# Protein Modeling using Hidden Markov Models: Analysis of Globins

David Haussler†, Anders Krogh†, I. Saira Mian§, Kimmen Sjölander†

† Computer and Information Sciences
§ Sinsheimer Laboratories
University of California, Santa Cruz, CA 95064, USA.

haussler@cse.ucsc.edu, krogh@cse.ucsc.edu

UCSC-CRL-92-23
June 1992, revised Sept. 1992

## Abstract

*We apply Hidden Markov Models (HMMs) to the problem of statistical modeling and multiple sequence alignment of protein families. A variant of the Expectation Maximization (EM) algorithm known as the Viterbi algorithm is used to obtain the statistical model from the unaligned sequences. In a detailed series of experiments, we have taken 400 unaligned globin sequences, and produced a statistical model entirely automatically from the primary (unaligned) sequences. We use no prior knowledge of globin structure. Using this model, we obtained a multiple alignment of the 400 sequences and 225 other globin sequences that agrees almost perfectly with a structural alignment by Bashford et al. This model can also discriminate all these 625 globins from nonglobin protein sequences with greater than 99% accuracy, and can thus be used for database searches.*

## 1 Introduction

Predicting the three-dimensional structure of a biological macromolecule purely from primary sequence data is still a largely unsolved problem. The rate of generation of sequence data far exceeds that at which structures are being determined. Fortunately, an accurate alignment of two or more sequences can provide a wealth of information to guide further experimentation, particularly if one of the aligned molecules has been well characterized biochemically or structurally. Furthermore, multiple sequence alignments are used to infer the evolutionary relationships between sets of protein (or nucleic acid) sequences. In both cases, any judicious inference from the alignment is critically dependent on the accuracy of that alignment.

Consider a family of protein sequences that all have a common three-dimensional structure, for example, the family of globins (proteins involved in the storage and transportation of oxygen). The common or core structure in these sequences can be defined as a sequence of positions in space where amino acids occur. In the case of globins, whose structure contains principally alpha helices, the 150 or so residues in a helical conformation have been named A1, A2, ..., A16, B1, ... etc., where the letter denotes which alpha helix the residue occurs in, and the number indicates the location within that helix (see e.g. [1]). For each of these positions there is a (distinct) probability distribution over the 20 amino acids that measures how likely it is that each amino acid will occur in that position in a typical globin, as well as the probability that there is no amino acid in that position. These have been called *profiles* [2, 3, 4]. A profile of globins can be thought of as a statistical model for the family of globins, in that for any sequence of amino acids, it defines a probability for that sequence, such that globin sequences tend to have much higher probabilities than non-globin sequences.

In this paper we define a class of structures related to profiles and propose them as statistical models of protein families such as globins. Similar models can also be applied to other, non-protein sequence fam-

ilies, such as DNA [5, 6]. The novel aspect of this work is that we are able to "learn" or "estimate" the stochastic model directly from raw, unaligned sequences, rather than starting with a multiple alignment of the sequences. Once we have built a statistical model for a family of sequences, we can use this model to discriminate other sequences in this family from sequences not in the family. We can also use the model to obtain a multiple alignment of all of the sequences in the family, by aligning each of the sequences to the model, rather than trying to align them to each other (see figures 2 and 3). Finally, we can study the model we have found directly, and see what it reveals about the common structure underlying the various sequences in the family.

We demonstrate the general technique on a set of 625 globin sequences from the SWISS-PROT database, release 19 [7]. By building a statistical model of globins based on 400 globin sequences selected at random, we are able to discriminate with very high accuracy between the remaining 225 globins and the 19,458 non-globins in the database. Specifically, with one probability cutoff, we can produce a model that misclassifies only 2 out of 225 globins as non-globins (and none of the "training set" of 400 globins), and misclassifies only 10 out of 19,458 non-globins as globins. With a more conservative cutoff it misclassifies 8 out of 225 globins and 0 out of 19,458 non-globins (3 globins in the training set of 400 are also misclassified in this case).

We also obtain a multiple sequence alignment for all of the 625 globin sequences that basically reproduces the alignment given for the seven globins in Bashford et al. [1], even though their method employed additional information on the three-dimensional structure of globins, whereas we have only used the primary sequence information. Our method of multiple alignment is quite different from conventional methods, which are usually based on pairwise alignments using the standard dynamic programming scheme with gap penalties (see e.g. [8]). The alignments produced by conventional methods often depend strongly on the particular values of the parameters required by the method, in particular the gap penalties [9]. Furthermore, a given set of sequences is likely to possess both fairly conserved regions and highly variable regions, yet in aligning them with the conventional global methods, the same penalties are used for all regions of the sequences. Substitutions, insertions, or deletions in a region of high conservation should ideally be penalized more than in a variable region, and some kinds of substitutions should be penalized differently in one position than in another. That is one of the motivations for the present work. The statistical model we propose corresponds to multiple alignment with variable penalties, depending on the position. Furthermore, the penalties we use are in large part learned from the data itself. Essentially, we build a statistical model during the process of multiple alignment, rather than leaving this as a separate task to be done after the alignment is completed. We believe the model should guide the alignment as much as the alignment determines the model.

The type of statistical model we propose is a *hidden Markov model* (HMM, or simply "model" for short). The HMM we build identifies a set of positions that describe the (more-or-less) conserved first-order structure in the sequences. In biological terms, this corresponds to identifying the core elements of homologous molecules. Each of these positions may be viewed as corresponding to a column in a multiple alignment of the sequences, or to a position in space, as described above. Often not all positions are modeled, but only the ones with the most significantly nonrandom statistics. As in a profile, for each position in the model, the relative probabilities of the different amino acids in that position are given, as is the probability that the amino acid at that position is deleted (i.e., missing in the sequence). For any amino acid $x$, the negative logarithm of the probability of $x$ at a given position in the model can be interpreted as a penalty for an alignment that puts $x$ in this position, and the negative logarithm of the probability of deletion at that position can be similarly interpreted as a penalty for an alignment that puts a "-" (gap character) in that position. Furthermore, for each pair of adjacent positions, the model includes the probability for initiating an insertion of one or more amino acids between these positions, and the probability for extending it. Other things are also included, such as the probability that an amino acid at a particular position is deleted, given that the amino acid at the previous position was deleted. This helps model the higher probability of consecutive deletions/insertions in certain regions. By taking negative logarithms as above, these methods provide a very flexible, position-dependent form of initiation and extension gap penalties.

Once the hidden Markov model is learned, a multiple alignment of the sequences is easily produced from it by aligning each sequence to the model separately using a dynamic programming method (the Viterbi algorithm). The process is similar to aligning a sequence to a profile or average sequence. To search a database for similar sequences, simply align all the sequences

to the model, and see how they score (the score is the probability of the sequence given that model).

The algorithm we use to estimate the model is an approximation of the Baum-Welch or more general EM (expectation maximization) method[1] [11]. The basic idea is to start with some initial model, possibly randomly constructed, align all the sequences to this model, and then reestimate the probabilities in the model based on how the sequences align to it. This process of alignment to the model and then reestimation of the parameters of the model continues until there are no further changes to the model. From a statistical perspective, the procedure can be viewed as a computationally efficient approximation to maximum likelihood estimation, or, if prior probabilities for the parameters of the model are incorporated, as a maximum *a posteriori* or Bayesian approach. A related EM procedure is applied in [5, 6] to model protein binding sites, and an explicit 2-state HMM model is constructed in [12] to distinguish coding from noncoding regions in DNA. Here, in an attempt to model entire proteins, we explore a significantly richer class of models than these. Independent work, closely related to ours, is also given in [13], where hidden Markov models for protein superfamilies are constructed. The models used there are smaller than ours, but have a more varied structure.

## 2 The protein HMM

A hidden Markov Model describes a series of observations by a "hidden" stochastic process – a Markov process; for an excellent review, see [10]. In speech recognition, where HMMs have been used extensively, the observations are sounds forming a word, and a model is one that by its "hidden" random process generates these sounds with high probability. Every possible sound sequence can be generated by the model with some probability, and thus the model defines a probability distribution over possible sound sequences. A good word model would assign high probability to all sound sequences that are likely utterances of the word it models, and low probability to any other sequence. In this paper we are proposing an HMM similar to the ones used in speech to model *protein families* – or families of other molecular sequences like DNA and RNA. When modeling proteins, the observations are the amino acids in the primary sequence of a protein (corresponding to sounds in a word). A model for a set of proteins is one that assigns high probability to the sequences in that particular set. In the following we will describe the model using the 20 amino acids as the underlying alphabet, but we stress again that any other alphabet reflecting primary sequence or any derived property can be used.

The HMM contains a sequence of $M$ states that we call match states. These correspond to positions in a protein or columns in a multiple alignment. Each of these states can generate a letter $x$ from the 20-letter amino acid alphabet according to a distribution $\mathcal{P}(x|\mathtt{m}_k)$, $k = 1 \ldots M$. The notation $\mathcal{P}(x|\mathtt{m}_k)$ means that each of the match states $\mathtt{m}_k$, $1 \leq k \leq M$, have different distributions. These match states form the main line in the model and are shown as boxes in figure 1. For each match state $\mathtt{m}_k$ there is a delete state $\mathtt{d}_k$ that does not produce any amino acid, it is a "dummy" state used to skip $\mathtt{m}_k$. Delete states are shown as circles in figure 1. Finally, there are insertion states between all the match states, $M + 1$ in total, shown as diamonds in figure 1. They generate amino acids in exactly the same way as the match states, but they use probability distributions $\mathcal{P}(x|\mathtt{i}_k)$.

From each state, there are three possible transitions to other states, also shown in figure 1. Transitions into match or delete states always move forward in the model, whereas transitions into insertion states do not. Note that multiple insertions between match states can occur because of the self loop on the insert state, meaning that a transition from an insert state to itself is possible. The transition probability from state $s_1$ to $s_2$ is called $\mathcal{T}(s_2|s_1)$. For convenience we have added a dummy begin state and a dummy end state, which are denoted $\mathtt{m}_0$ and $\mathtt{m}_{M+1}$, respectively. These states do not produce any amino acid. A sequence can be generated by a "random walk" through the model as follows: beginning at state $\mathtt{m}_0$ (BEGIN) choose a transition to $\mathtt{m}_1$, $\mathtt{d}_1$, or $\mathtt{i}_0$ randomly according to the probabilities $\mathcal{T}(\mathtt{m}_1|\mathtt{m}_0)$, $\mathcal{T}(\mathtt{d}_1|\mathtt{m}_0)$, and $\mathcal{T}(\mathtt{i}_0|\mathtt{m}_0)$. If $\mathtt{m}_1$ is chosen, generate the first amino acid $x_1$ from the probability distribution $\mathcal{P}(x|\mathtt{m}_1)$, and choose a transition to the next state according to probabilities $\mathcal{T}(\cdot|\mathtt{m}_1)$, where "$\cdot$" indicates any possible next state. If this next state is the insert state $\mathtt{i}_1$, then generate amino acid $x_2$ from $\mathcal{P}(x|\mathtt{i}_1)$ and select the next state from $\mathcal{T}(\cdot|\mathtt{i}_1)$. If delete ($\mathtt{d}_2$) is chosen next, generate no amino acid, and choose the next state from $\mathcal{T}(\cdot|\mathtt{d}_2)$. Continue in this manner all the way to the END state, generating a sequence of amino acids $x_1, x_2 \ldots x_L$ by following a *path* of states $y_0, y_1 \ldots y_N, y_{N+1}$ through the model, where $y_0 = \mathtt{m}_0$ (the BEGIN state) and $y_{N+1} = \mathtt{m}_{M+1}$ (the END state). Because the delete states do not

[1] Sometimes the algorithm is called the Viterbi algorithm, but following [10] we reserve that name for the estimation part of the algorithm.

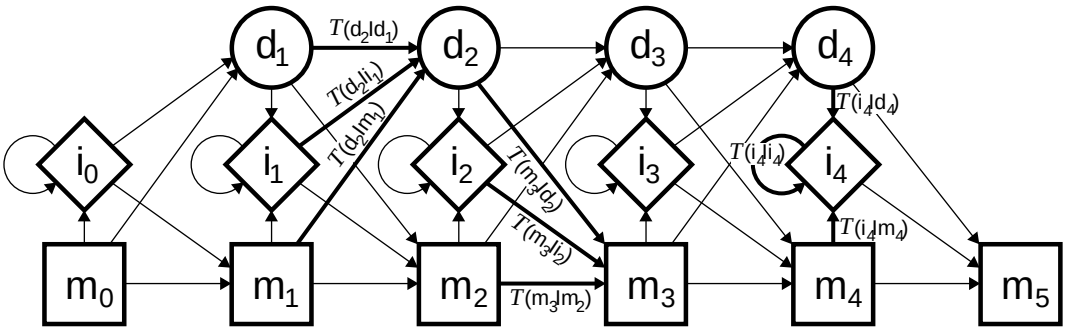



Figure 1: The model. The "begin" and "end" states are $m_0$ and $m_5$.

produce any amino acid, $N$ is larger than or equal to $L$. If $y_i$ is a match or insert state, we define $l(i)$ to be the index in the sequence $x_1 \ldots x_L$ of the amino acid produced in state $y_i$. The probability of the event that the path $y_0 \ldots y_{N+1}$ is taken and the sequence $x_1 \ldots x_L$ is generated is

$$\begin{aligned}&\text{Prob}(x_1 \ldots x_L, y_0 \ldots y_{N+1}) \\ &\quad = \mathcal{T}(\mathtt{m}_{N+1}|y_N) \times \prod_{i=1}^{N} \mathcal{T}(y_i|y_{i-1}) \mathcal{P}(x_{l(i)}|y_i), \quad (1)\end{aligned}$$

where we set $\mathcal{P}(x_{l(i)}|y_i) = 1$ if $y_i$ is a delete state. The probability of any sequence $x_1 \ldots x_L$ of amino acids is a sum over all possible paths that could produce that sequence, which we write as follows:

$$\begin{aligned}&\text{Prob}(x_1 \ldots x_L) \\ &\quad = \sum_{paths} \text{Prob}(x_1 \ldots x_L, y_0 \ldots y_{N+1}). \quad (2)\end{aligned}$$

In this way a probability distribution on the space of sequences is defined. The goal is to find a model, i.e. a proper model length and probability parameters, that accurately describes a family of proteins by assigning large probabilities to sequences in that family.

We have chosen this particular structure for the HMM because it is the simplest model that captures the structural intuition of a protein as a sequence of positions, each with its own distribution over the amino acids, along with the possibility for either skipping a position or inserting extra amino acids between consecutive positions, and allowing for the possibility that once a position is skipped, it may be more likely that positions following that one are also skipped. This choice appears to have worked well for modeling the globins, but other types of HMMs may be better at other tasks. The beauty of the HMM method is that it is completely general. One can choose any structure for the states and transitions that is appropriate for the problem at hand.

## 3 Estimating the HMM

All the parameters in the model (i.e., probabilities) could in principle be chosen by hand from an existing alignment of protein sequences, as in [2], or from information about the three-dimensional structure of proteins, as in [3]. The novel approach we take is to "learn" the parameters entirely automatically from a set of *unaligned* primary sequences, using an EM algorithm. The particular method we use to learn the model can be viewed as a computationally efficient approximation to Maximum A Posteriori (MAP) estimation of the parameters of the model.

In MAP estimation, we define the *posterior probability* Prob(model|sequences) of each particular setting of the model parameters (denoted simply "model" here), given the training sequences we have available (denoted simply "sequences" here). This posterior probability is defined using Bayes rule,

$$\begin{aligned}&\text{Prob(model|sequences)} \\ &\quad = \frac{\text{Prob(sequences|model)Prob(model)}}{\text{Prob(sequences)}}. \quad (3)\end{aligned}$$

Here Prob(sequences|model) is defined using equation (2), as described below, Prob(model) is a *prior* on the space of models, and Prob(sequences) can be viewed as a normalizing constant. The prior on the models contains prior beliefs on what a model should be like, and can be used to "penalize" models that are known to be bad or uninteresting. We discuss this further below. To find the best model one can optimize the posterior probability (3) with respect to the parameters of the model.

### The distance from sequence to model

First, we need a way to calculate the probability Prob(sequences|model). Call the sequences $s^1 \dots s^n$. Then, assuming independence

$$\text{Prob}(s^1 \dots s^n|\text{model}) = \prod_{\mu=1}^{n} \text{Prob}(s^\mu|\text{model}). \quad (4)$$

Equation (2) gives an expression for Prob($s^\mu$|model), which can be calculated using the "forward algorithm" [10]. Often the most probable path will have a much higher weight than any other path, so for simplicity and computational efficiency we have chosen to use the approximation

$$\text{Prob}(s|\text{model}) \simeq \max_{paths} \text{Prob}(s, \text{path}|\text{model}), \quad (5)$$

where Prob($s$, path|model) is given in (1). (In equation (1) and (2) we did not condition on the model, because it was assumed to be fixed.) Instead of maximizing the probability we find it convenient to minimize the negative logarithm of the probability. This minimum we will call the distance from the sequence to the model,

$$\begin{aligned} \text{dist}(s, \text{model}) &= -\min_{paths} \log \text{Prob}(s, \text{path}|\text{model}) \\ &= -\min_{paths} \sum_{i=1}^{N+1} [\log \mathcal{T}(y_i|y_{i-1}) + \log \mathcal{P}(x_{l(i)}|y_i)] \quad (6) \end{aligned}$$

Here we have translated back to the notation of (1) where the path is given by $y_0 \dots y_{N+1}$ and the sequence $s$ by $x_1 \dots x_L$. This distance from sequence to model is very similar to the standard "edit distance" from one sequence to another (with gap penalties), see e.g. [8]. The $-\log \mathcal{T}(y_i|y_{i-1})$ corresponds to a penalty for using the transition from $y_{i-1}$ to $y_i$ in the model. One of the main differences is that the penalties (e.g., for starting a gap) depend on the position in the model, whereas they would be fixed in the standard pairwise alignment method. Indeed, this distance can be efficiently calculated in quadratic time by a straightforward generalization of the dynamic programming method used for sequence comparison (see also [2]). In addition, the most probable path can be found using the usual backtracking technique. The algorithm is known as the Viterbi algorithm in the HMM literature (see e.g. [10]).

### Estimation algorithm

There is no known efficient way to directly calculate the best model, i.e., the one that maximizes the posterior probability (3). However, there are algorithms that iteratively re-estimate the model from some arbitrary starting point which guarantee that the posterior probability increases in each iteration, and thus find a local maximum. The most common one is the Baum-Welch or forward-backward algorithm [10], which is a version of the general EM (Expectation-Maximization) method often used in statistics. The full Baum-Welch algorithm is computationally rather intensive, but using the Viterbi approximation (5) makes it somewhat faster.[2]

The Viterbi-EM adaptation of the model to the sequences is an iterative process. From some more or less arbitrary starting point the best path through the model is found for all the sequences, and the new probabilities are basically set equal to the fraction of times that particular path was used, or that particular amino acid produced. More precisely, it proceeds as follows:

1. Choose an initial model. If one already knows some features present in the sequences, they can be encoded in the initial model.

2. Using the Viterbi algorithm, find the most probable path through the model for each of the sequences. The number of these paths that pass through state $y$ is denoted $n(y)$, the number of times the transition from $y$ to $y'$ occurs is denoted $n(y'|y)$, and the number of times an amino acid $x$ was generated in state $y$ is denoted $m(x|y)$. Obviously $n(\mathtt{m}_k) = n(\mathtt{m}_{k+1}|\mathtt{m}_k) + n(\mathtt{d}_{k+1}|\mathtt{m}_k) + n(\mathtt{i}_k|\mathtt{m}_k)$, and similarly for delete and insert states. Also $n(y) = \sum_{i=1}^{20} m(a_i|y)$ if $y$ is a match or insertion state and $a_1 \dots a_{20}$ are the amino acids.

3. Reestimate the transition probabilities and amino acid probabilities in the match and insertion states from

$$\mathcal{T}(y'|y) = \frac{n(y'|y) + R(y'|y)}{n(y) + \sum_{y''} R(y''|y)} \text{ and } \quad (7)$$

$$\mathcal{P}(x|y) = \frac{m(x|y) + R(x|y)}{n(y) + \sum_{i=1}^{20} R(a_i|y)}. \quad (8)$$

Here $R(y'|y)$ and $R(x|y)$ represent a *regularizer*, which comes from the model prior in (3). We assume that $R(y''|y) = 0$ if there is no transition from state $y$ to state $y''$. Often $R$ is just set equal to one in order to avoid zero probabilities, but it can be used to bias the model in a certain direction, as will be discussed later.

[2] Recently we have switched to using the full Baum-Welch algorithm for training the model instead of the Viterbi algorithm, because we find that it gives slightly better results, even though it takes about twice as long.

4. Repeat step 2 and 3 until the parameters change only insignificantly.

Further details are given in the appendix. The main difference between this method, which uses the Viterbi approximation, and the standard Baum-Welch algorithm is that in the latter all paths (not just the most probable) are considered, but weighted according to their probability.

### Initial model and local minima

As already mentioned, the algorithm discussed above does not guarantee convergence to the best model. It is basically a steepest-descent-type algorithm that climbs the nearest peak (local maximum) of the posterior probability function. Since finding the globally optimal model seems to be a difficult optimization problem in general [15], we have experimented with various heuristic methods to improve the performance of the method.

Probably the best method is to give the model a hint if something is already known about the sequences, which is often the case. A good starting point makes it much more likely that the nearest peak is at least close to optimal. This is done by setting the probabilities in the initial model to values reflecting that knowledge. If, for instance, an alignment of some of the sequences is available, it is straightforward to translate that into a model by simply calculating the relative frequency of the amino acids and the transition frequencies in each position. We can then use (7) and (8) to find the initial model.

It is of course even more interesting if the model can be found from a *tabula rasa*, i.e., using no knowledge about the sequences. For that we have used an initial model where all equivalent probabilities are the same, i.e., $\mathcal{T}(\mathtt{m}_k|\mathtt{m}_{k+1})$ is independent of the position $k$ in the model, and similarly for all other transition probabilities, and $\mathcal{P}(x|\mathtt{m}_k)$ is also independent of $k$. To avoid the smaller local maxima, noise is added to the model during the iteration before each reestimation. Initially quite a lot of noise is added, but over ten iterations the noise is decreased linearly to zero. Since noise is added directly to the model, it is not like the usual implementation of simulated annealing, but the principle is the same. The "annealing schedule" is presently rather arbitrary, but it does seem to give reasonable results if it is applied several times, and the best of the models found is used as the final model. Details are given in the appendix.

### Choosing the length of the model

The length of the model is a crucial parameter that needs to be chosen *a priori*. However, we have developed a simple heuristic that selects a good model length, and even helps in the problem of local minima. The heuristic is this: After learning, if more than a fraction[3] $\gamma_{del}$ of the optimal paths of the sequences choose $\mathtt{d}_k$, the delete state at position $k$, that position is removed from the model. Similarly, if more than a fraction $\gamma_{ins}$ make insertions at position $k$ (in state $\mathtt{i}_k$), a number of new positions are inserted into the model after position $k$. This number is equal to the average number of insertions the sequences make at that position. After these changes in the model, it is retrained, and this cycle is repeated until no more changes are needed. We call this "model surgery".

### Over-fitting and regularization

A model with too many free parameters cannot be estimated well from a relatively small data set of training sequences. If we try to estimate such a model, we run into the problem of *overfitting*, in which the model fits the training sequences very well, but gives a poor fit to related (test) sequences that were not included in the training set. We say that the model does not "generalize" well to test sequences. This phenomenon has been well documented in statistics and machine learning (see e.g. [16]). One way to deal with this problem is to control the effective number of free parameters in the model by *regularization*, which can be viewed as a way of implementing MAP estimation.

In our application of regularization to HMMs, the regularizer is closely related to the model prior, and it can be used to bias the model in some specific direction. The regularizer ($R$) is added to the frequency counts in (7) and (8) as if we actually saw more than $n(y'|y)$ or $m(x|y)$ events. For instance, in our models we believe *a priori* that sequences should spend most of their time in the match states on the main line of the HMM, so we incorporate this belief into the model prior by setting $R(\mathtt{m}_i|y)$ to a large value compared to the regularizer $R(y|\mathtt{m}_i)$, which is added to transitions diverging from the main line. In so doing, these parameters become less adjustable, and as a result we are less prone to overfit the data. We also tend to get better models because we are encouraging the learning algorithm to first explore the more *a priori* likely part of model space when trying to fit the data. A more detailed Bayesian interpretation of this method of regularization is beyond the scope of this paper.

[3] Currently we choose $\gamma_{del}$ and $\gamma_{ins}$ each to be 1/2.

In the extreme case, the regularizers $R(\cdot|\cdot)$ can be such large numbers that the parameters simply become "hardwired". We have found that the method works best when very large regularizers $R(x|y)$, proportional to the relative overall frequencies of amino acids in globin sequences, are used when $y$ is an insert state. This has the effect of fixing the distributions in the insert states, rather than trying to estimate them from the training sequences.

## 4 Experiments

The modeling has been tested on the globin family, which includes the hemoglobins, myoglobins, and other heme-binding proteins. Hemoglobin is the protein which, in vertebrates, functions as the principal carrier of oxygen from the lungs to the tissues. It is normally found within the erythrocytes, annucleate cells which are present in the plasma of the circulatory system. Myoglobin is a protein present in muscle cells of both vertebrates and invertebrates and participates in respiration, where it functions as a storage site for oxygen. In some invertebrates, erythrocruorin has a function similar to that of hemoglobin in higher animals, and, like other invertebrate heme-containing proteins, is extracellular. Mammalian hemoglobins are tetramers composed of four subunits: two alpha chains and two other subunits (usually beta, gamma, delta or theta). Myoglobin is a single chain and the remainder have different oligomeric states (homo- or heterodimers) and overall architecture.

Globins are a well studied and fairly homogeneous family with many sequences available. Our set of 628 globins was found in the SWISS-PROT database (release 19) by searching for the word "globin" (three of these later turned out to be non-globins, hence the number 625 given in the introduction). They have an average length of about 145 amino acids without too much variation. The sample of globins in the database is *not* the random sample of globins a statistician would prefer, but it is perhaps one of the best and largest collection of protein sequences from a homologous family. Searching for the words "alpha", "beta", "gamma", "delta", "theta", and "myoglobin" in the data file gave the following results: 224 alpha chains, 199 beta, 16 gamma, 8 delta, 5 theta, and 79 myoglobins, which adds up to 531 sequences. These should naturally be considered minimum numbers, but they give a good picture of how skewed the sample is.

Many of the globin sequences have been aligned by Bashford et al. [1] using seven different globins whose three-dimensional structures were known. This was done by aligning these seven sequences, and then aligning the rest (of the 226 they studied) to the closest of these seven. The alignment of the seven is shown in figure 2.

```
          AAAAAAAAAAAAAAAA   BBBBBBBBBBBBBBBBCCCCCCCCCCCC
                                                      DDDD
---------VLSPADKTNVKAAWGKVGA--HAGEYGAEALERMFLSFPTTKTYFPHF-DL
--------VHLTPEEKSAVTALWGKV----NVDEVGGEALGRLLVVYPWTQRFFESFGDL
---------VLSEGEWQLVLHVWAKVEA--DIAGHGQDILIRLFKHHPETLEKFDRFKHL
----------LSADQISTVQASFDKVKG------DPVGILYAVFKADPSIMAKFTQFAG-
PIVDTGSVAPLSAAEKTKIRSAWAPVYS--TYETSGVDILVKFFTSTPAAQEFFPKFKGL
--------GALTESQAALVKSSWEEFNA--NIPKHTHRFFILVLEIAPAAKDLFS-FLK-
---------GLSAAQRQVIAATWKDIAGADNGAGVGKDCLIKFLSAHPQMAAVFG-FSG-

 DDDDDDDEEEEEEEEEEEEEEEEEEE            FFFFFFFFFFF  FFGGG
                           F                      GG  GG
S-----HGSAQVKGHGKKVADALTNAVAHV---D--DMPNALSALSDLHAHKL--RVDPV
STPDAVMGNPKVKAHGKKVLGAFSDGLAHL---D--NLKGTFATLSELHCDKL--HVDPE
KSEAEMKASEDLKKHGVTVLTALGAILKK----K-GHHEAELKPLAQSHATKH--KIPIK
KDLESIKGTAPFETHANRIVGFFSKIIGEL--P---NIEADVNTFVASHKPRG---VTHD
TTADQLKKSADVRWHAERIINAVNDAVASM--DDTEKMSMKLRDLSGKHAKSF--QVDPQ
GTSEVPQNNPELQAHAGKVFKLVYEAAIQLQVTGVVVTDATLKNLGSVHVSKG---VADA
---AS---DPGVAALGAKVLAQIGVAVSHL--GDEGKMVAQMKAVGVRHKGYGNKHIKAQ

GGGGGGGGGGGGGGGG     HHHHHHHHHHHHHHHHHHHHHHHHH

NFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR------
NFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH------
YLEFISEAIIHVLHSRHPADFGADAQGAMSKALELFRKDIAAKYKELGYQG
QLNNFRAGFVSYMKAHT--DFA-GAEAAWGATLDTFFGMIFSKM-------
YFKVLAAVIADTVAAG---------DAGFEKLMSMICILLRSAY-------
HFPVVKEAILKTIKEVVGAKWSEELNSAWTIAYDELAIVIKKEMNDAA---
YFEPLGASLLSAMEHRIGGKMNAAAKDAWAAAYADISGALISGLQS-----
```

Figure 2: The alignment of seven representative globins from Bashford et al. [1]. Above the alignments are indicated which alpha helix each column belongs to. Bashford et al. identified 8 helices (A to H), but some regions are not well defined (especially CD, D, and FG). The sequences and their SWISS-PROT identifiers are from the top: Human $\alpha$ (HBA$HUMAN), human $\beta$ (HBB$HUMAN), sperm whale myoglobin (MYG$PHYCA), larval *Chironomous thummi* globin (GLB3$CHITP), sea lamprey globin (GLB5$PETMA), *Lupinus luteus* leghemoglobin (LGB2$LUPLU), and bloodworm globin (GLB1$GLYDI).

Our procedure was tested in two ways. First, to see if the model was at all appropriate for the description of the globins, an initial model was derived from the alignment of the seven globins as outlined in the previous section. The length of this model was 171, which is the same as the length of the alignment. A little noise was added to the model, and it was trained using all of the 628 sequences, but without annealing (i.e., noise was only added to the initial model). The distributions over the amino acids modeled in the insertion states were held fixed to the overall amino acid distribution in the globins by regularization, as described above, and were not reestimated from the training sequences. Also, no positions in the model were added or deleted in the process. The procedure

was repeated ten times, and virtually identical models were produced in all the runs. (The runs differed only in the specific noise added to the initial model.) To score a final model, we used the average distance between the sequences and the models. As discussed in the previous section, the distance between a sequence and a model is approximately the negative log probability of the sequence under the model. These average distance scores were consistently around 208 in all runs.

In the second and more interesting experiment, we used a homogeneous initial model that contained no knowledge about the globin family. Its probability parameters were set according to what we thought were reasonable guesses, but they were the same for all equivalent transitions (i.e., 9 different transition probabilities), and all amino acid probabilities (the $\mathcal{P}$ distributions) were set equal to the global distribution of the amino acids. Also, in this experiment 400 sequences were picked uniformly at random from the 628 sequences. These sequences were used as a "training set" to train the model. We held out the other 228 sequences so that we could test the model on data not used in the training process.

The model was trained using noise and "model surgery" ($\gamma_{del} = \gamma_{ins} = 0.5$), as described above. As in the previous experiment, the distributions in the insertion states were not reestimated from the training sequences. This was repeated 20 times, and the average run-time was around 25 cpu minutes on a Sun Sparc 2 station. The final scores varied considerably for these runs, and all of them were higher than for the previous model. The best score was 218. That model had a length of 147. We validated this model from this second experiment in two ways: from the alignments it produces, and by its ability to discriminate between globins and non-globins. The results are described below.

In the appendix we give a detailed description of the particular regularization values and noise added in these experiments. Our choice of parameters for these experiments is based on only one or two trials. Subsequent experiments have verified that when there are many training sequences, as for the globins, the method is not very sensitive the choice of these parameters. Furthermore, recently we tried the program with the exact same setting of all the regularization, noise rate and other parameters on an entirely different set of sequences: the set of 193 protein kinase sequences from the protein kinase catalytic domain database by Hanks and Quinn [17]. Preliminary analysis of these experiments indicate that the method works just as well on the kinases as it does on the globins, without any further tuning of the parameters. This is perhaps the best evidence for the robustness of the method. We will give details of these experiments in a future paper.

### Multiple sequence alignments

From a model it is possible to obtain a multiple alignment of all the sequences. It is in a sense an incomplete alignment, because insertions are not really aligned. It is obtained by finding the most probable path through the model for the sequences to be aligned. This gives an alignment of all the amino acids that align to the match states on the main line in the model. Between the match states there might be insertions of varying lengths, but by inserting enough spaces in all the sequences to accommodate the longest insertion, an alignment is obtained. Figure 3 shows the alignment of the seven sequences produced by the model from the second experiment, and gives a more detailed explanation. Once a model has been constructed, a similar multiple alignment can be produced for the entire set of 628 globin sequences (or any subset, however small) using this model. The alignments using the model from the first experiments are even better, and are almost identical to the structural alignments from [1]. This indicates that an extremely good model exists, and that the EM algorithm was able to find a good approximation to it. It also shows that the method will be very useful for inserting new sequences into existing alignments, since the previous alignment forms a good starting point for the EM algorithm in this case. As more sequences become available, the overall accuracy of the multiple alignment should improve, because better models will be found.

Although it is not perfect, the alignment found from the second experiment (starting from an unbiased model) agrees very well with the structurally derived alignment by Bashford et al. [1]. The main problem with the alignment, from a protein structure point of view, is that it does not correctly align the first important conserved histidine (H) of HBA$HUMAN (the first sequence).[4] Other than this, the only region in which the model did not give the same alignment as Bashford et al. is the region between the C and E helices (with the exception of the first part of the alpha chain, most of the E helix is correct). This region is

[4] In subsequent experiments using the full Baum-Welch algorithm instead of the Viterbi method for training, we have obtained correct alignments of this histidine. In fact, these alignments have been almost perfect. We will report them in a future paper.


Figure 3: The alignment of the globins depicted in figure 2, as obtained from our model trained on 400 randomly chosen globin sequences using a homogeneous initial model. The capital letters represent amino acids aligned to the main line of the model, "-" to deletions in the model, and lower-case letters to amino acids treated as insertions by the model. The "." is used as fill character to accommodate insertions. No attempt has been made to align the insertion regions. The stars above the alignments indicate almost perfect agreement with the structural alignment (the only accepted difference is reasonable displacements of gaps). The training set contained five of the seven globins, not HBA$HUMAN and GLB5$PETMA.

highly variable, since only some of the globins have the D helix. Five of the insertions the model chose are between helices or at the ends of helices, regions which are very variable. The last insertion (of length one) appears in the E helix in the only region not modeled very well.

### Database search: Discriminating globins from non-globins

The model found in the second experiment was tested by comparing it to all proteins in the SWISS-PROT database of length less than 2000 amino acids. For each of these sequences their distance to the model was computed. For the non-globins, this distance turns out to increase roughly linearly with the length of the sequence. By smoothing the non-globin distance data[5] we found a curve for the average distance as a function of sequence length. The standard deviation was also calculated as a function of length and smoothed the same way. Finally the deviation from the average, measured in standard deviations, was found for all the sequences. These are plotted in figure 4.

The model distinguishes almost perfectly between globins and non-globins. Choosing a cutoff of 4 standard deviations we would get 10 out of 19,458 "false globins" and miss 2 out of 625 globins. Choosing 5 standard deviations as a cutoff would give 0 false globins and miss 11 of the real globins. The "anomalous" globin sequences between 4 and 5 standard deviations from the average non-globin include those from invertebrates (arthropods, molluscs and annelids) that, unlike hemoglobin and myoglobin from higher organisms, are extracellular. The two sequences falling between 1 and 3 standard deviations from the average non-globin are protozoan (unicellular), whereas the other globins are metazoan.

It is also interesting to see where random sequences fall in the histogram. First, we generated about 2000 sequences at random with the same distribution of amino acids as the globins (length from 10 to 2000). They are all closer than 5 standard deviations to the average non-globin (see figure 4, last panel). Second, we generated 500 sequences at random *from the model*. This was done by taking random walks through the states of the model, as described above. Obviously they should fit the model very well, and they do. However, the majority of the globins fit the model even better. This may simply be because the data set is far from being a set of random and independent sequences.

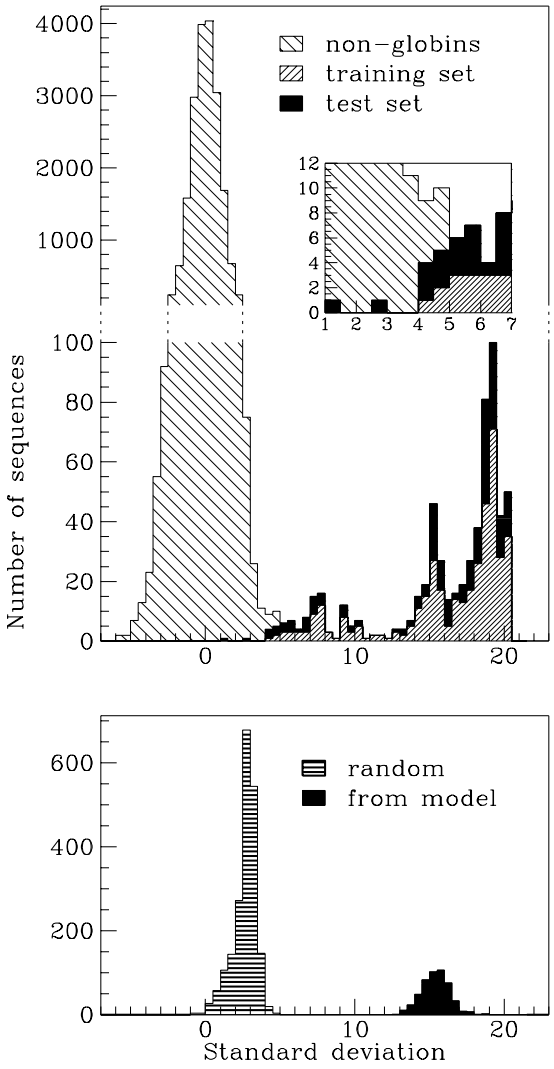
## 5 Conclusion

A new method to model protein families using hidden Markov models has been introduced. The method utilizes the tremendous amount of information contained in many sequences from the same family. For the simple case of globins, with this method it is possible to obtain multiple alignments that mirror structural alignments using only the unaligned primary sequences as input. Furthermore, we also did not need any sophisticated weighting schemes to compensate for the skewed dataset (which is dominated by alpha and beta chains), nor did we need a large number of

[5] The smoothing was done by assuming local linearity and averaging over a running window of 300 points.

Figure 4: Histograms of the distance to the model measured in standard deviations from the average distance to the model for non-globins (see text).
Top: The training set of 400 globins and test set of 225 globins. Four globin fragments of length 19–45 were removed from the data (three from worms and one from indian spiny-tailed lizard). We also removed three non-globin sequences in the globin file that we discovered when analyzing the outliers. All of these were close to zero standard deviations from the average.
Also shown are all the non-globins in the SWISS-PROT database of length less than 2000. From this data we removed on the order of ten sequences that contained a lot of Xs (unknown amino acids), because they spuriously match the model very well. (Currently we treat an unknown amino acid as being the most probable amino acid at the position it is matched to.) Note the change of scale of the y-axis. The insert shows the critical region around five standard deviations from the average.

Bottom: The distribution of 2000 randomly generated sequences with the same amino acid distribution as the globins (labeled "random"). These sequences have lengths 1,2,3,4,...,2000. Note that, even though they have the same amino acid distribution, these sequences are all less than 5 standard deviations from the average, which strongly supports a cutoff around that value. Also 500 random sequences generated from the model and afterwards compared to the same model are shown (labeled "from model"). They were generated by the kind of "random walk" described earlier.




precomputed regularization parameters, such as the approximately 200 parameters of the Dayhoff matrix normally used in multiple alignment programs. The model can also be used in database searches for putative analogs of sequences in a given protein family, and we believe that the model itself is a valuable tool for representing the family. We have demonstrated the model for globin sequences in the experiments described here, and are currently planning to do further experiments with the kinase, cytochrome c, serine protease, immunoglobulin, ATPase and helicase protein families.

The method requires that many sequences be available from the family one wants to model. Since the number of sequences in the protein databases is growing rapidly, this may be less of a problem in the future, but it will always be a serious issue. Currently only a relatively small number of sequences are available for most protein families. For the globin family, we found that 400 sequences is certainly enough. Preliminary results indicate that 200 is enough, and even as few as 70 suffice, if they are chosen carefully from our database of 628 (70 chosen at random will be nearly all alpha and beta chains). Using careful regularization this number might even be lowered further. However, there will be a limit on how small the number of available sequences can be if one hopes to get a reasonable model starting from a *tabula rasa*.

There are two possible answers to the problem of small sets of training sequences. The first is to add more prior knowledge into the training process. This can be done by starting with a better initial model,

and by using more involved kinds of regularization. We are currently exploring a regularization method that we call "soft tying" of the distributions in the states of the HMM. This combines the idea of tying states, see e.g. [10], in which the number of free parameters is reduced by having groups of states all share the same distribution on the output alphabet (the 20 amino acids in this case), and the idea of *soft weight sharing* from [18], in which the regularizer (in this case for the distribution of amino acids) is also adaptively modified during learning. Even more complex, position-specific kinds of regularization are possible. Switching from the alphabet of the primary sequences to a different representation based more on the structural or chemical properties of the amino acids in the sequence may also help.

The second answer is to give up trying to model the whole protein in cases where only few sequences are known, and instead try to model only pieces of the protein (motifs or domains) that have the same or similar structure in other proteins. In this case we can find more training sequences by adding in the corresponding pieces of these other proteins. In fact, the EM method can be used to locate the appropriate pieces by itself, given the entire sequences [5, 6]. We are currently experimenting with an extension of the method described here that does this, applying it to the kinase and EF hand domains. In this method we augment our models by having "free insertions" in both ends of a (short) model, so that only a subsequence is matched to the model. In addition, several models can also be linked together with free insertions in between, to model more than one subsequence in the molecules.

Finally, when a relatively large number of sequences are available, it is sometimes possible to get better results by dividing these sequences into clusters of similar sequences, and training a different model for each cluster. For example, we can train separate models for the alpha and beta chains in the globin dataset. It turns out that again, this partition into clusters of similar sequences can also be done automatically by the EM algorithm during training. This way of using EM is called mixture modeling in the statistics literature, and is known as competitive learning in the neural network literature. To implement this, we simply start with more than one model and force the models to "compete" for the sequences, and adapt to fit only the ones they "won". This method gives very good results on the globins, discovering the subclasses of alpha chains, beta chains and myoglobins with near perfect accuracy. Details will be reported in a future paper.

In summary, we believe that HMMs and the EM algorithm have tremendous potential in the area of statistical modeling of biological macromolecules. Currently, most of this potential remains to be realized.

## Acknowledgements

We would like to thank Jeff Keller for his comments and help with the software, and John Bridle, Peter Brown, Richard Durbin, Alan Lapedes, Richard Lippmann, Harry Noller, Christine Guthrie, Martin Vingron, and Michael Zuker for valuable comments on this work. This work was supported by NSF grants CDA-9115268 and IRI-9123692, ONR grant N00014-91-J-1162, NIH grant number GM17129, and a grant from the Danish Natural Science Research Council.

## Appendix: Technical Details

### Prior, initial model, regularizer, and noise

The model prior described in the paper is a probability distribution over model space, i.e. a multivariate distribution over all the probabilities in the model. We wouldn't like to have to specify that in detail, so instead we specify a model that we believe (a priori) is reasonable. This model is called *the prior model*, and can be thought of as (very loosely speaking) the peak of our prior distribution. The following transition probabilities have been used in the prior model in all our runs,

| from\to | delete | match | insert |
|---|---|---|---|
| delete | 0.10 | 0.88 | 0.02 |
| match | 0.02 | 0.96 | 0.02 |
| insert | 0.02 | 0.59 | 0.39 |

(Numbers are rounded off. The "funny" numbers come about, because for instance the last row is entered as 0.02, 0.6, and 0.4, and the program automatically normalizes the sum to one).

For all the amino acid distributions we used the empirical distribution of the training data. For the globin training set containing 400 randomly chosen sequences it looks like this:


Another natural choice would be to use probability 1/20 for all the amino acids. In our experiments it does not seem to make much difference, although we have not studied the difference systematically.

The regularizer ($R$ in (7) and (8)) is obtained directly from the prior model by multiplying all probabilities by a "confidence" parameter. For the transitions we used a parameter of 50, so for instance $R(\mathtt{d}_k|\mathtt{m}_{k-1}) = 50 \times 0.02 = 1$, $R(\mathtt{m}_k|\mathtt{m}_{k-1}) = 50 \times 0.96 = 48$, and $R(\mathtt{i}_k|\mathtt{m}_k) = 50 \times 0.02 = 1$. For the amino acid distributions in the match states a value of 20 was used. With the number of sequences as large as in the experiments reported here, the program does not seem very sensitive to the size of the regularizers.

The insertion states were kept basically fixed by using a huge confidence parameter (100,000). This is just a simple way to fix the distribution in the current implementation, and should not be thought of as regularization.

The *initial model* is a noisy version of the prior model. The noise can be added to the probabilities directly. For adding noise to a probability distribution the beta distribution, or more generally the Dirichlet distribution, can be used. However, for simplicity we are currently generating noise from the prior model itself. This is done by generating artificial "frequency counts" by doing random walks through the model, and counting how many times each transition is used and how many amino acids of each kind are produced at each position. Then the initial model is obtained from these artificial frequency counts by "reestimation" according to equations (7) and (8). In our experiments the initial model is obtained from 100 such random walks.

### The "annealing" procedure

The annealing method used to try to find a good model is currently very crude. It is almost certain that it can be improved by experimenting with other parameter settings, other kinds of noise, and by a more careful annealing schedule.

In the iterative reestimation procedure we add noise to the model in each step. The noise is generated in almost the same way we generate the initial model. After step 2 in the EM algorithm (page 5) artificial frequency counts are generated *from the prior model* by $K$ random walks, and then added to the real frequency counts ($n$ and $m$). However, the noise is weighted by a parameter we can call the noise level $T$, which is the analogue of the temperature in simulated annealing. So, instead of adding one each time a transition is used, $T$ is added. Initially $T$ is large, but after each reestimation of the model it is decreased according to the annealing schedule. Instead of gradually lowering $T$ one could decrease $K$, which would presumably have basically the same effect.

In the runs reported here we used $K = 100$. To start with we used $T = 1$, and lowered $T$ linearly to 0 over 10 iterations, i.e., $T = 1 - i/10$, if $i$ is the iteration number. As indicated earlier, this schedule is completely ad hoc, and only based on a few tests, but it does improve performance as compared to adding no noise. After the temperature reaches zero, the estimation process usually converges in about 4 iterations. We stopped the iterations if the change in the average distance of all the sequences to the model was less than 0.1.

### The "surgery" procedure

There are many ways one can imagine to change the length of the model. Of the heuristics we have considered, we chose to implement the simplest one: After learning, if more than a fraction $\gamma_{del}$ of the optimal paths of the sequences choose $\mathtt{d}_k$, the delete state at position $k$, that position is removed from the model. Similarly, if more than a fraction $\gamma_{ins}$ make insertions at position $k$ (in state $\mathtt{i}_k$), a number of new positions are inserted into the model after position $k$. The number of new positions is equal to the average number of insertions the sequences make at that position. In the experiments we used $\gamma_{del} = \gamma_{ins} = 0.5$. After this "surgery" the model was re-trained by going through the whole annealing process. When no surgeries were necessary anymore, the program was stopped. The method turned out to work surprisingly well, in the sense that it usually converges to stable length after just one or two surgeries (if the initial length is not too far off). In the globin experiments, an initial length of 171 was used. In almost all the runs the final length was between 142 and 149.

### Treatment of unknown characters

In the sequences we used there are three special characters for unknown amino acids: B, which means amino acid N or D, Z meaning Q or E, and X which means unknown amino acid. We have always given these characters the benefit of the doubt, e.g., by assigning a B the largest probability, $\max_{N,D}\{P(N|\mathtt{m}_k), P(D|\mathtt{m}_k)\}$, of the two possible at each state. This has the unfortunate consequence that a sequence consisting of only Xs will have the highest possible probability. Therefore, when running the whole database through the globin model, we found some very good matches that turned out to only match Xs to the model. Although this is not such a big problem with the data we used, one could consider using other methods if unknown amino acids are prevalent.

### The dynamic programming algorithm

The Viterbi algorithm is described in the paper by Rabiner [10] page 264. Here we give a translation to our notation. The description in [10] is nice, general, and compact, so only the reader who wants to see the connection to sequence comparison with gap penalties, or who wants to actually implement the algorithm will benefit from this translation.

A negative log of a probability we will call a cost. At each position $k$ in the model we calculate three quantities: The lowest cost $D_i(\mathtt{m}_k)$ of having the $i$th amino acid match to $\mathtt{m}_k$, the lowest cost $D_i(\mathtt{d}_k)$ of having the delete state $\mathtt{d}_k$ between the $i$th amino acid and the next, and the lowest cost $D_i(\mathtt{i}_k)$ of having the $i$th amino acid inserted in state $\mathtt{i}_k$. Also, backtrack information about the cheapest path into a state $y$ and generating amino acid $x_i$ is kept as $\phi_i(y)$, which is a pointer to the previous state that makes it cheapest to get to state $y$. If $y$ is a delete state $\phi_i(y)$ points to the cheapest state before $y$ in between generating the $i$th amino acid and the next. The procedure is as follows:

1. Initialization

$$
\begin{aligned}
&D_0(\mathtt{m}_0) = D_0(\mathtt{d}_0) = D_0(\mathtt{i}_0) = 0\\
&\phi_0(\mathtt{m}_0) = \phi_0(\mathtt{d}_0) = \phi_0(\mathtt{i}_0) = null\\
&D_0(\mathtt{d}_1) = -\log \mathcal{T}(\mathtt{d}_1|\mathtt{m}_0)\\
&D_1(\mathtt{i}_0) = -\log \mathcal{T}(\mathtt{i}_0|\mathtt{m}_0) - \log \mathcal{P}(x_1|\mathtt{i}_0)\\
&\phi_0(\mathtt{d}_1) = \phi_1(\mathtt{i}_1) = \mathtt{m}_0\\
&D_0(\mathtt{d}_k) = D_0(\mathtt{d}_1) - \sum_{j=2}^{k} \log \mathcal{T}(\mathtt{d}_j|\mathtt{d}_{j-1}) \text{ for } 2 \le k \le M
\end{aligned}
$$

$$\phi_0(\mathtt{d}_k) = \mathtt{d}_{k-1} \text{ for } 2 \le k \le M$$
$$D_i(\mathtt{i}_0) = D_1(\mathtt{i}_1) - \sum_{j=2}^{i} [\log \mathcal{T}(\mathtt{i}_0|\mathtt{i}_0) + \log \mathcal{P}(x_i|\mathtt{i}_0)] \text{ for } 2 \le i \le L$$
$$\phi_i(\mathtt{i}_i) = \mathtt{i}_{i-1} \text{ for } 2 \le i \le L$$

where $M$ is the length of the model, and $L$ is the length of the sequence.

2. Recursion

$$\phi_i(\mathtt{m}_k) = \operatorname*{argmin}_{\mathtt{m}_{k-1},\mathtt{d}_{k-1},\mathtt{i}_{k-1}} (D_{i-1}(y) - \log \mathcal{T}(\mathtt{m}_k|y))$$
$$D_i(\mathtt{m}_k) = D_{i-1}(\phi_i(\mathtt{m}_k)) - \log \mathcal{P}(x_i|\mathtt{m}_k) - \log \mathcal{T}(\mathtt{m}_k|\phi_i(\mathtt{m}_k))$$
$$\phi_i(\mathtt{d}_k) = \operatorname*{argmin}_{\mathtt{m}_{k-1},\mathtt{d}_{k-1},\mathtt{i}_{k-1}} (D_i(y) - \log \mathcal{T}(\mathtt{d}_k|y))$$
$$D_i(\mathtt{d}_k) = D_i(\phi_i(\mathtt{d}_k)) - \log \mathcal{T}(\mathtt{d}_k|\phi_i(\mathtt{d}_k))$$
$$\phi_i(\mathtt{i}_k) = \operatorname*{argmin}_{\mathtt{m}_k,\mathtt{d}_k,\mathtt{i}_k} (D_{i-1}(y) - \log \mathcal{T}(\mathtt{i}_k|y))$$
$$D_i(\mathtt{i}_k) = D_{i-1}(\phi_i(\mathtt{i}_k)) - \log \mathcal{P}(x_i|\mathtt{i}_k) - \log \mathcal{T}(\mathtt{i}_k|\phi_i(\mathtt{i}_k))$$

3. Termination

$$\phi_{L+1}(\mathtt{m}_{M+1}) = \operatorname*{argmin}_{\mathtt{m}_M,\mathtt{d}_M,\mathtt{i}_M} (D_L(y) - \log \mathcal{T}(\mathtt{m}_{M+1}|y))$$
$$\text{dist}(x_1 \ldots x_L, \text{model}) = D_L(\phi_{L+1}(\mathtt{m}_{M+1})) - \log \mathcal{T}(\mathtt{m}_{M+1}|\phi_i(\mathtt{m}_{M+1}))$$
$$\phi_{L+1}(\mathtt{d}_{M+1}) = \phi_{L+1}(\mathtt{i}_{M+1}) = null$$

4. Backtracking. The most probable path is found this way. First $y_N = \phi_{L+1}(\mathtt{m}_{M+1})$. Then iterate $y_{j-1} = \phi_{l(j)}(y_j)$ for $j = N$ to $j = 2$. Here $l(j)$ is the number of the amino acid generated in state $j$, as defined on page 4 (where we set $l(j) = l(j-1)$ if $y_j$ is a delete state).

In our implementation we used a 32 bit integer representation of the negative log probabilities to speed up the algorithm. In a sequence-to-sequence comparison with fixed parameters it is quite common to have two or several paths with exactly the same cost. This happens very seldom in this case, because almost all the parameters are different. If it happens our program will choose arbitrarily among these paths.

## References


[1] D Bashford, C Chothia and A M Lesk, *Journ Mol Biol* **196** (1987) 199–216.

[2] M Gribskov, R Lüthy, and D Eisenberg, *Methods in Enzymology* **183** (1990) 146–59.

[3] J U Bowie, R Lüthy, and D Eisenberg, *Science* **253** (1991) 164–70.

[4] M S Waterman and M D Perlwitz, *Bull. Math. Biol.* **46** (1986) 567–577.

[5] C E Lawrence and A A Reilly, *Proteins* **7** (1990) 41–51.

[6] L R Cardon and G D Stormo, *Journ Mol Biol* **223** (1992) 159–170.

[7] A Bairoch and B Boeckmann, *Nucl. Acids Res.* **19** (1991) 2247–2249.

[8] M S Waterman, In *Mathematical Methods for DNA Sequences*, ed M S Waterman, CRC Press, 1989.

[9] M Vingron and P Argos, dot-matrices. *Journ Mol Biol* **218** (1991) 33–43.

[10] L R Rabiner, *Proc IEEE* **77:2** (1989) 257–286.

[11] A P Dempster, N M Laird and D B Rubin, *J. Roy. Statist. Soc. B ,1977, Vol. 39, pp 1-38*

[12] G A Churchill, *Bull Math Biol* **51** (1989) 79–94.

[13] White, Stultz and Smith, preprint, Nov. '91.

[14] D Haussler, A Krogh, S Mian, and K Sjölander, UC Santa Cruz Tech. Rep. UCSC-CRL-92-23.

[15] N Abe and M K Warmuth, *Machine Learn., Spec. issue for COLT90, to appear, 1992.*

[16] S Geman, E Bienenstock, and R Doursat, *Neural Computation* **4** (1992) 1–58.

[17] S K Hanks and A M Quinn, *Methods in Enzymology* **200** (1991) 38-62.

[18] S J Nowlan and G E Hinton, In *Advances Neural Information Processing Systems 4*, ed. Moody, Hanson, and Lippmann, Morgan Kauffmann Publishers, San Mateo, CA, 1992.