statement. None declared.

REFERENCES

1. Meyer,I.M. and Durbin,R. (2002) Comparative ab initio prediction of gene structures using pair HMMs. *Bioinformatics*, **18**, 1309–1318.
2. Meyer,I. and Durbin,R. (2004) Gene structure conservation aids similarity based gene prediction. *Nucleic Acids Res.*, **32**, 776–783.
3. Stanke,M., Schoffmann,O., Morgenstern,B. and Waack,S. (2006) Gene prediction in eukaryotes with a generalized hidden Markov model that uses hints from external sources. *BMC Bioinformatics*, **7**, 62.
4. Finn,R., Tate,J., Mistry,J., Coggill,P., Sammut,S., Hotz,H., Ceric,G., Forslund,K., Eddy,S., Sonnhammer,E. *et al.* (2008) The Pfam protein families database. *Nucleic Acids Res.*, **36**, 281–288.
5. Nguyen,C., Gardiner,K. and Cios,K. (2007) A hidden Markov model for predicting protein interfaces. *J. Bioinform. Comput. Biol.*, **5**, 739–753.

6. Hosking,F., Sterne,J., Smith,G. and Green,P. (2008) Inference from genome-wide association studies using a novel Markov model. *Genet. Epidemiol.*, **32**, 497–504.
7. Searls,D. (1993) String variable grammar: a logic grammar formalism for the biological language of DNA. *J. Logic. Program.*, **24**, 73–102.
8. Birney,E. and Durbin,R. (1993) DYNAMITE: a flexible code generating language for dynamic programming methods used in sequence comparison. In *Fifth International Conference on Intelligent Systems in Molecular Biology*. IEEE Press.
9. Viterbi,A. (1967) Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. In *IEEE Transactions on Information Theory,* IT-13*,* pp. 260–269.
10. Steffen,P. (2006) compiling a domain specific language for dynamic programming. *Ph.D. thesis*. University of Bielefeld.
11. Eddy,S.R. http://hmmer.janelia.org/, last accessed 1st September, 2009.
12. Lunter,G. (2007) Hmmoc – a compiler for hidden Markov models. *Bioinformatics*, **23**, 2485–2487.
13. Schütz,F. and Delorenzi,M. (2008) MAMOT: hidden Markov modeling tool. *Bioinformatics*, **24**, 1399–1400.
14. Hirschberg,D. (1975) A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, **18**, 341–343.
15. Viterbi,A. (1967) Error bounds for convolutional codes and an assymptotically optimum decoding algorithm. In *IEEE Transactions on Information Theory,* pp. 260–269.
16. Altschul,S., Gish,W., Miller,W., Myers,E. and Lipman,D. (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.
17. Lam,T.Y. (2008) HMMConverter – a tool-box for hidden Markov models with two novel, memory efficient parameter training algorithms. *MSc Thesis*. University of British Columbia.
18. Durbin,R., Eddy,S., Krogh,A. and Mitchison,G. (1998) *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge.
19. Miklós,I. and Meyer,I. (2005) A linear memory algorithm for Baum–Welch training. *BMC Bioinformatics*, **6**, 231.
20. Churbanov,A. and Winters-Hilt,S. (2008) Implementing em and Viterbi algorithms for hidden Markov model in linear memory. *BMC Bioinformatics*, **9**, 224.
21. Baum,L. (1972) An equality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, **3**, 1–8.