Development of a modular HMM package for biological sequence analysis

Håkan Viklund

March 18, 2003

Development of a modular HMM package for biological sequence analysis
Abstract

A hidden Markov model (HMM) is a probabilistic finite state machine which is widely used in biological sequence analysis. We have constructed a versatile tool, called modhmm, for constructing HMMs of arbitrary design and applying them to different problems inside the field of biological sequence analysis.

One such sequence analysis problem is protein fold recognition. Using modhmm we have developed a new method for scoring amino acid sequences against HMMs using multiple sequence information in both the target- and query- sequences. Preliminary results show a definite improvement in remote homology detection compared to the standard HMM method of using multiple sequence information only in the query sequence.

Utveckling av ett modulbaserat HMM-paket för biologisk sekvensanalys
Sammanfattning

Dolda markovmodeller (förkortat HMM) är probabilistiska finit-tillståndsmaskiner, vilka används bland annat inom biologisk sekvensanalys. Vi har utvecklat modhmm, ett flexibelt verktyg för att konstruera HMM:er med godtycklig arkitektur och använda dessa på olika problem inom sekvensanalys. Ett viktigt sekvensanalysproblem är proteinveckningsigenkänning. Med hjälp av modhmm har vi utecklat en ny metod för att jämföra en proteinsekvens med en HMM. Denna metod använder multipel sekvensinformation i representationen av både fråge- och mål- sekvenser. Preliminära resultat visar en klar förbättring i att finna avlägset homologa proteiner jämfört med när man använder multipel sekvensinformation enbart i frågesekvensen, vilket är standardmetoden för HMM:er.

Contents

1 Introduction **5**
- 1.1 Biological sequence analysis and protein structure prediction 5
- 1.2 Hidden Markov models in sequence analysis 6
- 1.3 Aims of the project 8

2 Theory **10**
- 2.1 Theory of HMMs 10
  - 2.1.1 Markov chains and hidden Markov models 10
  - 2.1.2 The forward, backward and Viterbi algorithms 11
  - 2.1.3 Expectation maximization and the Baum-Welch training algorithm 14
  - 2.1.4 HMM scoring methods 15
  - 2.1.5 Prior distributions 17
- 2.2 Architectures of HMMs 18
- 2.3 Profiles and multiple sequence alignments 21

3 The modhmm package **23**
- 3.1 Introduction 23
- 3.2 Architecture - modhmmc 23
- 3.3 Training - modhmmt 26
- 3.4 Scoring - modhmms 29

4 Profile-profile comparison **30**
- 4.1 Introduction 30
- 4.2 Methods 30
- 4.3 Results 33

5 Discussion **36**
- 5.1 modhmm package 36
- 5.2 Profile-profile comparison 36

References **38**

1 Introduction

1.1 Biological sequence analysis and protein structure prediction

Biological sequence analysis belongs to a group of problems where given a set of strings from some alphabet, the goal is to find which of these strings are related, or in some way similar to each other. In biology this type of classification is useful since it can help to organize proteins-, DNA- or RNA-sequences into groups that are related on a structural or functional level. And since there currently exists a large amount of these sequences for which little is known about their structures and functions, being able to derive such knowledge from them is very useful.

One field of research where sequence analysis plays an important role is protein structure prediction, i.e. deciding the three dimensional structure of a given protein (amino acid sequence). There are several approaches to this problem. One of the most promising has been to identify a protein with known structure which has the same *fold* as the query protein. This is known as *fold recognition*. Folds are categories of three dimensional structures that are used to cluster proteins with similar 3D structures into groups. Fold recognition methods can be divided into two groups based on the type of information they use to recognize the fold, *structure-based* methods and *prediction-based* methods. However, these groups contain many fairly diverse subtypes, and there have also been approaches that combine methods from the two groups.

The basic idea in structure-based methods is to try to find the best structure for a probe sequence by threading it through a library set of folds and measuring an energy function for each possible alignment of the sequence onto that fold. An example of a method based on this idea is THREADER [15].

For prediction based methods the evolutionary term *homology* plays an important role. Two proteins are homologous if they share the same fold because of their evolutionary history, i.e. they have been developed from the same ancestor through a series of mutations. The point is that even though homologous proteins may have undergone substantial sequence changes, they are still similar enough to exhibit more or less the same structural properties, and having conserved their structure they have many times conserved much of their biological functionality as well. Therefore, predicting a protein's structure can often help to assign a putative function to a new protein sequence.

The classical way of performing fold recognition based on homology has been to take a new sequence and search a database of proteins of known fold for a sequence similar to the query sequence. Traditionally, this has been done using dynamical programming algorithms based on substitution matrices (section 2.3) like the Needleman-Wunch- [10], or the Smith-Waterman-algorithms [23] or by fast heuristic algorithms like BLAST [1]. However, when the sequence identity[1] falls below 20%, into the so called twilight zone of sequence similarity, these methods start having difficulties detecting sequence homologies.

One way of improving the sensitivity of homology detection is by using existing information of the three dimensional structure of the proteins. This can for instance be done using predicted secondary structure information (below). The secondary structure of a protein can be correctly predicted to a degree of around 76% and even though this is not completely accurate, including this information has been shown to increase homology detection [11].

Another fruitful approach is to incorporate information from multiple related sequences to achieve a more robust representation of a protein sequence to use when searching a database. Using this type of information it is possible to detect some protein homologies where the sequence identity is low. A very popular method based on this idea is PSI-BLAST [2], which performs iterative database searches, where information from matches detected in one iteration is used when searching the database in the next iteration.

The field of protein structure prediction also includes more than fold recognition. One example is the earlier mentioned *secondary structure prediction* which aims to predict not the whole three dimensional structure of a protein but rather wishes to classify the amino acid residues into what type of structural region they belong to.

There is also the area of *topology prediction*, where the goal is to predict the protein's topology. For example, for transmembrane proteins (proteins that go through one of the membranes in the cell) methods have been developed to predict the number of transmembrane helices and the proteins' in/out orientation relative to the membrane [17].

1.2 Hidden Markov models in sequence analysis

As described in the previous section, fold recognition can be improved by defining sequence similarity on the basis of a set of sequences rather than on single native sequences. The question is how to represent this multiple sequence information. There are several ways to do this. Two common examples are multiple sequence alignments and sequence profiles based on substitution matrices. Yet another method is using a hidden Markov model, or HMM. An HMM is a probabilistic model (originally used for speech recognition) which has turned out to be very suitable for modeling multiple sequence information. Basically an HMM is a probabilistic finite state machine. It consists of a set of interconnected states, each of which emits an observable

[1]Identity is measured as the share of residues for which two sequences have the same amino acid.

output symbol. To each state there are two types of parameters, emission probabilities, which are the probabilities of emitting each symbol from some alphabet, and transition probabilities, which are the probabilities of moving from that state to a new state. An HMM can be used to represent a set of sequences by adjusting the transition- and emission- probabilities so that when "walking" through the HMM from a start state to an end state by randomly selecting the transitions and emissions to use, the probability of producing any of the sequences in this set is high. Since HMMs are probabilistic models, sequences that are similar to the ones in the set for which the HMM has been adjusted will also be produced with high probability, whereas sequences dissimilar from them will be produced with low probability. This feature is of course the same as the basic principle in most other machine learning approaches, such as artificial neural networks and support vector machines. The name *hidden Markov model* comes from the fact that when doing this type of "random walk" through an HMM one gets, together with an observable sequence of output symbols, a hidden sequence of state symbols, which constitutes a first order Markov chain.

There are a couple reasons why HMMs are suitable for representing multiple sequence information. Firstly, they rest on a solid theoretical foundation from probability theory. Compared to corresponding sequence modeling methods this is especially an advantage in the scoring procedure where HMMs automatically have a non ad hoc scoring method. Secondly, HMMs have good flexibility for emphasizing the importance of different regions of a sequence differently, both in the architecture and in the possibility of giving more importance to similarity in some states than others. A variety of algorithms exist for doing local similarity mesurements.

Because many problems in computational biology reduce to some sort of linear sequence analysis, HMMs have been used for many different types of problems such as genefinding, radiation hybrid mapping, phylogenetic analysis and more. However, HMMs are generally most suited to problems which in themselves closely resemble linear sequence analysis problems, i.e problems with little dependence on correlations between sequence residues. For problems with many such correlations HMMs are often outperformed by for instance neural networks.

In protein structure prediction, there are many cases when HMMs make excellent tools. One successful HMM application is the so called profile HMM [16] which is a specific HMM architecture, well suited to represent profiles of multiple sequence alignments (section 2.3). Profile HMMs can be used both for performing homology searches and for aligning a "match" sequence to a profile model. Another example of a succesful HMM application is the HMMSTR, which is an HMM designed to find recurrent local features of protein sequences and structures that transcend protein family boundaries [4]. And in topology prediction, the TMHMM is the state of the art for predicting the membrane regions in transmembrane proteins [17]. Recently, an increased amount of hybrid methods has also come about. These are methods that are partly based on HMMs and partly on something else. One example is the GIOHMM, which is a combination of an HMM and a neural network used for predicting protein contact maps [20].

1.3 Aims of the project

As described in the last section, HMMs can be used in many different ways in the area of biological sequence analysis. Although HMMs have been around for a while in bioinformatics the belief is that they have not yet been used to their full potential, especially not in the hybrid combinations with other methods mentioned earlier. The aim of this project was therefore to create a flexible and easy to use platform for creating HMMs of different types to apply on different biological sequence analysis problems with emphasis on the area of protein structure prediction.

Some software packages related to this exist already, for instance hmmer[2], SAM[3] and HMMpro[4]. However, there is as of yet no software of this type which both has the desired flexibility features and comes with freely distributed source code. This is of importance in this case since an integral part of the usage of this HMM-package is to support the addition of new features that might become useful as research in the sequence analysis field progresses. The implementation of training- and search- algorithms in this HMM-package builds for the most part on existing theory. However, the simulated annealing and momentum term optimization methods in the training algorithm are independent work.

A further aim for this project was to extend the basic HMM-algorithms with a method for building and scoring HMMs using multiple sequence alignments (as supposed to plain sequences) as input data. This part of the project builds on an earlier attempt to improve the protein structure prediction performance of profile HMMs by adding multiple sequence information to the target sequence done by Björn Larsson and Arne Elofsson [18].

The reasons for incorporating this into the project were twofold. First, it is an interesting attempt to try to improve structure prediction, as of yet not tried in this form. As mentioned before, there exists a common belief that multiple sequence information better describes the significant features of a protein sequence than a single sequence does. The idea of using multiple sequence information both in the query and in the target is however rather new. Some work has been done to develop methods for this type of comparison with promising results [14] [24] [25] [26]. There also exists an HMM-method for doing this type of comparison (no results

[2]http://hmmer.wustl.edu/
[3]http://www.cse.ucsc.edu/research/compbio/HMM-apps/HMM-applications.html
[4]http://www.netid.com/index.html

published yet).[5] The difference between this method and the method we use in this project is that in our method, only one of the profiles are represented as an HMM, while they compare two HMMs. The second reason for this part of the project was to test the performance of the HMM package by using it to solve a real problem and thereby help identify what areas of the system need improvement and to so aid the further development of the program.

[5]http://supfam.mrc-lmb.cam.ac.uk/PRC/

2 Theory

2.1 Theory of HMMs

2.1.1 Markov chains and hidden Markov models

A Markov chain can be defined as a pair $M = (Q, A)$, where $Q = \{q_0, \ldots, q_s\}$ is a set of states and A = $\{a_{qq'}\}$ is a set of transition probabilities for which $\sum_{q'} a_{qq'} = 1$ and $a_{qq'} = P(q_i = q'|q_{i-1} = q)$. M generates a state path $\pi = \pi_1, \pi_2, \ldots$ where π_i is the state in step i. We can write the probability of a state sequence π as

$$\begin{aligned} & P(\pi) \\ &= P(\pi_L, \pi_{L-1}, \ldots, \pi_1) \\ &= P(\pi_L|\pi_{L-1}, \ldots, \pi_1)P(\pi_{L-1}|\pi_{L-2}, \ldots, \pi_1) \cdots P(\pi_1) \end{aligned}$$

by using $P(X, Y) = P(X|Y)P(Y)$ a number of times. Further, the key property of the Markov chain, the Markov condition, is that the probability of each state in the sequence depends only on the value of the preceding state. This means that $P(\pi_i|\pi_{i-1}, \ldots, \pi_1) = P(\pi_i|\pi_{i-1}) = a_{\pi_{i-1}\pi_i}$. The previous equation can therefore be rewritten

$$P(\pi) = P(\pi_L|\pi_{L-1})P(\pi_{L-1}|\pi_{L-2}) \cdots P(\pi_2|\pi_1)P(\pi_1) = P(\pi_1) \prod_{i=2}^{L} a_{\pi_{i-1}\pi_i}.$$

A more detailed description of Markov chains can be found in Cox & Miller [5].

A hidden Markov model is an extension to the classical Markov chain and can be defined as a 6-tuple $M = (Q, \Sigma, A, E, q_0, q_s)$ where $Q = \{q_0, \ldots, q_s\}$ is a set of states, $\Sigma = \{\sigma_1, \ldots, \sigma_r\}$ is the alphabet, $A = \{a_{qq'}\}$ is a set of transition probabilities (with the same conditions as for Markov chains), $E = \{e_q\}$ is a set of symbol distributions such that $\forall q \in Q \neq q_0, q_s$: $\sum_{i=1}^{r} e_q(\sigma_i) = 1 \wedge 0 \leq e_q(\sigma_i) \leq 1, q_0$ is the start state and q_s is the end state. The addition of specific begin and end states is not strictly necessary but facilitates the notation regarding the probabilities for the beginnings and ends of sequences.

M generates a state path $\pi = \pi_0, \pi_1, \ldots$ and a symbol sequence $x = x_1, x_2, \ldots$, where the state path is hidden, starts in q_0 and ends in q_s. As before, the state chain is characterized by parameters

$$a_{qq'} = P(q_i = q'|q_{i-1} = q).$$

For the symbol sequence however, we have a new set of parameters

$$e_{q_j}(\sigma_i) = P(x_k = \sigma_i|\pi_i = q_j)$$

or informally $e_{q_j}(\sigma_i)$ is the probability that symbol σ_i is seen when in state q_j.

The joint probability of a specific state-symbol sequence of M is

$$P(x, \pi) = a_{q_0 \pi_1} \prod_{i=1}^{L} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}}$$

where $\pi_{L+1} = q_s$. This equation can be proven by induction over L. For $L = 0$ we get

$$P(x, \pi) = P(\pi_0, \pi_1) = a_{q_0 q_s}$$

which can be interpreted as the probability of transiting from the start state directly to the end state, not emitting anything. Assuming the equation holds for $L - 1$ we get

$$\begin{aligned}
& P(x_1, \ldots, x_L, \pi_0, \ldots, \pi_{L+1}) \\
= P(x_1, \ldots, x_L, \pi_0, \ldots, \pi_{L+1} | x_1, \ldots, x_{L-1}, \pi_0, \ldots, \pi_L) * \\
& P(x_1, \ldots, x_{L-1}, \pi_0, \ldots, \pi_L) \\
\overset{Mark.Cond.}{=} & P(x_L, \pi_{L+1} | \pi_L) * P(x_1, \ldots, x_{L-1}, \pi_0, \ldots, \pi_L) \\
\overset{Ind.Hyp.}{=} & P(x_L, \pi_{L+1} | \pi_L) * a_{q_0 \pi_1} \prod_{i=1}^{L-1} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}} \\
= & a_{q_0 \pi_1} \prod_{i=1}^{L} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}}.
\end{aligned}$$

For a more indepth description of HMMs see for example Rabiner [21].

2.1.2 The forward, backward and Viterbi algorithms

When working with HMMs it is often necessary to be able to retrieve the state paths for the sequences. For this purpose three dynamic programming algorithms are at the core of every HMM implementation: Viterbi, forward and backward. The Viterbi algorithm is used to calculate the most probable state path given a sequence of symbols, while forward and backward are used to calculate the total probability for a sequence to be produced by an HMM by adding the probabilities for all possible state paths producing that sequence.

As we saw in the previous section, the formula for the joint probability of an observed sequence x and a state sequence π is

$$P(x, \pi) = a_{q_0 \pi_1} \prod_{i=1}^{L} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}}.$$

The Viterbi algorithm calculates the specific state path that has the highest probability. This can of course not be done by examining all possible paths, since the number of possible paths is exponential to the number of states. Instead the Viterbi algorithm uses recursion. Let the most probable path for a sequence x be $\pi^* = argmax_\pi P(x, \pi)$. Now suppose that the probability $v_q(i)$ of the most probable path ending in state q with observation x_i is known for all states q. Then the probability for observation x_{i+1} in each state q' can be calculated by

$$v_{q'}(i+1) = e_{q'}(x_{i+1}) max_q(v_q(i) a_{qq'}).$$

This can be motivated since

$$\begin{aligned} v_{q'}(i+1) = P(x_1, \ldots, x_{i+1}, q_{i+1} = q') \\ = max_q(P(x_1, \ldots, x_{i+1}, q_{i+1} = q' | x_1, \ldots, x_i, q_i = q) * P(x_1, \ldots, x_i, q_i = q)) \\ = max_q(P(x_{i+1}, q_{i+1} = q' | q_i = q) * P(x_1, \ldots, x_i, q_i = q)) \\ = max_q(e_{q'}(x_{i+1}) * a_{qq'} * v_q(i)) \\ = e_{q'}(x_{i+1}) * max_q(a_{qq'} * v_q(i)). \end{aligned}$$

Since all sequences must start in state q_0, the initial condition is that $v_{q_0}(0) = 1$. The algorithm terminates when $i = L$, which is the length of the sequence. If an HMM has a non-emitting end state we must also incorporate $v_{q_s}(L + 1) = max_q(v_q(L) a_{qq_s})$. By keeping backward references the state sequence can then be found by backtracking.

The time complexity for this algorithm is $O(|Q|^2 L)$, which can be motivated as follows. To decide $v_{q'}(i + 1)$ we need to examine all probabilities $v_q(i)$, which are $|Q|$. The number of $v_{q'}$'s to decide for each value of i is also $|Q|$ and finally the number of i's is equal to the length of the sequence, which is L.

Sometimes it is not enough to just calculate the most probable path through an HMM for a sequence. To calculate the total probability of a sequence one must calculate the sum of the probabilities for all possible paths that produce the sequence. This can be done using the same approach as in the Viterbi algorithm. The forward algorithm is in fact exactly similar to Viterbi, only replacing $max_q(v_q(i) a_{qq'})$ with $\sum_q v_q(i) a_{qq'}$. Informally this means that the total probability of having produced a sequence up to and including observation x_{i+1} ending in state q' is derived by taking the probability of having produced the sequence up to (but not including) x_{i+1} ending in q multiplied with the probability of moving from q to q'. We then sum this product over all states q. The total sum is then multiplied with the probability of emitting symbol x_{i+1} in q'. Or to state it formally

$$f_{q'}(i+1) = e_{q'}(x_{i+1}) \sum_q f_q(i) a_{qq'}.$$

The backward algorithm performs the same task as the forward algorithm with the difference of (as the name implies) starting the calculation at the back of the sequence. This leads to a slightly different recursion step:

$$b_q(i) = \sum_{q'} b_{q'}(i+1) * a_{qq'} * e_{q'}(x_{i+1}).$$

Informally, the probability of being in state q having produced the part of the sequence from x_{i+1} to the end can be calculated by taking the probability of being in state q' having produced the part of the sequence from x_{i+2} to the end, multiplied with the probability of moving from q to q'. This is multiplied with the probability of emitting x_{i+1} in q' and summed over all states q'. Apart from this the forward and backward algorithms are basically the same.

Using both the forward and backward algorithms it is possible to derive such information as what the probability is that observation x_i came from state q given the observed sequence x, that is $P(\pi_i = q|x)$. To calculate this, we need to know $P(x, \pi_i = q)$, which we can get quite easily since

$$\begin{aligned} & P(x, \pi_i = q) \\ = {} & P(x_1, \ldots, x_i, \pi_i = q)P(x_{i+1}, \ldots, x_L|x_1, \ldots, x_i, \pi_i = q) \\ \overset{Mark.Cond.}{=} {} & P(x_1, \ldots, x_i, \pi_i = q)P(x_{i+1}, \ldots, x_L|\pi_i = q) \\ = {} & f_q(i) * b_q(i) \end{aligned}$$

So we get

$$P(\pi_i = q|x) = \frac{P(x, \pi_i = q)}{P(x)} = \frac{f_q(i) * b_q(i)}{P(x)}$$

where $P(x)$ is calculated for instance by forward.

A common implementation problem regarding these algorithms is that the accumulated probabilities tend to get so small that they cause underflow errors. Two schemes exist to account for this, either one can use the logarithm of the probabilities or one can rescale them to some reasonable value in each step of the algorithm.

There also exists a possibility of extending the standard HMM specification to include *silent*. These are states which do not emit any symbols but behave as regular states in all other aspects. To account for silent states all three algorithms need to be changed slightly. For the Viterbi algorithm, the change is the following:

1. For all regular states q', calculate $v_q(i+1)$ as before.
2. For all silent states q', put $max_q(v_q(i+1)a_{qq'})$ in $v_{q'}(i+1)$ for all regular states q.
3. Starting from the lowest numbered silent state q' put $max_q(v_q(i+1)a_{qq'})$ in $v_{q'}(i+1)$ for all states $q < q'$ if the value is larger than the value stored there in step 2.

The numbering of the silent states is done so that any transition between silent states goes from a lower numbered state to a higher numbered one. This is to avoid loops consisting entirely of silent states, which make the situation more complicated, however not unsolvable. The changes for the forward and backward algorithms are done in the same fashion as for Viterbi.

Further details and a more extensive description of Viterbi, forward and backward, can be found in Durbin et al [6].

2.1.3 Expectation maximization and the Baum-Welch training algorithm

When creating an HMM to describe a set of sequences the goal is to make the likelihood for the HMM to produce these sequences high and the likelihood for it to produce all other sequences low, preferably with a distinct border between the two groups. To help accomplishing this, there are two parts. One is the architecture of the HMM, i.e. the possible state transitions and possible emissions. The other is the fitting of the $a_{qq'}$ and $e_q(\sigma)$ parameter values to maximize the likelihood of the sequence set being produced by the HMM. Since one rarely has access to or knowledge of all sequences that belong to the set one wishes to model, fitting the parameters is usually done by adjusting them to fit a subset this set, commonly known as *training* the HMM. There are two situations to handle when doing parameter optimization. The state paths may be known or the state paths may be unknown.

When the state paths are known, the parameter estimation is fairly straight forward. Let $A_{qq'}$ be the number of times the transition from q to q' is used in the set of training sequences and $E_q(\sigma)$ be the number of times σ is produced in state q in the same set. Then the optimal parameter configuration for $a_{qq'}$ and $e_q(\sigma)$ is given by

$$a_{qq'} = \frac{A_{qq'}}{\sum_{i=0}^{s} A_{qq_i}} \quad \text{and} \quad e_q(\sigma) = \frac{E_q(\sigma)}{\sum_{\sigma'} E_q(\sigma')} \tag{1}$$

which can be proven formally (a proof can for instance be found in Durbin et al. [6]), but which is also intuitively reasonable.

The case when the state paths are unknown is sligthly harder. The values of $A_{qq'}$ and $E_q(\sigma)$ are not known a priori, but must be calculated somehow. The most common method for this is a variant of the expectation maximization (EM) algorithm called the Baum-Welch training algorithm. This is an iterative method which first estimates the $A_{qq'}$ and $E_q(\sigma)$ values by considering the paths for the training sequences using the current $a_{qq'}$ and $e_q(\sigma)$ values and then estimates new $a_{qq'}$ and $e_q(\sigma)$ values using (1). This procedure is then repeated until some stopping criterion is reached. $A_{qq'}$ and $E_q(\sigma)$ are calculated using the forward and backward algorithms in the way described in the previous section. The probability that $a_{qq'}$ is used at position i in sequence x^j is

$$P(\pi_i = q, \pi_{i+1} = q'|x^j, \theta) = \frac{f_q(i) * a_{qq'} * e_{q'}(x^j_{i+1} * b_{q'}(i+1)}{P(x^j)}$$

where θ is the $a_{qq'}$ and $e_q(\sigma)$ parameter values. From this, the exact number of times that $a_{qq'}$ is used can be derived by summing over all positions i and over all training sequences x.

$$A_{qq'} = \sum_j \frac{1}{P(x^j)} \sum_i P(\pi_i = q, \pi_{i+1} = q'|x^j, \theta).$$

Similarly, the expected number of times a symbol σ is emitted in state q can be found through

$$E_q(\sigma) = \sum_j \frac{1}{P(x^j)} \sum_{\{i:x^j_i=\sigma\}} f^j_q(i) * b^j_q(i).$$

It has been proven that the algorithm is guaranteed to improve the overall likelihood of the model in each iteration, which means that it converges towards a local maximum. For details, please see Durbin et al. [6]. What local maximum the algorithm ends up in is however greatly dependent on the starting values of the parameters.

The time complexity for one iteration is $O(\sum_{j=1}^{S}(|Q|^2 * L_j))$, where S is the number of sequences in the training set and L_j is the length of the j'th sequence. The overhead of this algorithm is however fairly large. The number of iterations performed depends on the stopping criterion. Since the algorithm operates in continuous space it is possible to iterate for an infinite number of times. Usually however, the algorithm converges pretty close to a maximum in less than 20 iterations.

There has been some research on how to optimize HMM training, mostly methods for avoiding getting stuck in local maxima. Some example optimization methods that have been tried for HMMs are simulated annealing [7] and noice injection [13].

2.1.4 HMM scoring methods

In an HMM context scoring means measuring how well a particular sequence fits an HMM. The simplest way of doing this is to run the Viterbi or forward algorithm and let the result be the score. To avoid underflow problems one usually works with the logarithm of the result. This result is called the *log-likelihood score* (or *LL-score*). A drawback with this method is that the score is highly dependent on the length of the sequence. A short random sequence is likely to get a better score than a long sequence which belongs to the family modeled by the HMM. The common way to account for this is to divide the LL-score with the length of the sequence (and also put a minus sign in front of the result) to get what is called the *normalized log-likelihood*. This improves the result, but still the scores are too fuzzy to be convenient to use as discrimination functions.

The problem with normalized log-likelihood is that it is a non-related measurement. An amino acid sequence might for instance consist mostly of the overall most common amino acids, which will tend to give a high normalized log-likelihood score in many different HMMs. But this will not be seen when scoring the sequence against one HMM with the result that one will think that the sequence is a "match". To deal with this problem it is common to use a scoring method called *log-odds*. In log-odds the resulting score is the logarithm of the quotient between the likelihood of a sequence being generated by the HMM and the likelihood of it being generated by a random model, usually refered to as a null model.

$$logodds(x) = \log \frac{LLscore}{LLscore_{null}}$$

One drawback with log-odds scoring is however that the significance of a particular log-odds score is not necessarily the same for all HMMs, that is, the distributions of the log-odds scores differ between HMMs. To deal with this problem one can use yet another measurement method called *P-value*. The P-value is obtained by treating the log-odds score S of a random sequence as a random variable and then estimating the density of $max_N(S)$, meaning the maximum of N independent random variables S. The P-value for a query sequence with log-odds score s is then measured as the probability of getting a higher log-odds score by chance given the density of $max_N(S)$. Formally:

$$Pvalue(s) = Prob(max_N(S) \geq s)$$

for some value of N. The distribution of $max_N(S)$ can be estimated either analytically or empirically. For local pairwise non-gapped alignments it has been proven formally that it is extreme value distributed. As for gapped pairwise alignments, multiple alignments and HMMs, no such formal proofs exist, but given observed scores for both real and artificial data it is reasonable to believe that $max_N(S)$ is extreme value distributed in these cases as well.

Closely related to the P-value is the *E-value*, which is a function of the score s and the size of the database k, which calculates the expected number of sequences in a database of size k consisting of random sequences which would have a higher score than s.

2.1.5 Prior distributions

A common problem with HMMs (and machine learning approaches in general) is that if there is to little data to train the HMM on, it will become overspecialized. If, for instance, all protein sequences in an HMM training set has either A or L as their first residue, the probability of the trained HMM emitting something other than A or L in the first state will be zero. With a very large training set, this may be what one wants, but for a small training set, it is probable that there may exist some sequence that belongs to the family modeled by the HMM and does not start with A or L. In this way such a sequence cannot be found.

The standard method used to account for this problem is to add some kind of pseudo-counts when estimating how many times each alphabet symbol is emitted in each state. The most simple, called *simple pseudocount* method is to add one for each symbol of the alphabet, which means that one pretends that all symbols have been emitted in each state at least once. This corresponds to changing the updating formula $e_{q'}(\sigma') = \frac{E_{q'}(\sigma')}{\sum_\sigma E_{q'}(\sigma)}$ to $e_{q'}(\sigma') = \frac{E_{q'}(\sigma')+1}{\sum_\sigma (E_{q'}(\sigma)+1)}$. This takes care of the zero-probability problem but is of course a very crude measurement. On the upside however, is the fact that this method gives a natural decrease in importance to the pseudo-counts as the size of the training set increases since the proportion of the pseudo-count to the real count decreases. This feature is also kept in the more advanced pseudo-count schemes.

A more sofisticated pseudo-count method is to incorporate more prior knowledge of what we are trying to model into the pseudo-counts. For amino acids for example, it is known that isoleucine is commonly found in buried beta strand environments and that leucine and valine often substitutes for it in those environments. So if our training data tells us that the probability for some state emitting isoleucine is high, it would perhaps be a good idea to give this state a fairly high probability of emitting leucine and valine as well.

A simple way of incorporating more information into the pseudo-counts is to change the 1 to a weight W and a proportion p_σ given some kind of background distribution, yielding $e_{q'}(\sigma') = \frac{E_{q'}(\sigma')+Wp_{\sigma'}}{\sum_\sigma (E_{q'}(\sigma)+Wp_\sigma)}$. Although a better approach, this is not enough to capture such correlations as described above. What is needed is a method which adjusts the pseudo-counts after the observed data. To achieve this, a method for using a mixture of Dirichlet distributions as the prior was developed by Brown et al. [3]. The basic idea is to have a set of different prior distributions $\alpha_\Sigma^1, \ldots, \alpha_\Sigma^L$ where a_σ^l corresponds to Wp_σ above and then let the different distributions in the set model different background environments, such as the buried beta strand environment mentioned above. Given the real counts $E_q(\sigma)$ the likelihood of each prior distribution is estimated and used to decide each prior distribution's contribution to the pseudo-count. Formally the updating formula becomes

$$e_{q'}(\sigma') = \sum_l P(l|E_{q'}) * \frac{E_{q'}(\sigma') + \alpha^l_{\sigma'}}{\sum_\sigma (E_{q'}(\sigma) + \alpha^l_\sigma)}$$

where $E_{q'}$ is the vector of observed counts in q' and $P(l|E_{q'})$ is calculated using Bayes' rule,

$$P(l|E_{q'}) = \frac{p_l P(E_{q'}|l)}{\sum_{i=1}^{L} p_{l_i} P(E_{q'}|l_i)}$$

where p_l are the prior probabilities for each mixture component and $P(E_{q'}|l)$ is the probability of the data according to Dirichlet mixture l. Further details and an explanation of how to derive $P(E_{q'}|l)$, can be found in Durbin et al. [6] or Sjölander et al. [22].

2.2 Architectures of HMMs

For HMMs, the term *architecture* means what states there are, the possible state transitions and to some extent the alphabet. The architecture of an HMM is very important. It is the backbone that decides what is possible to model, and it has great influence on the training algorithm, both regarding speed and performance. As there are no simple rules regarding how to design an HMM given a specific problem, HMM-design is as much a form of art as it is science.

A simple example to show the importance of architecture is the following HMM-comparison. Suppose two different HMM-parts, A and B (figures 1 and 2), were built to model the same set of sequences, which we assume to be between two and five symbols long. A and B model the "gaps" in the sequences in two different ways. In A there are transitions from each state to all states to the right of that state. In B the circles represent silent states, i.e. states that do not emit any symbol. Both these models "accept" the same sequences (given that their alphabets are the same). However, model B has the quality of having a number of transitions which is linear to the number of states, while model A has a number of transitions which is in the order of the square of the number of states. For larger models with about 200 states this fact will make models of type A impossible to use because both training and searching will take too long. Model A on the other hand has the possibility of having high probabilities for the transitions from state 1 to state 5 and from state 2 to state 4, while having low transition probabilities for transitions from 1 to 4 and from 2 to 5. This is a flexibility model B does not have.

The above example shows a common tradeoff problem of flexibility versus speed. Another thing to consider in a model is generality versus level of detail. If there is prior information regarding the sequences that are to be

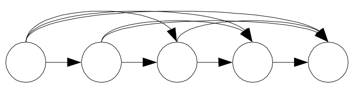

Figure 1: Model A

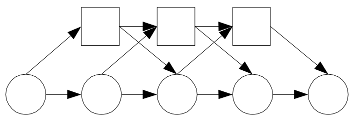

Figure 2: Model B. Silent states are represented by squares.

modeled, which tells us that some regions of the sequences are more specific to the set than others this should be reflected in the HMM-architecture. For instance it might be a good idea to model the specific regions with many states while the non-specific regions are modeled with fewer states. The idea is to make the modeling of the non-specific regions general enough not to pick up any irrelevant pattern that still might be found in the training set. It may also be the case that different regions are equally characteristic but have totally different properties. This too will have influence on the architecture.

The TMHMM [17] is a good example of an HMM designed to model data with known specific regions. TMHMM is designed to recognize membrane proteins (proteins that go through one of the cell membranes). The most characteristic regions of such proteins are the membrane regions themselves. The amino acid composition in the membrane regions differs from the one on the outside or the inside if the cell, since the chemical environment differs. The key to detect whether a protein is a transmembrane protein or not lies therefore in the modeling of the membrane regions. This is reflected in the TMHMM architecture (figures 3 and 4) which is more specific for these regions while the non-transmembrane (globular) regions (which not uncommonly make up the larger part of the protein) are modeled in a very simple way.

Another HMM architecture that must be mentioned in this context is the profile HMM. This is an architecture invented by Anders Krogh for the specific task of modeling protein families [16]. Profile HMMs have proven to be a very good tool for this task as they can turn a multiple sequence alignment

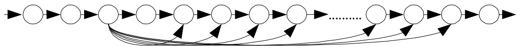

Figure 3: The part of the TMHMM that models the transmembrane helix regions

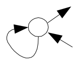

Figure 4: The part of TMHMM that models the globular regions

into a position-specific scoring system suitable for searching large databases for remotely homologue sequences. The profile HMM architecture (figure 5) corresponds closely to the concept of multiple sequence alignments since it tries to model which residues of a protein sequence are conserved, which have been inserted during evolution and where deletions have occured. For this they have three different types of states, match-, insert-, and delete-states. Match and insert states are regular states whereas delete states are silent. Each path that produces a specific sequence represents a possible alignment of that sequence onto the model, which can be conveniently transcribed to a regular multiple sequence alignment notation using the labels of the states in the path. There exist some variations to the standard profile HMM architecture. For instance it is possible to not allow direct transitions between delete states and insert states.

Another advantage of the profile HMMs is that the number of transitions is proportional to the number of states, which makes the basic algorithms (Viterbi, forward and backward) run in $O(|Q| * L)$ instead of $O(|Q|^2 * L)$ as for general HMMs. Profile HMMs belong to a subgroup of HMMs called left-right models, which means that these models are possible to depict with all transitions going from left to right.

A weekness with HMMs is that is that it is very difficult for them to model dependencies in the data. For instance, in a 10 letter sequence the letter in position 10 may be dependent on what letter there is in position 1, meaning that the probability distribution of state number 10 in a linear HMM is dependent on what letter is emitted in state 1. The only way to model this with an HMM is to use a parallel architecture, i.e. several parallel paths through the HMM that are in no contact with each other. But this increases the number of states in the HMM exponentially to the number

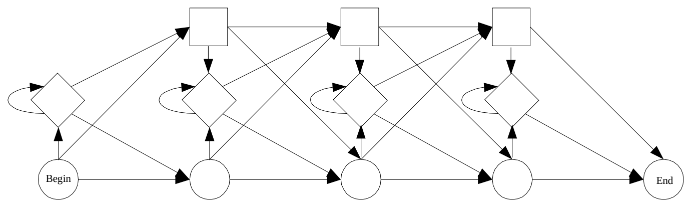

Figure 5: The standard architecture of a profile HMM. Circles represent match states, diamonds represent insert states and squares represent delete states. The states labeled “Begin” and “End” are the special start- and stop- states.

of dependencies that one wishes to model. Therefore, this approch is not possible to use except for a very small number of dependencies.

2.3 Profiles and multiple sequence alignments

Profiles and multiple sequence alignments are two common ways of representing multiple sequence information. Multiple sequence alignments are a way of modeling the evolutionary process where residues of a protein sequence are mutated, deleted and inserted. What makes multiple sequence alignments useful is the fact that when aligning two homologous protein sequences to each other they will exhibit more identity than when aligning two non-homologous sequences. However, there is no unique method to find the "best" alignment between two proteins. This is due to the fact that there are different ways of assessing how much penalty should be given for gaps in the alignment (inserts and deletions) and for non-matching (mutated) residues. To estimate how probable it is for one amino acid to substitute for another it is common to use a *substitution matrix*, which assigns a score to each possible amino acid mutation, usually positive for likely mutations and negative for unlikely ones. What a likely mutation is may be estimated either from the chemical properties of the amino acids or from empirical observations. The total score of an alignment measures how likely it is for two sequences to be related. Moreover, the problem of finding an optimal multiple sequence alignment given a scoring method is proven to be NP-complete.

```
AHC---D--DEK--FAJSAJ
AHG---D--FJKAIGAFAAJ
IBGBCJGCCCGD--AMLKFJ
BHI---A--DLL--AAFAAJ
```

Figure 6: A small multiple sequence alignment of four sequences from a fictive alphabet. '-' is used to represent both insert- and delete- gaps. In some alignment notation inserts and deletes are represented by separate symbols. Sequence 1 and 2 are close homologues. Sequence 4 is a distant homologue to sequence 1 and 2. Sequence 3 is not related to the other three.

Profiles are another way of representing multiple sequence information. A profile is usually based on a multiple sequence alignment. For each position in the alignment, one counts the number of times each amino acid appears in that column and then finds the probability of seeing that amino acid in the column by dividing the number of times it is seen with the size of the column. This number is normalized by accounting for the number of gaps that exist in the column and usually some sort of pseudocount scheme is applied at the end. A profile is then defined as the series of probability distributions one receives after applying this procedure to all columns in the alignment.

A:	0.50	0.00	0.00	0.00	0.00	0.00	0.25	...
B:	0.25	0.25	0.00	1.00	0.00	0.00	0.00	...
C:	0.00	0.00	0.25	0.00	1.00	0.00	0.00	...
D:	0.00	0.00	0.00	0.00	0.00	0.00	0.50	...
E:	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...
F:	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...
G:	0.00	0.00	0.50	0.00	0.00	0.00	0.25	...
H:	0.00	0.75	0.00	0.00	0.00	0.00	0.00	...
I:	0.25	0.00	0.25	0.00	0.00	0.00	0.00	...
J:	0.00	0.00	0.00	0.00	0.00	1.00	0.00	...

Figure 7: The profile for the first seven residues of the alignment in figure 6.

3 The modhmm package

3.1 Introduction

The basic part of the HMM package developed in this project, called modhmm, consists of three subparts: modhmmc, modhmmt and modhmms. modhmmc is a program for building an HMM by putting together small HMM parts, called modules. modhmmt is a program for training an HMM constructed in modhmmc on a set of training sequences. Finally modhmms is a program for scoring sequences against HMMs.

The common idea for the modhmm package was to make it as general as possible. The reason for this was to enable modhmm to be used in as many areas of the sequence analysis field as possible. Therefore we tried not to limit the architectural possibilities, allowing users to create HMMs of arbitrary design, both regarding states, state transitions and alphabet. At the same time the goal was to implement the training and search algorithms as efficiently as possible, given the limiting factor of the programs' generality.

3.2 Architecture - modhmmc

modhmmc is the program for designing an HMM. It lets the user specify alphabet, states, transitions, relations between transitions, relations between states and if any files containing prior distributions are to be associated with the HMM. So far the functionality of modhmmc has been given priority over the user interaction. The result of this is that the program allows a more detailed input than it is possible for a user to specify. This will be corrected later with the addition of a more advanced user interface, which however lies outside the scope of this project. What is described below is the functionality of the program.

The alphabet of a particular HMM is specified as a set of letters, where each letter is a word of up to 4 characters. The reason for allowing letters to consist of more than one symbol is to make more specialized alphabets possible. An example is an alphabet of the 20 amino acids, where each amino acid is split into several letters depending on which environment it exists in. A possible set of environments is cell-membrane, cell-inside and cell-outside. A possible alphabet is then 'Ai', 'Ao', 'Am', 'Ci', 'Co' , 'Cm' and so on.

The states of an HMM are specified as a collection of *state modules* with transitions between them. A module is a set of states which are interconnected in a predecided fashion. The idea behind modules is to make the creation of large HMMs easier. HMMs with several hundred states are not uncommon in sequence analysis and specifying each and every state and transition in such a case is impractical. modhmmc currently has 6 module types to choose from: singlenode, singleloop, forward, cluster, profile9 and profile7 (figures 8 to 13). The user specifies what modules to use and how to interconnect them. modhmmc allows for the creation of both regular and silent states. Which states are regular and which are silent may be specified entirely by the user, but default settings are built into the modules. The modhmm package also implements start- and stop- states as a necessary part of all HMMs. Loops consisting entirely of silent nodes are not allowed.

The initial (before training) probabilities of all transitions and emissions may also be specified entirely by the user. However, transition probabilities are set by default inside the modules to correspond to the intrinsic properties of that module (see descriptions of the modules). Transition probabilities between modules are by default set uniformly, so that the probabilities of going from a module to another are equally distributed among all its neighbours. Emission probabilities for each state are either set manually by the user or according to a chosen distribution. The choices are to set the values uniformly, randomly or to zero for all letters, which creates a silent state. modhmmc also has the possibility of connecting the emission probabilities of a set of states so that during training, the emission probabilities for all letters remain the same inbetween these states, i.e. the probabilities themselves may change, but the probability of emitting each letter is the same for all the connected states. There is also a possibility of connecting a set of states to make them be seen as “the same state” when constructing an alignment from the HMM. With this feature it is possible to have parallel architectures (figure 14) and still make it possible to align letters emitted in different branches to the same column.

Singlenode The singlenode module is the most basic module of modhmmc. It consists of only one state, regular by default, but the user may specify it as silent. All other modules may be built using collections of singlenodes. The singlenode has no input parameters.

Singleloop The singleloop module consists of one state with a transition to itself. It is necessarily regular, since no loops of silent states are allowed. The singleloop has one input parameter. The user specifies the expected length of the loop, which sets the loop transition probability to $p_{trans} = e^{\frac{ln0.5}{l}}$, where l is the specified loop length. This results in the probability of a loop length longer than or equal to l being equal to the probability of a loop length shorter than l, that is, l is the expected median loop length.

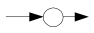

Figure 8: Singlenode module

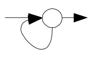

Figure 9: Singleloop module

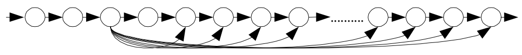

Figure 10: Forward module

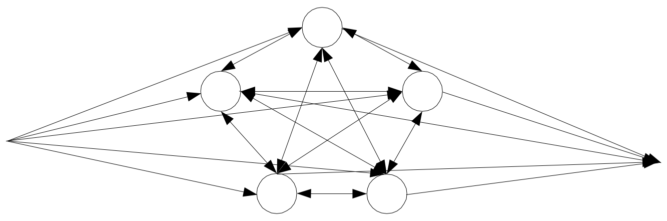

Figure 11: Cluster module

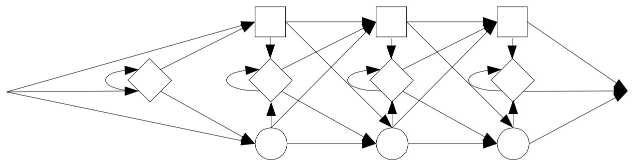

Figure 12: Standard profile hmm module, profile9

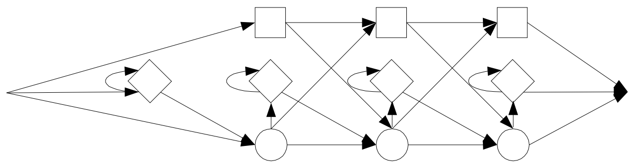

Figure 13: Profile hmm module with no direct connections between insert states and delete states, profile7

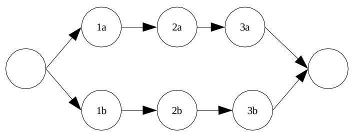

Figure 14: Parallel HMM-architecture example. Connecting each a-state with its b-counterpart may in this case give multiple sequence alignments which better represent the underlying data.

Forward The forward module is a set of states connected in a straight line, indexed 1 to n. All states are regular by default. The two input parameters (m, n) specify the shortest and longest possible routes through the module. The number of states in the module is equal to the length of the longest possible route n. For all states with index $m - 1$ to $n - 2$ there is a transition to the last state. The transition probabilities are set so that the total probability is equal for all possible paths through the module. Any state that connects to this module will connect to the state with index 1, and all outgoing connections from this module go from the state with index n.

Cluster The cluster module is a fully interconnected set of states. Every state has a transition to every other state. The transition probabilities are evenly distributed by default. All states are regular. The input parameter is the number of states. Incoming transitions connect to all states, and outgoing connections go from all states.

Profile9 The profile9 module is equal to the standard profile-HMM architecture (section 2.2). The input parameter specifies the length of the module, i.e. the number of match states. Incoming transitions connect to the first state of each type (delete, insert, match), while outgoing transitions go from the last state of each type. The input parameters specify the expected gap length and the expected insert length. The expected gap length sets the transition probability of the delete→delete transitions according to the same formula as for the singleloop module. The expected insert length sets the insert→insert loop transition in the same fashion. As for the probabilities of opening up a gap or an insert these probabilities (match→delete and match→insert transitions) may be specified by the user, but are otherwise set by default to 0.025 in an ad hoc fashion, leaving 0.95 as the probability for the match→match transitions. The probabilities of the delete→match and the delete→insert transitions (which depend on the expected gap length parameter) are set so that $p_{d \to m} = 0.95 * (1 - p_{d \to d})$ and $p_{d \to i} = 0.05 * (1 - p_{d \to d})$. According to the same principle, the insert→match and the insert→delete transitions are set to $p_{i \to m} = 0.95 * (1 - p_{i \to i})$ and $p_{i \to d} = 0.05 * (1 - p_{i \to i})$.

Profile7 The profile7 module is equal to the profile9 module in every way, except for it not having any delete→insert or insert→delete transitions.

3.3 Training - modhmmt

The modhmm-package program for training an HMM is modhmmt. modhmmt implements four variations of the Baum-Welch traning algorithm: standard, annealing, random-walk and momentum. It is possible to train an HMM either on a set of regular sequences or on a multiple sequence alignment. The user has the option of using prior information in the form of Dirichlet priors for the emissions and simple pseudocounts for the transitions.

Baum-Welch standard This is a straight forward implementation of the Baum-Welch algorithm as described in the theory section (see section 2.1.3). By only examinining existing transitions with non zero probabilities in the forward and backward recusion steps the effective time complexity for these algorithms is reduced to $O(T * L)$, where T is the number of transitions. In the worst case, this is of course equal to $O(|Q|^2 * L)$.

Input training data may either be a set of sequences or a multiple sequence alignment. When using ordinary sequences, the algorithm works exactly as described in the theory section. When using a multiple sequence alignment however, there are some slight modifications. Since the idea of training on an alignment suggests that all letters belonging to the same column should be produced in the same state, each "letter" of the alignment is a probability distribution representing a column. This leads to a slightly different recursion step for the forward and backward algorithms

$$f_{q'}(i+1) = e_{q'}^{msa}(x_{i+1}^{msa}) \sum_{q} f_q(i) a_{qq'}.$$

and

$$b_q(i) = \sum_{q'} b_{q'}(i+1) * a_{qq'} * e_{q'}^{msa}(x_{i+1}^{msa})$$

where e^{msa} represents a formula for calculating the probability for the column x^{msa} to be produced in a state q. In modhmmt e^{msa} is implemented in two ways. The first is the dot product

$$e_q^{msa}(x_i^{msa}) = \sum_{\sigma} e_q(\sigma_j) * x_i^{msa}(\sigma_j).$$

This formula is easy to use and only increases the time complexity of the forward and backward algorithms by a factor the size of the alphabet, which is usually compensated for since multiple sequence alignment training is performed using only one sequence. The probabilistic interpretation of this score is that it represents the probability that the same letter is drawn if drawing independently from both the state- and the profile- distributions.

The second implementation of e^{msa} is using a variant of the so called *log-average score*, which originally is

$$score_{logaverage}(\alpha, \beta) = log \sum_{i=1}^{|\Sigma|} \sum_{j=1}^{|\Sigma|} \alpha_i \beta_j \frac{p_{rel}(i,j)}{p_i p_j},$$

where α and β are the two probability distributions, p_i is the background amino acid distribution for amino acid i and p_{rel} is a probability distribution of “related” amino acid pairs. For details, please see von Öhsen et. al. [25] [26]. To suit the strict probability environment of HMM scores, this scoring method ha been modified slightly in the modhmm implementation.

$$e_q^{msa}(x_i^{msa}) = \sum_{j=1}^{|\Sigma|} \sum_{k=1}^{|\Sigma|} e_q(\sigma_j) * x_i^{msa}(\sigma_k) * submtx(j,k).$$

Here, $submtx(j,k)$ is an a priori probability that amino acids j and k are related derived from a user specified substitution matrix of the BLOSUM series [12]. Recent results indicate that this method performs better than the dotproduct [25] [26]. However, it is at least a factor $|\Sigma|$ slower, since the single sum is replaced by a double sum.

Baum-Welch random walk The random walk optimization of the Baum-Welch algorithm (originally developed by Richard Hughey and Anders Krogh [13]) tries to avoid getting stuck in local maxima by generating a number R of random sequences from the initial model. Given a noise level N_i, each of the R paths through the model adds a value to the respective $A_{qq'}$ and $E_q(\sigma)$ parameters. The N parameter is decreased for each iteration of the algorithm, either linearly or exponentially.

The idea is to run the algorithm several times, finishing in different local maxima, choosing the result model which has the highest likelihood. This is an ad hoc - optimization method which has been shown to work well in practice [13].

Baum-Welch annealing The annealing optimization method uses a variant of simulated annealing to help overcome local maxima. It has a temperature T and a cooling factor c. For each iteration, each $A_{qq'}$ and $E_q(\sigma)$ value is scrambled by multiplying it with a random number between $\frac{1}{T}$ and T. After each iteration the temperature is decreased by multiplying it with the cooling factor.

Like random walk, annealing is used to optimize the performance of an HMM. It may also be run several times on the same training data, letting the user choose the result model with the highest likelihood.

Baum-Welch momentum The third optimization method builds on a concept borrowed from artificial neural networks which has been adapted to fit HMMs. The basic idea is to let each parameter (transition and emission probabilities) keep a momentum from how it updated in one iteration to the next iteration. This will enhance the size of its update if it continues to change in the same direction. Otherwise it will reduce it. If the momentum term is larger than a move in the opposite direction, the parameter will continue to move in the momentum direction. Momentum is used both as a speed optimizer and as a mean of avoiding local maxima. The size of the momentum term is set as a fraction of the size of the update in the last iteration. After adding the momentum terms, all updated values are normalized to make the probability sums equal to 1.0.

3.4 Scoring - modhmms

modhmms is the program for scoring a sequence against an HMM built by modhmm. Scoring can be done using either the forward- or Viterbi-algorithm. The score obtained is a log likelihood score which may be turned into a log odds score using either a user specified- or default- null model. The default null model consists of a singleloop module with equal probabilities for all letters of the alphabet and a loop transition probability that corresponds to the length of the sequences (the average length if there are multiple sequences) that are being scored against the HMM.

It is possible to use either regular sequences or multiple sequence alignments as input to modhmms. A multiple sequence alignment “letter” is a probability distribution over the alphabet representing the distribution of a column in the same way as for alignment training. The user may choose what columns of an alignment should be used in the search. The alternatives are to use either the columns corresponding to any of the sequences that are part of the alignment (i.e. the columns where this particular sequence has a non gap character) or to use all alignment columns. For alignment scoring it is also possible to adjust the column distributions using Dirichlet priors. When calculating the emission probabilities of a column, one may use either the dotproduct- or the logaverage- methods described in the previous section.

The time complexity for scoring a sequence is equal to the time complexity for running the forward- or Viterbi- algorithm, that is $O(|Q^2| * L)$ for regular sequences, $O(|Q^2| * L * |\Sigma|)$ for alignments using the dotproduct and $O(|Q^2| * L * |\Sigma|^2)$ for alignments using the log-average, optimized to $O(T * L)$, $O(T * L * |\Sigma|)$ and $O(T * L * |\Sigma|^2)$ respectively.

When scoring multiple sequences against multiple HMMs one can use either the sequences or the HMMs as the “query sequences”. Using the HMMs as queries results in an output which for each HMM shows a list containing the scores for all sequences and vice versa.

4 Profile-profile comparison

4.1 Introduction

The goal of this study was to evaluate if using multiple sequence information in the target sequences improves the structure prediction performance of profile HMMs compared to the standard method of using single sequences. Recent results using other profile-profile methods[6] ([14] [24] [25] [26]) show an overall increase in finding remotely homologous sequences (i.e. homologous sequences with less than approximately 20% sequence similarity) compared to methods using multiple sequence information only in the query sequence. Such results suggest that it is possible to achieve a similar improvement with HMMs.

4.2 Methods

Data set The SCOP (Structural Classification of Proteins) database [19] is a hierarchical database for storing information on proteins of known structure. Each protein in SCOP belongs to a *family*, a *superfamily* and a *fold*. Fold is the highest level in the hierarchy, followed by superfamily and family, that is, a fold can consist of many superfamilies and a superfamily may consist of many families. Proteins belonging to the same family are close homologues (high sequence identity) while proteins related only on the superfamily level are more distantly related (medium to low sequence identity), but still homologues. Proteins related only on the fold level may or may not be homologues. SCOP also divides proteins into eleven different fold classes, which is one step above fold in the hierarchy.

For this experiment, a randomly selected data set from the fold class *a* of SCOP 1.57 consisting of 484 sequences was used. PSI-BLAST [2] was used to create multiple sequence alignments of these 484 sequences. The threshold for including a sequence in the alignment was set at E-value = 10^{-3}. 17 sequences were filtered out at this stage since they did not produce any alignment. The remaining 467 were kept. The number of sequences in the alignments ranged from 2 to 579. The average number of sequences per alignment was 136. 72 sequences had alignments of fewer than 10 sequences and 126 sequences had alignments of fewer than 25 sequences.

HMMs An HMM was built from each of the 467 alignments in the following way:

1. An untrained HMM was created using modhmmc. The architecture was a single profile7-module (figure 15) with length equal to the length of the template sequence of the alignment.

[6] *Profile-profile* is a common term for sequence comparison methods which use multiple sequence information both in the target- and query- sequence.

2. The HMM was trained using the standard Baum-Welch algorithm, where the input-sequence was a profile using the columns from the alignment corresponding to the positions where the template sequence had a non-gap character.

3. Each delete→delete probability was set to the share of the already started gaps that continued over this column, adjusted with a simple pseudocount, $dd_i = \frac{gaps_{cont}+1}{gaps_{cont}+gaps_{ending}+2}$. The delete→match probability was set to $dm_i = 1 - dd_i$.

4. The match→match-, match→delete- and match→insert- probabilities were calculated using the share of inserts and deletes started between two template sequence columns (where the template sequence had non gap characters), adjusted with a simple pseudocount:

$$md_i = \frac{gaps_{starting} + 1}{matches_{cont} + inserts_{starting} + gaps_{starting} + 3},$$
$$mi_i = \frac{inserts_{starting} + 1}{matches_{cont} + inserts_{starting} + gaps_{starting} + 3},$$
$$mm_i = 1 - (md_i + mi_i).$$

5. For the insert states where inserts occur in the alignment (i.e. for the states where the corresponding template sequence columns are non adjacent), insert→insert- and insert→match- probabilities were calulated using the average length of the inserts between two template sequence columns in the following way,

$$ii_i = \frac{\sum_{allinserts}(insertlength - 1) + 1}{\sum_{allinserts}(insertlength - 1) + nrofinserts + 2},$$

and

$$im_i = 1 - ii_i.$$

6. For all other insert states, $ii_i = 0.5$ and $im_i = 0.5$.

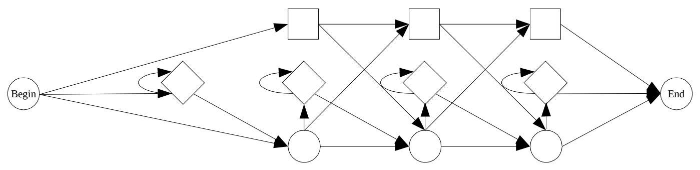

Figure 15: Circles represent match states, diamonds represent insert states and squares represent delete states. The states labeled “Begin” and “End” are the special start- and stop- states.

Test procedure Since the object of this experiment was to determine whether or not multiple sequence information in the target sequence improves structure prediction performance for HMMs, the 467 (query) HMMs were scored against the 467 (target) sequences in two ways. The first, called msaHMM, uses the multiple sequence information in the target sequence alignments. The second, called sinHMM, just scores the target sequences themselves. The scoring method was log-odds (section 2.1.4) on both occasions. When scoring the alignments, the dotproduct method (section 3.3) was used.

To compare the results with the current state of the art for profile HMMs, the sequences were also scored using hmmer-2.1.1[7]. hmmer-HMMs were built for all sequences and scored against the 467 sequences. These results are headlined with “hmmer”. hmmer uses a regular scoring method, i.e. without multiple sequence information in the target sequences.

Evaluation method Results were evaluated in two ways. Both the share of correctly classified sequences and the share of true positives compared to the share of false positives was measured.

The definition of correct classification used on the family level was the following: A sequence s_1 is classified to the correct fold if the sequence with the highest score when compared to s_1 is from the same family as s_1. The score for s_1 itself is disregarded. In the same way, a sequence s_2 was defined to be correctly classified on the super family level if the sequence with the highest score was from the same super family. Here the scores for the sequence itself and all sequences from the same family as s_2 were disregarded. The definition of correct classification on the fold level was done accordingly.

The share of true positives on the family level was defined as the share of sequences from the same family as the sequence scored against that were found given a certain threshold score value, and analogous for false positives. Sequences from the correct fold, but not from the same family were counted as neither positives nor negatives. The share of true positives compared to the share of false positives can be said to represent a ratio between how many sequences were correctly classified and how many that were falsely classified to a certain fold, given a specific threshold value. The definition for the superfamily and family levels were done accordingly, again disregarding true positives that belong to a lower category in the hierarchy, e.g. on the super family level true positives belonging to the same family as the sequence scored against were disregarded, as were sequences from the correct fold but from a different superfamily.

[7]http://hmmer.wustl.edu/

4.3 Results

Correct classification As can be seen in table 1, using multiple sequence information clearly improves distant homology detection. i.e. detection on the super-family- and fold- levels. Furthermore, the results show that the msaHMM method performs better also for close homology detection, for which all methods perform well. The *total* category measures the total number of correctly classified proteins, calculated as a weighted average of the scores for the three sub categories. This category also includes the 17 sequences for which no alignment was found, i.e. the theoretical max score for this category was 96.5%. It also looks as if the modhmm package performance is on par with hmmer as the results for the comparable scoring methods sinHMM and hmmer are roughly the same for all three categories.

Share correct

Category	sinHMM	msaHMM	hmmer
family	88.0%	89.5%	88.9%
superfamily	41.5%	49.0%	38.8%
fold	10.0%	21.8%	12.9%
total	71.8%	74.8%	72.7 %

Table 1: The share of correctly classified protein sequences

True positives vs. false Positives As for the share correct-classification, true positives vs. false positives (figures 16, 17, 18 and table 2) show that distant homology detection as well as close homology detection clearly improves when using multiple sequence information in the target sequences. The results for sinHMM are slightly lower than the results for hmmer here, but still on the same level. The shapes of the curves suggest that the msaHMM method is good at improving the scores for related sequences which already score fairly well using sinHMM or hmmer, but seems to give even lower scores for the related sequences with initially low scores.

True positives at 1% false

Category	sinHMM	msaHMM	hmmer
family	87.3%	93.2%	91.1%
superfamily	30.6%	42.6%	32.4%
fold	4.0%	8.1%	3.8%

Table 2: The share of true positives found at 1% false positives found

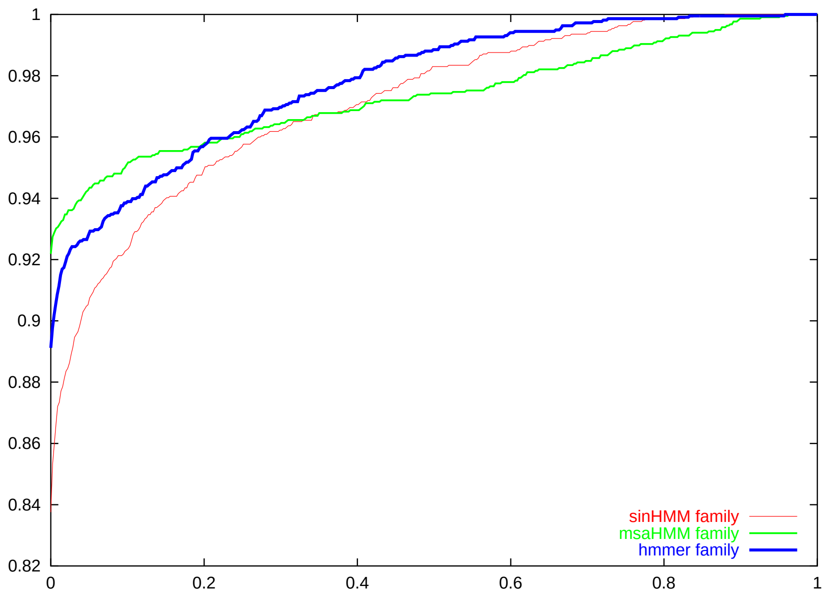

Figure 16: The true positives (y-axis) vs. false positives (x-axis) diagram for sinHMM, msaHMM and hmmer on the family level

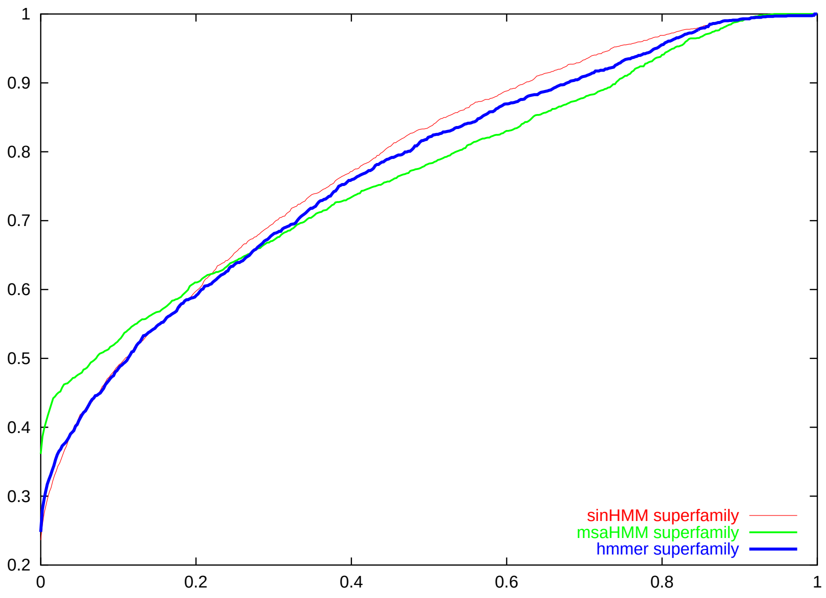

Figure 17: The true positives (y-axis) vs. false positives (x-axis) diagram for sinHMM, msaHMM and hmmer on the superfamily level

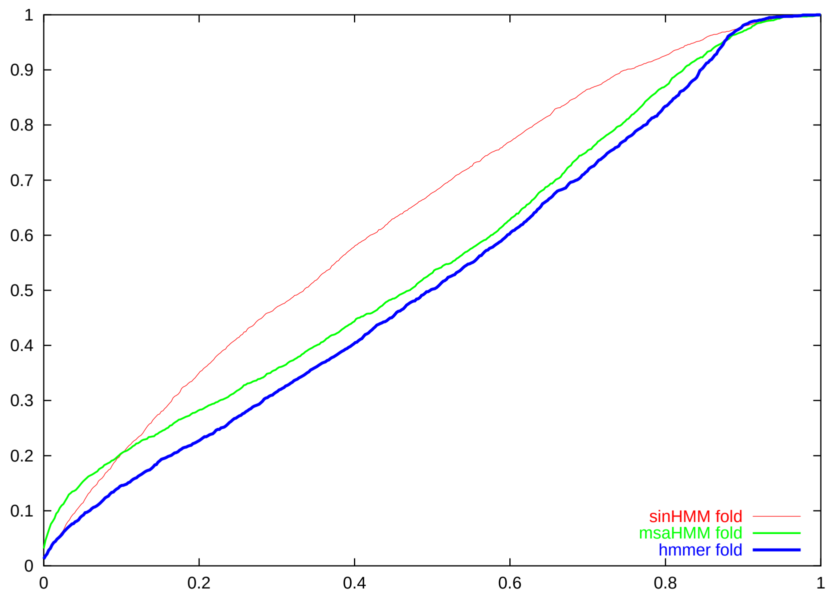

Figure 18: The true positives (y-axis) vs. false positives (x-axis) diagram for sinHMM, msaHMM and hmmer on the fold level.

5 Discussion

5.1 modhmm package

The primary goal of this project was to implement a package for constructing and training HMMs as well as scoring sequences aginst HMMs. To accomplish this, three programs were built, modhmmc (construction), modhmmt (training) and modhmms (scoring). The three basic algorithms forward, backward and Viterbi, as well as the standard training algorithm, Baum-Welch, were implemented to support an arbitrary HMM-architecture. The possibility of using prior information was incorporated into the training- and scoring- algorithms. In addition to this, the possibility of training and scoring using multiple sequence alignments as input was implemented.

Judging from the profile-profile experiment and other tests, the modhmm package seems to work well. As indicated by the results from the profile-profile test, modhmm seems to perform on par with hmmer[8]. A slight problem with modhmm is the speed of modhmms and modhmmt programs. Comparisons with hmmer during the profile-profile experiment show that modhmm is approximately 4 times slower than hmmer. To some extent this is to be expected, since hmmer is a very fast program customized for profile-hmms and modhmm is more general, which prohibits certain “shortcuts” in the algorithms. However, modhmm would need to be optimized in order for it to be useful for large database searches. As mentioned earlier, the user interface of modhmmc is also rudimentary and needs further development. One possibility is to add a graphical user interface. Furthermore, no real evaluation of the optimization methods for the training algorithm has been done.

5.2 Profile-profile comparison

The second goal of this project was to test whether including multiple sequence information in both query- and target- sequences improves the fold recognition of HMMs compared to using multiple sequence information only in the query. The results show that this clearly is the case. Especially remote homology detection is improved using this method. Overall, the results for msaHMM (the profile-profile method) at the superfamily- and fold- levels are good. 49.0% and 21.8% respectively of correctly classified sequences are at the same level as the top methods, although it is not really possible to make such comparisons over different test sets. It would be interesting to try the msaHMM method on a larger and more diverse test set and also to compare the results with the scores of other methods. For this test set, the 21.8% on the fold level is probably raised some due to the composition and size of the test set. The score of a sequence related on the fold level is

[8]http://hmmer.wustl.edu/

usually not that much higher than the score of a non related sequence. With a larger and more diverse test set, the probability of getting a score from a random sequence which is higher than the score for the best scoring fold level-related sequence increases. This problem could also affect the results on the superfamily- and family- levels slightly, but most certainly not to a significant degree, since the scores for more closely related sequences usually differ more from the scores of the unrelated sequences.

Some possible improvements to the results of msaHMM and HMM are also possible. For instance, it may be better to use the log average scoring method instead of the dot product. The drawback with this is the speed decrease. Another possible improvement would be to use the information from the sequence alignments to set the emission probabilities of the insert states. In the experiment performed here, training was done without using these columns of the alignments, resulting in emission probabilities for the insert states more or less equal to the Dirichlet background distribution. Furthermore, all scoring in this experiment was done globally, i.e. a whole sequence was scored against a whole HMM. It is possible to change the profile-HMM architecture, as well as the alignments to enable both global-local and local-local scoring. Since local scoring is less affected by differences in sequence lengths than global scoring, this would probably lead to slightly better and more stable results.

Acknowledgements I would like to thank my supervisor at SBC, Arne Elofsson, for giving me the opportunity to work with this project and for introducing me to the field of bioinformatics.

References

[1] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., Lipman, D. J. (1990). Basic Local Alignment Search Tool. *J. Mol. Biol* **215**, 403-410.

[2] Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W., Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research Vol.25* **17**, 3389-3402.

[3] Brown, M., Hughey, R., Krogh, A., Mian, I. S., Sjölander, K., Haussler, D. (1993). Using Dirichlet mixture priors to derive hidden Markov models for protein families. *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology*, 47-55. AAAI Press.

[4] Bystroff, C., Thorsson, V., Baker, D. (2000). HMMSTR: a Hidden Markov Model for Local Sequence-Structure Correlations in Proteins. *J. Mol. Biol.* **301**, 173-190.

[5] Cox, D. R., Miller, H. D. (1965). *The theory of stochastic processes.* Chapman & Hall.

[6] Durbin, R., Eddy, S. R., Krogh, A., Mitchison, G. (1998). *Biological sequence analysis, probabilistic models of proteins and nucleic acids.* Cambridge University Press, Cambridge.

[7] Eddy, S. R. (1995). Multiple alignment using hidden Markov models. *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, 114-120. AAAI Press.

[8] Eddy, S. R. (1996). Hidden Markov models. *Current Opinion in Structural Biology* **6**, 361-365.

[9] Eddy, S. R. (1998). Profile Hidden Markov models. *Bioinformatics* **14**, 755-763.

[10] Goto, O. (1982). An improved algorithm for matching biological sequences. *J. Mol. Biol* **162**, 705-708.

[11] Hargbo, J., Elofsson, A. (1999). Hidden Markov models that use predicted secondary structures for fold recognition. *Proteins: Structures, Function and Genetics* **36**, 68-76.

[12] Henikoff, S., Henikoff J. G. (1992). Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci.* **89(22)**, 10915-10919.

[13] Hughey, R., Krogh, A. (1996). Hidden Markov models for sequence analysis: extension and analysis of the basic method. *CABIOS*, **12** (2), 95-107.

[14] Jaroszewski, L., Rychlewski, L., Godzik, A. (2000). Improving the quality of twilight-zone alignments. *Protein Science* **9**, 1487-1496.

[15] Jones, D. T., Taylor, W. R., Thornton, J. M. (1992). A new approach to protein fold recognition. *Nature*, **358**, 86-89.

[16] Krogh, A., Brown, M., Saira Mian, I., Sjölander, K., Haussler, D. (1994). Hidden Markov models in computational biology: Applications to protein modeling. *J. Mol. Biol.* **235**, 1501-1531.

[17] Krogh, A., Larsson, B., von Heijne, G., Sonnhammer, E. L. L. (2001). Predicting Transmembrane Protein Topology with a Hidden Markov Model: Application to Complete Genomes. *J. Mol. Biol.* **305**, 567-580.

[18] Larsson, B. (1999). *Development of HMMs that use multiple sequence alignments*. Master thesis project, Stockholm Bioinformatics Center.

[19] Murzin A. G., Brenner S. E., Hubbard T., Chothia C. (1995). SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.* **247**, 536-540.

[20] Pollastri, G., Baldi, P. (2002). Prediction of contact maps by GIOHMMs and recurrent neural networks using lateral propagation from all four cardinal corners. *Bioinformatics* **18**, S62-S70.

[21] Rabiner, L. R. (1989). A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE, vol.77*, **2**.

[22] Sjölander, K., Karplus, K., Brown, M., Hughey, R., Krogh, A., Mian, I. S., Haussler, D. (1996). Dirichlet Mixtures: A method for improving detection of weak but significant protein sequence homology. *CABIOS*, **12** (4), 327-345.

[23] Smith, T. F., Waterman, M. S. (1981). Identification of common molecular subseuences. *J. Mol. Biol* **147**, 195-197.

[24] Yona, G., Levitt, M. (2002). Within the Twilight Zone: A sensitive profile-profile comparison tool based on information theory. *J. Mol. Biol* **315**, 1257-1275.

[25] von Öhsen, N., Zimmer, R. (2001). Improving profile-profile alignments via log average scoring. *WABI 2001*

[26] von Öhsen, N., Sommer, I., Zimmer, R. (2003). Profile-profile alignment: A powerful tool for protein structure prediction. *Pac Symp Biocomput.*, 252-63