Hidden Markov models for sequence analysis: extension and analysis of the basic method

Richard Hughey*and Anders Krogh†

December 6, 1995

Running title: Hidden Markov models for sequence analysis

Keywords: Hidden Markov model, parallel computation, multiple sequence alignment, protein modeling, motif modeling.

Abstract

Hidden Markov models (HMMs) are a highly effective means of modeling a family of unaligned sequences or a common motif within a set of unaligned sequences. The trained HMM can then be used for discrimination or multiple alignment. The basic mathematical description of an HMM and its expectation-maximization training procedure is relatively straight-forward. In this paper, we review the mathematical extensions and heuristics that move the method from the theoretical to the practical. Then, we experimentally analyze the effectiveness of model regularization, dynamic model modification, and optimization strategies. Finally it is demonstrated on the SH2 domain how a domain can be found from unaligned sequences usign a special model type. The experimental work was completed with the aid of the Sequence Alignment and Modeling software suite.

1 Introduction

Since their introduction to the computational biology community (Haussler *et al.*, 1993; Krogh *et al.*, 1994a), hidden Markov models (HMMs) have gained increasing acceptance as a means of sequence modeling, multiple alignment, and profiling (Baldi *et al.*, 1994; Eddy *et al.*, 1995; Eddy, 1995). A hidden Markov model is a statistical model similar to a profile (Gribskov *et al.*, 1987; Bucher & Bairoch, 1994), but can also be estimated from *unaligned sequences.*

*Computer Engineering, University of California, Santa Cruz, CA 95064, USA, rph@cse.ucsc.edu.

†NORDITA, Blegdamsvej 17, 2100 Copenhagen, Denmark. Present address: The Sanger Centre, Hixton, Cambs CB10 1RQ, UK, krogh@sanger.ac.uk.

A linear HMM for a family of nucleotide or amino acid sequences is a set of positions that roughly correspond to columns in a multiple alignment. Typically, each position will have three states: match, insert, and delete, corresponding to matching a single character to a column of the multiple alignment, emitting characters not modeled by the HMM, or skipping over that column and proceeding to the next. Probabilities or costs (negative log-probabilities) are associated with each character emission and each transition between states. The alignment of a sequence is simply the highest-probability, or lowest-cost, path through the HMM.

The primary advantage of HMMs over, for example, profiles, is that they can be automatically estimated, or trained, from unaligned sequences. The most basic mathematical method of training an HMM does not work well without the addition of several mathematical extensions and heuristics. The body of this paper is a study of three of the most important extensions to the basic model presented previously (Krogh *et al.*, 1994a): regularizers, dynamic model modification, and 'free insertion modules'. After reviewing these in turn, we present an experimental evaluation of the utility of these and other extensions.

2 System and methods

The Sequence Alignment and Modeling (SAM) software suite is written in ANSI C for Unix workstations. Additionally, we have implemented the inner loop in MPL for the MasPar family of parallel computers. The system has been tested on DEC Alpha, Decstation, IBM RS/6000, SGI, and Sun Sparc workstations. The research described herein was performed on a Sun Sparc 30/10 and a MasPar MP-2204 connected to a DEC Alpha 5000/300.

Newly revised and documented workstation and MasPar versions of SAM are available from our WWW site, http:/www.cse.ucsc.edu/research/compbio/sam.html. We recently created a server for using SAM to align and model sequences, which is accessible from the SAM WWW page. Questions concerning the software and its distribution can be mailed to sam-info@cse.ucsc.edu.

3 Algorithm

This section discusses the basic theory and use of HMMs. Although most aspects of our linear HMMs have been described previously (Krogh *et al.*, 1994a), this section reexamines several operational features as the foundation for the experimental evaluation that follows.

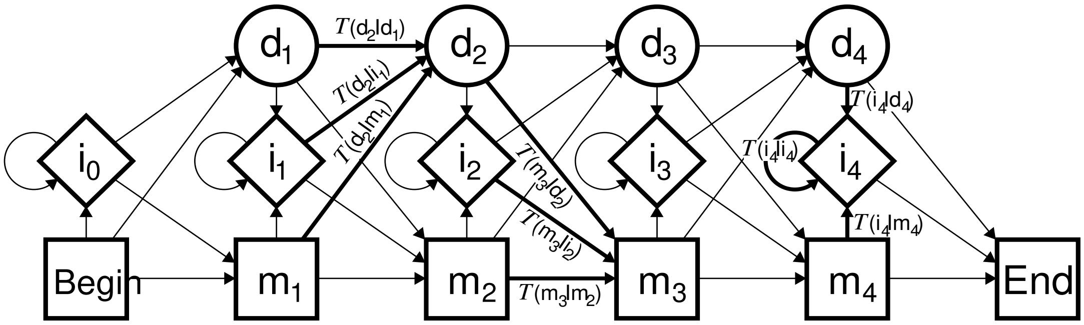


Figure 1: A linear hidden Markov model.

3.1 The hidden Markov model

This section will briefly describe the structure and basic theory of the type of hidden Markov model used in this work. For a general introduction to hidden Markov models, refer to (Rabiner, 1989). For additional details on our hidden Markov models, refer to (Krogh *et al.*, 1994a).

The basic model topology is shown in Figure 1. Each position, or module, in the model has three states. A state shown as a rectangular box is a match state that models the distribution of letters in the corresponding column of an alignment. A state shown by a diamond-shaped box models insertions of random letters between two alignment positions, and a state shown by a circle models a deletion, corresponding to a gap in an alignment. States of neighboring positions are connected, as shown by lines. For each of these lines there is an associated 'transition probability', which is the probability of going from one state to the other.

This model topology was chosen so as to feature insertions and deletions similar to biological sequence alignment techniques based on affine gap penalties. Other model topologies can certainly be considered, see *e.g.* (Baldi *et al.*, 1994; Krogh *et al.*, 1994b), but for reasons of efficiency the software described here is limited to the topology shown in Figure 1. Gibbs sampling is an alternate approach that does not allow arbitrary gaps within aligned blocks (Lawrence *et al.*, 1993).

Alignment of a sequence to a model means that each letter in the sequence is associated with a match or insert state in the model. Two 5-character sequences, A and B are shown in a 4-state model in Figure 2, along with the corresponding alignment between the sequences.

One can specify such an alignment by giving the corresponding sequence of states with the restriction that the transition lines in the figure must be followed. For example, to match a letter to the first match state (m_1) and the next letter to match state number 3 (m_3) can only be done by using the intermediate delete state (d_2), so that part of the alignment

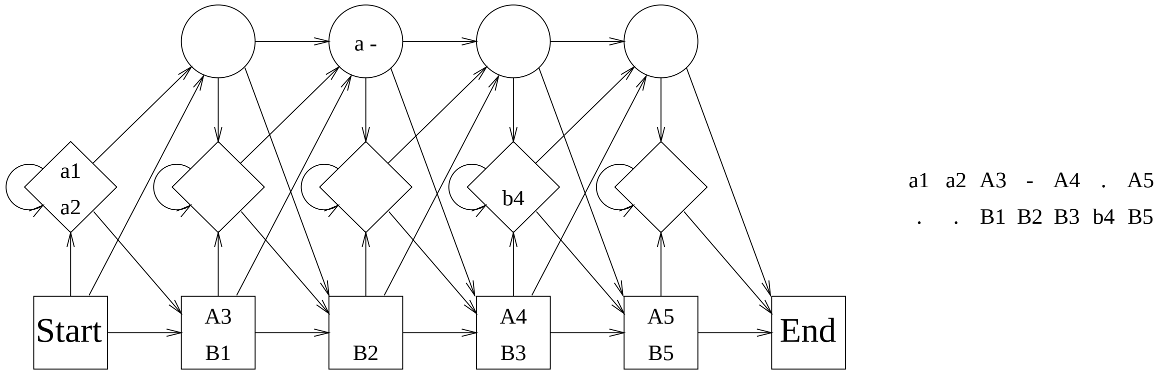


Figure 2: An example of two sequences whose characters are matches to states in an HMM, and the corresponding alignment.

can be written as $m_1 d_2 m_3$. In HMM terminology such an alignment of a sequence to a model is called a path through the model. A sequence can be aligned to a model in many different ways, just as in sequence-to-sequence alignments, or sequence-to-profile alignments. Each alignment of a given sequence to the model can be scored by using the probability parameters of the model.

In the following, a sequence of L letters is denoted $x_1 \dots x_L$. The path specifying the particular alignment is denoted $q_1 \dots q_L$, where q_k is the kth state in the path. To simplify notation, as compared to (Krogh *et al.*, 1994a), only the match and insert states are included in the path; for two states not directly connected, delete states are filled in as needed. Because the first state in a path is always the begin state m_0, and the last one is always the end state m_{M+1}, these two trivial states are also not included in the state sequence. The number of match positions in the model is called M, and is referred to as the length of the model. The probability of an alignment of a sequence to a model is the product of all the probabilities met in the path, written as

$$\mathrm{Prob}(x_1 \dots x_L, q_1 \dots q_L | \mathrm{model}) = \left[\prod_{i=1}^{L} \mathcal{T}(q_i | q_{i-1}) \mathcal{P}(x_i | q_i) \right] \times \mathcal{T}(q_{L+1} | q_L) \qquad (1)$$

where $\mathcal{T}(q_i|q_{i-1})$ is the probability associated with the transition between states q_{i-1} and q_i. The probability of matching letter x to state q is called $\mathcal{P}(x|q)$. If two states in the path are not directly connected by a transition, the transition probability has to be calculated by filling in the appropriate delete states. For example, the probability of a transition from the match state at position 1 (m_1) to the insert state at position 4 (i_4), passing through delete states d_2, d_3, and d_4, is

$$\mathcal{T}(i_4|m_1) = \mathcal{T}(d_2|m_1)\mathcal{T}(d_3|d_2)\mathcal{T}(d_4|d_3)\mathcal{T}(i_4|d_4) \qquad (2)$$

By taking the negative logarithm of equation (1) it can be turned into a more familiar type of alignment 'cost':

$$-\log \mathrm{Prob}(x_1 \ldots x_L, q_1 \ldots q_L|\mathrm{model}) = \sum_{i=1}^{L}[-\log \mathcal{T}(q_i|q_{i-1}) - \log \mathcal{P}(x_i|q_i)] - \log \mathcal{T}(q_{L+1}|q_L). \tag{3}$$

Now the numbers can be interpreted as 'penalties'. A term like $-\log \mathcal{T}(d_{i+1}|m_i)$ is a positive penalty for initiating a deletion after position i, and $-\log \mathcal{T}(d_{i+1}|d_i)$ is the penalty for continuing a deletion at position i. Some of the other terms have similar interpretations.

3.2 Estimation of the model

One great advantage of HMMs is that they can be estimated from sequences, without having to align the sequences first. The sequences used to estimate or *train* the model are called the training sequences, and any reserved sequences used to evaluate the model are called the test sequences. The model estimation is done with the forward-backward algorithm, also known as the Baum-Welch algorithm, which is described in (Rabiner, 1989). It is an iterative algorithm that maximizes the likelihood of the training sequences. We have modified the algorithm to implement maximum *a posteriori* (MAP) estimation.

In the basic algorithm the expected number of times a certain transition or letter emission is used by the training sequences is calculated (Rabiner, 1989). For a letter emission probability $\mathcal{P}(x|q)$ this is called $n(x|q)$. Then the reestimated parameter is

$$\hat{\mathcal{P}}(x|q) = \frac{n(x|q)}{\sum_{x'} n(x'|q)}, \tag{4}$$

where the sum is over all the characters x' in the alphabet, such as the 20 amino acids. The reestimation formula for the transition probabilities are similar. To begin with all the $n(x|q)$ values are found from an arbitrary initial model. Next, a new set of parameters is found using equation (4), and the similar formula for the transition probabilities. Then, using the new model, the reestimation procedure is repeated until the change in the parameters is insignificant.

3.2.1 Prior distribution and regularization

When estimating a model from data, there is always the possibility that the model will overfit the data — it models the training sequences very well, but will not fit other sequences from the same family. This is particularly likely if there are few training sequences. With only one training sequence, a perfect model would have a match state for each residue in which that residue would have unity probability and all other residues zero probability. Such a model would give zero probability to all other sequences than the training sequence! For larger sets of training data, similar problems are still present but not as extreme.

To avoid this problem a *regularizer* can be used. Regularization is a method to avoid overfitting the data, and in Bayesian statistics it is tightly connected with the so-called *prior* distribution. The prior distribution is a distribution over the model parameters; for the HMM it is a probability distribution over probability distributions. The prior contains our prior beliefs about the parameters of the model. In our work we use Dirichlet distributions for the prior (Berger, 1985; Santner & Duffy, 1989). For a discrete probability distribution $p_1, \ldots, p_M$ a Dirichlet distribution is described by M parameters $\alpha_1, \ldots, \alpha_M$. The mean of the Dirichlet distribution is $p_i = \alpha_i/\alpha_0$, where $\alpha_0 = \sum_i \alpha_i$, and the variance is inversely proportional to α_0. If α_0 is large, it is highly probable that $p_i \simeq \alpha_i/\alpha_0$. For each probability distribution in the HMM, a Dirichlet distribution is used as a prior, and we call the corresponding α values $\alpha(x|q)$ for the distributions over characters, and $\alpha(q_i|q_j)$ for the transition probabilities. The reestimation formula corresponding to (4) is

$$\hat{\mathcal{P}}(x|m_k) = \frac{n(x|m_k) + \alpha(x|m_k)}{\sum_{x'}[n(x'|m_k) + \alpha(x'|m_k)]}. \tag{5}$$

The set of all the αs is called the regularizer. We call this the *maximum a posteriori* (MAP) estimate, although the correct MAP formula has $\alpha(x|m_k) - 1$ instead of $\alpha(x|m_k)$. Equation (5) is really a least squares estimate (see (Krogh *et al.*, 1994a)), but one can also view it as a MAP estimate with redefined αs.

Even without the theoretical justification this formula is appealing. For each parameter in the model a number (α) is added to the corresponding n before the new parameter is found. If n is small compared to α, as when there is little training data, the regularizer essentially determines the parameter, and

$$\hat{\mathcal{P}}(x|m_k) \simeq \frac{\alpha(x|m_k)}{\sum_{x'} \alpha(x'|m_k)}. \tag{6}$$

(This is the average of the Dirichlet distribution). The size of the sum $\sum_{x'} \alpha(x'|m_k)$ determines the strength of the regularization, or the strength of the prior beliefs. If this sum is small, say 1, just a few sequences will be enough to 'take over' the model. On the other hand, if the sum is large, say 1000, then on the order of 1000 training sequences will be needed to significantly make the model differ from the prior beliefs.

This type of regularization is convenient when modeling biological sequences because we have prior knowledge from conventional alignment methods. For example, in both pairwise and multiple alignments the penalty for starting a deletion is usually larger than for continuing a deletion. This 'prior belief' that match to delete transitions are less probable than delete to delete transitions can easily be built into an HMM by setting $\alpha(d_i|m_{i-1}) < \alpha(d_i|d_{i-1})$.

The SAM system can also use the more complicated Dirichlet mixture priors for regularization. These priors include several different distributions for some number of different types of columns, such as hydrophobic and hydrophilic positions. For more information on these distributions, please refer to our previous work (Brown *et al.*, 1993).

By normalizing the regularizer as in (6), a valid model is obtained. Since this model represents prior beliefs, it is natural to use it as the initial model for the estimation process. Usually noise is added to this initial model for reasons discussed next.

3.2.2 Problems with local maxima

A serious problem with any hill-climbing optimization technique is that it often ends up in a local maximum. The same is true for the forward-backward procedure used to estimate HMMs by maximizing the likelihood (or the *a posteriori* model probability). In fact, it will almost always end up in a local maximum.

To deal with this problem, we start the algorithm several times from different initial models. The resulting models then represent different local maxima, and we pick the one with the highest likelihood. The different initial models are obtained by taking the normalized regularizer and adding noise to all the parameters. The noise is added to a model in the following way. A number R of sequences are generated from the regularizer model, stepping from state to state and generating characters according to the regularizer's transition and match distributions. Given a noise level N_i, each of these R paths through the model is added to the counts with a weight of N_i/R, before MAP reestimation. By default, R is set to 50 random sequences.

One can also add noise to the models during their estimation and decrease the noise level gradually in a technique similar to the general optimization method called simulated annealing (Kirkpatrick *et al.*, 1983). The initial level of the noise in this annealing process is called N_0 as above. During the estimation process the annealing noise is decreased by a speed determined by r. We have tested two method for decreasing the noise:

Linearly: If $r \geq 1$ the noise is decreased linearly to zero in r iterations, so in iteration i

$$N_i = N_0(1 - i/r) \text{ for } i < r \text{ and } N_i = 0 \text{ for } i \geq r.$$

Exponentially: If r is less than 1, the noise is decreased exponentially by multiplying the noise with r in each iteration,

$$N_i = N_0 r^i.$$

An alternative and very elegant way of using simulated annealing is described in (Eddy, 1995).

3.3 Multiple sequence alignment

The probability (1) or the score (3) can (in principle) be calculated for all possible alignments to a model, and thus the most probable (*i.e.*, the *best*) can be found. There exists a dynamic programming technique, called the Viterbi algorithm (Rabiner, 1989), that can find the best alignment and its probability without going through all the possible alignments. It is this best alignment to the model that is used to produce multiple alignments of a set of sequences. For each of the sequences the alignment to the model is found. The columns of the multiple alignment are then determined by the match states. All the amino acids matched to a particular match state are placed under each other to form the columns of the alignments. For sequences that do not have a match to a certain match state, a gap character is added (Figure 2).

3.4 Searching

By summing the probabilities of all the different alignments of a sequence to a model, one can calculate the total probability of the sequence given that model,

$$\mathrm{Prob}(x_1 \ldots x_L|\mathrm{model}) = \sum_{q_1 \cdots q_L} \mathrm{Prob}(x_1 \ldots x_L, q_1 \ldots q_L|\mathrm{model}), \tag{7}$$

where the sum is over all possible alignments (paths) $q_1 \ldots q_L$, and the probability in the sum is given by (1). This probability can be calculated efficiently without having to explicitly consider all the possible alignments by the forward algorithm (Rabiner, 1989). The negative logarithm of this probability is called the negative log-likelihood score,

$$\mathrm{NLL}(x_1 \ldots x_L|\mathrm{model}) = -\log \mathrm{Prob}(x_1 \ldots x_L|\mathrm{model}) \tag{8}$$

Any sequence can be compared to a model by calculating this NLL score. For sequences of equal length the NLL scores measures how 'far' they are from the model, and it can be used to select sequences that are from the same family. However, the NLL score has a strong dependence on sequence length and model length, see Figure 3. One means of overcoming this length bias is using Z-scores, or the number of standard deviations each NLL is away from the average NLL of sequences of the same length, but which are not part of the family being modeled, or do not contain the motif being modeled.

When searching a database like SWISS-PROT (Bairoch & Boeckmann, 1994) with an HMM, the smooth average and the Z-scores are calculated as follows. For a fixed sequence length we assume that the NLL scores are distributed as a normal distribution with some *outliers* representing the sequences in the modeled family. The smooth average should be the average of the normal distribution, and it is found by iteratively removing outliers:

1. For k larger than or equal to the minimum sequence length, find the minimum length $e_k \geq k$ such that the length interval $[k, e_k]$ contains at least K sequences. We usually use $K = 1000$. For all such intervals calculate the average sequence length t_k and the average NLL score.

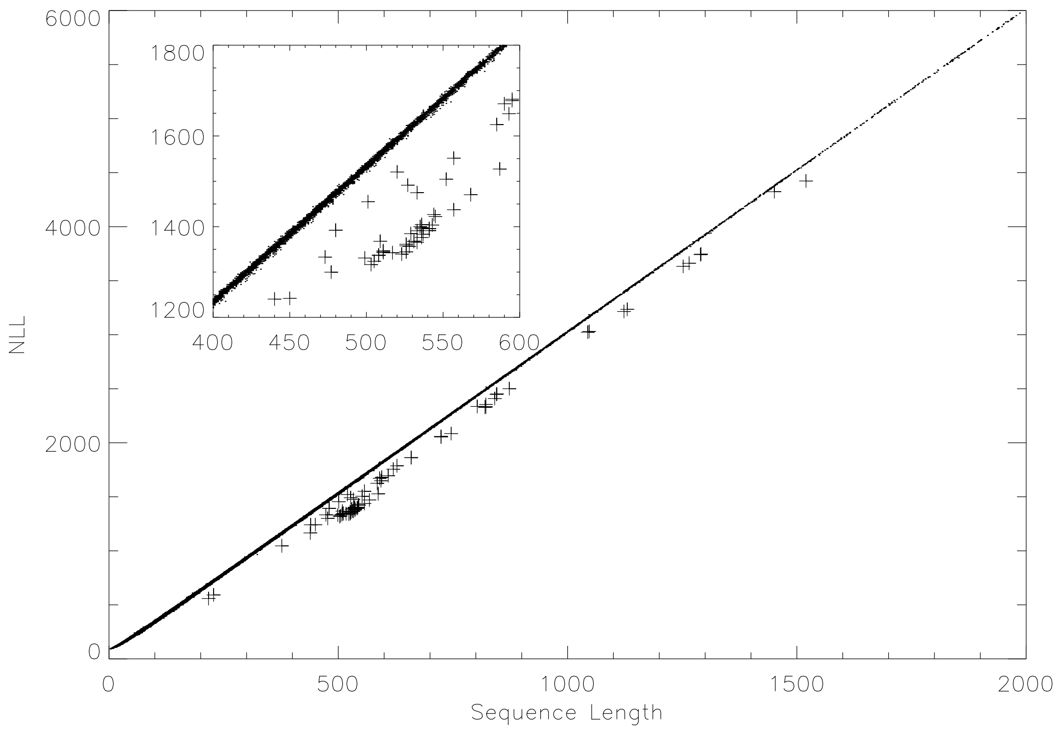


Figure 3: NLL scores vs. sequence length for a search of SWISS-PROT using a model of the SH2 domain described in section 5.3. The $\times$ show the SH2 containing sequences (the training set), and dots show the about 30.000 remaining sequences. All sequences containing the letter 'X' are excluded.

2. For any integer length l the smoothened average NLL score is found by linear interpolation of the average NLLs corresponding to the closest $t_k < l$ and t_{k+1}. For l either smaller than the smallest t_k or larger than the largest t_k, linear extrapolation from the two closest points is used.

3. For each interval the standard deviation of the NLL scores from the averages is found. For any integer length the smoothened standard deviation is found by linear interpolation or extrapolation as above.

4. All sequences with an NLL score more than a certain number of standard deviations (usually 4) from the smooth average are considered outliers and excluded in the next iteration.

5. If the excluded sequences are identical to the ones excluded in the previous iteration, the process is stopped. Otherwise it is repeated unless the maximum number of iterations have been performed.

This procedure often produces excellent results on a large database like SWISS-PROT, but there is no guarantee that it works. It is easy to detect when it is not working, because the sequences in the family, such as the training sequences, have low Z-scores. In this case, the training sequences and other obvious outliers can be removed by hand, and the above process repeated. This method always yields good results.

In some sequences there are unknown residues that are indicated by special characters. A completely unknown residue is represented by the letter X in proteins and by N in DNA and RNA. For proteins also the letters B, meaning amino acid N or D, and Z (Q or E) are taken into account. For DNA and RNA the letters R for purine and Y pyrimidine are recognized. All other letters that are not part of the sequence alphabet or equal to one of these wild card characters are taken to be unknown, *i.e.*, changed to X or N depending on the sequence type. The probability of a wild card character in a state of the HMM is set equal to the maximum probability of all the letters it represents. It has the unfortunate side effect that sequences with many unknowns automatically receive a large probability, and these sequences have to be inspected separately. Another solution would be to set the probability equal to the average probability of the letters the wild card represents, but then the opposite problem might occur. This can also be chosen as an option in SAM.

3.5 Model surgery

It is often the case that during the course of learning, some match states in the model are used by few sequences, while some insertion states are used by many sequences. Model surgery is a means of dynamically adjusting the model during training.

The basic operation of surgery is to delete modules with unused match states and to insert modules in place of over-used insert states. In the default case, any match state used by fewer than half the sequences is removed, forcing those sequences to use an insert state or to significantly change their alignment to the model. Similarly, any insert state used by more than half the sequences is replaced with a number of match states approximating the average number of characters inserted by that insert state. After surgery the model is retrained to adjust to the new situation. This process can be continued until the model is stable.

For finer tuning of the surgery process, the user can adjust the fraction of sequences that must use a match state, the fraction of sequences that are allowed to use an insert state, and the scaling used when replacing an insert state with one or more match states. While the default 50% usage heuristic stabalizes after a few surgery iterations, careless setting of these fine-tuning parameters can result in a model that oscillates, switching back and forth between match and insert states.

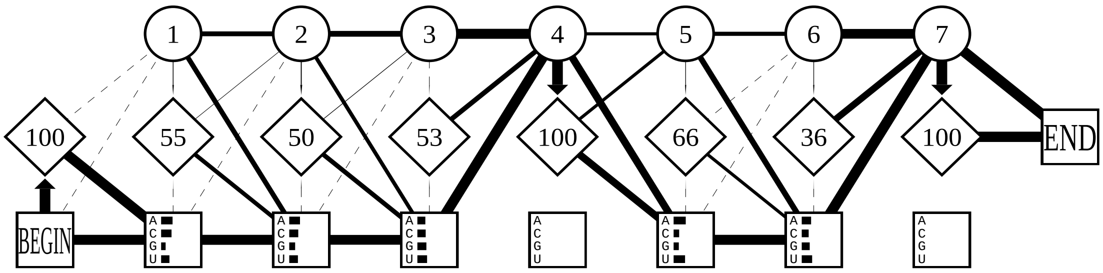


Figure 4: A model with a FIM in both ends and one at module 4. The width of a line is proportional to the probability of the transition. This example uses the RNA alphabet, and the probabilities in the modules other than the FIMs are set according to our standard regularizer, which strongly favors the transitions from match state to match state and has low probability of starting insertions or deletions.

3.6 Modeling domains and motifs

Imagine that the same domain appears in several sequences. Then to build an HMM of that domain and using it for searching or aligning requires a few extensions. We augment the HMM by insertion states in both ends that have no preference for which letters are inserted (*i.e.*, they have uniform character distributions). The cost of aligning a sequence to such a model does not strongly depend on the position of the pattern in the sequence, and thus it will pick up the piece of the sequence that fits the model the best. A new parameter then has to be set, namely the the probability for making an insertion in these flanking insertion modules. If this probability is very low, deletions in the main part of the model is encouraged and insertions discouraged, whereas a high insertion probability encourages the opposite behavior, because it becomes 'expensive' to use the flanking insertion states. The setting of this parameter is most important when *estimating* a model like this, because there is a danger of finding a model in which for instance only the delete states are used if it is too 'cheap' to make inserttion in the flanking modules. These flanking modules are called a *free insertion modules* (FIMs), because the the self-transition probability in the insert state is set to unity. Other values can of course be used.

FIMs are implemented by only using the delete state and the insert state in a standard module. All the transitions from the previous module go into the delete state, which is ensured by setting the other probabilities to zero. From the delete state there is a transition to the insert state with the probability set to one. In the insert state all characters have the same probability and the probability of insert to insert is set to one. From the delete and insert states there are transitions to the next module which have the same probabilities (delete to match and insert to match have the same probability, and so on). Note that the probabilities do not sum up to one properly in such a module. A model with three FIMs is shown in Figure 4.

These special modules can be used to learn, align, or discriminate domains that occur once per sequence. Typically, FIMs are used at the beginning and the end of a model to allow an arbitrary number of insertions at either end. To train a model to find several (different) domains, one can also add FIMs at different positions in the model. Note, however, that only domains always occouring in the same order and the same number of times can be modelled this way. To model domains that come in different order or in a variable number, a model with backward transitions is required, which is something that we might add to SAM at a later stage.

There are two different ways to model subsequences. The first method is to cut out the subsequences from their host molecules, build a model from these, and then add a FIM in both ends afterwards. The second method is to use the full sequences and train with the FIMs. The second method is generally easier, but also slower in terms of CPU time, especially if the sequences are much longer than the subsequences. In (Krogh *et al.*, 1994a) the first method was used to model the protein kinase catalytic domain and the EF hand domain, and in this paper the second method will be demonstrated on the SH2 domain.

In (Lawrence & Reilly, 1990) and (Lawrence *et al.*, 1993), two related methods of automatically finding common patterns in a set of sequences are described. Those papers deal only with gap-less alignments, *i.e.*, patterns without insertions or deletions. The method described here can be viewed as an extension to alignments with gaps.

3.7 Additional features

Several additional features also aid the learning performance of the SAM system. First, SAM performs the first training pass (before any surgery occurs) on multiple models. The best of those models is selected for the remaining surgery and training iterations. Second, SAM also supports Viterbi training, in which the model is trained according to the best path of each sequence through model, rather than by the probability distributions over all possible paths. Although this is slightly faster, the results are generally not as good. Third, when models are derived from an alignment or an existing profile, training of a module's match table or transitions can be turned off and it can be insulated from the surgery procedure. This last feature was not used for the results of this paper, apart from the related protection of FIMs from surgery.

4 Implementation and Performance

The inner loop of the algorithm is an $O(n^2)$ dynamic programming algorithm that calculates the forward and backward values for each of the three states at a given pair of model and sequence indices. The serial implementation of this algorithm is straightforward. For Cray vectorization, the two inner loops were modified to calculate counter-diagonals in order (i.e., $(0,1),(1,0),(0,2),(1,1),(2,0),\ldots$) to make effective use of the vector pipelines.

Table 1: HMM training on various machines.

System	Rel. Porting Difficulty	Performance (kCUPS)	Performance (Sun 4/50s)
Sun 3/110	0	3.2	0.1
Sun 4/50	0	37.1	1.0
Decstation 5000/240	0	39.2	1.1
SGI 440VGX, 1 CPU	0	59.0	1.6
Dec Alpha 3000/500	0	107	2.9
C-Linda, 7 Decstation 5000s (240 and 125)	4	147	4.0
Cray Y-MP, 1 CPU, vectorized	8	167	4.5
8K MP-1, optimized	60	1530	41
4K MP-2, optimized	60	1580	43

The MasPar parallel processor features a 2-dimensional mesh of processing elements (PEs) and a global router (Nickolls, 1990). For large numbers of sequences, the best algorithm mapping can be to perform a complete model-sequence dynamic programming in each PE; unfortunately, the 64 KBytes of local memory per PE is too small for such coarse-grain parallelism. Instead, the array is treated as collection of linear arrays of PEs, where each linear array contains one model. Thus, if the models are of length 100, forty sequence-against-model calculations are performed at a time using 4000 of the 4096 processing elements.

The linear arrays are used as follows. During the forward part of the dynamic programming, the (i, j) value is computed during step $i + j$ in i-th PE of one of the linear arrays. This calculation depends on values from $(i-1, j)$, $(i, j-1)$, and $(i-1, j-1)$, all of which have already been calculated in either PE_i or PE_{i-1}. The characters of the sequence are also shifted through the linear array, one PE at a time, to ensure that character j is in PE_i at time step $i + j$. During the backward calculation, this process is reversed, calculating each diagonal at time $n^2 - i - j$. After all sequences have been compared to the model, the frequencies are combined using log-time reduction, and uploaded to the host computer for high-level processing such as surgery and noise injection.

Performance can be rated in terms of dynamic programming cell updates per second, or CUPS. Performing both the forward and backward calculations at a single dynamic programming cell (one character from one sequence against one model node) is a single cell update. The total number of cell updates, which depends greatly on parameter settings, is the sum of model lengths over all reestimation cycles multiplied by the total number of characters in all the sequences. For the experiments described in the next section, a typical training run on 50 globins of average length 146 and a model length of 145 with 40 reestimation cycles, required 50 seconds on our 4096-PE MasPar MP-2 or about 7 minutes on a DEC Alpha 3000/400 (longer runs runs heighten the difference between the two machines to factors of 10–15). Experiments based on 400 training sequences and an earlier version of the code are summarized in Table 1.

5 Results and Discussion

While several reports have shown the general effectiveness of HMMs (Krogh *et al.*, 1994a; Brown *et al.*, 1993; Baldi *et al.*, 1994; Eddy *et al.*, 1995; Eddy, 1995), this section takes a close look at effectiveness of each extension to the basic method.

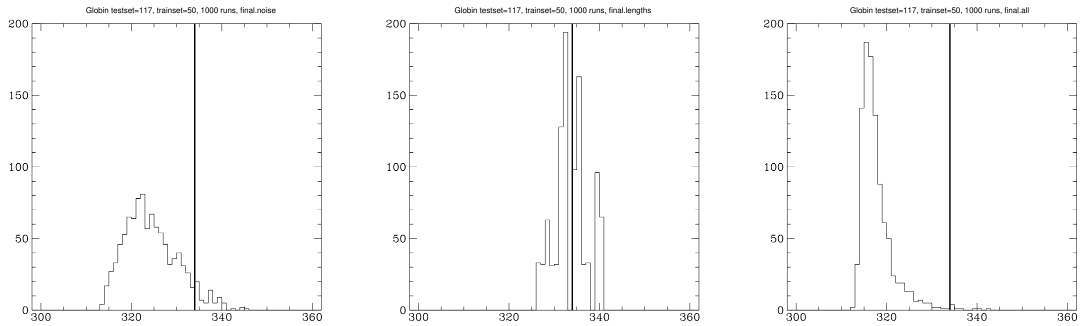
We choose the globin family for the first of these illustrative experiments because of our previous familiarity with the family. From a set of 624 globins, close homologues were removed using a maximum entropy weighting scheme (Krogh & Mitchison, 1995) by removing all sequences with a very small weight ($< 10^{-5}$), which left us with 167 globins. For the experiments, our group of 167 sequences was randomly divided into a training set of 50 sequences and a test set of 117 sequences, except in the experiments on training set size.

The statistical goodness of an HMM is tied to the final probability result of the test set. SAM reports this as a negative-log-likelihood ($-\ln P$), or NLL, score. This section considers the effects of each of the more important extensions on NLL scores. Ideally, we would like small NLL scores that, with multiple runs using different random seeds, are sharply peaked. Such a peaked distribution implies that far fewer than the thousands of runs performed in these experiments are required to generate a good model.

5.1 Noise

To find default noise settings, we ran 50 random seeds for all combinations of nine annealing schedules and seven noise values. Average performance over the runs was typically 5% better than without noise, while the best NLL score over the 50 runs was 12% better than without noise. Given that the typical mode of running SAM is to generate many models and pick the best, this 12% value is quite an improvement. Our chosen default is 5 sequences worth of noise using an exponential annealing schedule with factor 0.8. This is a somewhat arbitrary choice based on the range of scores obtained — no clear winner among the settings emerged. The tested setpoints added between 20% and 350% more reestimation cycles over the noiseless case. If less time is available, we suggest a linear schedule with 1 noise sequence. In general, as many models should be created as possible and then the best one further refined. This procedure is automated in SAM.

The histograms in Figure 5 show average test set NLL scores for 1000 training runs on 50 training globins with just default noise, random model lengths without noise, and all heuristics (noise, random model lengths, and surgery). The vertical bar at 334 indicates the NLL score for training without noise. Note in particular how the combination of noise and surgery both improves the test set scores and sharpens their distribution, indicating that far fewer than 1000 runs are needed to generate good models.



(a) (b) (c)

Figure 5: Test NLL scores (average over 117 test sequences) from running SAM 1000 times on 50 other globin sequences with (a) default noise, (b) random starting model lengths and (c) all heuristics including surgery. The solid vertical line at 334 is the average test sequence score without any random heuristics (in this case, surgery has no effect on the non-random training routine).

5.2 Regularization

The effect of regularization is even more dramatic than the effects of noise and surgery. SAM supports both Dirichlet and Dirichlet mixture regularization. To gauge the effects of regularization, we compared no regularization to four other choices: the default simple regularizer, the original 9-component mixture (Brown *et al.*, 1993), and more recent 9- and 21-component regularizers (Karplus, 1995). Histograms of the test set NLL scores for these experiments, which used the noise settings determined in the previous experiments, are shown in Figure 6. Clearly, regularization is needed. With 50 training sequences, however, the distinction between the different regularizers is not high, as expected, since the model is built from a reasonably-sized training set.

Where regularization truly shines is with small training sets. Figure 7 shows the performance of the regularization methods for small training sets. In this case, each test point represents the average test set NLL score over 20 training runs, each based on a different random training set of the indicated size

The Dirichlet mixture priors require additional running time, but have somewhat improved performance over the single-component prior. All four of these regularizers are available with the SAM distribution.

5.3 Case-study: Modeling the SH2 domain with FIMS

In this section we demonstrate the use of FIMs for modeling domains. We stress that this is mostly for illustrative purposes, so we will not go deep into any biological implications

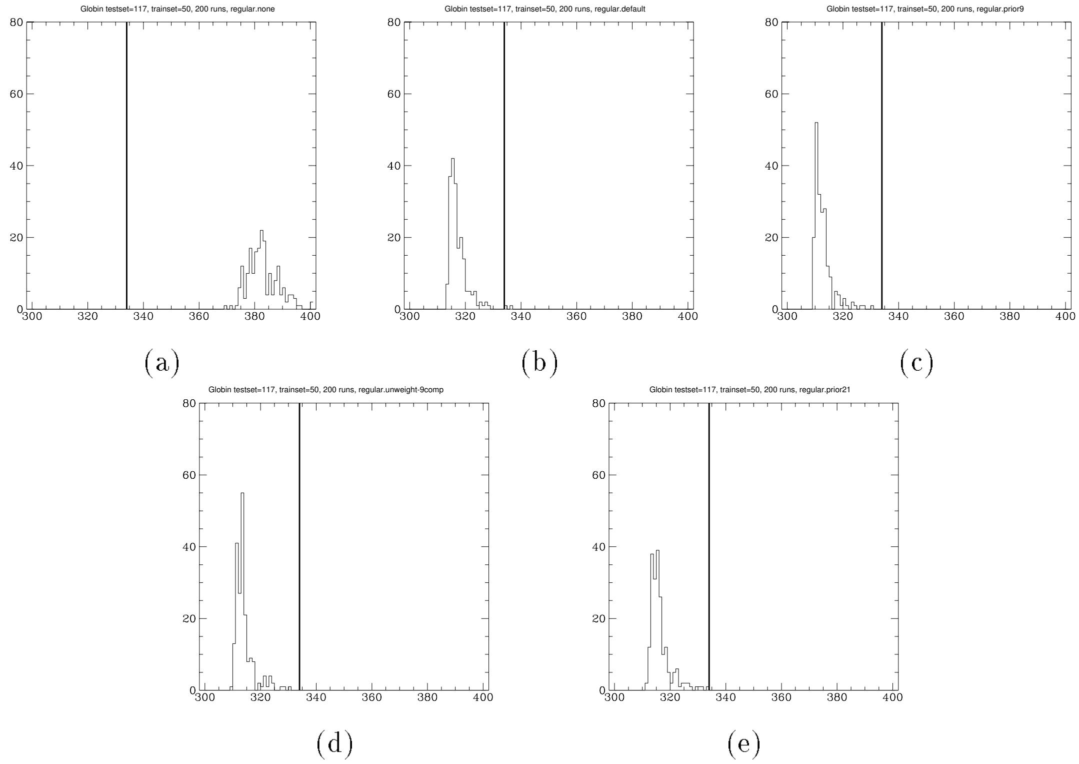


Figure 6: NLL scores on the test set from running SAM 200 times on 50 globin sequences with no regularization (a), and single-component (b), original 9-component (c), revised 9-component (d), and 21-component (e) regularizers. The solid vertical line at 334 is the score with no random heuristics and the default regularizer.

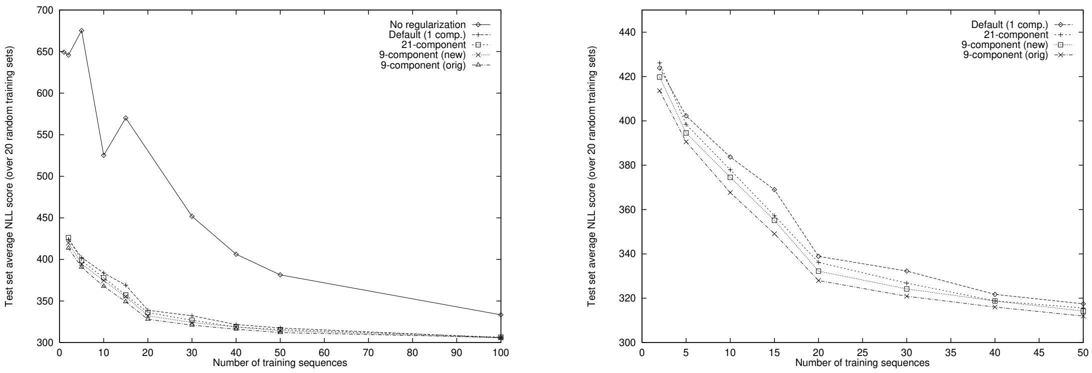


Figure 7: The effect of training set size and regularization on model building. Especially for small training sets, regularization greatly improves modeling with respect to the larger test set. The differences in test set scores between the available regularizers is shown in the blowup to the right.

of the model or alignment. We use the SH2 domain, which is found in a variety of proteins involved in signal transduction, where it mediates protein–protein interactions. For a review see (Kuriyan & Cowburn, 1993). The domain has a length of about 100.

Initially a file was created with 78 SH2 containing proteins by searching SWISS-PROT (release 30) for the keyword 'SH2 domain', see Table 2. 50 models of length 100 were trained in batches of 10 with FIMs in both ends and without surgery. For each batch the model with the best overall score was examined. Of these 5 models the best scoring one happened to cover the SH2 domain almost entirely. It started about 20 amino acids prior to the domain and ended about 20 amino acids early as compared to the the alignment in (Kuriyan & Cowburn, 1993). Some of the other models also covered part of the domain, whereas others had picked up a different signal. This signal was probably the kinase catalytic domain of some of the proteins in the file. It is quite remarkable that the model can find the domain completely unsupervised, and that might not always be the case.

Using this first model, a search was made of the entire SWISS-PROT database and all sequences scoring better than a Z-score of 7 were examined (not taking sequences with many 'X' characters into account), see Table 2. All the sequences in the training set were among those high-scoring ones, except a fragment of length 17 which was then deleted from the data set. Of these high-scoring sequences, 10 had Z-scores of 12 or more and by checking the alignment, we consider it to be certain that they contain SH2. The last sequence with a score of 7.1 we also believe contains SH2. All these high-scoring sequences were now included in the data set. The old and new sequences in the training set are listed in Table 2, and all sequences with Z-scores larger than 4 is shown in Table 3.

Now the model was modified by deleting the first 20 modules and inserting 30 'blank' modules in the end, so it could better fit the domain. Starting from this modified model, 20 new models were trained on the new set of 88 protein sequences, and the best one selected. This was the final model for the SH2 domain, see Figure 8. Using this model a new search was performed. The new model picked up all of the 88 sequences in the training set, and the smallest score was significantly higher than that of the first model (13.8 compared to 7.1), see Table 2. It also found a new one (KFLK_RAT) with a Z-score of 8.4 that is a fragment and in SWISS-PROT it is described as containing part of the SH2 domain. All other sequences in SWISS-PROT had Z-scores below 5.5, and in the highest scoring ones (Table 3) we did not see any signs of the SH2 domain.

The SH2 domian often occours several times in the same protein, which is not modelled properly by such a model (see Section 3.6). When training, all domains in a given protein contribute to the model (because all paths are taken into account), but when aligning only the occourence matching the model best will be found. There are ways to find all occourences by finding suboptimal paths or masking domains already found.

Table 2: All the sequences with a Z-score greater than 7 using the first model. The score is shown in the columns labeled 'Z 1'. All except the ones marked by '***' were part of the training set initially extracted from SWISS-PROT. All the ones marked by '***' were included in the training set. The column labeled 'Z 2' show the Z-scores for the new model. The first training set also contained the fragment KLCK_RAT of length 17. It had a Z-score of 0.18 with the first model, and it was then removed from the training set.

ID	length	Z 1	Z 2	ID	length	Z 1	Z 2
KSRC_AVISS	568	67.569	57.941	PIP5_RAT	1265	41.204	41.236
KSRC_AVIS2	587	66.610	59.765	PTNB_MOUSE	585	38.255	36.095
KSRC_AVIST	557	66.187	56.073	PIP5_HUMAN	1252	38.087	38.653
KSRC_CHICK	533	63.475	56.653	KATK_HUMAN	659	37.859	31.787
KYES_XENLA	537	63.177	55.735	PTNB_HUMAN	593	37.700	36.920
KSRC_AVISR	526	63.007	57.519	GTPA_HUMAN	1047	37.255	35.081
KSRC_RSVSR	526	63.006	57.519	GTPA_BOVIN	1044	36.794	34.776
KSRC_RSVPA	523	62.815	58.016	KATK_MOUSE	659	36.593	31.475
KSRC_HUMAN	536	62.782	56.550	P85A_MOUSE	724	35.106	35.299
KSRN_MOUSE	541	62.609	56.243	P85A_BOVIN	724	35.104	35.299
KYES_CHICK	541	62.162	55.339	P85A_HUMAN	724	35.042	35.389
KHCK_MOUSE	503	62.012	51.266	KABL_MLVAB***	746	34.189	44.282
KFGR_MOUSE	517	61.846	53.802	KCSK_RAT	450	34.115	30.778
KFYN_HUMAN	536	61.811	52.437	CSW_DROME	841	33.930	36.004
KFYN_CHICK	533	61.799	51.222	PTN6_MOUSE	595	33.681	31.583
KYES_AVISY	528	61.798	55.872	PTN6_HUMAN	595	32.909	30.885
KHCK_HUMAN	505	61.731	50.077	KLYK_HUMAN	620	32.546	29.987
KFYN_XENLA	536	60.899	51.891	P85B_BOVIN	724	31.012	34.030
KYES_MOUSE	541	60.894	55.201	KSR2_DROME	590	30.245	30.025
KYES_HUMAN	543	60.808	54.594	SEM5_CAEEL	228	29.742	25.657
KSR1_XENLA	531	60.516	54.921	SHC_HUMAN	473	28.570	24.458
KSR2_XENLA	531	60.320	54.793	KABL_CAEEL***	557	28.330	32.172
KYRK_CHICK	535	59.673	50.890	KFES_FSVST***	477	28.243	35.183
KSRC_RSVH1	526	59.639	54.652	KFES_HUMAN	822	28.214	40.554
KFYN_XIPHE	536	59.610	50.490	NCK_HUMAN	377	28.194	26.543
KSRC_RSVP	526	59.370	53.892	GRB2_HUMAN	217	27.546	24.833
KLCK_HUMAN	508	58.265	49.467	KFES_FELCA	820	27.264	40.511
KLCK_MOUSE	508	58.265	49.287	VAV_MOUSE	845	25.937	29.161
KLYN_HUMAN	511	58.265	49.868	KFER_HUMAN	822	25.905	35.117
KFGR_HUMAN	529	58.024	50.468	KFES_FSVGA***	609	25.838	36.585
KLYN_MOUSE	511	57.843	49.468	GAGC_AVISC	440	25.058	24.473
KLYN_RAT	511	57.843	48.955	VAV_HUMAN	846	24.997	28.856
KYES_XIPHE	544	57.477	50.010	KFES_MOUSE***	820	23.101	38.987
KFGR_FSVGR***	545	55.885	51.428	KSYK_PIG	628	22.652	28.247
KBLK_MOUSE	499	55.812	45.981	KAKT_MLVAT	501	21.399	18.600
KSTK_HYDAT	509	47.469	43.094	KFPS_AVISP***	533	21.396	33.346
KABL_MOUSE	1123	47.440	46.331	KFPS_FUJSV***	873	21.063	36.728
KABL_HUMAN	1130	47.408	46.458	KRAC_MOUSE	480	20.673	17.461
KABL_DROME	1520	46.822	41.191	KTEC_MOUSE	527	20.547	26.322
KABL_FSVHY***	439	43.836	40.175	KRAC_HUMAN	480	20.509	17.461
KSR1_DROME	552	43.509	38.564	KRCB_HUMAN	520	16.613	16.221
PIP4_HUMAN	1290	42.121	39.355	KFPS_DROME***	803	12.358	25.804
PIP4_RAT	1290	41.892	39.526	SPT6_YEAST	1451	9.461	13.805
PIP4_BOVIN	1291	41.533	39.341	YKF1_CAEEL***	424	7.138	19.053

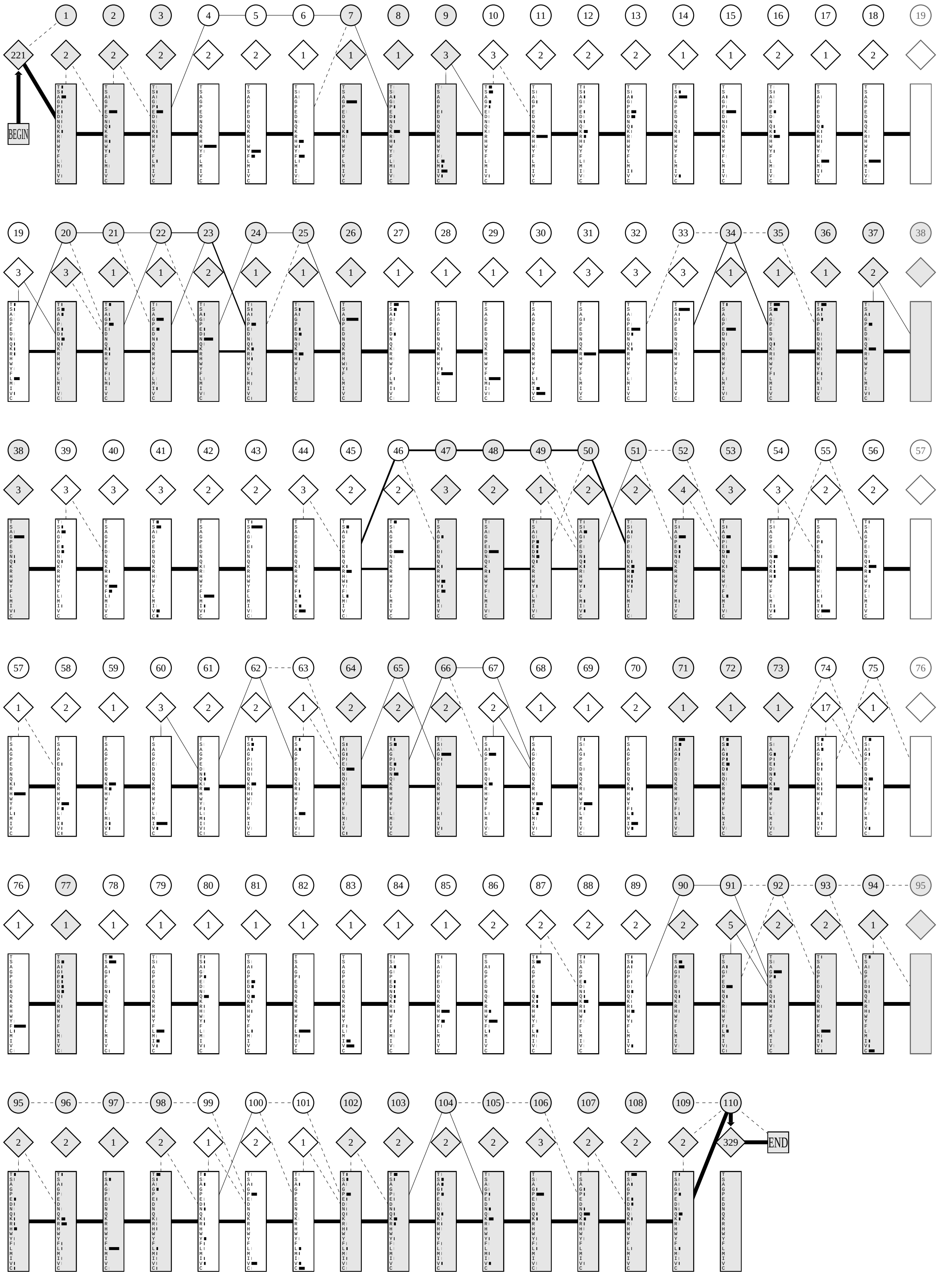


Figure 8: The second (and final) model of the SH2 domain. The *unshaded* modules correspond to secondary structure elements as given in (Kuriyan & Cowburn, 1993). These elements are βA, αA, βB, βC, βD, βE, βF, αB, and βG. The figure (except the shading) was produced with the program drawmodel in SAM. It is almost impossible to see the actual amino acid distributions at this magnification, but the most important things to notice are that the distributions are quite peaked and that the delete states are used very rarely in the conserved secondary structure elements, which are both indications of a good model.

Table 3: All the sequences that have a Z-score of more than 4, but not shown in 2. The column labeled 'Z 1' contains the scores from the search with the first model and the one labeled 'Z 2' the scores with the second model.

ID	length	Z 1	ID	length	Z 2
MYCN_CHICK	441	6.067	KFLK_RAT	323	8.445
HLYA_SERMA	1608	5.507	PIP1_DROME	1312	5.432
PSBO_CHLRE	291	5.426	MYSD_MOUSE	1853	5.245
MYSN_DROME	2017	5.187	GSPF_ERWCA	408	5.227
MYCN_SERCA	426	5.111	DRRA_STRPE	330	4.775
TRP_YEAST	707	4.994	NFL_COTJA	555	4.750
DRRA_STRPE	330	4.958	MPP1_NEUCR	577	4.718
COAT_RCNMV	339	4.937	CHLB_CHLHU	431	4.603
GLNA_AZOBR	468	4.831	VG24_HSVI1	333	4.587
FIXL_BRAJA	505	4.654	UFO_MOUSE	888	4.521
MXIC_SHIFL	182	4.593	TPM_SCHPO	161	4.477
ENL_HUMAN	559	4.592	POL_MMTVB	899	4.413
K2C1_XENLA	425	4.503	RM02_YEAST	371	4.378
RF3_SACUV	476	4.500	UFO_HUMAN	887	4.364
TF3A_XENLA	344	4.467	RS15_PODAN	152	4.349
POL_SIVGB	1009	4.422	RL9_ARATH	197	4.336
TBP1_YEAST	434	4.421	SY61_DISOM	427	4.246
ORA_PLAFN	701	4.407	ADX_HUMAN	184	4.215
PIP1_DROME	1312	4.347	SYT1_RAT	421	4.171
ATPF_BACSU	170	4.322	SYT1_HUMAN	422	4.149
TPM4_RAT	248	4.294	TTL_PIG	379	4.138
ODO2_YEAST	475	4.291	DNIV_ECOLI	184	4.084
VSI1_REOVL	470	4.285	TTL_BOVIN	377	4.068
ADX_HUMAN	184	4.276	RL15_SCHPO	29	4.056
H2B_EMENI	139	4.241			
HLY4_ECOLI	478	4.199			
UBC6_YEAST	250	4.164			
BNC3_RAT	318	4.134			
PSBO_SPIOL	332	4.094			
RF3_YEAST	559	4.094			
MERA_BACSR	631	4.078			

6 Conclusion

In principle the hidden Markov model method is a simple and elegant way of modeling sequence families. For practical purposes, however, various problems arise, of which the most important ones were discussed in this paper. A variety of heuristics and extensions to overcome the problems which are all part of the SAM software package were presented. Most of these techniques were mentioned in our original paper (Krogh *et al.*, 1994a), but here we have discussed them in more detail, and treated it from a more practical perspective, whereas we have not focussed on the biological results at all.

The algorithm for estimating HMMs is a simple hill-climbing optimization technique which will very often find suboptimal solutions resulting in inferior models. Three methods were introduced to deal with the problem. Firstly, several different models can be trained and the best one selected, based on the over-all NLL score. Secondly, noise of slowly decreasing size can be added during the estimation process in a fashion similar to simulated annealing, and thirdly the surgery method for adding and deleting states from the model was shown to also help to obtain models with better NLL scores.

The general theory for HMMs does not tell how to choose the topology of the model. In our work we have chosen a model structure that we believe fits the biological sequences particularly well, and most of the probability parameters can be interpreted as penalties familiar from other alignment methods, such as gap penalties. For choosing the length of the model the above mentioned surgery heuristics was introduced, which deletes parts of the model not used very much and inserts more states where the existing states are 'over-loaded.'

In any parameter estimation process over-fitting is a danger, in particular when there are many parameters and little data. This can be overcome using Dirichlet and Dirichlet mixture prior distributions to regularize the models. They are particularly useful because they incorporate prior biological information and insight into the model but can be overcome by sufficient numbers of sequences.

For the problem of modeling domains and other subsequences we introduced the free insertion modules (FIMs). These modules treat the part of the sequences outside the domain as completely random sequences, and by an example it was shown that an HMM can locate domains automatically. By using several FIMs one can model proteins with many subdomains (always appearing in the same order). By using long backward transition, one can in principle model domains occurring several times in different order, but it is not currently implemented in the software package.

SAM is an evolving system. Future additions will include repeated motif location, alternate scoring methods using null models, subsequence to submodel training, sequence weighting, an improved user interface, and use of our new algorithm for parallel sequence alignment in limited space to greatly extend the capabilities of the MasPar code (Grice *et al.*, 1995).

7 Acknowledgments

We would like to thank Saira Mian for many valuable suggestions, and David Haussler and the rest of the computational biology group in Computer and Information Sciences UCSC for many suggestions and contributions. We also thank the anonymous referees for their excellent suggestions. This research was supported in part by NSF grants CDA-9115268 and BIR-9408579, a grant from CONNECT, Denmark, and a grant from the Novo-Nordisk Foundation. SAM source code, the experimental server, and SAM documentation can be accessed on the World-Wide Web at `http://www.cse.ucsc.edu/research/compbio/sam.html` or

by sending electronic mail to `sam-info@cse.ucsc.edu`.

References

Bairoch, A. & Boeckmann, B. (1994). The SWISS-PROT protein sequence data bank: current status. *Nucleic Acids Research,* **22**, 3578–3580.

Baldi, P., Chauvin, Y., Hunkapillar, T., & McClure, M. (1994). Hidden Markov models of biological primary sequence information. *Proceedings of the National Academy of Sciences,* **91**, 1059–1063.

Berger, J. (1985). *Statistical Decision Theory and Bayesian Analysis.* New York: Springer-Verlag.

Brown, M. P., Hughey, R., Krogh, A., Mian, I. S., Sjölander, K., & Haussler, D. (1993). Using Dirichlet mixture priors to derive hidden Markov models for protein families. In: *Proc. of First Int. Conf. on Intelligent Systems for Molecular Biology*, (Hunter, L., Searls, D., & Shavlik, J., eds) pp. 47–55, Menlo Park, CA: AAAI/MIT Press.

Bucher, P. & Bairoch, A. (1994). A generalized profile syntax for biomolecular sequence motifs and its function in automatic sequence interpretation. In: *Proc. Int. Conf. on Intelligent Systems for Molecular Biology* pp. 53–61, Stanford, CA: AAAI/MIT Press.

Eddy, S. (1995). Multiple alignment using hidden Markov models. In: *Proc. Int. Conf. on Intelligent Systems for Molecular Biology* pp. 114–120, Cambridge, England: AAAI/MIT Press.

Eddy, S., Mitchison, G., & Durbin, R. (1995). Maximum discrimination hidden Markov models of sequence consensus. *J. Comput. Biol.* **2**, 9–23.

Gribskov, M., McLachlan, A. D., & Eisenberg, D. (1987). Profile analysis: Detection of distantly related proteins. *Proceedings of the National Academy of Sciences of the United States of America,* **84**, 4355–4358.

Grice, J. A., Hughey, R., & Speck, D. (1995). Parallel sequence alignment in limited space. In: *Proc. of Third Int. Conf. on Intelligent Systems for Molecular Biology* , Menlo Park, CA: AAAI/MIT Press.

Haussler, D., Krogh, A., Mian, I. S., & Sjölander, K. (1993). Protein modeling using hidden Markov models: Analysis of globins. In: *Proceedings of the Hawaii International Conference on System Sciences* volume 1 pp. 792–802, Los Alamitos, CA: IEEE Computer Society Press.

Karplus, K. (1995). Regularizers for estimating distributions of amino acids from small samples. In: *Proc. of Third Int. Conf. on Intelligent Systems for Molecular Biology* , Menlo Park, CA: AAAI/MIT Press. Full version available as UCSC Technical Report UCSC-CRL-95-11.

Kirkpatrick, S., Jr., C. G., & Vecchi, M. (1983). Optimization by simulated annealing. *Science,* **220**.

Krogh, A., Brown, M., Mian, I. S., Sjölander, K., & Haussler, D. (1994a). Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology,* **235**, 1501–1531.

Krogh, A., Mian, I. S., & Haussler, D. (1994b). A hidden Markov model that finds genes in *e. coli* DNA. *Nucleic Acids Research,* **22**, 4768–4778.

Krogh, A. & Mitchison, G. (1995). Maximum entropy weighting of aligned sequences of proteins or dna. In: *Proc. of Third Int. Conf. on Intelligent Systems for Molecular Biology* , Menlo Park, CA: AAAI/MIT Press.

Kuriyan, J. & Cowburn, D. (1993). Structures of SH2 and SH3 domains. *Current Opinion in Structural Biology,* **3**, 828–837.

Lawrence, C., Altschul, S., Boguski, M., Liu, J., Neuwald, A., & Wootton, J. (1993). Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science,* **262** (5131), 208–14.

Lawrence, C. & Reilly, A. (1990). An expectation maximization (em) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins,* **7** (1), 41–51.

Nickolls, J. R. (1990). The design of the Maspar MP-1: A cost effective massively parallel computer. In: *Proc. COMPCON Spring 1990* pp. 25–28, Los Alamitos, CA: IEEE Computer Society.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE,* **77** (2), 257–286.

Santner, T. J. & Duffy, D. E. (1989). *The Statistical Analysis of Discrete Data.* New York: Springer Verlag.