Manual for HMMConverter 1.0

Tin Yin Lam and Irmtraud M. Meyer
Centre for High-Throughput Biology,
Department of Computer Science and
Department of Medical Genetics, 2366 Main Mall,
University of British Columbia, Vancouver V6T 1Z4, Canada,

Contact: irmtraud.meyer@cantab.net
Web: http://www.cs.ubc.ca/~irmtraud/hmmconverter

May 2009

1 Introduction

Hidden Markov Models (HMMs) and their variants are very useful, but not easy to implement. Especially for sophisticated applications in Bioinformatics, the implementation of the model as well as the prediction and parameter training algorithms can be time-consuming and error-prone.

HMMCONVERTER is a toolbox for HMMs which allows you to focus on model and analysis design and shields you from the implementation of the model and the algorithms. It provides several, computationally efficient prediction and parameter training algorithms which are easily invoked via a user-specified XML file which contains the model's description as well as the types of analysis to be performed. HMM translates this description into efficient C++ code. Our package supports several special features that are particularly appealing for Bioinformatics applications, but is no way limited to applications in Bioinformatics.

If you find HMMCONVERTER useful please

- send your feedback to `irmtraud.meyer@cantab.net`
- cite Tin Yin Lam and Irmtraud M. Meyer "HMMCONVERTER : a toolbox for hidden Markov models" (2009, submitted)
- cite Tin Yin Lam and Irmtraud M. Meyer "Efficient parameter training for hidden Markov models using posterior sampling training and Viterbi training" (2009, submitted)
- cite Tin Yin Lam's M.Sc. thesis "HMMCONVERTER - A tool-box for hidden Markov models with two novel, memory efficient parameter training algorithms" (UBC 2008)

This document is a detailed manual of HMMCONVERTER. Its primary purpose is to explain how to set up different HMMsand how to invoke different special features, prediction and training algorithms. Please refer to

- the distributed HMMCONVERTER tar-ball for information on how to install HMMCONVERTER, to see several examples and for information on the license under which HMMCONVERTER is made available
- the HMMCONVERTER web-site http://www.cs.ubc.ca/~ irmtraud/hmmconverter for the latest HMMCONVERTER release and news on updated versions
- the publication by Tin Yin Lam and Irmtraud M. Meyer "HMMCONVERTER : a toolbox for hidden Markov models" (2009, submitted) for the HMMCONVERTER paper
- the publication by Tin Yin Lam and Irmtraud M. Meyer "Efficient parameter training for hidden Markov models using posterior sampling training and Viterbi training" (2009, submitted) for a detailed description of the new training algorithms implemented into HMMCONVERTER and a practical performance evaluation

- Tin Yin Lam's M.Sc. thesis "HMMCONVERTER - A tool-box for hidden Markov models with two novel, memory efficient parameter training algorithms" (UBC 2008) for the most detailed description of the algorithms and features of HMMCONVERTER

2 Installation and requirements

HMMCONVERTER is an executable file in C++ that is executed from the command line. Users can start it using the following command:

HMMCONVERTER input_directory/ output_directory/ HMM.xml

, where HMMCONVERTER is the name of the executable file and HMM.xml is the XML input file specifying the HMM and the analyses to be performed. Note that the name of the XML file is not restricted to HMM.xml. All input files including this XML input file must be contained in the input_directory. Similarly, all output files will be written to the output_directory.

In order to construct an HMM with HMMCONVERTER, the user only has to supply the corresponding input files, no compilation of code is needed.

3 Supported features

- simple and user-friendly XML input file format
- Viterbi algorithm [12] and Hirschberg algorithm [5] for generating predictions by sequence decoding
- HMM and pair-HMM as well as their generalized versions, i.e. models whose states read more than one symbol from the input sequence at a time
- models with silent states provided they do not form a circular route
- banding, i.e. heuristically restricting the search space in the sequence dimensions of a pair-HMM, and a memory-efficient data structure to incorporate banding into all prediction and parameter training algorithms
- special emission probabilities to incorporate sequence-position specific prior information on the input sequences into the prediction and parameter training algorithms
- parameterized transition and emission probabilities which is respected by the parameter training algorithms
- computationally efficient parameter training algorithms: linear-memory Baum-Welch training [10, 3], linear-memory Viterbi training and linear-memory posterior sampling training [6, 7] (the latter two algorithms are new training algorithms)

4 Special features

Some features listed above are uniquely supported by HMMCONVERTER, we will discuss them briefly in the following section. For more theoretical details of the features, please refer to Lam 2008 [6].

4.1 Banding

Banding allows running sequence decoding and parameter training algorithms in a restricted search space along the sequence dimensions. We call the restricted search space a "tube". If the tube is well chosen, the memory and time requirements of the algorithms can be greatly improved while keeping the performance unchanged.

In HMMCONVERTER, there are two ways to construct a tube:

(i) specify the restricted search space explicitly using a tube file.

(ii) derive a tube from local matches generated by TBLASTX or BLASTN [1]

In order to derive a tube from BLAST matches, the user has to specify a parameter called **radius** which defines the width of the tube around the selected set of matches. HMMCONVERTER using a dynamic programming routine to derive the highest-scoring sub-set of mutually compatible matches from an initial set of local BLAST matches. Figure 1 illustrates a tube derived from two matches.

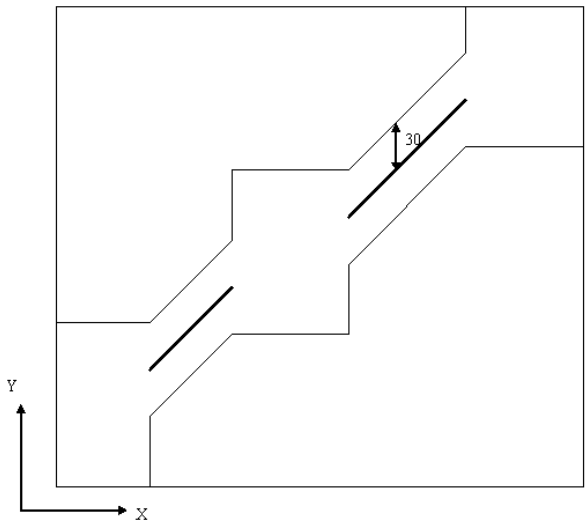

Figure 1: The two thick lines in the middle are the best matches that were selected by the dynamic programming routine as the highest scoring sub-set of mutually compatible BLAST matches (the discarded BLAST matches are not shown here). The **radius** is specified as 30 (by the user). If banding is used, all prediction and parameter training algorithms will then only explore the inside the tube derived from the matches.

We now consider specifying the tube explicitly in the tube file. HMMCONVERTER defines a tube via a set of user defined 4-tuples, $\{(x_1, y_1, x_2, y_2) | 1 \leq x_1 \leq x_2 \leq L_x, 1 \leq y_1 \leq y_2 \leq L_y\}$. Each 4-tuples defines two points (x_1, y_1) and (x_2, y_2) on the sequence plane. These two points can either define the end points of a match, or a rectangular block in the sequence plane of the Viterbi matrix. Users can explicitly define a tube either by specifying matches and a **radius** or by specifying blocks of restricted area. Figure 2 shows the two different ways of explicitly defining a tube using the same set of 4-tuples. It also shows how HMMCONVERTER specifies the final tube which connects the specified blocks via maximum areas such that the final tube connects the start of the two sequences to the end of the two sequences.

4.2 Annotation label sets and special emission

HMMCONVERTER allows constructing an HMM or a pair-HMM that can integrate prior information of input sequences into sequence decoding and parameter training. This is done by biasing the nominal emission probabilities with the corresponding position-specific prior probability using the concept of *annotation label set* and *special emission*. This concept was employed in the context of comparative gene prediction in PROJECTOR [9].

We first describe the concept of *multiple annotation label sets*, and then explain *special emissions*.

Multiple annotation label sets In order to integrate prior probabilities along the input sequences into the prediction process, a mapping has to be defined between the labels of the states in the HMM and the different types of prior information. We call the prior information along the sequences "the annotation of the input sequences" and the labels of the states the "annotation labels of the states". The mapping has to explain how an annotation label of the sequence is mapped to the annotation labels of the states. An annotation label set defines some annotation labels, for example, an HMM for gene prediction may have an annotation label set $\text{Feature} = \{\text{Exon}, \text{Intron}, \text{Intergenic}\}$, where the annotation labels represent gene structures. Users can define arbitrary annotation label sets for an HMM. Any non-silent state in the HMM is assigned exactly one annotation label from each annotation label set to each letter that it reads from each input sequence.

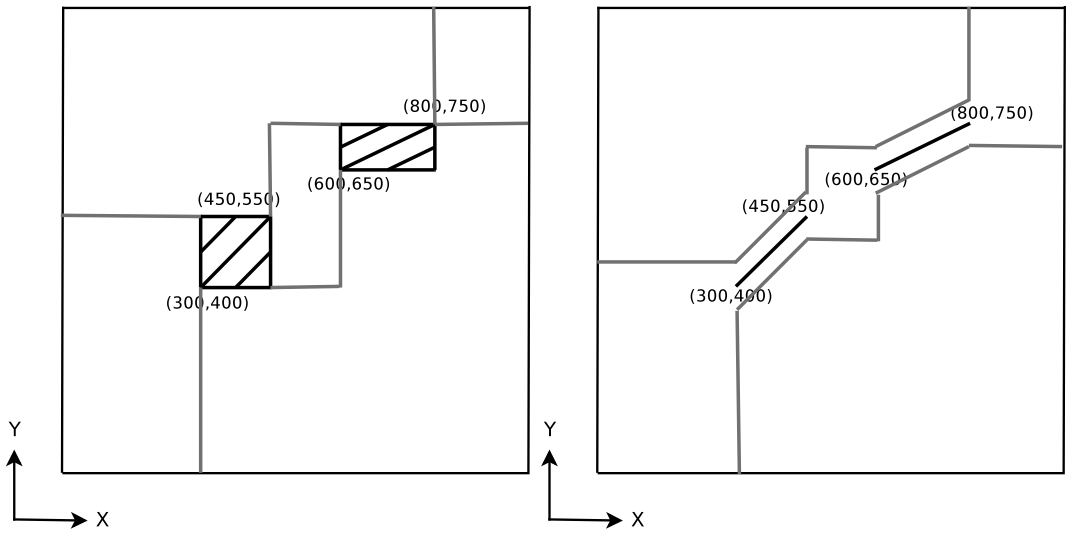

Figure 2: The right part of the figure shows the matches defined by the tube file, with the radius set to 50. The tube around the matches is derived in the same way as for matches generated by BLAST. The left figure shows the same specified 4-tuples interpreted as blocks and how the final tube is derived from these blocks.

Let us consider as an example a pair-HMM for comparative gene prediction with the following two annotation label sets, i.e. $\mathrm{Set\_1} = \{\mathrm{Exon}, \mathrm{Intron}, \mathrm{Intergenic}, \mathrm{Start\_Condon}, \mathrm{Stop\_Codon}\}$ and $\mathrm{Set\_2} = \{\mathrm{Match}, \mathrm{Emit\_X}, \mathrm{Emit\_Y}\}$. Let is consider a state called **Match_Exon** (see Figure 3). This state reads three letters from each of the two input sequences at a time, the annotation labels $\mathrm{Exon} \in \mathrm{Set\_1}$ and $\mathrm{Match} \in \mathrm{Set\_2}$ are the annotation labels for each letter that this state reads. Two types of prior information, $\mathrm{Set\_1}$ and $\mathrm{Set\_2}$, can be specified for the input sequences. Note that the annotation labels for different letters of one state can differ. For example, we could define a third annotation label set $\mathrm{Set\_3} = \{\mathrm{NoPhase}, \mathrm{Phase0}, \mathrm{Phase1}, \mathrm{Phase2}\}$ to specify the reading frame of a nucleotide in the input sequence. The **Match_Exon** state would assign the annotation labels $(\mathrm{Phase0}, \mathrm{Phase1}, \mathrm{Phase2})$ to the sequence positions (x_1, x_2, x_3), respectively, and to the sequence positions (y_1, y_2, y_3), respectively.

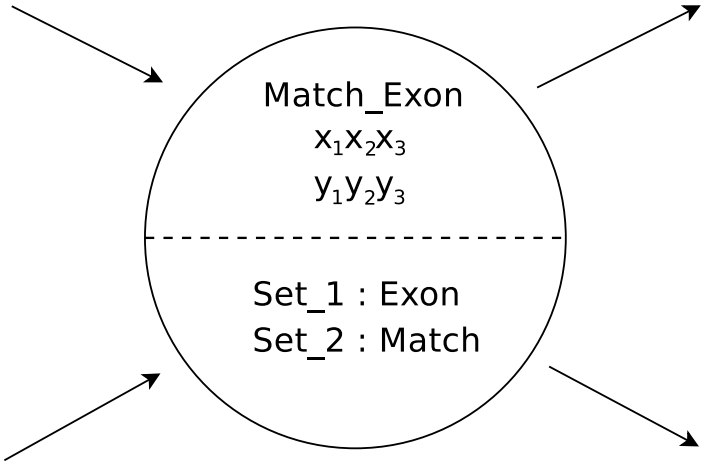

Figure 3: A state in a generalized pair-HMM which carries labels from two different annotation label sets: $\mathrm{Exon} \in \mathrm{Set\_1}$ and $\mathrm{Match} \in \mathrm{Set\_2}$. This state reads six letters at a time, three from input sequence X and three from input sequence Y, and assigns to all six letters the same pair of annotation labels, namely Exon and Match.

In order to use *special emissions*, users have to specify prior information along the input sequences using the labels of the different annotation label sets. The labels of the states in the model have to be matched to those labels specified along the input sequences. The following paragraph explain the details of how the *special*

emissions are used.

Special emissions HMMCONVERTER provides a very flexible way for integrating prior information. Users can defined any number of annotation label sets for an HMM. The labels from different annotation label sets have to be mutually exclusive, but the labels within an annotation label set do not need to be mutually exclusive. Consider the annotation label set $\mathrm{Set\_1}$, where we know that every start codon is also an exon. If there are prior probabilities specified for the label $\mathrm{Start\_Codon}$ for an interval of sequence positions, the same interval may also have a prior probability for the label Exon (in this particular case, the prior probability for the label Exon should be equal to or larger than the prior probability for the label $\mathrm{Start\_Codon}$).

Figure 4 illustrates how different prior probabilities can be assigned to an input sequence, it also shows that the users have to take care of the correct normalization of the prior probabilities. This means that the sum of the prior probabilities for mutually exclusive annotation labels in one annotation label set assigned on the same interval of sequence positions has to be at most one (and exactly one, if the annotation labels of one set cover all possibilities). The requirement of normalization for prior probabilities for any sequence position becomes clearer once we explain the mechanism for biasing the emission probabilities with prior probabilities.

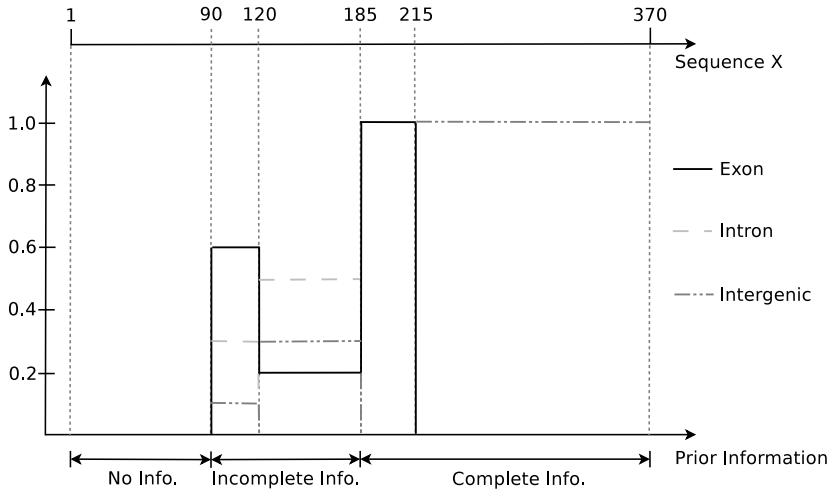

Figure 4: Example of prior information for input sequence X of length L_x for one annotation label set $\mathcal{S} = \{\mathrm{Exon}, \mathrm{Intron}, \mathrm{Intergenic}\}$. For sequence interval [1,89], no prior information on the annotation of the sequence is available. This stretch of the sequence would thus be analyzed with an HMM 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. 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 are mutually exclusive and that each sequence position has to fall into exactly one of these three categories.

We illustrate the mechanism by considering the state **Match_Exon** in Figure 3 and the prior probabilities assigned to an input sequence X as shown in Figure 4. We consider four different cases for explaining how prior probabilities are incorporated into the prediction and parameter training algorithms:

(i) reading sequence position 89

(ii) reading sequence position 90

(iii) reading sequence position 220

(iv) reading sequence position 186 with an additional prior probability of 70% for the label Match in annotation label set $\mathrm{Set\_2}$

In case (i), there is no prior information assigned to the sequence position 89 of sequence X. In this case, no bias of the nominal emission probabilities is introduced for the corresponding sequence position, i.e. when

dealing with this sequence position, the algorithms will use the nominal emission probabilities of the HMM.

In case (ii), sequence position 90 of the input sequence X has a 60% probability of being an exon. The decoding or training algorithm then integrates these prior probabilities into the emission probabilities when a state path passes through this sequence position and a state of the model which carries the annotation label Exon (in particular, the **Match_Exon** state) by multiplying the nominal value of the emission probabilities with an additional factor of 0.7. On the other hand, there is a 30% prior probability of being Intron at the same sequence position. A factor of 0.3 will thus be multiplied to the nominal value of the emission probabilities of the states which carry the label Intron when a state path passes through this sequence position. The nominal emission probabilities for states which carry the label Intergenic will be similarly biased when a state path passes through this sequence position. Note that the three annotation labels Exon, Intron and Intergenic of the same annotation label set are mutually exclusive labels, so the sum of the prior probabilities assigned to these labels for any sequence position has to be 1. This additional responsibility for the user provides a highly flexible way taking prior probabilities into account in order to improve prediction and parameter training. For the same annotation label set, users can define annotation labels that are related to each other (e.g. Exon and Start_Codon) as well as mutually exclusive labels (e.g. Exon and Intron).

In case (iii), sequence position 220 has 100% probability of being an Intergenic region. As this is the only prior probability specified for this sequence position, the prior probabilities of other annotation labels (i.e. Exon and Intron) in the same annotation label set (i.e. Set_1) are set to 0 for this sequence position by default. A factor of 0 will be multiplied to the nominal value of the emission probabilities of the **Match_Exon** state when a state path passes through this sequence position. In an other words, the **Match_Exon** state cannot be used to read sequence position 220. This case is different from case (i), where there is no prior probability assigned to any annotation labels in this annotation label set. In HMMCONVERTER, once a prior probability for an annotation label and a specific sequence position is specified (in this case, label Intergenic of Set_1), all the other annotation labels in the same annotation label set for which no prior probabilities are explicitly specified are set to 0 (in this case, for Exon and Introng). So, it is very important for users to supply normalized prior probabilities for any sequence positions.

Finally, in case (iv), sequence position 186 has a prior probability of 100% for being an Exon. A factor of 1 will thus be multiplied to the nominal value of the emission probabilities of the **Match_Exon** state when a state path passes through this sequence position (i.e. the nominal value of the emission probability of this state is not biased for this sequence position). In this case, however, we also have an additional prior probability of 70% for label Match of annotation label set Set_2 for this sequence position. In order to take this into account, a further factor of 0.7 is multiplied to the nominal emission probabilities of the **Match_Exon** state for sequence position 186. More precisely, let the sequence symbol read by the **Match_Exon** state at sequence position 186 be γ, and the corresponding emission probability for this state be $e(\gamma)$, then when a state path passes through the sequence position 186, the emission probability for passing through the **Match_Exon** state will become $e'(\gamma) = e(\gamma) \cdot 1 \cdot 0.7$. More factors would be multiplied to the nominal emission probability of a state in case there were multiple types of prior probabilities assigned to that sequence position. Once again, the user is required to provide normalized prior probabilities for all the different annotation label sets defined in an HMM for all input sequences.

4.3 Free transition and emission parameters

In HMMCONVERTER , users can parameterize the transition and emission probabilities for the HMM. This feature is very useful for reducing the number of free parameters for complex HMMs and for increasing the chances of successful parameter training and avoiding over-fitting problems. The rules for parameterizing transition probabilities and emission probabilities differ. We will therefore describe them separately in the following.

Parameterization of transition probabilities We first present the power of parameterization by looking at the complex generalized pair-HMM of DOUBLESCAN [8] which is employed for comparative, alignment-free gene prediction. It consists of 153 transitions, but can be efficiently parameterized using only 19 free transition parameters.

In HMMCONVERTER, a transition probability can be defined by an arithmetic formula. The formula may use any of the basic arithmetic operators: '+','−', '∗','/' and brackets, with the id of the free transition param-

eters and numerical values as operands. The free transition parameters need not correspond to probabilities, i.e. their values are not restricted too the interval $[0, 1]$.

The user has to ensure, however, that the parameterization of all transition probabilities implies that $\sum_j t_{i,j} = 1$ for all states i in the HMM. For example, for a state i with three transitions, $i \to j_1$, $i \to j_2$ and $i \to j_3$, and the derivation formula (functions of free parameters) $t_{i,j_1} = f_1$, $t_{i,j_2} = f_2$ and $t_{i,j_3} = f_3$, respectively, the sum of the three transition probabilities as function of the free parameters is always 1 (as $f_1 + f_2 + f_3 = 1$).

HMMCONVERTER applies consistency checks to ensure that $\sum_j t_{i,j} = 1$ and $\sum_\gamma e_i(\gamma) = 1$ for all states i in the HMM and reports error messages and terminates the program otherwise.

Derivation of emission probabilities For the emission probabilities, we only allow free parameters that correspond to probabilities, i.e. that have values in $[0, 1]$. In HMMCONVERTER, a group of emission probabilities that defines the set of emission probabilities of one state is regarded as one emission parameter. All the free emission parameters are specified in a special free emission parameter file.

HMMCONVERTER provides two functions, **SumOver** and **Product**, for deriving emission probabilities. Consistency checks on the correct dimensions of the parameters and the normalization are enforced by HMMCONVERTER.

There are four ways of defining the emission probabilities of a state:

i. Directly defining the emission probability An emission parameter can be used directly by several states. This means that the emission probabilities of a state can be defined by pointing directly to an emission parameter in the free emission parameter file.

ii. SumOver This function can be used to derive a lower dimensional emission probability from a higher dimensional emission probability. The dimension of an emission probability is the sequence dimension, i.e. the total number of letters read at by the corresponding state at a time. Consider as example three states: **match_xy**, **emit_x** and **emit_y** of dimension 6, 3 and 3, respectively. The emission probabilities of **emit_x** and **emit_y** can be derived from those of **match_xy** by summing over several sequence dimensions:

$$\begin{aligned} P_{emit_x}(x_1, x_2, x_3) &= \sum_{(y_1, y_2, y_3)} P_{match_{xy}}(x_1, x_2, x_3, y_1, y_2, y_3) \\ P_{emit_y}(y_1, y_2, y_3) &= \sum_{(x_1, x_2, x_3)} P_{match_{xy}}(x_1, x_2, x_3, y_1, y_2, y_3) \end{aligned}$$

iii. Product This function allows the user to express a high dimensional emission probability as product of several lower dimensional emission probabilities. Consider as example two states, **triple** and **single**, of dimension 3 and 1, respectively. The emission probabilities of **triple** can be expressed as function of those of **single** in the following way:

$$P_{triple}(x_1, x_2, x_3) = P_{single}(x_1) * P_{single}(x_2) * P_{single}(x_3)$$

iv. SumOver & Product The two functions can be combined to form more complex rules, for example for three states of dimension 3, 1 and 4, respectively:

$$P_{3-tuple}(x_1, x_2, x_3) = \left(\sum_{(y_1, y_2)} P_{4-tuple}(x_1, x_2, y_1, y_2) \right) \cdot P_{single}(x_3) \tag{1}$$

The above rules are easy to apply and flexible enough for most applications. It is easy to prove mathematically that these rules derive a normalized set of emission probabilities which fulfill $\sum_\gamma e_i(\gamma) = 1$ for all states i in the HMM.

The methods of deriving transition probabilities and emission probabilities are different, we will discuss them separately:

4.3.1 Derivation of transition probabilities

The transition probabilities can be specified as numerical values or arithmetic formulae. The formulae can consists of the basic arithmetic operators: '$+$','$-$','$*$','$/$' and brackets, with the id of the free parameters and numerical values as operands. The free parameter is not necessarily to a probability itself, i.e. it need not be in the interval $[0, 1]$. Note that the normalization of the transition probabilities has to be implicitly enforced by the arithmetic formulae, in this way HMMCONVERTER can apply consistency checks for the normalization of transition and emission probabilities to ensure robustness.

4.3.2 Derivation of emission probabilities

The derivation of emission probabilities is more complicated, because the normal arithmetic rules may not be well-defined. The free emission parameters itself has to define an emission probability. Furthermore, the dimension of the parameters has to be matched according to the derivation rules.

HMMCONVERTER provides two functions: "SumOver" and "Product" for deriving emission probabilities. Consistency checks concerning the dimensions of the parameters and the normalization of probabilities enforced within HMMCONVERTER when the two functions for deriving emission probabilities are used.

There are four ways of defining the emission probabilities of a state, assuming a set of free emission parameters has already been defined:

Getting the emission probability directly A set of emission probabilities can be shared with several states, if they have the same dimension. This allows the emission probabilities of a state to be directly derived from a set of emission parameters or a set of emission probabilities of another state.

SumOver Using this function allows to derive a lower dimensional emission parameter from a higher dimensional one. Consider the example of three states: *Match_xy*, *emit_x* and *emit_y* of dimension 6, 3 and 3, respectively. The emission probabilities of *emit_x* and *emit_y* can be derived from that of *match_xy* in the following, intuitive way:

$$P_{emit_x}(x_1, x_2, x_3) = \sum_{(y_1,y_2,y_3)}(P_{match_xy}(x_1, x_2, x_3, y_1, y_2, y_3))$$
$$P_{emit_y}(y_1, y_2, y_3) = \sum_{(x_1,x_2,x_3)}(P_{match_xy}(x_1, x_2, x_3, y_1, y_2, y_3))$$

Product This function allows deriving a high dimensional parameter from several lower dimensional parameters. Consider an example of two states: *triple* and *single* of dimension 3 and 1 respectively. The emission probabilities *triple* can be derived from that of *single* in the following, intuitive way:

$$P_{triple}(x_1, x_2, x_3) = P_{single}(x_1) \cdot P_{single}(x_2) \cdot P_{single}(x_3)$$

SumOver & Product The two functions described above can be used in combination to define more complex rules. Consider, for example, three states of dimension 3, 1 and 4, respectively:

$$P_{complex}(x_1, x_2, x_3) = P_{single}(x_3) \cdot \sum_{(y_1,y_2)}(P_{4-tuple}(x_1, x_2, y_1, y_2))$$

The above rules are easy to apply and flexible enough for most useful applications. They are well-defined to guarantee normalized sets of emission probabilities.

4.4 Hirschberg algorithm

HMMCONVERTER provides the Hirschberg algorithm [5] as memory-efficient alternative to the Viterbi algorithm [12] for sequence decoding. It reduces the memory usage of the Viterbi algorithm by an order of the length of one input sequence and at most doubles the time requirement of the Viterbi algorithm. In HMMCONVERTER the Hirschberg algorithm can be combined with banding, i.e. it can be used in a restricted search space for even more memory efficiency, see Figure 5.

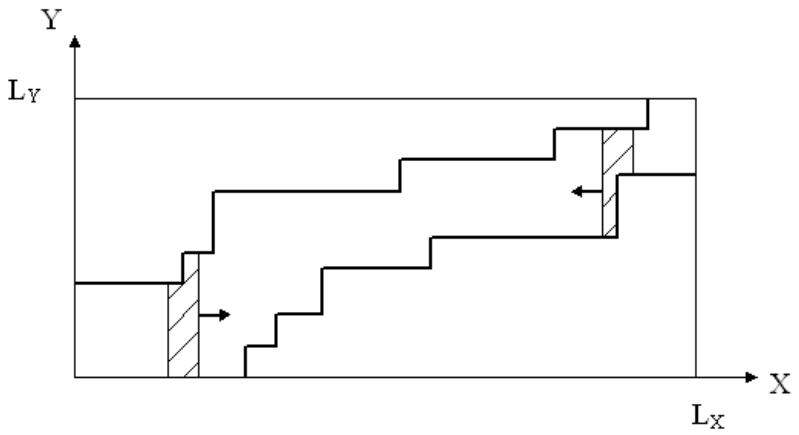

Figure 5: Employing the Hirschberg algorithm with banding, i.e. inside a tube, to further increase the computational efficiency. The two high-lighted Hirschberg strips only search the restricted search space.

4.5 Linear memory Baum-Welch training algorithm

Baum-Welch parameter training [2] is implemented in the memory efficient way introduced by Meyer and Miklós [10, 3]. This new algorithm reduces the memory requirement of Baum-Welch training for an order of the length of input sequence and thereby renders parameter training even for long training sequences and complex models feasible.

For using this type of parameter training algorithm in HMMCONVERTER, the user has to specify the maximum amount of memory available to run this algorithm. HMMCONVERTER then uses this value to optimize the number of parameters to be simultaneously trained in one iteration. As it is often not necessary or desired to train all the parameters of the model, the user can explicitly specify the parameters to be trained. Prior information along input sequences can be integrated into the training process by switching on the feature of special emissions. This can increase the chances of successful parameter training. In addition, the linear-memory Baum-Welch training algorithm can be combined with banding to further increase the time efficiency of the parameter training while maintaining the same accuracy.

In HMMCONVERTER, three types of parameters that can be trained:

- transition probabilities
- free transition parameters
- emission probabilities and emission parameters

Pseudo-probabilities can be added for training transition and emission probabilities. These pseudo-probabilities are added to the corresponding parameters after deriving the probabilities from the corresponding counts in each iteration of the Baum-Welch training procedure and before the corresponding parameter is normalized. HMMCONVERTER allows specifying a pseudo-probability instead of pseudo-count because the counts depend to a larger extent on the number of training sequences in the training set and the length of sequences. It is thus usually more intuitive for user to come up with a reasonable pseudo-probability for a parameter rather than a pseudo-count.

In order to train free transition parameters, the user has to provide a derivation formula for each free parameter to be trained. In this case, pseudo-counts can be set for the parameters, which are added to the observed counts for that parameter after each iteration in the training process. Pseudo-counts need not have integer values and can even be a negative, because a free transition parameter can be any number (as we have explained in the previous section), and the normalization of the transition probabilities are implicitly included in the derivation formulae. More details on the specification of training free parameters are described in section 5.2.

4.6 Two novel, memory efficient parameter training algorithms

Linear-memory Posterior Sampling Training algorithm This new type of parameter training algorithm is implemented in HMMCONVERTER [7]. This algorithm uses a similar iterative process as Viterbi training, but samples several state paths from the posterior distribution rather than taking one a single Viterbi state path into account. This strategy improves the parameter estimation in training and reduces the over-fitting problem that often occurs in Viterbi training. Posterior sampling training is implemented in a computationally efficient algorithm [6, 7].

Linear-memory Viterbi Training algorithm We implement a significantly more efficient and new algorithm for Viterbi training in HMMCONVERTER [6, 7]. Our new algorithm reduces the memory requirements of the classical Viterbi training algorithm [4] by a factor of the length of one of the input sequences.

5 Specifying an HMM or a pair-HMM with HMMCONVERTER

HMMCONVERTER takes up to six input files if all special features are used. We first give a brief description of these files:

- XML input file: This is a compulsory input file which describes the states and connections of the HMM or pair-HMM.
- Input sequence file: This compulsory file stores the input sequences.
- Free emission parameter file: This is a compulsory flat text file which specifies the emission probabilities of the model.
- Free transition parameter file: This file is only required if the transition probabilities are parameterized.
- Tube file: This file is only required if *banding* is used with a user-defined tube.
- Prior information file: This file is only required if *special emissions* are employed, i.e. if prior information on the input sequences shall be taken into account for generating predictions or for parameter training.

In this section, the format of all the input files and the output files generated by HMMCONVERTER is described in detail. We also introduce several HMMCONVERTER applications included in the software package which demonstrate how to set up different HMMs with different features in HMMCONVERTER.

6 The XML input file

HMMCONVERTER takes an XML input file and translates it into efficient C++ code. The XML input file consists of two large components, defined by the <model> and the <sequence_analysis> tag. In this section, we explain the XML tags by first showing an example of how to specify the tag, and then describing its attributes. The term "boolean attribute" used in this section refers to an attribute that only accepts two discrete values "0" and "1" where "0" means "No" and "1" means "Yes".

6.1 The <model> component of the XML file

This component consists of five compulsory tags for specifying the model, and two optional tags for using special features.

[i] <Model_Type> This tag defines some general parameters of the HMM.
Example tag:

<Model_Type name="Gene Prediction" pair="1" SpecialEmission="0" />

Attributes of <Model_Type>:

- *name*: This compulsory attribute defines the name of the model.

- *pair*: This optional boolean attribute defines whether the model is a pair-HMM or an HMM. The default value is ”0” (HMM).
- *SpecialEmission*: This optional boolean attribute defines whether the model uses special emission or not. The default value of this attribute is ”0” (no special emissions).

[ii] <**Alphabets**> This tag defines the set of symbols, i.e. the alphabet, the set of symbols that the model can read from the input sequences
Example tag:

<Alphabets set=”ACGT” cases=”0”>

Attributes of <Alphabets>:

- *set*: This compulsory attribute defines the set of symbols, i.e. the alphabet, that the model can read.
- *cases*: This optional boolean attribute defines whether the model is case sensitive (”1”) or not (”0”). The default value of this attribute is ”0”.

[iii] <**Emission_Probs**> This tag defines the attributes of the set of free emission parameters on which the emission probabilities of the model depend. The free emission parameters themselves are defined in a separate input file, we call it the free emission parameter file.
Example tag:

<Emission_Probs id=”FEP” size=”7” file=”emission_probs_mouse_human.txt” >

Attributes of <Emission_Probs>:

- *id*: This compulsory attribute defines the id of the set of free emission parameters. This id is used to find the corresponding emission parameters in the free emission parameter file.
- *file*: The name of the free emission parameter file.
- *size*: Compulsory attribute which specifies the number of free emission parameters contained in the free emission parameter file.

In this example, ”FEP” is the id of the set of 7 free emission parameters (size=”7”), and the ids of the free emission parameters specified in the free emission parameter file would be ”FEP.0”,”FEP.1”,$\cdots$,”FEP.6”.

[iv] <**States**> This tag defines the attributes of all states in the model. For a model with N states, the **start** state should be defined with id S.0 and the **end** state should be defined with id S.$(N-1)$.
Example tag:

<State id=”S.14” name=”sample_state” xdim=”2” ydim=”3” special=”1” >
<Set1>
<label idref=”Set1.2”/>
</Set1>
<Set2>
<label idref=”Set2.0”/>
<label idref=”Set2.0”/>
<label idref=”Set2.1”/>
<label idref=”Set2.1”/>
<label idref=”Set2.2”/>
</Set2>
<State_Emission_Probs $\cdots$ >
$\vdots$
<State_Emission_Probs />
</State>

Attributes of <State>:

- *id*: This compulsory attribute takes a string value. The format of the string starts with "S.", followed by an integer number in the set $\{0, 1, \cdots, N-1\}$, where N is the number of states in the model.
- *name*: This compulsory attribute defines the name of the state.
- *xdim*: This optional attribute defines the number of characters that this state reads from input sequence x. The default value is "0".
- *ydim*: This optional attribute defines the number of characters that this state reads from sequence y, the default value is "0". This attribute is only required for states in pair-HMMs.
- *special*: This optional boolean attribute defines whether the special emission of this state is switched on ("1") or not ("0").

The state (namely "sample_state") described in the above example tag is shown in Figure 6. This state reads five characters, two from input sequence X and three from input sequence Y, we call this a state of dimension five. For the annotation label set "Set1" defined for the HMM, only one label "Set1.2" (this is the id of the annotation label defined in this annotation label set) is specified, which means that all the five sequence positions of the state have the same annotation label "Set1.2" from this set. For "Set2" (another annotation label set defined for the HMM), different labels are specified for each of the five sequence positions $(x_1, x_2, y_1, y_2, y_3)$, i.e. the annotation label in "Set2" for y_2 is "Set2.1".

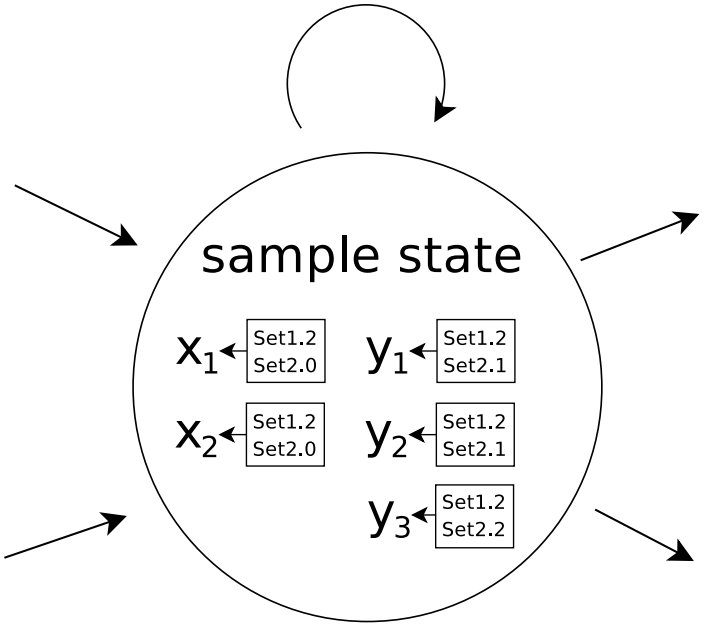

Figure 6: The figure shows the state defined in the above example tag. This state reads five letters at a time, two from input sequence X and three from input sequence Y. Each of the sequence position is assigned two annotation labels, one from the annotation label set "Set1" and one from "Set2". In this particular example, all five sequence positions are assigned the same annotation label "Set1.2" from the annotation label set "Set1", while different annotation labels from annotation label set "Set2" are assigned to each sequence positions. For example, the sequence position y_2 is assigned with "Set1.2" from annotation label set "Set1" and "Set2.2" from annotation label set "Set2".

The name of the <Set1> (and <Set2>) tag has to match with the name of the corresponding annotation label set specified in the <Annotation_Label> tag. The annotation labels defined in this <state> tag are only for specifying the annotation labels for this particular state, the complete annotation label sets for the HMM and the annotation labels defined in each annotation label set are specified in the <Annotation_Label> tag. The <Annotation_Label> tag is an optional tag inside the <model> component which will be explained after describing all the compulsory tags.

The attribute of the <label> tag inside the <state> tag is:

- *idref*: This attribute stores the id of an annotation label in a string. The string has to start with the name of the corresponding annotation label set, followed by a "." character and then a number between 0 to the total number of labels defined in this annotation label set.

The <State_Emission_Prob> tag defines the emission probabilities of a state. There are two ways to define this tag, the simpler way is the following:

<State_Emission_Probs GetFrom="S.2" >

The attribute *GetFrom* specifies the id of a free emission parameter defined in the free emission parameter file or the id of a state in the HMM. If a state is specified, this means the two states share the same set of emission probabilities.

The second way to define this tag is to specify how the emission probabilities are derived from free emission parameters. In that case, the <State_Emission_Prob> tag has to be defined in the following way:

<State_Emission_Probs SumOver="1" Product="1">
<SumOver From="S.3" FromPos="0 1 3 4" ThisPos="0 1 4 5"/>
<Product From="S.14" FromPos="0 1 2 3" ThisPos="2 3 6 7"/>
</State_Emission_Probs>

The attributes of the <State_Emission_Prob> tag are:

- *SumOver*: This boolean attribute defines if the **SumOver** function is used ("1") or not ("0"). The default value is "0".
- *Product*: This boolean attribute defines if the **Product** function is used ("1") or not ("0"). The default value is "0".

The concept of these two functions has already is explained in detail by Lam [6]. We now describe how to set up the functions. Consider an example where we use the **SumOver** function to derive the emission probabilities of state **S.3** from those of state **S.6**:

$$P_{S.3}(x_1, x_2, x_3) = \sum_{(y_1, y_2, y_3)} P_{S.6}(x_1, x_2, x_3, y_1, y_2, y_3) \tag{2}$$

This could be specified by:

<State_Emission_Probs SumOver="1">
<SumOver From="S.6" FromPos="0 1 2" ThisPos="0 1 2"/>
</State_Emission_Probs>

ThisPos="0 1 2" and FromPos="0 1 2" means that the 0-th, 1-th and 2-th character read by **S.3** match with the 0-th, 1-th and 2-th character read by **S.6** respectively, i.e. that we sum over the 3-rd, 4-th and 5-th character of state **S.6** to obtain the emission probabilities of state **S.3**.
The attributes of the <SumOver> tag are:

- *From*: The id of a state or an emission parameter.
- *FromPos* and *ThisPos*: *FromPos* is a vector of positions of the state or emission parameter specified in the *From* attribute. The *FromPos* vector and the *ThisPos* vector have to have the same length as they map the sequence positions in the *From* state to the sequence position in this state. Here, "0" represents the first character read by the state,"1" and "2" represents the second and third character respectively.

For the **Product** function, consider an example with states **S.3** and **S.1** of dimension 3 and 1 respectively where the emission probabilities of state **S.3** depend in the following way on those of state **S.1**:

$$P_{S.3}(x_1, x_2, x_3) = P_{S.1}(x_1) \cdot P_{S.1}(x_2) \cdot P_{S.1}(x_3) \tag{3}$$

The corresponding specification is:

<State_Emission_Probs Product="1">
<Product From="S.1" FromPos="0" ThisPos="0"/>
<Product From="S.1" FromPos="0" ThisPos="1"/>
<Product From="S.1" FromPos="0" ThisPos="2"/>
</State_Emission_Probs>

The attributes of the <Product> tag are the same and have the same meaning as those of <SumOver> tag. In the above example, the 0-th, 1-th and 2-th character read by state **S.3** are all mapped to the 0-th character read by state **S.1**.

The **Product** and **SumOver** functions can be combined, as the following example shows where we have three states **S.3**, **S.1** and **S.4** with dimension 3, 1 and 4, respectively.

$$P_{S.3}(y_1, y_2, y_3) = (\sum_{(x_1,x_2)} (P_{S.4}(x_1, x_2, y_1, y_2))) \cdot P_{S.1}(y_3) \tag{4}$$

The corresponding specification would be:

<State_Emission_Probs SumOver="1" Product="1">
<SumOver From="S.4" FromPos="2 3" ThisPos="0 1" />
<Product From="S.1" FromPos="0" ThisPos="2"/>
</State_Emission_Probs>

Users can also use the combinations of **Product** and **SumOver** with one sequence position involved in more than one operation.

The <State_Emission_Prob> tag is not required in two cases:

(i) A silent state is a state that does not read any symbols from any input sequence.

(ii) All the emission probabilities of the HMM are specified explicitly in the free emission parameter file, the numbering of emission probability sets start with "FEP.0" ("FEP" is the id of the set of free emission parameter).

Regarding case (ii), if this state (**S.14**) is not defined with the <State_Emission_Prob> tag, the emission probabilities of this state is assigned the free emission parameter "FEP.13" defined in the free emission parameter file (as **S.0** is the **start** state which does not read any letter, and the emission probabilities of **S.1** will be assigned with "FEP.0"). This is implemented for convenient set-up of the model, if the emission probabilities of the states in the HMM are not parameterized.

If the emission probabilities are not parameterized, the user has to specify a set of emission probabilities for each state. In this case, the user may skip the <State_Emission_Probs> tag in the XML file and specify the set of emission probabilities in the txt file in the same order as the states appear in the XML file. HMM Converter will then map the emission probabilities in the emission probabilities txt file to the states (following the same order). Please see the XML file of the p53transcription example for how this works in practice.

[v] <Transitions> This tag defines the topology of the model, i.e. the transitions between states in the HMM and the corresponding probabilities.
Example tag:

<Transitions train="1">
<from idref="S.0">
<to idref="All" exp="1/53"/>
</from>
⋮
<from idref="S.3" >
<to idref="S.3" exp="0.6" pseudoprob="0.2"/>
⋮
<to idref="S.53" exp=0.01" />
<from idref="S.4" train="1" >
<to idref="S.3" exp="FTP.15*(1−FTP.2)"/>
<to idref="S.4" exp="(1−FTP.15)∗(1−FTP.2)"/>
<to idref="S.53" exp="FTP.2"/>
</from>
⋮
</Transitions>

The attributes of the <Transitions> tag are:

- *train*: This optional attribute takes the value ”0”, ”1” or ”All”. The default value of this attribute is ”0” which means that none of the transition probabilities is trained. The value *train* =”1” indicates that some transitions are to be trained. These transitions have to be matched by setting the *train* value in the corresponding ”from” state to ”1”. A *train*=”All” values implies that all transitions of the HMM are to be trained.

The attributes of the <from> tag are:

- *idref*: This compulsory attribute defines the state from which this transition is. The value of *idref* has to match the id of the ”from” state.
- *train*: This optional boolean attribute defines whether the transition probabilities of the ”from state” need to be trained or not. The default value is ”0”.

The attributes of the <to> tag:

- *idref*: This compulsory attribute specifies the ”to” state of the transition. The value of *idref* has to either match the id of the ”to” state, or is set to ”All” which means that the ”from” state is connected to all other states of the HMM except the **start** and the **end** state using the same transition probability.
- *exp*: This compulsory attribute defines the transition probability. It takes a string with either of the following three formats:
  - Decimal number
  - Arithmetic formula with decimal numbers as operands. The formula accepts operators: ’$+$’,’$-$’,’$*$’ and ’$/$’ representing addition, subtraction, multiplication and division respectively. Brackets are also allowed in the expression.
  - Arithmetic formula with decimal numbers and free transition parameters id as operands.
- *pseudoprob*: This optional attribute defines a pseudo-probability to be used for this transition during parameter training.

All the five tags described above, i.e. <Model_Type>, <Alphabets>, <Emission_Probs>, <States> and <Transitions>, are compulsory for specifying an HMM. Two optional tags can be used for setting up special features with HMMCONVERTER.

[vi] <**Annotation_Labels**> HMMCONVERTER can use *special emissions* to integrate prior information along the input sequences into the sequence decoding and the training algorithms. Prior probabilities are specified with the name of the annotation labels along the input sequences, those names should match with the name of the annotation labels defined in the states of the HMM. The details of these annotation label sets for the HMM are defined in this <Annotation_Labels> tag.
Example tag:

```
<Annotation_Labels>
        <Annotation_Label name=”Feature” score=”1” >
                <label id=”Feature.0”name=”Undefined” />
                <label id=”Feature.1” name=”Intergenic” />
                ⋮
                <label id=”Feature.6” name=”Stop_Codon” />
        </Annotation_Label>
</Annotation_Labels>
```

The attributes of the <Annotation_Label> tag are:

- *name*: This compulsory attribute defines the name of the annotation label.
- *score*: This optional boolean attribute defines whether the annotation label set accepts a prior probability in $[0, 1]$ (”1”) or only discrete values ”on” or ”off” (”0”). The default value is ”0”.

The attributes of the <label> tag are:

- *id*: This compulsory attribute defines the id of the annotation label. The string has to start with "Annotation_Label_name.", followed by an integer in the set $\{0, L-1\}$, where L is the size of the annotation label set.
- *name*: This compulsory attribute defines the name for the label.

[vii] <**Transition_Probs**> This tag defines a set of free transition parameters, the values of the free transition parameters are defined in a separate input file, we call it the free transition parameter file.
Example tag:

<Transition_Probs id="FTP" size="19" file="transition_parameters_mouse_human.txt" />

The attributes of the <Transition_Probs> tag are:

- *id*: This compulsory attribute defines the id of this set of free transition parameters. This id is used to find the corresponding values in the free transition parameter file.
- *size*: This compulsory attribute specifies the number of parameters in the free transition parameter file.
- *file*: The file name of the free transition parameter file.

6.2 The <sequence_analysis> component of the XML file

The purpose of this tag is to specify the decoding and training algorithms that are to be used with the model and to specify the algorithms parameters.

This tag consists of two main tags, the <sequence_decoding> and the <parameter_training> tag. We explain these tags in order in this section.

6.2.1 The <sequence_decoding> tag

All parameters for sequence decoding algorithms are specified in the this tag. It consists of three sub-tags:

- <input_files>
- <algorithm>
- <output_files>

<**input_files**> This tag specifies the names of different input files for the sequence decoding algorithm.
Example tag:

<input_files SeqFile="seq1.fasta" AnnFile="priorinfo1.txt" TubeFile="tube.txt" />

The attributes of the <input_files> tag are:

- *SeqFile*: This compulsory attribute specifies the name of the input sequence file.
- *AnnFile*: This optional attribute specifies the name of the prior information file. This attribute is not required if *special emissions* are not used.
- *TubeFile*: This optional attribute specifies the name of the tube file that defines the restricted search space. This tag is only required if algorithm 3 is chosen.

<**algorithm**> This tag specifies the sequence decoding algorithm to be used and its parameters.
Example tag:

<algorithm alg="1" MaxVolume="300000" radius="30" />

The attributes of the <algorithm> tag are:

- *alg*: This compulsory attribute defines the decoding algorithm being used, it has four options:
  - 0: A combination of normal Viterbi algorithm and the Hirschberg algorithm will be used depending on the maximum amount of memory specified by the user to optimize the time requirement.

  - 1: Derives a tube from BLASTN matches and run algorithm 0 inside the resulting tube.
  - 2: Derives a tube from TBLASTX matches and run algorithm 0 inside the resulting tube.
  - 3: Runs algorithm 0 inside a user-defined tube-defined by an input file.

- *MaxVolume*: This compulsory attribute defines the maximum amount of memory in bytes to be allocated for running any decoding algorithm.

- *radius*: This attribute specifies the radius of the tube generated from matches given by the blast programs, see Figure 1, i.e. this attribute is only required if algorithm 1 or algorithm 2 above are chosen.

<**output_files**> This tag specifies the name of the output files which store the results generated by the sequence decoding algorithm.
Example tag:

<output_files OutFile="viterbi_result.txt"/>

The attribute of the <output_files> tag is:

- *OutFile*: This compulsory attribute specifies the name of the output file. The file will be created if it does not exist in the output directory, otherwise, the existing file will be overwritten.

HMMCONVERTER generates at most two output files for storing the result of sequence decoding algorithm. The output files generated are both in simple text formats. The first output file shows the optimal state path explicitly, and the second one shows the resulting annotation of the sequences. If no annotation label set is defined, the second file is not generated. If both output files are generated, the output files name are **viterbi_result_1.txt** and **viterbi_result_2.txt** respectively in the above example.

6.2.2 The <parameter_training> tag

This tag specifies the parameters of the training algorithms. The general format is similar to that of the <sequence_decoding> tag, except this tag has an additional sub-tag while specifies the derivation rules for training free transition parameters. For this reason, only the tags and attributes that are different from the <sequence_decoding> tag are explained in detail.
The <parameter_training> tag consists of four sub-tags:

- <input_files>
- <algorithm>
- <train_free_parameters>
- <output_files>

<**input_files**> This is the same tag as for the <sequence_decoding> tag, it specifies the names of the different input files for parameter training. The attribute *TubeFile* in this <input_files> tag is different from that in the <sequence_decoding> tag. In addition to specifying the name of the user-defined tube file, it also accepts the value "Blastn" and "TBlastX", which means the tubes for the parameter training would be generated from the matches provided by the specified BLAST program.

<**algorithm**> The <algorithm> tag for parameter training is similar to that for sequence decoding. We only describe the attributes that are different from those for sequence decoding:
Example tag:

<algorithm alg="1" MaxVolume="100000" Maxiter="5" threshold="0.01" SamplePaths="10" />

The attributes for the <algorithm> tag are:

- *alg*: This attribute specifies which training algorithm is to be used, it accepts two values:
  - 0: The linear memory Baum-Welch algorithm [10, 3]
  - 1: The linear memory posterior sampling training [6, 7]

- 2: The linear memory Viterbi training [6, 7]

- *threshold*: This attribute specifies a threshold score for terminating the Baum-Welch training algorithm. Its default value is 0, the algorithm will be terminated if the average log-odd ratio in the current iteration is less than this threshold value. Note that this attribute does not apply to training algorithms 1 and 2.
- *Maxiter*: This attribute specifies the maximum number of iterations for running the training algorithm, the default value is 10.
- *SamplePaths*: This attribute specifies the number of state paths that are to be sampled for every training sequence and every iteration in algorithm 1.

The training algorithm stops as soon as the first of the two convergence criteria are met.

parameter_training This tag is needed for training the free transition parameters. As discussed in the previous section, the user has to specify rules for deriving free transition parameters from transition probabilities. Take a look at Figure 7 again, we want to train the "Phase0" parameter by counting the transitions $3 \rightarrow 14$, $3 \rightarrow 17$ and $3 \rightarrow 20$, and then normalizing by all transitions from state 3 to any of the states $14 - 22$.

The following example shows how to specify the above "Group transition probabilities":

```
<Parameters_training>
    <FreeTransitionParameters>
        <FTP idref="FTP.0" exp="GTP.1" />
                ⋮
        <FTP idref="FTP.2" exp="GTP.2+GTP.3" />
    </FreeTransitionParameters>
    <GroupTransitions id="GTP">
                ⋮
        <GTP id="GTP.1">
            <from idref="S.3">
                <to idref="S.14">
                <to idref="S.17">
                <to idref="S.20">
            </from>
            <Overfrom idref="S.14">
                <Overto idref="All">
            </Overfrom>
            <Overfrom idref="S.17">
                <Overto idref="All">
            </Overfrom>
            <Overfrom idref="S.20">
                <Overto idref="All">
            </Overfrom>
        </GTP>
                ⋮
    </GroupTransitions>
</Parameters_training>
```

In the above example, **GTP** stands for "Group transition probabilities". For example, the parameter **GTP.1** specifies the group of transitions for training **Phase0**. In the above example, another free transition parameter **FTP.2** is derived by adding two group transition probabilities. One important point to mention here is that in order to train a free transition parameter, the user has to specify this in the tag <FreeTransitionParameters>. Furthermore, an appropriate expression which should be an arithmetic formula using group transition probabilities and numerical values as operands should be given. In the above example, **FTP.1** is not specified so the parameter is not being included in the parameter training.

No rule is needed to specify the training of the free emission parameters, as discussed previously. The counts from appropriate states for an emission parameter can be determined within the parameter training process. For a more detailed explanation of this strategy, see Lam [6].

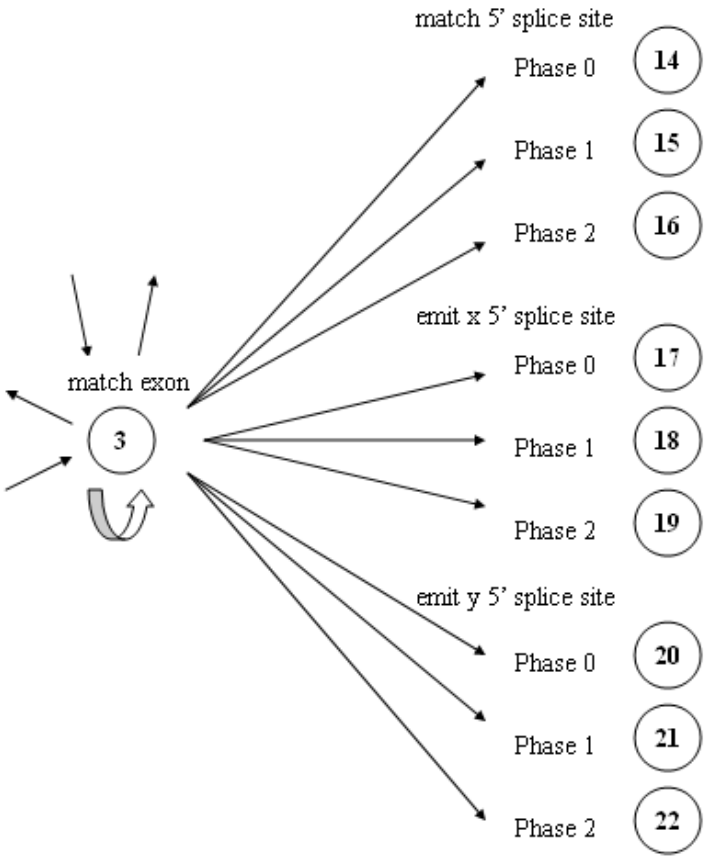

Figure 7: In order to train the free transition parameter **Phase0** with id **FTP.0**, all transitions from state 3 to any state presenting a splice site of phase 0 have to be counted, i.e. transitions $3 \rightarrow 14$, $3 \rightarrow 17$ and $3 \rightarrow 20$. The group transition **GTP.1** is set up correspondingly.

output_files This tag specifies the output file for storing the result of parameter training. At most three output files are generated.
Example tag:

<output_files XMLFile="trained_HMM.xml"
TProbFile="trained_transition.txt"
EProbFile="trained_emission.txt"/>

The attributes for the <output_files> tag are:

- *XMLFile*: This attribute specifies the name of the XML output file for describing the HMM after the training process. i.e. if transition probabilities are trained, the updated values will appear in this output XML file. This is convenient for users to compare the HMM before and after parameter training and to set up to new HMM for data analyses using one of the decoding algorithms.
- *TProbFile*: This attribute specifies the name of output file that stores the trained free transition parameters. This file has the same format as that of the input free transition parameter file.
- *EProbFile*: This attribute specifies the name of output file for storing the trained emission parameters. This file has the same format as that of the input free emission parameter file.

7 Other text input files

HMMCONVERTER takes several flat text files as input for setting up several features of an HMM. In this section, we describe the format of these text input files in detail.

7.1 The input sequence file

This file stores the input sequences for decoding or training algorithms in a fasta-like format. Once again, HMMCONVERTER is not limited to Bioinformatics applications. All sequences in this input sequence file are considered for the specified algorithm. The following is an example of an input sequence file:

```
>Mm.X13235.5  1-637
gggaatgaagtttttctgcaggatttaaatgtggtctttaagagacaccgcatgcaaaga
atagctggggcttgctagccaatgaaaacattcagattccaatgacgcatccttttttct
ccacccccttccaagacccggattcggaaaccccgcctaacgctctagttttcaaccagg
tccgcagaaggcctatttaaagggacgattgctgtctccctgctgtcataaccatgtctg
gacgtggcaagggtggtaaaggccttgggaaaggcggcgctaagcgccaccgtaaggttc
tccgcgataacatccagggcatcaccaagcctgccatccgccgcctggcccggcgcgggg
gagtgaagcgcatctccggcctcatctacgaggagacccgcggtgtgctgaaggtgttcc
tggagaacgtgatccgcgacgccgtcacctacacggagcacgccaagcgcaagaccgtca
ccgccatggacgtggtctacgcgctcaagcgccagggccgcactctctacggattcggcg
gttaatcgactaacaaacgattttccactgtcaacaaaaggccctttcagggccaccca
caaattcctagaaggagttgttcacttaccgaagctt
>Hs.M16707.H4F2.8  85-1098
ccgtcctccttcggcgaagatccctggcgcgcgtccttgaggtcgccttcggtgttgacc
tcatcgtcggaacggcgcttcctgaagctttatataagcacggctctgaatccgctcgtc
ggattaaatcctgcgctggcgtcctgccagtctctcgctccatttgctcttcctgaggct
ccctccagagacctttcccttagcctcagtgcgaatgcttccgggcgtcctcagaaccag
agcacagccaaagccactacagaatccggaagcccggttgggatctgaattctcccgggg
accgttgcgtaggcgttaaaaaaaaaaaagagtgagagggacctgagcagagtggaggag
gagggagaggaaaacagaaaagaaatgacgaaatgtcgagagggcggggacaattgagaa
cgcttcccgccggcgcgctttcggttttcaatctggtccgatactcttgtatatcagggg
aagacggtgctcgccttgacagaagctgtctatcgggctccagcggtcatgtccggcaga
ggaaagggcggaaaaggcttaggcaaaggggcgctaagcgccaccgcaaggtcttgaga
gacaacattcagggcatcaccaagcctgccattcggcgtctagctcggcgtggcggcgtt
aagcggatctctggcctcatttacgaggagacccgcggtgtgctgaaagtgttcttggag
aatgtgattcgggacgcagtcacctacaccgagcacgccaagcgcaagaccgtcacagcc
atggatgtggtgtacgcgctcaagcgccaggggagaaccctctacggcttcggaggctag
```

```
gcgccgctccagctttgcacgtttcgatcccaaaggccctttttgggccgaccacttgct
catcctgaggagttggacacttgactgcgtaaagtgcaacagtaacgatgttggaaggta
actttggcagtggggcgacaatcggatctgaagttaacggaaagacataaccgc
```

If a pair-HMM is used for an analysis, the entries in the input sequence file will be interpreted and analyzed in pairs, i.e. the first two sequence are regarded as a pair, and then the third and the forth, etc. The header line has to start with the character ">", followed by the name of the sequence (a string without white spaces) and the start and end position of the sequence below that header line, the start position always has to be smaller or equal to the end position and both numbers have to be integers. The name and the positions are separated by a space or a tab character. Other information about the sequence can be put on the header line after the range, which is for users reference only.

7.2 The free emission parameter file

This flat text format file is required for setting up the emission probabilities of an HMM. We will call a grouped set of emission probabilities which defines the emission probabilities for a state in the HMM a free emission parameter. All free emission parameters are stored in this file.

We specify a free emission parameter with a header line with the following format:

FEP.1	Stop_exon	6	train

The first three columns contain the id, the name of the emission parameter and the dimension of the parameter, respectively, the forth column tells whether this parameter is to be included in the training process. The four columns are separated by a space or a tab character.

The id has to be identical to the one that was defined in the XML input file, the name of the emission parameter is optional and defaults to "Not defined". Similarly, the dimension specified in the header line has to be consistent with the dimensions of the emission probabilities follow the header line. If the parameter set does not needed to be trained, then the word "train" should be omitted from the header line. The program can correctly interpret a header line with 3 entries by recognizing the position of the key word "train" and the position of the dimension parameter, which is the only parameter with numerical value.

Below the header line, the emission probabilities are defined using the following format:

taataa	0.5082690	0.10
taatag	0.0632332	0.05
taatga	0.0943092	0.05
tagtaa	0.0632332	0.05
tagtag	0.0607797	0.05
tagtga	0.0160113	0.01
tgataa	0.0943092	0.05
tgatag	0.0160113	0.01
tgatga	0.0838441	0.05

The first column is the string of characters that corresponds to the emission probability in the second column. The third column is the value of an optional pseudo-probability to be used in training which defaults to 0 if it is not specified. The three columns are separated by a space or a tab character.

Emission probabilities of unspecified character combinations are set to zero. We can use this feature to specify the emission probabilities of some states in a very compact way. For example, if a **Match_Start_Codon state** reads only "atgatg" with emission probability larger than zero, we can specify its emission probabilities by:

FEP.1	Match_Start_Codon	6
atgatg	1	

After specifying one free emission parameter, there has to be an empty line before specifying another parameter.

7.3 The free transition parameter file

This file defines the free transition parameters. This input file is not required if the transition probabilities are not parameterized. Each line in the file defines a free transition parameter using the following format:

FTP.0	Phase0	0.4387	0.3333
FTP.1	Phase1	0.3870	0.3333
	⋮		
FTP.16	Match_exon_to_stop_exon	0.003	0.01
FTP.17	Match_exon_to_emit_exon_x	0.01	
FTP.18	Match_exon_to_emit_exon_y	0.01	

The four columns contain the id, the name, the value and the pseudo-probability of the transition parameter respectively. The id has to be identical to the one defined in the XML input file. The name is an optional attribute where default value is "Not defined". The default value for the pseudo-probabilities is 0. In this free transition parameter file, each line represents one free transition parameter, there is no need to have an empty line between two parameter entries.

7.4 The tube file

This input file stores the user-defined restricted search space in the sequence dimension(s) to be used for sequence decoding or parameter training. Note that the *banding* feature only applies to pair-HMMs.

The tubes for all input sequence pairs have to be specified in a single tube file. A header line is required for specifying a tube for each sequence pair, using the following format:

>Mm.X13235.5&Hs.X60484.H4/e.6 radius:50

Each header line starts with the character ">" followed by the name of the sequence pair to which the tube will be applied to. The name of the two sequences are connected by the character '&', i.e. in the above example, the names of the two input sequences are Mm.X13235.5 and Hs.X60484.H4/2.6. The radius is defined in the next field of the header line, if the specified coordinates below represent matches.

The coordinates for specifying the tube itself are given in the following format:

300	400	450	550
600	650	800	750

The four columns of each line are "xstart", "ystart", "xend" and "yend", respectively, which together define two points in the X-Y plane: (xstart,ystart) and (xend,yend). Depending on the desired interpretation, each line thus either specifies a rectangular area (see left of Figure 8) or a match (see right of Figure 8). Figure 8 also shows two possible ways of defining a tube from the same set of coordinates. Note that if radius is not defined in the header line, the coordinates are interpreted as rectangles rather than matches.

7.5 The prior information file

This input file stores prior information along the input sequences to be used with the *special emissions* feature of HMMCONVERTER. Each line of this input file corresponds to one piece of information for one sequence interval along one input sequence. It is specified using the following format:

SeqX	Source1	100	150	Feature:Exon:Intron	0.3:0.7
				Phase:NoPhase:Phase0	.:.

The above example actually corresponds to a single line in the prior information file, we only split it into two lines for better readability. Each line contains one piece of prior information for one sequence interval, it consists of several columns separated by a space or a tab character.

The first four columns specify:

- SeqX: The name of the sequence to which this piece of information applies.
- Source1: The source of this piece of information, e.g. a database name or the name of a human annotator.

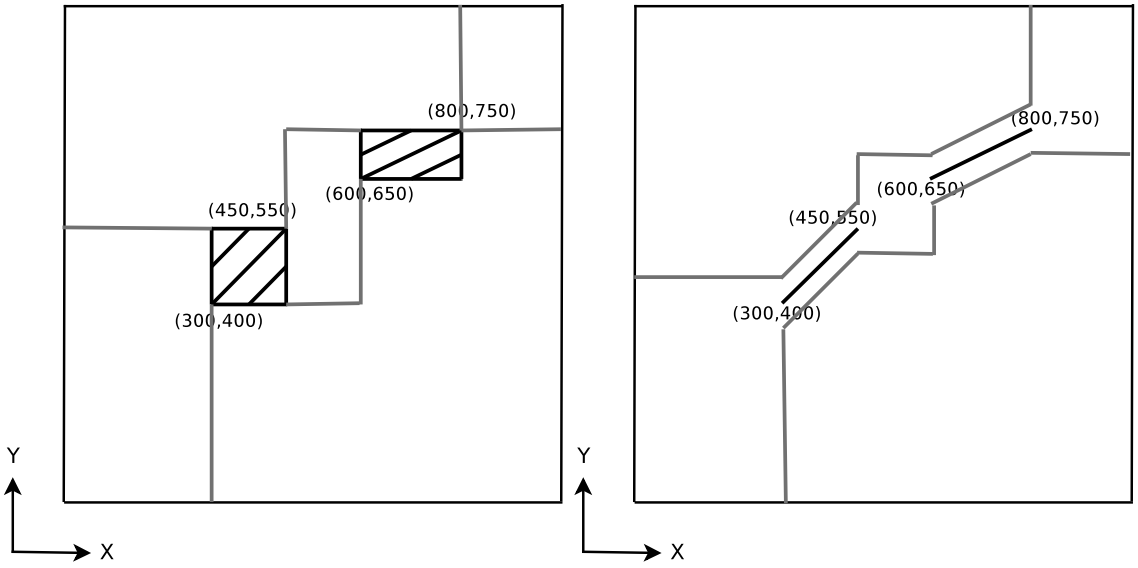

Figure 8: The tube in the right part is constructed by interpreting the coordinates as matches (with radius around them 50), while the tube on the left is constructed by interpreting the coordinates as rectangles.

- 100: The start of the sequence interval to which this information applies, i.e. the first sequence position to which this information applies.
- 150: The end of the sequence interval to which this information applies, i.e. the last sequence position to which this information applies.

The prior information itself is defined starting in the fifth column. The above example defines two types of prior information, information of type "Feature" and of type "Phase":

(i) Feature:Exon:Intron 0.3:0.7

(ii) Phase:NoPhase:Phase0 .:.

For each type of prior information, the first entry specifies the annotation label set and the annotation labels, e.g. "Feature:Exon:Intron", and following entry specifies the corresponding prior probabilities, e.g. 0.3 for "Exon" and 0.7 for "Intron". Users can define any number of types of prior information within a line as long as they concern the same sequence interval.

We now consider prior information for annotation label set "Feature" in (i) above. It means that sequence interval $[100, 150]$ has a prior probability of 0.3 for being "Exon" and a prior probability of 0.7 for being "Intron". The prior information for annotation label set "Phase" in (ii) has the same format, but the prior probability is '.' instead of a numerical value, because the annotation label set "Phase" is an on/off label set. In this example it means that the sequence positions in the interval $[100, 150]$ has value "NoPhase" or "Phase0".

Finally, suppose there was a third annotation label set "State_Type" defined for the HMM. As no information for this label set is defined for sequence interval $[100, 150]$, there the nominal emission will not be biased by this label set. The concept of *special emissions* in HMMCONVERTER and how to specify valid prior probabilities are discussed in detail in Lam [6].

8 The output files of HMMCONVERTER

HMMCONVERTER provides two simple output files for storing the results of sequence decoding. The first file stores the sequence of states in the optimal state path, and the second optional file stores the resulting annotation of the input sequences.

Figure 9 shows a simple, 5-states pair-HMM. This pair-HMM consists of two annotation label set, "Set1" $=$ {"align","gap"} and "Set2" $=$ {"match","emitX","emitY"}. Each state (except the **start** state and the **end** state) of the pair-HMM are assigned with one annotation label from each of the annotation label set as shown in

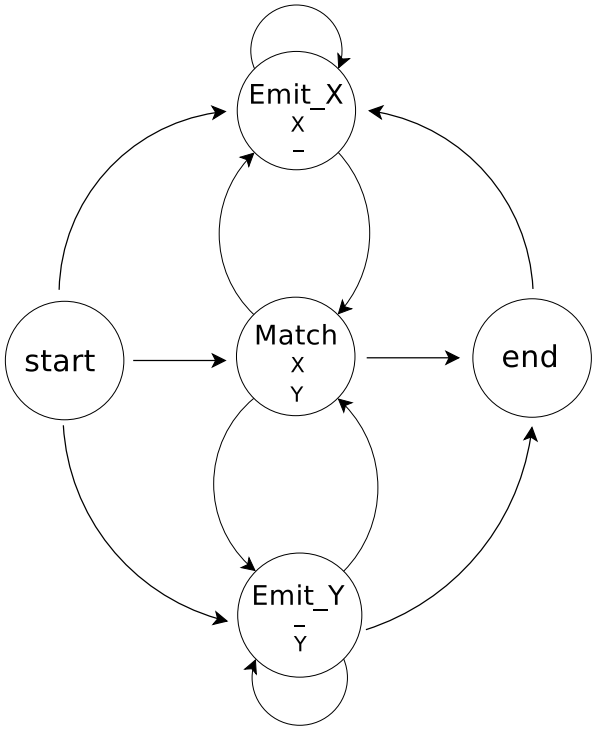

Figure 9: A simple, 5-state pair-HMM which comprises two annotation label sets, “Set1” and “Set2”. Each sequence position of a state in the pair-HMM is assigned one annotation label from each of the annotation label set. For example, every X and Y sequence position in the **Match** state is assigned label ”align” and ”match” from annotation label set “Set1” and annotation label “Set2”, respectively.

the figure. In this example, two output files are generated after running the Viterbi algorithm on a pair of input sequences **SeqX** and **SeqY** of length 10 and 15 respectively, the first file looks like:

SeqX&SeqY:

0	0	start	xdim= 0	ydim= 0				
1	3	Emit_Y	xdim= 0	ydim= 1	Set1_y0=gap	Set2_y0=emitY		
2	3	Emit_Y	xdim= 0	ydim= 1	Set1_y0=gap	Set2_y0=emitY		
3	3	Emit_Y	xdim= 0	ydim= 1	Set1_y0=gap	Set2_y0=emitY		
4	3	Emit_Y	xdim= 0	ydim= 1	Set1_y0=gap	Set2_y0=emitY		
5	3	Emit_Y	xdim= 0	ydim= 1	Set1_y0=gap	Set2_y0=emitY		
6	1	Match	xdim= 1	ydim= 1	Set1_x0=align	Set1_y0=align	Set2_x0=match	Set2_y0=match
7	2	Emit_X	xdim= 1	ydim= 0	Set1_x0=gap	Set2_x0=emitX		
8	1	Match	xdim= 1	ydim= 1	Set1_x0=align	Set1_y0=align	Set2_x0=match	Set2_y0=match
9	1	Match	xdim= 1	ydim= 1	Set1_x0=align	Set1_y0=align	Set2_x0=match	Set2_y0=match
10	2	Emit_X	xdim= 1	ydim= 0	Set1_x0=gap	Set2_x0=emitX		
11	2	Emit_X	xdim= 1	ydim= 0	Set1_x0=gap	Set2_x0=emitX		
12	1	Match	xdim= 1	ydim= 1	Set1_x0=align	Set1_y0=align	Set2_x0=match	Set2_y0=match
13	1	Match	xdim= 1	ydim= 1	Set1_x0=align	Set1_y0=align	Set2_x0=match	Set2_y0=match
14	1	Match	xdim= 1	ydim= 1	Set1_x0=align	Set1_y0=align	Set2_x0=match	Set2_y0=match
15	3	Emit_Y	xdim= 0	ydim= 1	Set1_y0=gap	Set2_y0=emitY		
16	3	Emit_Y	xdim= 0	ydim= 1	Set1_y0=gap	Set2_y0=emitY		
17	3	Emit_Y	xdim= 0	ydim= 1	Set1_y0=gap	Set2_y0=emitY		
18	1	Match	xdim= 1	ydim= 1	Set1_x0=align	Set1_y0=align	Set2_x0=match	Set2_y0=match
19	3	end	xdim= 0	ydim= 0				

Let the optimal state path $S := (S_0, S_2, \cdots, S_{N'})$, the first output file then contains $N' + 1$ lines, one line for each state in the state path, the first line always represents the **start** state and the last line always represents the **end** state. The first column shows the line number, which is also the index in the state path. The second column and third column show the number and the name of the state in the HMM. The forth and fifth column specifies the number of characters read from sequence x and the number of characters read from sequence y respectively. Starting from the fifth column, each column contains the annotation label for the corresponding sequence position read by the state. For example, the position 6 (the 7-th line in the output file) of the optimal state path is a **Match** state, which the sequence position X and sequence position Y of the state are both assigned with annotation label ”align” from annotation label set “Set1” and ”match” from annotation label set “Set2”.

The second format shows the annotation of the sequence. If no annotation label set is defined for the HMM, this output file will not be generated. The following is the second output file of the same state path for the same pair of input sequences:

SeqX	SimplePair-HMM	1	1	Set1=align	Set2=match
SeqX	SimplePair-HMM	1	1	Set1=align	Set2=match
SeqX	SimplePair-HMM	2	2	Set1=gap	Set2=emitX
SeqX	SimplePair-HMM	3	4	Set1=align	Set2=match
SeqX	SimplePair-HMM	5	6	Set1=gap	Set2=emitX
SeqX	SimplePair-HMM	7	10	Set1=align	Set2=match
SeqY	SimplePair-HMM	1	5	Set1=gap	Set2=emitY
SeqY	SimplePair-HMM	6	11	Set1=align	Set2=match
SeqY	SimplePair-HMM	12	14	Set1=gap	Set2=emitY
SeqY	SimplePair-HMM	15	15	Set1=align	Set2=match

Each line presents a contiguous sequence interval that is assigned the same annotation, where all the annotation labels have the same values. Regarding the format, the first column is the sequence name, second column is the name of the HMM specified by users in the XML input file. The third and forth columns indicate the interval of the sequence positions that have the same annotations, the corresponding annotation labels are showed after that.

If the HMM is used with any of the parameter training algorithms, HMMCONVERTER writes three output files to store the results of the parameter training, one XML file and two flat text files. The XML output file is the same as the XML input file that describes the HMM, but contains the trained transition probabilities. The two flat text files are the files with the trained free transition parameters and the files with the trained free emission parameters. They have the same format as the corresponding input files, but contain the trained parameter values.

9 Examples

The HMMCONVERTER software package comprises several examples with all the input and output files. These examples illustrate the range of features supported by HMMCONVERTER.

CpG-Island prediction This simple HMM is introduced in Durbin [4] for predicting the CpG-Islands in a potentially long genomic DNA sequence. This simple example shows how to specify a simple HMM with annotation labels.

Dishonest Casino The example of a dishonest casino HMM is introduced in Durbin [4]. A variation of this HMM using *special emissions* is also included. The *special emissions* version of the HMM can be interpreted as prior knowledge by a casino-insider who provides additional information on when the loaded rather than the fair dice was used. This simple example illustrates how to use *special emissions*.

p53-transcription site prediction This HMM can be used to predict p53 transcription factor binding sites given genomic DNA input sequences. This example shows how parameter training with HMMCONVERTER works. We use the same training set and the test set as MAMOT [11].

Simple pair-HMM This is a simple pair-HMM as shown in Figure 9. This simple example illustrates how to specify a pair-HMM, and how to train its parameters. This example comprises an artificially created training set.

References

[1] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

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

[3] A. Churbanov and S. Winters-Hilt. Implementing em and viterbi algorithms for hidden markov model in linear memory. *BMC Bioinformatics*, 9, 2008.

[4] Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge University Press, Cambridge, 1998.

[5] D.S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18:341–343, 1975.

[6] Tin Yin Lam. *HMMConverter - A tool-box for hidden Markov models with two novel, memory efficient parameter training algorithms*. PhD thesis, University of British Columbia, 2008.

[7] T.Y. Lam and I.M. Meyer. Efficient parameter training for hidden markov models using posterior sampling training and viterbi training. *submitted*, 2009.

[8] I. M. Meyer and R. Durbin. Comparative ab initio prediction of gene structures using pair hmms. *Bioinformatics*, 18(10):1309–1318, Oct 2002.

[9] I.M. Meyer and R. Durbin. Gene structure conservation aids similarity based gene prediction. *Nucleic Acids Research*, 32(2), 2004.

[10] I. Miklós and I.M. Meyer. A linear memory algorithm for baum-welch training. *BMC Bioinformatics*, 6, 2005.

[11] F. Schütz and M. Delorenzi. Mamot: hidden markov modeling tool. *Bioinformatics*, 24(11), 2008.

[12] A. Viterbi. Error bounds for convolutional codes and an assymptotically optimum decoding algorithm. *IEEE Trans. Infor. Theor.*, pages 260–269, 1967.