

# 1 Introduction

In protein fold recognition, the fold of a probe amino acid-sequence is determined using a library of known folds, by searching for a clear homology. Traditionally this has been done using dynamical programming algorithms based on substitution matrices like the Needleman-Wunch- [10], or Smith-Waterman-algorithms[15], or by fast heuristic algorithms like BLAST[1] or FASTA[11]. During the last years several methods have been developed to increase the sensitivity of homology recognition between proteins, many of which makes use of machine learning algorithms like neural networks or hidden Markov models. Basically these newer methods can be divided into two groups, namely methods that uses multiple sequence information derived from multiple alignments, and methods that uses information about the 3 dimensional structure of proteins.

3D-structural information can be used in many different ways, such as finding the best matching fold by threading the probe sequence through a library set of folds and calculating the energy for each possible alignment of the probe sequence onto that fold, so called threading based methods. Some of the most efficient threading methods are based on predicted properties of the probe sequence. In works by Fisher & Eisenberg [6] and Rice & Eisenberg [12] algorithms are described where classical sequence alignment methods used, but if the predicted secondary structure of a residue of the probe sequence belongs to the same secondary structure class as the residue it is aligned with, the score of the alignment will be more favourable than else, and vice versa.

Multiple sequence information have e.g. been combined with neural networks to increase the accuracy of protein secondary structure predictions [13]. Hidden Markov models (HMMs) are probabilistic models usefull for string matching applications that are widely used in speech recognition. Recently HMMs have been introduced in molecular biology as a very efficient way of representing the information of a multiple sequence alignment (MA)[5], and are thus an excellent tool for incorporating statistical information about each particular protein family/fold into the library families/folds, which can be represented as HMMs[4].

In an earlier study by Hargbo & Elofsson [7] the sensitivity of protein fold recognition with hidden Markov models was increased by incorporating information about predicted secondary structures. The aim of this project was to investigate whether a corresponding improvement of the sensitivity of HMMS for protein fold recognition could be obtained by using multiple sequence information in the probe sequence as well as in the library sequences. This was in practice carried out by developing algorithms for the alignment of two multiple sequence alignments, for cases where one of the multiple alignments is represented as a hidden Markov model. The big problem connected with this approach is that in contrary to alignment of a single probe sequence with a HMM, which can recursively be broken down into smaller problems, in this case recursion breaks down, which means that if  $K$  denotes the sequence length,  $L$  the length of the probe sequence/multiple sequence alignment, and  $P$  the number of sequences in the alignment, the time complexity of aligning a MA with a HMM will be  $O(P*K*L^2)$  whereas the time-complexity of an ordinary single probe sequence -HMM alignment is  $O(K*L)$ . In addition the leading term coefficient will be larger to for the MA-HMM alignment algorithm, which means that for datamining purposes, when we are searching large datasets this algorithm is practically useless unless one posses a high performance parallell computer. To avoid the drawbacks of this approach we have tried to find  $O(P*K*L)$  suboptimal heuristical approximations of the MA against HMM alignments that could increase the sensitivity in compairison to single sequence against HMM alignment. We have tried two different approaches to the approximation problem, which are here termed reference sequence search and "in flight" state distribution estimation.

## 2 Theory

Two books that together provide an excellent general introduction to the field of machine learning and probabilistic modelling in molecular bioinformatics are [?] and [3]. These two books provide the foundation for this theory-section.

Hidden Markov models (HMMs) are probabilistic models which has proven to be useful for problems that can be viewed as generalized (stochastic) string matching problems. In the case of family/fold recognition of proteins a special architecture of hidden Markov models, called profileHMMs are widely used. The profileHMMs were introduced by David Haussler, Anders Krogh and co-workers[5]. In this work, plan 7, a modification of the original Krogh-Haussler architecture, developed by Sean Eddy[5] has been used.

Since HMMs are probabilistic models they, as all probabilistic models, contains a number of parameters which has to be estimated from data. Subsection 2.1 of the theory section therefore contains an introduction to the methods of parameter estimation developed in mathematical statistics. To build models, assumptions must often be made about the distributions of data, some of the most important distributions used for profileHMMs purposes are briefly described in subsection 2.2. In subsection 2.3, hidden Markov models are formally defined, profileHMMs and the plan 7 architecture are introduced, and the problems of sequence scoring and model building are discussed at length. Although this discussion is meant to be general, in some cases restrictions must be made, since a general discussion would be too voluminous. Subsection 2.4 finally contains an overview over methods used to determine the significance of the match between a sequence and a model.

### 2.1 Statistical parameter estimation

#### 2.1.1 Maximum Likelihood-estimation

One of the most central problems in science and engineering is to build models  $M$  for systems that explains some set of measurements,  $\overline{D}$  in some optimal sense. Usually this model critically depends on sets of free parameters  $\overline{\theta}$ , which has to be estimated. The classical way of estimating this set of parameters in statistics has been through *maximum-likelihood* (ML) estimation. The set of measurements is considered to consist of  $p$  observations  $d_1, \dots, d_p$  of random variables  $X_1, \dots, X_p$ . The likelihood-function,  $L(\overline{\theta}|\overline{D} \cap M)$  is then defined as the conditional probability of generating the observations given the current set of parameters:

$$L(\overline{\theta}|\overline{D} \cap M) = P(\overline{D}|\overline{\theta} \cap M) = P(X_1 = d_1 \cap \dots \cap X_p = d_p|\overline{\theta} \cap M) = f_{\overline{X}}(d_1, \dots, d_p|\overline{\theta} \cap M)$$

where  $f_{\overline{X}}(d_1, \dots, d_p|\overline{\theta} \cap M)$  is the simultaneous probability mass/density function for the set of random variables,  $\overline{X} = \{X_1, \dots, X_p\}$ . The usual terminology is to refer to  $f_{\overline{X}}(d_1, \dots, d_p|\overline{\theta} \cap M)$  as a probability mass/density function when it is a function of  $d_1, \dots, d_p$  that are unknown values of future observations of the stochastic variables for a given set of parameters  $\overline{\theta}$ , and as a likelihood function when it is a function of the values of the parameters  $\overline{\theta}$  when  $d_1, \dots, d_p$  is a given set of already measured realizations of the corresponding stochastic variables. In maximum likelihood estimation this function (the likelihood-function) is maximized with respect to the free parameters. In this text the maximum likelihood estimate of a parameter  $\xi$ , will be denoted  $\tilde{\xi}$ , i.e.

$$\tilde{\overline{\theta}} = \operatorname{argmax}_{\overline{\theta}}(f_{\overline{X}}(d_1, \dots, d_p|\overline{\theta} \cap M))$$

The maximum likelihood-estimate has the pleasant property that it is invariant under non-linear transformations, i.e. under very general conditions the maximum likelihood-estimate of  $\varphi = h(\theta)$ , where  $h$  is some non-linear function of  $\theta$ , is  $\tilde{\varphi} =$

$h(\tilde{\theta})$ . In other words the ML-estimate of the transformed parameters equals the transformation of the ML-estimate of the original parameters. A more rigorous discussion about the invariance property of ML-estimates can be found in [2].

A very common problem is to estimate the probabilities of occurrence of each outcome from a finite set of mutually exclusive and exhaustive outcomes  $\Xi$  for a random event. Let  $\alpha$  be an index over  $\Xi$  and let  $p_\alpha$  be the probability that outcome  $\alpha$  will occur. It can be shown that the ML-estimates of the  $p_\alpha : s$  are the relative frequencies by which the different  $\alpha : s$  occur in the set of measurements. Because of the frequent usage of this result in the context of hidden Markov models the proof given in [3] will be cited here. Let  $\bar{n} = \{n_\alpha | \alpha \in \Xi\}$  be the set of the number of times each outcome occurred among the  $N$  independent observations, and let  $\bar{p}$  be defined analogously as the set of probabilities of occurrence of each outcome in  $\Xi$ ,  $\bar{p} = \{p_\alpha | \alpha \in \Xi\}$ .  $\tilde{\bar{p}} = \{\tilde{p}_\alpha = \frac{n_\alpha}{N}\}$  is then the ML-estimate of  $\bar{p}$  if  $P(\bar{n}|\tilde{\bar{p}}) > P(\bar{n}|\hat{\bar{p}})$  for all other sets of estimates  $\hat{\bar{p}} = \{\hat{p}_\alpha\}$ , this is equivalent to showing that  $\log[P(\bar{n}|\tilde{\bar{p}})/P(\bar{n}|\hat{\bar{p}})] > 0$  for all other estimates:

$$\log \frac{P(\bar{n}|\tilde{\bar{p}})}{P(\bar{n}|\hat{\bar{p}})} = \log \frac{\prod_\alpha (\tilde{p}_\alpha)^{n_\alpha}}{\prod_\alpha (\hat{p}_\alpha)^{n_\alpha}} = \sum_\alpha n_\alpha * \log\left(\frac{\tilde{p}_\alpha}{\hat{p}_\alpha}\right) = N * \sum_\alpha \tilde{p}_\alpha * \log\left(\frac{\tilde{p}_\alpha}{\hat{p}_\alpha}\right) > 0$$

The last inequality follows from the fact that relative entropy between two discrete stochastic variables is always non-negative and equals zero only if the variables are equally distributed. The proof of the non-negativity of the relative entropy for discrete finite state random variables can be found in [4].

### 2.1.2 Bayesian estimation

When data are very sparse, or the number of free parameters in the model is large, the maximum likelihood-estimate will suffer from a high variance, which means that although the average of a large number of estimates could be unbiased, each particular estimate could be very poor. To overcome this problem, some sort of prior knowledge must be incorporated in the parameter estimation. Bayesian statistics provides the framework for doing this. In this formalism we have to view the parameters  $\bar{\theta}$  as an observation of stochastic variables  $\bar{\Theta}$ , i.e. we represent our a priori knowledge about the system as a a priori probability distribution,  $g_{\bar{\Theta}}(\bar{\theta}) = P(\bar{\Theta} = \bar{\theta} | M)$  for the parameters of the model. According to Bayes law, the a posteriori probability of a parameter-vector  $\bar{\theta}$ ,  $P(\bar{\Theta} = \bar{\theta} | \bar{D} \cap M)$ , given a model  $M$  and a set of observation  $\bar{D}$ . is given by:

$$P(\bar{\Theta} = \bar{\theta} | \bar{D} \cap M) = \frac{P(\bar{D} | \bar{\Theta} = \bar{\theta} \cap M) * P(\bar{\Theta} = \bar{\theta} | M)}{P(\bar{D} | M)}$$

where  $P(\bar{D} | \bar{\Theta} = \bar{\theta} \cap M)$  is the usual likelihood-function. In Bayesian statistics there are two ways of doing parameter (point) estimation, namely maximum a posteriori(MAP)-estimation and mean a posteriori-estimation(MPE). According to the MPE- method  $\bar{\theta}$  is estimated by,  $\bar{\theta}_{PME}^*$ , the expectation value of  $\bar{\theta}$  over the a posteriori distribution:

$$\bar{\theta}_{PME}^* = \int \bar{\theta} * g_{\bar{\Theta}}(\bar{\theta}) * P(\bar{D} | \bar{\Theta} = \bar{\theta} \cap M) d\bar{\theta}$$

In the case of discrete valued  $\bar{\theta}$  the integral is of course replaced by a sum. The MAP-estimate of  $\bar{\theta}$ ,  $\bar{\theta}_{MAP}^*$  is the value of  $\bar{\theta}$  that maximizes the a posteriori probability.

$$\bar{\theta}_{MAP}^* = \operatorname{argmax}_\theta (g_{\bar{\Theta}}(\bar{\theta}) * P(\bar{D} | \bar{\Theta} = \bar{\theta} \cap M))$$

Since the probability  $P(\bar{D}|M)$  does not depend on the parameters  $\bar{\theta}$ , it is only a normalization constant with respect to optimization or integration, and does not affect the calculation, although it affects the numerical value of the estimate.

A useful concept in the Bayesian paradigm is the concept of conjugated families. According to the definition in [2] if  $\Phi$  is a family (set) of probability mass functions (pmf) or probability density functions (pdf),  $\phi(x, \theta)$ ,  $\Phi = \{\phi(x, \theta)\}$  and  $\Pi$  is family of a priori pmfs or pdfs,  $\pi$  such as that  $\forall \phi \in \Phi$  and  $\forall \pi \in \Pi$ , the corresponding posteriori pmf or pdf,  $\varpi \in \Pi$ , the families of distributions are termed *conjugated*. In other words if the a posteriori distribution belongs to the same family of distributions as the a priori-distribution for all the members of a distribution family, then these families are conjugated. This is usefull because if one can prove that two distributions are conjugated, then one would know the of the a posteriori distribution.

Bayesian estimates have long been viewed with a certain suspicion among statisticians for at least two reasons [3]. Firstly and most importantly because Bayesian estimates are not generally invariant under non-linear transformations, i.e. if we let  $\theta^*$  denote a Bayesian estimate of  $\theta$ , then, to reuse the notation of the discussion about the invariance of ML-estimates, in general  $\varphi^* \neq h(\theta^*)$  because of the volume change due to the coordinate transformation (remember  $d\varphi = |h'| * d\theta$ ). Secondly, the assumption that “constant” parameters have probability distributions raises a number of philosophical questions. For the purposes of bioinformatics we must however be pragmatic and put on our engineering caps to conclude that Bayesian estimation, despite these short-comings, in practice in the situation of very sparse data is clearly superior to ML-estimation. A good introduction to the general theory of statistical point estimation is found in [2].

### 2.1.3 Maximum Likelihood-estimation of hidden data - The EM/GEM algorithms

A general approach for computing maximum likelihood-estimates in the case of incomplete or hidden data was given by Dempster, Laird and Rubin [3]. This algorithm called the *Expectation Maximization* algorithm, is an iterative algorithm that alternates between two steps, the prediction step and the estimation step. In the case of HMMs the hidden data is obviously the paths. What we really are interested in is the ML estimate of  $\theta$  from  $P(\bar{D}|\bar{\theta})$ , but because of the hidden data this can not be directly calculated. To compute the ML-estimate we have to maximize over the sum of all possible values of the simultaneous likelihood function including the hidden data  $P(\bar{D} \cap \bar{H}|\bar{\theta})$ . According to the definition of conditional probability we can write the simultaneous likelihood of the visible  $\bar{D}$  and hidden  $\bar{H}$  data given a set of parameters  $\bar{\theta}$  (represented as a vector), as:

$$P(\bar{D} \cap \bar{H}|\bar{\theta}) = P(\bar{H}|\bar{D} \cap \bar{\theta}) * P(\bar{D}|\bar{\theta})$$

By taking the logarithm of both sides of and doing some reshuffling:

$$\log[P(\bar{D}|\bar{\theta})] = \log[P(\bar{D} \cap \bar{H}|\bar{\theta})] - \log[P(\bar{H}|\bar{D} \cap \bar{\theta})]$$

Let  ${}_i\bar{\theta}^*$  denote the present estimate of the parameter vector. Multiply with  $P(\bar{H}|\bar{D} \cap {}_i\bar{\theta}^*)$  and integrate/sum over all possible values of  $\bar{H}$ , i.e. we calculate some sort of entropy-measure. Of course, since the left side of does not depend on  $\bar{H}$  this will be equivalent to multiply the left side with 1. The rest of this discussion will be restricted to the case of discrete valued hidden data. Let  $\mathbf{H}$  be the set of possible values of  $\bar{H}$ . The difference between  $\log[P(\bar{D}|\bar{\theta})]$  for the true parameter-vector and  $\log[P(\bar{D}|{}_i\bar{\theta}^*)]$  for the estimated  ${}_i\bar{\theta}^*$  is then calculated. The rest of this discussion

will be restricted to the case of discrete valued hidden data.

$$\log[P(\bar{D}|\bar{\theta})] - \log[P(\bar{D}|\bar{\theta}^*)] = \sum_{\bar{H} \in \mathbf{H}} [P(\bar{H}|\bar{D}\bar{\theta}) * (\log[P(\bar{D}\bar{H}|\bar{\theta})] - \log[P(\bar{H}|\bar{D}\bar{\theta})])] - \sum_{\bar{H} \in \mathbf{H}} [P(\bar{H}|\bar{D}\bar{\theta}^*) * (\log[P(\bar{D}\bar{H}|\bar{\theta}^*)] - \log[P(\bar{H}|\bar{D}\bar{\theta}^*)])]$$

We now define  $Q(\bar{\theta}|\bar{\theta}^*)$  to be:

$$Q(\bar{\theta}|\bar{\theta}^*) = \sum_{l=1} (P(\bar{H}|\bar{D}\bar{\theta} \cap \bar{\theta}^*) * \log[P(\bar{D}\bar{H}|\bar{\theta})]) - \sum_{l=1} (P(\bar{H}|\bar{D}\bar{\theta}^* \cap \bar{\theta}^*) * \log[P(\bar{D}\bar{H}|\bar{\theta}^*)])$$

We can now write the difference  $\log[P(\bar{D}|\bar{\theta})] - \log[P(\bar{D}|\bar{\theta}^*)]$  as:

$$\log[P(\bar{D}|\bar{\theta})] - \log[P(\bar{D}|\bar{\theta}^*)] = Q(\bar{\theta}|\bar{\theta}^*) - Q(\bar{\theta}^*|\bar{\theta}^*) + \sum_{\bar{H} \in \mathbf{H}} (P(\bar{H}|\bar{D}\bar{\theta} \cap \bar{\theta}^*) * \log(\frac{P(\bar{H}|\bar{D}\bar{\theta} \cap \bar{\theta}^*)}{P(\bar{H}|\bar{D}\bar{\theta} \cap \bar{\theta})}) - P(\bar{H}|\bar{D}\bar{\theta}^* \cap \bar{\theta}^*) * \log(\frac{P(\bar{H}|\bar{D}\bar{\theta}^* \cap \bar{\theta}^*)}{P(\bar{H}|\bar{D}\bar{\theta}^* \cap \bar{\theta}^*)}))$$

But since the relative entropy of two stochastic variables are always non-negative:

$$\log[P(\bar{D}|\bar{\theta})] - \log[P(\bar{D}|\bar{\theta}^*)] > Q(\bar{\theta}|\bar{\theta}^*) - Q(\bar{\theta}^*|\bar{\theta}^*)$$

with equality only if  $\bar{\theta} = \bar{\theta}^*$ , so by choosing  $\bar{\theta}_{i+1} = \text{argmax}_{\theta} (Q(\bar{\theta}|\bar{\theta}^*))$  the difference  $\log[P(\bar{D}|\bar{\theta}_{i+1})] - \log[P(\bar{D}|\bar{\theta}^*)]$  will always be positive, until the maximum is reached. This means that the algorithm will with probability 1 converge to a local maximum. The two steps of the algorithm can then be formulated:

1. Prediction step. Calculate the  $Q(\bar{\theta}|\bar{\theta}^*)$  function.
2. Estimation step. Maximize this with respect to  $\theta$ , and choose the argument of the maximum to be the new estimate of  $\theta$ .

$$\bar{\theta}_{i+1} = \text{argmax}_{\theta} [Q(\bar{\theta}|\bar{\theta}^*)]$$

Iterate till convergence.

This discussion is based on and closely follows the descriptions of the EM/GEM algorithms found in [3].

## 2.2 A probability theory primer

### 2.2.1 The binomial and multinomial distribution

The binomial distribution,  $\text{Bin}(N, p)$ , is the distribution of the number of times one of two mutually exclusive and exhaustive outcomes occur during  $N$  independent random trials, if the probability of the outcome is  $p$ . The example of the binomial distribution is  $N$  flippings of the same coin. The probability mass function of the binomial distribution is the well-known:

$$P(n|\text{Bin}(N, p)) = \binom{N}{n} p^n (1-p)^{N-n}$$

The multinomial distribution is a generalization of the binomial distribution for the case of more than two possible outcomes of a random event. Also in this case it is assumed that the  $N$  realizations of this event are independent. A good example of multinomial distributed data would be the number of times each number was of  $N$  independent rolls of a dice. If  $n_1, \dots, n_M$  denotes the number of times each of the  $M$  possible outcomes occur, and  $p_1, \dots, p_M$  are the corresponding probabilities, the probability mass function of the multinomial distribution can be written as:

$$p(n_1, \dots, n_M) = \frac{N!}{n_1! * \dots * n_M!} \prod_{i=1}^M p_i^{n_i}$$

### 2.2.2 The Dirichlet distribution

The Dirichlet distribution is conjugate to the multinomial distribution. In other words this means that the Dirichlet distribution is a distribution over the probabilities for the different symbols, whereas the multinomial distribution is a distribution for the exponents. Formally the M-dimensional Dirichlet-distribution is a probability density distribution over the set of all possible probability vectors  $\bar{p}$  satisfying  $\sum_{i=1}^M p_i = 1$  and  $p_i \geq 0 \forall i$ , defined by a set of parameters  $\bar{\alpha} = (\alpha_1, \dots, \alpha_M)$ . Following the definition in [3] the probability density function for the M-dimensional Dirichlet distribution can be written:

$$D(\bar{p}|\bar{\alpha}) = \frac{1}{Z(\bar{\alpha})} \prod_{i=1}^M p_i^{\alpha_i-1}$$

where  $Z(\bar{\alpha})$  is given by:

$$Z(\bar{\alpha}) = \frac{\prod_{i=1}^M \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^M \alpha_i)}$$

algebraically identical to the multinomial distribution. The expectation values of the different probabilities  $p_i$  is given by the relative fraction of the sum  $\kappa = \sum_j \alpha_j$  caused by  $\alpha_i$ .

$$E(p_i) = \frac{\alpha_i}{\sum_i \alpha_i}$$

The expectation value of  $p_i$  does thus not depend on the magnitude of  $\kappa$  as long as the relative proportions of the  $\alpha_j$ s are unchanged. The variances and covariances do however depend on the magnitude of  $\kappa$ . The higher value of  $\kappa$ , the sharper the distribution of  $p_i$  around  $E(p_i)$  becomes. The magnitude of  $\kappa$  can be thought of as a way of expressing our confidence in the prior value, the stronger our confidence is, the higher value should we choose.

### 2.2.3 The Extreme Value Distribution

The Extreme Value Distribution (EVD) is the limiting distribution for the largest or smallest value of a large sample of realizations of identically distributed random variables. What is usually referred to when using the term extreme value distribution is the Fisher-Tippett -distribution also called the log-Weibull distribution. In this text the terms extreme value distribution and Fisher-Tippett distribution will be used interchangeably. It can be proven that the limiting distribution for the minimum or maximum value of large samples of identically distributed random variables belongs one of three classes, one of which is the Fisher-Tippett distribution. Moreover the Fisher-Tippett distribution is the limiting distribution for many of the most practically important distributions e.g. the Gaussian distribution and many of the distributions in the exponential family. In addition to its usefulness in evaluating the significance of score in biological sequence analysis the EVD has been used in e.g. solid mechanics to model the breaking point of chains. The Fisher-Tippett distribution is characterized by two parameters a and b, defining the probability density function  $f_X(x)$  to be:

$$f_X(x) = \frac{e^{(a-x)/b} e^{-e^{(a-x)/b}}}{b}$$

The cumulative distribution function  $F_X(x)$  of the Fisher-Tippett distribution is:

$$F_X(x) = e^{-e^{(a-x)/b}}$$

The mean  $\mu$  and variance  $\sigma^2$  of the Fisher-Tippett distribution is:

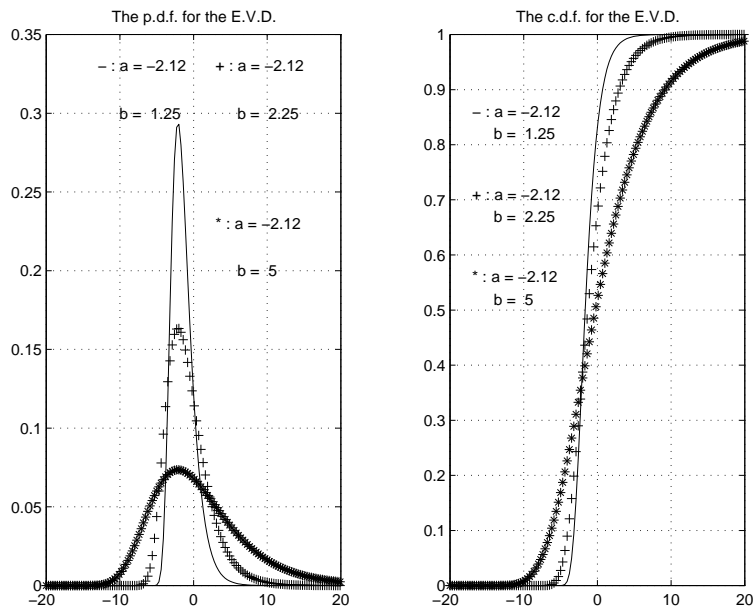
$$\mu = a + b * \gamma$$

$$\sigma^2 = \frac{1}{6} * \pi^2 * b^2$$

Where  $\gamma$  is Eulers constant.

In the case  $a=0$  and  $b=1$ , the Fisher-Tippett distribution is usually refered to as the Gumpel distribution.

A plot of the probability density function and the cumulative probability function for some values of  $a$  and  $b$  are shown below. Note that around the maximum of the density curve, the Fisher-Tippett distribution is strongly reminiscent of the Gaussian distribution, though it is not symmetric as the Gaussian.



## 2.3 Hidden Markov Models

### 2.3.1 Markov chains and the Markov condition

A Stochastic process is a family of functions that maps the sample space of a random event, i.e. the set of possible outcomes, to some set of numbers. In contrast to ordinary stochastic variables the mapping is time-dependent, thus for . For probabilistic modelling in pattern analysis of macromolecules a special type of stochastic processes, namely Hidden Markov Models is of special importance.

Markov Models (Chains) have been an important tool in science and engineering for almost 90 years. They are usefull in cases when the system being studied can be modelled as a discrete time random walk between a finite or infinite number of discrete states. In fact the theory can be generalized to handle even time-continuous systems. Only discrete time, finite state Markov chains are however of interest in biomolecular sequence analysis, therefor this discussion will be restricted to such cases. We denote the  $m$  possible states of the system  $a_1, \dots, a_m$ , and introduce  $a(t)$  to be a variable representing the actual state of the system at time  $t$ ,  $a(t) \in \{a_1, \dots, a_m\} \forall t$ .

Let us assume that the state of the system in question at time  $t$ ,  $a(t)$ , can be modelled as an observation of a discrete time, finite state stochastic process  $A(t)$ .

The stochastic process  $A(t)$  can be represented as a Markov Model if it fulfills the Markov Condition, i.e. that the conditional probability mass function of  $A(t)$  at any given time  $t$ , given that we know the probability mass functions for  $n$  previous times  $t-1, \dots, t-n$ ,  $P(A(t) = a(t) | A(t-1) = a(t-1) \cap \dots \cap A(t-n) = a(t-n))$ , equals the conditional probability mass function for  $A(t)$  given the probability distribution at time  $t-1$ ,  $P(A(t) = a(t) | A(t-1) = a(t-1))$ .

$$P(A(t) = a(t) | A(t-1) = a(t-1) \cap \dots \cap A(t-n) = a(t-n)) = P(A(t) = a(t) | A(t-1) = a(t-1))$$

Only the case of stationary Markov chains will be of interest in this work. This implies that for each pair  $a(t) = a_i$ ,  $a(t-1) = a_j$ , there exist a constant transition probability  $t_{ij} = P(A(t) = a_i | A(t-1) = a_j)$ . The transition probabilities of a Markov chain are usually represented in a transition matrix  $T = (t_{ij})$ . The states of an Markov Model are considered to be directly observable, i.e. from the output of the system at any given time the state can be concluded.

### 2.3.2 Hidden Markov Models

Hidden Markov Models are a generalization of the usual Markov Models in the sense that there is no longer a bijective (1 to 1) relation between the state of the system and the observed output of the system for a given time step. This means that  $a(t)$  is no longer directly retrievable from the output of the system  $e(t)$ . For Hidden Markov Models (usually abbreviated HMMs) we will only consider the case where the output generated by the system at each time-step is a symbol from a finite alphabet,  $E$ , consisting of  $k$  symbols denoted  $e_1, \dots, e_k$ . (Although the case of outputs drawn from continuous probability density functions is straight-forward from a theoretical viewpoint, it is not interesting for our purposes, and will thus not be considered.) This means that the output of a HMM,  $e(t)$  can be viewed as a realization of a doubly stochastic process  $E(t)$ , for each state,  $a_i$ , defining a second constant conditional probability mass function  $b_i(e(t)) = P(E(t) = e(t) | A(t) = a_i)$ , that accounts for the probabilities of emitting each particular symbol. We can now represent the conditional emission probabilities in a emission matrix,  $B = (b_{ij})$ , the elements of which are defined as:  $b_{ij} = P(E(t) = e_j | A(t) = a_i)$ . A system modelled by a Hidden Markov Model with stationary transition and emission probabilities can then be represented by a transition matrix,  $T$ , an emission matrix  $B$  and an initial state probability mass function, which together constitute the model  $\bar{M}$ . NOTE the difference in notation. From here on the term model  $\bar{M}$  refers to the supersection of the type of model  $M$  and a set of parameters  $\bar{\theta}$ , the reason for this will be obvious when the notation of profileHMM is introduced.

There are three main questions that has to be addressed when using HMMs, as was pointed out in an excellent tutorial on HMMs by Rabiner and Juang [6]:

1. Calculating the likelihood of obtaining a particular finite sequence of  $n$  symbols,  $\bar{e} = (e(1), \dots, e(n))$  from a model  $\bar{M}$ ,  $P(\bar{e} | \bar{M})$ . Referred to as the scoring problem.
2. Finding the path (sequence of states) through a given model  $\bar{M}$  for a sequence of emitted symbols that is optimal according to some given criterium, and the likelihood of emitting the sequence along this path. Referred to as the alignment problem.
3. Building a model from the data. Referred to as the estimation problem.

### 2.3.3 The scoring problem

To calculate the likelihood of obtaining a particular finite fixed sequence of  $n$  symbols,  $\bar{e} = (e(1), \dots, e(n))$  from a given model  $\bar{M}$ ,  $P(\bar{e} | \bar{M})$ , the naive approach would be to sum over the entire set of state-sequences(paths)  $\bar{a}$  possible for the particular model:

$$P(\bar{e} | \bar{M}) = \sum_{\bar{a}} P(\bar{e} | \bar{M} \cap \bar{a})$$

If all the elements in the transition matrix are non-zero, the time-complexity of this calculation is  $O(n * m^n)$  where  $m$  is the number of states and  $n$  is the length of the sequence. This is clearly infeasible for all but the very shortest sequences. A much better algorithm is the *forward algorithm* according to which the likelihood of the sequence being emitted by the given model is recursively calculated. To understand this algorithm we first recall that for an ordinary Markov chain the probability of being in state  $a_i$  at time  $t$ , given a certain model,  $P(A(t) = a_i | M)$ , can according to the Markov condition, be recursively calculated as:

$$P(A(t) = a_i | M) = \sum_{j=1}^m (t_{ij} * P(A(t-1) = a_j | M))$$

This is of course valid also for hidden Markov models. What is interesting in this case is however not the states themselves, but in the symbols emitted by the system. We therefore introduce the forward variables  $f_i(t) = P(e(1), \dots, e(t) \cap (A(t) = a_i))$  which in analogy with the expression above can be recursively calculated simply by multiplying the conditional probability of emitting symbol  $e_j$  from state  $a_i$ , with the probability that the system makes a transition to state  $a_i$  having emitted symbols  $e(1), \dots, e(t)$ :

$$P(e(1), \dots, e(t) \cap A(t) = a_i | M) = b_i(e(t)) * \sum_{j=1}^m (t_{ij} * P(e(1), \dots, e(t-1) \cap A(t-1) = a_j | M))$$

Since the states of the system are mutually exclusive, i.e. the system can only be in one state at any timestep, we can now calculate  $P(\bar{e} | \bar{M})$  by summing  $f_i(n)$  over the  $m$  possible states of the system.

$$P(\bar{e} | \bar{M}) = \sum_{i=1}^m P(e(1), \dots, e(n) \cap A(n) = a_i | M) = \sum_{i=1}^m f_i(n)$$

It is trivial to show that the time-complexity of this algorithm is  $O(n * m^2)$  for a fully connected model and in general  $O(p * n * m)$ , where  $p$  denotes the connectivity of the model.

### 2.3.4 The alignment problem

To find an optimal path, an optimality criterion suitable for the problem in question must first be specified. For the purpose of biomolecular pattern analysis, the criterion to use is the path,  $\hat{a}$  that maximizes  $P(\bar{e} \cap \bar{a} | \bar{M})$ :

$$\hat{a} = \operatorname{argmax}_{\bar{a}} [P(\bar{e} \cap \bar{a} | \bar{M})]$$

It can be formally proven that the *Viterbi algorithm*, a dynamical programming algorithm, gives the path that is optimal in this sense. Let  $a_i(t)$  denote the subpath of the  $t$  first states ending in  $a(t) = a_i$ , represented as a vector, that is optimal in the same sense. In addition to just finding the optimal path we are often interested in

the conditional likelihood that the sequence was generated from the present model along the found path. We now introduce the Viterbi-variables  $v_i(t) = P(e(1) \cap \dots \cap e(t) \cap \widehat{a_i}(t) | \overline{M})$ . According to Bellmans theorem, we can calculate the  $v_i(t)$  and the  $\widehat{a_i}(t)$  recursively as:

$$v_i(t) = b_i(e(t)) * \max_j [t_{ij} * v_j(t-1)]$$

$$\widehat{a_i}(t) = (a_k(t-1), i), k = \operatorname{argmax}_j [t_{ij} * v_j(t-1)]$$

The optimal path,  $\widehat{a}$  can then be found as:

$$\widehat{a} = a_k(n), k = \operatorname{argmax}_i [v_i(n)]$$

and the probability of this path is simply:

$$P(\overline{e} \cap \widehat{a} | \overline{M}) = \max_j [v_j(n)]$$

Also in this case it is trivial to show that the time-complexity of this algorithm is  $O(n * m^2)$  for a fully connected model and in general  $O(p * n * m)$ , where p denotes the connectivity of the model.

### 2.3.5 The estimation problem

The problem of building an optimal model from data can be divided into two cases depending on whether the state sequences (paths) of the training set are known or not.

If the paths are known, the model can be estimated relatively easy. Let  $T_{ij}$  denote the number of transitions from state no. j to state no. i, and let  $E_{ki}$  denote the number of times letter no. k is emitted from state no. i. As it was shown above the maximum likelihood-estimates of the emission ( $\tilde{e}_{ki}$ )- and transition ( $\tilde{t}_{ij}$ )-parameters, are:

$$\tilde{t}_{ij} = \frac{T_{ij}}{\sum_l T_{lj}}$$

for the transition probabilities,  $t_{ij}$ , and:

$$\tilde{e}_{ki} = \frac{E_{ki}}{\sum_l E_{li}}$$

for the emission probabilities,  $e_{ki}$ . For small samples, however the variance of the maximum likelihood estimate is high, thus making these estimates poor. In such cases Bayesian estimation is a good alternative.

If on the other hand the state sequences are not known, then there is no known way of finding a model that is optimal in the maximum likelihood-sense analytically. Instead one has to rely on e.g. the Baum-Welch expectation maximization algorithm for doing recursive maximum likelihood-estimation. To derive this algorithm we need to introduce the backwards-variables,  $b_i(t)$ , defined as:

$$b_i(t) = P(e(t+1), \dots, e(n) | A(t) = a_i \cap \overline{M})$$

The backwards variables can be recursively calculated as:

$$b_i(t) = \sum_{l=1}^m [t_{il} * e_l(t+1) * b_l(t+1)], t \geq 1$$

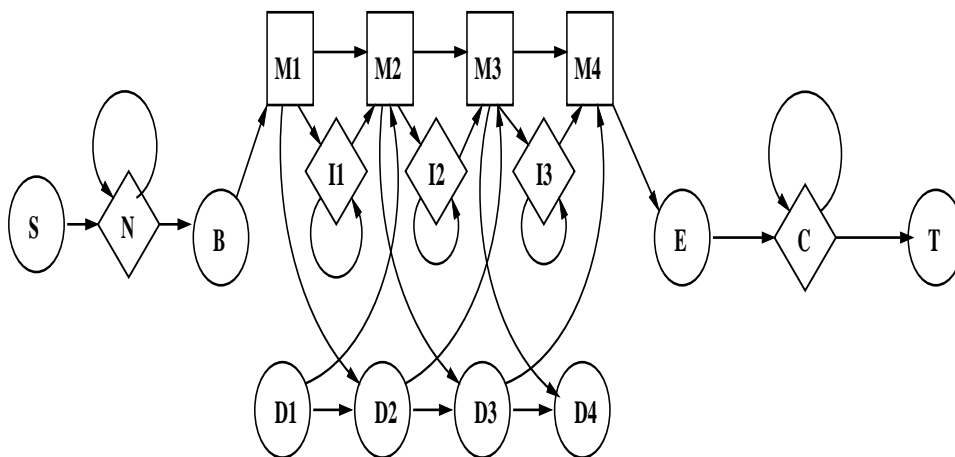
The problem of finding estimating a model can be thought of as a problem with hidden data. The Baum-Welch algorithm is guaranteed to converge to a local optimum, however there is no guarantee whatsoever that this will indeed be the global maximum. Since the number of local optima usually is very large, some sort of stochastic search algorithm is usually required to improve the performance of the optimization technique.

### 2.3.6 Numerical considerations

Direct implementations of the forward- or Viterbi-algorithms would suffer from immense numerical problems due to underflow. To illustrate this let's consider the case of calculating the total probability of traveling a certain path of length 150, not an unusual length of the aminoacid sequence of a protein, through a HMM, emitting a certain sequence of symbols. If we let  $\theta_{max}$  denote the largest of all the transition- and emission-probabilities, then the total probability will be  $\leq \theta_{max}^{300}$ , which could be a very small number indeed. The exponential decline of overall probabilities makes underflow almost inevitable on even the largest computers. To solve the problem with underflow, the algorithms must either be carried out in log-space, which is especially straight-forward for the Viterbi-algorithm since it only turns multiplications into additions, or some re-scaling method must be applied, as is described in [3]. Re-scaling usually works quite well, but if there are silent non-emitting states in the system, underflow might still occur. Because of these numerical considerations, the Viterbi-algorithm is usually faster than the forward algorithm, since the number of operations in each step is larger for the forward-algorithm. This is equivalent of saying that although the asymptotic time-complexity is the same for both algorithms, the constant of the leading term is larger for the forward-algorithm.

### 2.3.7 profileHMMs

To model evolutionary and structural similarities between proteins belonging to the same family, profileHMMs, a special class of HMMs with a very sparse transition-matrix, have been introduced by Haussler and Krogh et al [5]. In these models a temporal order, in signal processing often termed left-to-right order, is imposed, such as that only transitions to states with the same or higher indices are allowed. Three different kind of states are introduced, namely match-, insert- and delete-states, in the following discussion denoted M-, I- and D-states respectively. The sequence of match-states should correspond to the consensus sequence of the model. The insert-states should model insertions in a sequence in comparison to the model, whereas the delete-states should model deletions of consensus states in the particular sequence. Although it is tempting to view these states as a representation of the corresponding biological processes, it is probably safest to regard them as formal string operation on an imagined string corresponding to the consensus sequence of the protein family.

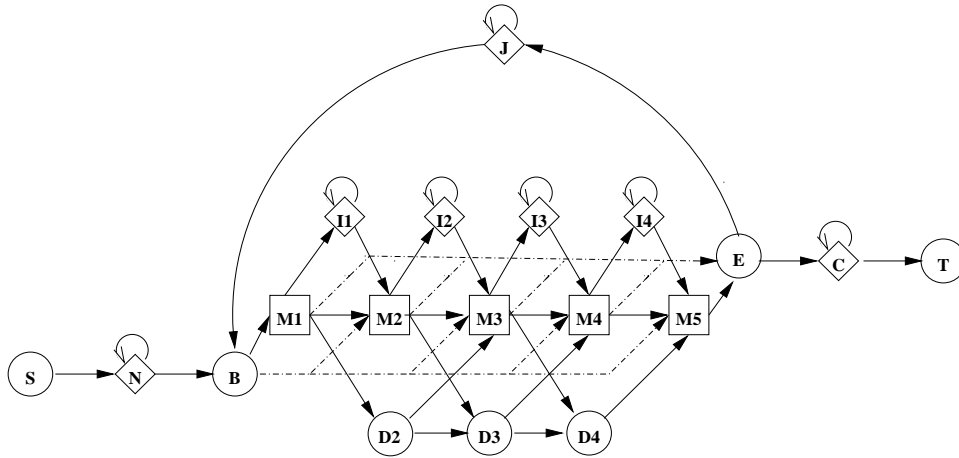


The parameters of a profileHMM are usually estimated from an trusted existing multiple alignment, although in principle profileHMMs can be build directly

from unaligned sequences, for instance by using the Baum-Welsh Expectation Maximization algorithm. The later approach is however a very difficult non-smooth optimization problem with a very large number of suboptimal alignments and has in practice not shown any significant improvement in alignment performance in comparison to classical fast heuristical multiple alignment methods (like Clustal W) while significantly increasing the time requirement. To overcome the problem with local optima stochastic optimization techniques like simulated annealing [3] or noise injection [5] have been tried.

### 2.3.8 The plan 7 architecture

In this work a modification of the usual Krogh-Haussler architecture, by Sean Eddy, has been used. This model, denoted plan 7, further restricts the number of possible transitions so that no delete-state to insert-state or insert-state to delete-state transitions are allowed.



To be able to do local alignment with respect to *the sequence* two flanking states, in this work denoted N and C, are introduced. Local alignment with respect to *the model* is made possible by allowing transitions from the N-state to M-states with higher indices than 1. By introducing constraints on the transition probabilities, i.e. by choosing which transition probabilities should be set to zero, when building the model from data, the alignment type can be chosen. The case of repeating domain proteins finally is treated by introducing a state joining the E(nd)-state with the B(egin)-state. The reduction of transition possibilities in the part of the model corresponding to the traditional Krogh-Haussler architecture generally does not have a serious effect on performance, but significantly reduces the number of free parameters in the model.

In general the notation of HMM states in this discussion follows that of the HMMer version 2.1 User's Guide.

## 2.4 Scoring and inference of alignments/searches

To be able to detect related sequences one must find a way to measure similarity between sequences, i.e. a measure to discriminate sequences that are related from those that are not. A number of crit

eria have been suggested as discrimination functions for scoring the significance of a match in biological sequence analysis. These include the log-likelihood of the data given the model, log-likelihood ratios, the Z-score and P- or E-values.

### 2.4.1 The log-likelihood score

The simplest criterion is to simply use the value of the logarithmized likelihood-function for a particular sequence calculated with the forward or Viterbi-algorithm. (When the Viterbi-algorithm is used, what is calculated is formally not the likelihood of the data, but rather the conditional likelihood given a certain state path.) However the average value of the log-likelihood(LL)-function (*LL-score*) for large sets random sequences is highly length-dependent. For short sequences this dependence can be highly non-linear, whereas it for long random sequences is almost linear. In addition the variance of the LL-score of random sequences increase as the sequence length increases. A more detailed discussion on these matters can be found in [?]. Because of this length-dependence present in the LL-score, it is not good enough to use as a discrimination function, since the LL-score for a long random sequence can be just as high as the LL-score for a short homologous sequence. Normalizing the LL-score over sequence length improves the situation but the discrimination is still not good enough to make the LL-score usefull as a discrimination tool.

### 2.4.2 Log-likelihood ratios

To deal with the problem of the lenght dependence present in the LL-score, the log-likelihood ratios frequently used as “distance” measures in statistics, as well as in genetics have been applied. In this case the logarithmized quotient between the likelihood of the sequence/alignment being generated by the model and the likelihood of the sequence/alignment being generated by a random(Null)-model is used as the discrimination function.

In the context of scoring similarites the log-likelihood ratios are usually refered to as *log-odds scores* [3].

### 2.4.3 The Z-score

Another approach to deal with the length dependence of the log-likelihood score is to, for each sequence length, or rather for each of a number of short sequence length intervall, estimate the mean and variance score for a large set of random sequences. The Z-score,  $Z$  is defined as the difference between the score of a sequence of a particular length and the corresponding estimated mean divided by the estimated standard deviation [3].

$$Z(i) = \frac{(LL - \mu_i)}{\sigma_i}$$

In other words the Z-score is the normalized statistical distance between a sequence and the average random sequence of the same length. It has been reported that using a log-odds score instead of the raw log-likelihood score, greatly reduces the noise level [3].

### 2.4.4 The P-value and the E-value

Although solving the problem connected with length-dependence to an acceptable degree, using log-odds scoring as a discrimination function in the case of profile HMMs suffers from one potential drawback. Although the log-odds score of a sequence is no longer strongly length dependent, the significance of a certain log-odds score can differ between different HMMs, i.e. the distribution of log-odds scores can differ from HMM to HMM.

The idea behind the P-value measure is that we treat the score, either log-likelihood or more preferably log-odds, of a random sequence,  $S$ , as a random

variable. If the distribution of the score variable  $S$  can be estimated either experimentally or analytically, the significance of a similarity can be measured as the probability of obtaining a larger score by chance. The formal definition of the P-value for a score of a sequence  $s$ ,  $Pvalue(s)$  is:

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

In the case of classical dynamical programming local pairwise non-gapped alignment it has been formally proven that the log-odds score is extreme value distributed [8]. For gapped pairwise alignments no formal proof exist, but the assumption seems reasonable. To my knowledge no analytical result regarding the distribution of log-odds ratios exist in the case of profile HMMs either, but also in this case the distribution of scores for real and artificial data seems to fit extreme value distributions relatively good.

Closely related to the P-value is the E-value, which is a function of the score  $s$  and the database size, that measures the expected number of sequences in a database of that size consisting of random sequences that would have a higher score. Thus in contrast to the P-value the E-value takes into account the size of the database searched.

### 3 Materials & Methods

In subsection 3.1 some of the keyterms used in this section are defined. Subsection 3.2 contains a description of the algorithms developed. The methods used to evaluate the performance of the different algorithms are discussed in subsection 3.3. Subsection 3.4 describes the databases from which our testsets were created. The criteria used by the methods discussed in subsection 3.3, as measures of fold/family recognition capability are defined in subsection 3.5. The testsets used and how they were created are described in subsection 3.6. In subsection 3.7 the computer software/hardware resources used for this work are described. The functionalities of the HMMER version 2.0 software package are briefly discussed in subsection 3.8.

#### 3.1 Definitions

In the methods and results-sections of this work the following definitions will be used. The term *testset* denotes the set of samples of sequences/multiple alignment and HMMs upon which the performance of a algorithm/model is evaluated, and *dataset* denotes the set of results for the algorithm/model on the corresponding testset. The set of *true* samples in a dataset is the set of samples that fullfills some given criterion. The set of *false* samples in the dataset is the set of samples that do not fullfill this criterion. All samples in the dataset scoring higher than a certain threshold in an alignment or search are termed *positive* hits, all other *negative* hits. According to this notation a true sample scoring less than the threshold is a true negative hit, whereas a false sample scoring less than threshold is termed false negative. This notation is to my knowledge the standard in computational biology, but *note* that this notation differs from that of mathematical statistics, where a sample fullfilling the desired condition and scoring less than threshold would be termed false negative (and vice versa) since it is falsely classified as not fullfilling the condition, whereas in our notation it is called true negative since it is a true hit, though scoring less than threshold.

#### 3.2 Algorithms

In the algorithms developed in this work, we generally do not allow through J-states, of course the algorithms could be generalized to handle transitions through J-states

as well, but ....

### 3.2.1 The optimal MA-HMM alignment

In the case of single sequence-HMM alignment, a unique path through the model can be found. Contrary to this case in a multiple alignment-HMM alignment all rows in each column of the multiple alignment can not generally be assigned to the same state, which means that a state distribution must be modelled. However we must make the assumption that all symbols in a column of the multiple alignment have been *emitted* from the same state in the HMM, i.e. all emitting sequences must have passed through that state. The state distributions are therefore denoted  $D_j(X_i)$ , where  $j$  is the column in the multiple alignment to which the state distribution is associated and  $X_i$  is the state, of type  $X \in \{M, I, N, C\}$ . indexed  $i$  in the model, from which the symbols are modelled to have been emitted. Because of this restriction, each column in the multiple alignment can be labeled by the state from which the symbols of that column has been generated. These labels are termed *superstates*. To allow modelling non-adjacent match-states in the hidden Markov model to generate the symbols of adjacent columns in the multiple alignment, the concept of superstates is generalized to include delete-superstates. by allowing modelling of imagined (non-visible) mute columns in multiple alignment between the adjacent columns. Delete-superstates are then defined as the state labels of these imagined mute columns. Thus the problem of aligning a multiple alignment with a HMM in some optimal sense can be recasted into finding an optimal sequence of superstates. Note however that the superstate of a column and the statedistribution of a column are not equivalent concepts, since for column labelled by I-, N- or C-superstates no state distribution can be defined, since the states of sequences containing gaps in those columns are undefineable.

We furthermore assume that the  $N$  sequences in the multiple alignment were generated by independent random walks through the model. Since the sequences can be weighted in order to compensate for evolutionary correlation, (which would violate the assumption of independence)  $N$  is not necessarily an integer. Let  $L_j$  be the number of symbols in the  $j$ :th column of the multiple alignment, and let  $\bar{L}_j = \{l_{qj} | q \in \mathbf{A}\}$  be a set over the number of times  $l_{qj}$  each symbol  $q$  belonging to the alphabet  $\mathbf{A}$  occurred in column  $j$ . According to the last assumption, the simultaneous probability of emitting the  $L_j$  symbols from a particular state  $X_i$ , in the model  $\bar{M}$ ,  $Prob(\bar{L}_j | X_i \cap L_j \cap \bar{M})$ , can be modeled by a generalized (remember that  $L_j$  might not be an integer) multinomial distribution as:

$$Prob(\bar{L}_j | X_i \cap L_j \cap \bar{M}) = L_j! * \prod_{q \in \mathbf{A}} \frac{(b_{X_i}(e_q))^{l_{qj}}}{l_{qj}!}$$

The generalization of the emission probabilities is thus fairly straight-forward. The problem is however to find generalizations of the state transitions, which in this case become state-distribution transitions. An obvious generalization of single sequence transition probabilities would then be to calculate the simultaneous probability of making the transitions implied by the assignment of the two adjacent columns in the multiple alignment. Under the assumption of independence between sequences, the probability of making a transition to a state distribution  $D_{j+1}$  assigned to column  $j + 1$  given the state distribution  $D_j$  assigned to column  $j$  can be calculated from a multinomial distribution by using the number of transitions of each kind between column  $j$  and column  $j + 1$  in the multiple alignment, given the assignments. Let be the set of possible transitions and  $\bar{n} = \{n_{kl}\}$  be a set of variabels denoting the number of times each possible transition is observed then the conditional probability of making the transition to state-distribution  $D_{j+1}$  from state-distribution  $D_j$  can

be calculated as:

$$Prob(D_{j+1}|D_j \cap \overline{M}) = N! * \prod_{n_{kl} \in \overline{n}} \frac{(t_{kl})^{n_{kl}}}{n_{kl}!}$$

However in the case when the symbols in one or both of the column are modelled to have been emitted by e.g. an insert-state (or equivalently by a N-state or a C-state), recursion breaks down. Of course, as mentioned above, no state distribution can be defined for an insert-state, this could however be circumvented by changing the definition of state distribution. The real problem is that whereas each match-state can only be modelled to have emitted the symbols of at most one column, each insert-state can be modelled to have generated the symbols of an arbitrarily number of columns.. Since the number of autorecursive self-loops can differ between the sequences, the states of gaps in column in which the symbols are modelled to have been emitted by insert-states can not be unambiguously defined, since they can either be due to sequences in which no insertions occurred, or due to sequences in which fewer insertion-loops occurred. This ambiguity can not be resolved by information from assignment of the previous column alone, since it requires knowledge of the assignment of the next column also, thus recursion breaks down. This implies that it is only for transitions between columns for which the symbols are modelled to have been generated by match-states that an unambiguous assignment of states to each sequence is possible, without knowing the assignment of the surrounding columns.

For such columns the likelihood of assigning the symbols of column  $j + a$  to match-state  $i + 1$  given that we have assigned the symbols of column  $j$  to match-state  $i$ , can still be calculated by calculating the the total probability of the transitions from the state-distribution assigned to column  $j$  to that assigned to column  $j + a$  according to the equation above, if we replace the transition probability for each sequence with total probability of the succession of transitions and emissions for the sequence, implied by the assignments. These successions of transitions and emissions are termed *tracks*. The columns must no longer be adjacent if the symbols of the columns in between are modelled to have been generated by the insert-state connecting the two adjacent match-states. If  $\tau_k$  denotes the probability of track  $k$ , where  $k$  is an index over the set of possible tracks from match-state  $i$  to match-state  $i + 1$ ,  $\Omega_i^{i+1}$ , and  $n_k$  is the number of times track  $k$  is chosen, in analogy with above, the total likelihood of modelling the symbols of column  $j + a$  in the multiple alignment to have been generated by match-state  $i + 1$  given that we have modelled the symbols of column  $j$  to have been generated by match-state  $i$  can be calculated as:

$$Prob(D_{j+a}(M_{i+1})|D_j(M_i)) = N! * \prod_{k \in \Omega_i^{i+1}} \frac{(\tau_k)^{n_k}}{n_k!}$$

Since we in practice use likelihood-ratios in stead of raw likelihoods and work in log-space, the prefactors in the multinomial distributions will cancel, since they will be the same for the random-model and the probabilities will be replaced with probability ratios. The log-likelihood ratio of labelling statevector indexed  $j + a$  by state  $X_{i+1}$  given that we have labelled statevector  $D_j$  by state  $X_i$ , given the model and the random model can thus be calculated as:

$$C_{X_i X_{i+1}}(j + a|j) = \log \left[ \frac{Prob(D_{j+a}(M_{i+1})|D_j(M_i))}{Prob(D_{j+a}(R)|D_j(R))} \right] = \sum_{k \in \Omega_i^{i+1}} [n_k * \log(\tau_k)]$$

This formula is valid for transitions to a match-superstate from the S-superstate and from a match-superstate to a T-superstate. Likewise, the simultaneous probability of emitting the symbols of column  $j$  from a given state  $X_i$  become a total emission

score  $Q_{X_i}(j)$  which can be calculated as:

$$Q_{X_i}(j) = \sum_{q \in \mathbf{A}} [l_{qj} * \log(b_{X_i}(e_q))]$$

However implementing equation is not the optimal way of computing the conditional log-odds of bl a bl a bl a. To make the results comparable for alignments of different size

To achieve this we introduce an algorithm reminiscent of the algorithm used to find an MAP-estimate of model length, described in the theory section.

In summary, the assumptions made in this algorithm are:

1. The sequences were generated by independent random walks through the Hidden Markov model.
2. The symbols of each column in the multiple alignment has been emitted from one state of the Markov model i.e. the same assumption as was made when building the profileHMM.

To derive this algorithm we introduce two sets of variables  $\{\widehat{M}_i(j)\}$  and  $\{\widehat{D}_i(j)\}$ , where  $\widehat{M}_i(j)$  is defined as is the maximal likelihood of the emissions in columns  $1, \dots, j$  given that the symbols of column  $j$  were emitted by the match-state,  $M_i$ , and  $\widehat{D}_i(j)$  is the maximal likelihood of the emissions in column  $1, \dots, j$  given that the match-state indexed  $i$  does not generate any symbols in the multiple alignment. (Equivalently match-state  $i$  generate a mute column in the multiple alignment.) To make the calculation in the algorithms complete we have to introduce a set of variables  $\{\widehat{T}_i(j)\}$ , defined as the maximal likelihood of the given that the symbols of column  $j$  were emitted by match-state  $i$  and that the symbols of the remaining columns (if any) are modelled to have been generated by the C-state. The simultaneous likelihood of modelling the symbols of the  $j$ :th column of the multiple alignment to have generated by the first match-state. Can be modelled analogously with the N-state instead of an I-state.

In log-space the can recursively be calculated as:

$$\begin{aligned} \widehat{M}_1(j) &= Q_{M_1}(j) + C_{SM_1}(j, 1) \\ \widehat{M}_i(j) &= Q_{M_i}(j) + \max \begin{cases} \max_k [\widehat{M}_{i-1}(k) + C_{M_{i-1}M_i}(j, k)], 1 < k < j \\ \widehat{D}_{i-1}(j-1) + C_{D_{i-1}M_i}(j, j-1) \\ C_{SM_i}(j, 1) \end{cases} \quad j > 1 \\ \widehat{D}_i(j) &= \max \begin{cases} \widehat{D}_{i-1}(j) + C_{D_{i-1}D_i}(j, j-1) \\ \widehat{M}_{i-1}(j) + C_{M_{i-1}D_i}(j, j-1) \end{cases} \quad 1 < j < N \\ \widehat{T}_i(j) &= \widehat{M}_i(j) + C_{M_iT}(j) \end{aligned}$$

In these equations the  $C$ :s denotes the total log-likelihood ratios of the paths between the state distributions, where the subscripts are the kind of the start-and end -state distributions of the paths, the superscript denotes the index in the model, and the argument denotes the column in the multiple alignment.  $Q_i(j)$  is the total likelihood of emitting the symbols of the  $j$ :th column of the multiple alignment from the  $i$ :th match-state in the model. According to the discussion above the various transition- and emission-likelihoods are: Since we must do a recursion for each match-state back to the previous match-states,

### 3.2.2 Reference sequence search

In the reference sequence search approach, one of the sequences in the multiple alignment (by default the longest) is selected. All the columns in which this sequence contains a symbol is then marked. The idea behind the algorithm is to align this sequence with the hidden Markov model, but in contrast to ordinary alignment of a single sequence with a HMM, to use multiple sequence information obtained from a multiple alignment of the sequence in question to compute the emission scores. If the columns are kept fixed aligning the longest sequence of the multiple alignment with the hidden Markov model is equivalent of aligning the multiple alignment with the HMM.

In the following discussion the term score will be used to denote the log-likelihood ratio. To incorporate the multiple sequence information the same assumptions as in subsection 3.2.1 are made. Under these assumptions all the symbols in each of the marked columns of the multiple alignment are modelled to have been independently generated by the same state as generated the symbol in the marked sequence. The symbols of each column of the multiple alignment can thus be viewed as a sample from the conditional emission probability mass function of a particular state in the hidden Markov model.

To incorporate the multiple sequence information in the alignment of the selected sequence, the score of emitting the symbol of the marked sequence in column  $j$  of the multiple alignment from state  $X_i$ , (where as in the previous sub-subsection  $X$  denotes the type of the state and  $i$  is the index in the model) is replaced by an estimate of the expectation value of the emission score of state  $X_i$ , estimated from the symbols in the  $j$ :th column of the multiple alignment. Let  $p'_r(j)$  be the estimated probability of seeing symbol  $r$  in column  $j$  of the multiple alignment. We can then calculate the estimated expectation value of the score  $E_j^*(s(X_i))$  by summing over the alphabet  $\mathbf{A}$

$$E_j^*(s(X_i)) = \sum_{r \in \mathbf{A}} [p'_r(j) * s_r(X_i)]$$

Since the score is a log-likelihood ratio  $s_r(X_i) = \frac{p_r(X_i)}{q_r}$ , where  $p_r$  is the conditional probability estimate of emitting symbol  $r$  from state  $X_i$  in the model estimated from the data used to build the HMM given the model and  $q_r$  is the corresponding probability estimate for the random model, this measure is a relative entropy measure.

$$E_j^*(s(X_i)) = \sum_{r \in \mathbf{A}} [p'_r(j) * \frac{p_r(X_i)}{q_r}]$$

To estimate the set of conditional emission probabilities for each column in the multiple alignment a straight-forward approach would be to simply do maximum likelihood-estimation. As was proved in the theory-section the ML-estimates of the probabilities of a set of exclusive and exhaustive outcomes of a random event are the relative frequencies by which they occur in the data. The problem connected with this estimation is that in many cases there are gaps present in a column, which means that the samples in those cases are incomplete since data are missing. Another problem is of course that if the alignment contains few sequences the emission probabilities will be estimated to zero for many columns. To address the first problem a simple approach is to simply make ML-estimates from the data available in each column regardless of how many symbols that actually are present in each column, that is we simply ignore the gaps. The problem connected with this approach is that if one ..... This latter approach has no direct probabilistic interpretation, since the sum of the probabilities for the different symbols in the alphabet might not sum to one, if gaps are present in a column.

### 3.2.3 "In flight" estimation of state distribution

This algorithm is an attempt to partly solve the problem connected with the increase in time-complexity of the algorithm in subsection 3.2.1 by simply ignoring the fact that the recursion breaks down for states for which autorecursive transitions are allowed, and thus approximate the state distributions "in flight". The formulas for the different transition scores will under the same assumptions made in subsection 3.2.1 be:

## 3.3 Methods used for dataset analysis

In the litterature a large number of methods have been suggested to evaluate the performance of algorithms and models on the dataset created using the algorithm/model in question from some testset. In this work two different methods namely Spec-Sens-plots and TopN-hit-analysis have been used. They differ in the respect that Spec-Sens-plots measure the overall performance of an algorithm/method whereas the TopN-value only measures the performance for the top-ranking hits. The evaluation was in both cases based on runs of all sequences against all Hidden Markov Models in each testset, in this text called all-to-all runs .

### 3.3.1 Spec-Sens plots

In statistical terms the *sensitivity* is the probability that a statistical test will yield a positive result for a true statistic, and the *specificity* the probability that the test will yield a negative result for false statistic. These probabilities are of course dependent on the threshold used to determine which values of the test statistic that are positive. As was shown in the theory-section the ML-estimates of these probabilities are the relative fractions by which they occur in the data, i.e. if we denote the number of true positives for a given threshold  $TP(\text{threshold})$ , the number of false positives,  $FP(\text{threshold})$ , and so on, the ML-estimates of the sensitivity, *sens*, and the ML-estimates of the specificity, *spec* are:

$$sens(\text{threshold}) = \frac{TP(\text{threshold})}{TP(\text{threshold}) + FN(\text{threshold})}$$

$$spec(\text{threshold}) = \frac{TN(\text{threshold})}{TN(\text{threshold}) + FP(\text{threshold})}$$

A spec-sens plot is a plot of the trajectory in the specificity-sensitivity plane of the estimated (specificity, sensitivity) pair as a function of threshold. As threshold or cut-off, any of the inference criteria discussed in subsection 2.4 of the theory section can be used. The ranking of the algorithms/models is then based on the sensitivities of each algorithm/model in the specificity-intervall of interest. The best model is defined to be the model that for each specificity-level in the region of interest has the highest sensitivity. This definition of spec-sens plots follows that of Rice and Eisenberg, [12]

### 3.3.2 TopN hits

As an alternative measure of performance the TopN-values are hereby introduced, defined as the sum over all HMMs in the testset of the number of true hits found among the first n hits for each HMM, when doing an all-to-all run. Though clearly sub-optimal in the sense that no attention is paid to the significance of each hit, this has some benefits, see the discussion in the results-section.

## 3.4 Databases

### 3.4.1 The Scop database

The Scop (Structural Classification Of Proteins) database is a manually constructed database aimed to provide a comprehensive description of the structural and evolutionary relationships of proteins with known 3D-structure. For a closer description of the Scop database, see Murzin et al [?, ?, ?]. The Scop database consist of 4 hierarchial levels, class, fold, superfamily and family, represented as numbers. This means that in this database a set of 4 descriptors along with the name of the protein uniquely identifies a certain protein.

The pdb40 subset of Scop version 1.37 is a subset of Scop 1.37, in which no proteins have more than 40 % sequence identity to any other member of the subset [?, ?].

### 3.4.2 The HSSP database

The HSSP (Homology-derived Secondary Structure of Proteins) database[14], is a database of proteins with known structure to which all clearly homologous proteins has been aligned, developed by Chris Sander and Reinhard Schneider at EMBL Heidelberg. Tertiary structures of the aligned sequences are implied but not explicitly modelled. For each protein of known structure, a multiple alignment with related sequences, secondary structure information, sequence variability and a sequence profile are available.

## 3.5 Definition of true hits

To be able to do a spec-sens analysis we must calculate the total number of true hits a priori, since we need the total number of true hits to calculate the specificity (TP+TN=number of true hits), and since the true negative hits are not detectable as they fall below the cut-off. In order to calculate the number of true hits the criterion of a true hit must first be specified. The criterion will off course depend on what the interesting property is, in this case the ability of family or fold recognition. In

### 3.5.1 Family recognition

In the case of family recognition we define, according to the Scop classification procedure, a *family* to be the subset of sequeces/multiple alignments in the testset used, having identical 4-tuples of Scop descriptors. To make the test of family recognition capability more challengeing, we define the set of *true* hits in family recognition to be the set of ordered pairs of (sequences/multiple alignments, HMMs) in the dataset whose corresponding sequences/multiple alignments and HMMs in the testset *belongs to the same family but do not have the same name*, i.e. we ignore the obvious self-hits.

According to this criterion we can calculate the total number of true hits,  $_{Fam}T$ , to be the sum of the number of true hits corresponding to each family,  $T_i, i \in \{1, \dots, K\}$ , where  $K$  denotes the number of families.

$$_{Fam}T = \sum_{i=1}^K T_i$$

For each family each sequence/multiple alignment will form a true hit with each hidden Markov model except the one corresponding to itself. If we let  $n_i$  denote the

number of sequences/multiple alignments belonging to a certain family, then  $T_i$  can be calculated as:

$$T_i = n_i * (n_i - 1)$$

The total number of true hits can thus be calculated as:

$$_{Fam}T = \sum_{i=1}^K n_i * (n_i - 1)$$

### 3.5.2 Fold recognition

For fold recognition purposes we define a *fold* to be the subset of the testset used, that have the same initial 2-tuple of Scop-descriptors (i.e. class-fold). In this case, the set of *true* hits is defined to be the set of ordered pairs of (sequences/multiple alignments, HMMs) in the dataset for which the corresponding sequences/multiple alignments and HMMs *have the same fold, but do not belong to the same family*.

To calculate the number of true hits let  $_{Fold}T$  be the total number of true hits, and let  $T_j$  be the number of true hits corresponding to fold number  $j, j \in \{1, \dots, L\}$ . Let  $m_j$  be the number of sequences/multiple alignments belonging to foldtyp indexed  $j$ , and let  $n_i$  be defined as above. Let  $D_j$  be the set of families compatible with a certain fold. For each family belonging to a certain foldclass, the number of true hits will then be the number of sequences/multiple alignments with that fold minus the number of sequences/multiple alignments belonging to the same family, i.e.

$$T_j = [\sum_{i \in D_j} n_i * (m_j - n_i)] = m_j^2 - \sum_{i \in D_j} n_i^2$$

Summing over  $j$  gives:

$$_{Fold}T = \sum_{j=1}^L [m_j^2 - \sum_{i \in D_j} n_i^2]$$

## 3.6 Testsets

A total of three testsets were used in this work.

The first and biggest was created by doing an intersection between the pdb40 subset of the SCOP-database version 1.37 and the HSSP-database (release date September 8, 1997) yielding 1130 different sequences/multiple alignments, in the following discussion denoted TestSetI. The names of the sequences/multiple alignments contained in this testset are listed in appendix 1.

Because of the size of this testset an all-to-all run using multiple alignment-HMM alignment is quite time-consuming. Due to this circumstance two smaller testsets are generated from the original testset. The first subset denoted TestSetIII consisted of 379 sequences/multiple alignments chosen randomly from fold categories of the original testset having at least 5 members, these are listed in appendix 2. The reason for making this restriction was that we wanted to preserve the maximum amount of fold/family recognition potential simultaneously as we wanted to choose a testset as small as possible. Although this might bias the result, it is the fold categories with many members that will really have a large impact on performance since the dependence is quadratic in terms of size of fold-class.

Since the fold-class 3-1 (according to Scop-notation) which had 30 members in TestSetIII distributed over  $x$  families, was responsible for nearly one third of the number of true hits in fold recognition using TestSetIII, any difference in the performance between different algorithms for this fold category might have an enormous impact on the result of an all-to-all run. Therefore a subsection of TestSetIII in which the members of this fold category were deleted, named TestSetV was also used.

### 3.7 Source codes distributions and computer software/hardware.

All of this work was carried out using modifications of various versions of the HMMer HMM software package.

In an early study the default performance of version 1.8.4 was compared with the default performance of version 2.0. All further studies were carried out by making various extensions and modifications of version 2.0, except for the last part, in which a comparison was made with the performance of a MA-HMM alignment algorithm developed by Sean Eddy and included in version 2.1 of the HMMER package. In this study a slightly modified version 2.1 was used.

The HMMer package has been developed by Sean Eddy, now at Washington University, Saint Louis, earlier at the Sanger Research Center, Cambridge, UK, and is implemented in C. The various versions of the HMMer package have been obtained as source code distributions, from a ftp-server at Washington University, which can be reached through the WWW-page (<http://hmmerr.wustl.edu>). The modifications of the source code have been carried out using GNU xemacs-editors.

For the compilation of both original and modified source code, the GNU g++ C-compiler, has been used in most cases, whereas in some cases the system-supplied cc C-compiler have been used. Most of the computation has been carried out using various PCs running Red Hat Linux version 4.2, though some calculations were performed on Dec Alpha workstations. The plots have been generated using xmgr, a plotting program running under X-windows.

### 3.8 The HMMER 2.x package

This description is valid for HMMER 2.0. For a description on HMMER version 2.0 contains the following main functionalities:

- **hmmalign** - A program that aligns single sequences to a given hidden Markov model. If several sequences are sent as an input, then the result is a multiple alignment. Any previous alignment of the sequences is overridden.
- **hmmbuild** - A program that builds hidden Markov models from an existing multiple alignment. Prior information can be incorporated either by means of Dirichlet mixtures or by using heuristic pseudocounts-schemes based on substitution matrices.
- **hmmcalibrate** - A program that generates a large number of random sequences and from those fits the EVD-parameters of the model.
- **hmmconvert** - A program that converts HMM-save files. The HMMER version 1.x files are binary files, whereas version 2.0/2.1 save-files are ASCII-files. The hmmconvert-program converts binary files to ASCII files and vice versa.
- **hmmemit** - A program that generates sequences by simulated random walks through the model.
- **hmmpfam** - A program that enables the user to search a single sequence against a whole database of hidden Markov models. Either the Viterbi- or the forward-algorithm can be used. Default is the Viterbi-algorithm.
- **hmmsearch** - A program that enables the user to search a single sequence against a single hidden Markov model. Also in this case this could be done using either the Viterbi- or the forward-algorithm can be used. Default is Viterbi.

This is only a very brief description of the functionalities in the HMMER package for more detailed information see the HMMER version 2.1 user's guide, which can be retrieved from <http://hmmer.wustl.edu/hmmer.html>. In addition to these programmes, the plan was to add the following:

- **hmmsearch** - A program that would enable the user to search a hidden Markov model with multiple sequence information, choosing between two different algorithms.

## 4 Results

Initially a comparison was made between the default performances of the 1.8.4 and 2.0 versions of the HMMER-package. As was expected according to the online user's manual of HMMER version 2.1 (<http://hmmer.wustl.edu/hmmer.html>), the default performance of the 2.0 version, clearly outperformed the 1.8.4 version (data not shown).

Initially spec-sens plots were drawn with log-likelihood cutoff as independent parameter, but a check showed that this is clearly inferior to doing spec-sens plots with  $\lambda$ .

## 5 Discussion

### References

- [1] Altschul S. F., Gish W., Miller W., Myers E. W. and Lipman D. J. 1990. Basic local alignment search tool. *Journal of Molecular Biology* 215:403-410.
- [1] Baldi P. and Brunak S. 1998. *Bioinformatics -The machine learning approach*. MIT Press.
- [2] Baldi P. and Chauvin Y. 1994. Hidden Markov models of the G-coupled receptor family. *J. Comput. Biol.* 1:311-335.
- [2] Brenner S. Chothia C. and Hubbard T. 1998. Assessing sequence comparison methods with reliable structurally identified evolutionary relationships. *Proc. Natl. Acad. Sci. USA* 95(11):6073-6078.
- [2] Casella G. and Berger R.L. 1990. *Statistical Inference*. Duxbury Press.
- [3] Dempster A.P., Laird N.M. and Rubin D.B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statist. Soc. B* 39:1-22.
- [3] Durbin R., Eddy S., Krogh A. and Mitchison G. 1998. *Biological sequence analysis*. Cambridge University Press.
- [3] Eddy S.R. 1995. Multiple Alignment using hidden Markov models. *Proc. Third Int. Conf. Intelligent Systems for Molecular Biology*, C. Rawlings et al., eds. AAAI Press, Menlo Park. pp. 114-120.
- [4] Eddy S.R. 1996. Hidden Markov models(review). *Current Opinion in Structural Biology*, 6:361-365.
- [5] Eddy S.R. 1998. Profile hidden Markov models (review). *Bioinformatics* (in press).
- [6] Fischer D. and Eisenberg D. 1996 *Protein Sciences* 5:947-955.

- [4] Gray R.M. 1990. Entropy and Information Theory. Springer Verlag: New York.
- [7] Hargbo J. and Elofsson A. 1998 A study of hidden Markov models that use predicted secondary structures for fold recognition. submitted.
- [8] Karlin S. and Altschul S. F. 1990. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. Proc. natl. Acad. Sci. USA 87:2264-2268.
- [5] Krogh A., Brown M., Mian I. S., Sjoelander K and Haussler D. 1994. Hidden Markov Models in Computational Biology: Applications to Protein Modelling. Journal of Molecular Biology 235:1501-1531.
- [9] Murzin A. G., Brenner S. E., Hubbard T. and Chothia C. 1995. Scop: a structural classification of proteins database for the investigation of sequences and structures. Journal of Molecular Biology 247:536-540.
- [10] Needleman S. B. and Wunsch C. D. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology 48:443-453.
- [11] Pearson W. R. and Lipman D. J. 1988. Improved tools for biological sequence comparison. Proceedings of the National Academy of Sciences of the USA 4:2444-2448.
- [6] Rabiner L.R and Juang B. H. 1986. An introduction to Hidden Markov Models. IEEE ASSP Magazine 3:4-16.
- [12] Rice D. and Eisenberg D. 1997. A 3D-1D substitution matrix for protein fold recognition that includes predicted secondary structure of the sequence. Journal of Molecular Biology 267:1026-1038.
- [13] Rost B. and Sander C. 1993. Improved prediction of protein secondary structure by use of sequence profiles and neural networks. Proceedings of the National Academy of Sciences of the USA 90:7558-7562.
- [14] Sander C. and Schneider R. 1991. Database of homology derived protein structures and the structural meaning of sequence alignment. Proteins 9(9):56-88.
- [15] Smith T. F. and Waterman M. S. 1981. Identification of common molecular subsequences. Journal of Molecular Biology 147:195-197.