Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computational Biology and Chemistry
Computational Biology and Chemistry 27 (2003) 77–84
www.elsevier.com/locate/compbiolchem

Brief communication

Hidden Markov models and optimized sequence alignments

L. Smith *, L. Yeganova, W.J. Wilbur

Computational Biology Branch, National Center for Biotechnology Information, National Library of Medicine, Rm. 614D, Bldg. 38A, 8600 Rockville Pike, Bethesda, MD 20894, USA

Received 8 November 2002; received in revised form 14 November 2002; accepted 14 November 2002

Abstract

We present a formulation of the Needleman–Wunsch type algorithm for sequence alignment in which the mutation matrix is allowed to vary under the control of a hidden Markov process. The fully trainable model is applied to two problems in bioinformatics: the recognition of related gene/protein names and the alignment and scoring of homologous proteins.

Keywords: Algorithm; Optimization; Gene synonyms; Sequence alignment

1. Introduction

The Needleman–Wunsch algorithm (Needleman and Wunsch, 1970) or the more efficient Gotoh version (Gotoh, 1982) are dynamic programing algorithms that align two sequences of symbols to achieve a minimum cost. The cost of an alignment is the sum of costs of the symbols that are paired by the alignment, where one or other symbol may be a 'gap.' These depend on a predetermined mutation matrix of symbol-pair cost values, and so the usefulness of the resulting alignment depends on how accurately the model and the values reflect the origin of the sequences.

We reformulate the alignment problem in terms of a random process with a probability model. Probabilities govern state transitions in a hidden Markov model and the output of paired and gapped sequence elements. This allows a sequence alignment to be the output of a hidden Markov process in a manner similar to the way speech is viewed as the output of a HMM part-of-speech tagger (Jurafsky and Martin, 2000) or an amino acid sequence is viewed as the output of an HMM protein family model (Krogh et al., 1994; Durbin et al., 1998). Our formulation is closely related to the pair HMMs described in (Durbin et al., 1998) chapter 4. However, we carry the methodology further in presenting a training algorithm for our HMM approach which is unique. For training, one specifies a collection of pairs of sequences, without any corresponding alignments. One also assigns some initialization of the parameter values (possibly random or uniform). The training then takes place iteratively to learn the parameters which will produce overall maximal forward probabilites for the set of training pairs.

As an application, we apply our methods to two problems of related sequence detection. One problem is from computational linguistics. Given a particular gene name occurring in the literature, how can one identify other forms of the same name or closely related names. The second problem arises in computational biology. Given a particular amino acid sequence in one organism how can one find its homolog in a different organism. In both cases we are able to demonstrate learning from training sets which generalizes to independent test sets.

2. Sequence alignment hidden Markov model

In this section, we describe a hidden Markov process controlling the alignment of two sequences.

* Corresponding author. Tel.: +1-301-594-2845; fax: +1-301-435-7794.
E-mail address: lsmith@ncbi.nlm.nih.gov (L. Smith).

1476-9271/02/$ - see front matter.
PII: S1476-9271(02)00096-8

2.1. Definition

Let S denote a finite alphabet and let * be a special symbol denoting a gap, and $S' = S \cup \{*\}$. If $x_1, \ldots, x_t \in S' \times S'$ then $x_1 \& \ldots \& x_t$ denotes the pair of sequences of symbols in S obtained by deleting the gap symbols. For example, with:

$$\begin{array}{ccccc} a & a & c & * & g \\ * & * & c & t & g \\ \hline x_1 & x_2 & x_3 & x_4 & x_5 \end{array}$$

then:

$$x_1 \& x_2 \& x_3 \& x_4 \& x_5 = \begin{bmatrix} a & a & c & g \\ c & t & g & \end{bmatrix}$$

Let $N \geq 1$ denote the number of states of a Markov process, with initial probabilities π_i and transition probabilities a_{ij} for $1 \leq i, j \leq N$. In each state i, there is a probability for the symbol-pair $x \in S' \times S'$ denoted by $b_i(x)$. If $X_1, \ldots, X_t$ are output pairs obeying this model, then O_t denotes the pair of sequences $X_1 \& \ldots \& X_t$. The random variables defining the state at each time are denoted by $q_1, \ldots, q_t$.

The probability of observing outputs $x_1, \ldots, x_t$ and states $i_1, \ldots, i_t$ is:

$$\Pr\begin{bmatrix} x_1 & x_2 & \ldots & x_t \\ i_1 & i_2 & \ldots & i_t \end{bmatrix} = \pi_{i_1} b_{i_1}(x_1) \cdot a_{i_1 i_2} b_{i_2}(x_2) \cdots a_{i_{t-1} i_t} b_{i_t}(x_t)$$

Therefore:

$$\Pr[O_t = O \text{ for some } t] = \sum_{t=1}^{\infty} \sum_{\substack{x_1,\ldots,x_t,i_1,\ldots,i_t \\ x_1 \& \ldots \& x_t = O}} \Pr\begin{bmatrix} x_1 & \ldots & x_t \\ i_1 & \ldots & i_t \end{bmatrix}$$

2.2. Forward–backward algorithm

The forward–backward and Viterbi algorithms can be derived following (Rabiner, 1989), modified for multiple output streams. Suppose that two sequences of symbols in S are observed, denoted A and B, with lengths $M = (M_1, M_2)$. A position in the observation is given by coordinates $r = (r_1, r_2)$ such that $0 \leq r_1 \leq M_1$ and $0 \leq r_2 \leq M_2$. The observation corresponding to the position r is the pair of sequences consisting of r_1 symbols from the first sequence, $A_1, \ldots, A_{r_1}$ and r_2 symbols from the second sequence, $B_1, \ldots, B_{r_2}$. This pair of sequences is denoted by $O(0 \to r)$.

The forward and backward variables can be defined for each position r and state i,

$$\alpha_r(i) = \Pr[O_t = O(0 \to r),\ q_t = i \text{ for some } t]$$

$$\beta_r(i) = \Pr[O_{t'} = O \text{ for some } t' \geq t | O_t = O(0 \to r),\ q_t = i \text{ for some } t] \quad (1)$$

We also need a notation for the various ways of moving from one position to another. A move, denoted by ϵ is one of (0, 1), (1, 0) or (1, 1) (this condition will be denoted by $0 \lneq \epsilon \leq 1$). From a given position r a move ϵ indicates a move from the position r to the position $r + \epsilon$, if this is valid. The output corresponding to this move is denoted by $O(r \to r + \epsilon)$ which is:

$$O(r \to r + \epsilon) = (*,\ B_{r_2+1}) \quad \text{if } \epsilon = (0,\ 1)$$

$$O(r \to r + \epsilon) = (A_{r+1},\ *) \quad \text{if } \epsilon = (1,\ 0)$$

$$O(r \to r + \epsilon) = (A_{r_1+1},\ B_{r_2+1}) \quad \text{if } \epsilon = (1,\ 1)$$

To further facilitate writing formulas, define:

$$\pi_r(i) = \Pr[O_{t-1} = O(0 \to r),\ q_t = i \text{ for some } t] \quad (2)$$

so that:

$$\pi_0(i) = \pi_i$$

$$\pi_r(i) = \sum_{j=1}^{N} \alpha_r(j) a_{ji}$$

and for convenience let $\pi_r(i) = 0$ if any coordinate of r is negative. Then the formulas (1) can be written:

$$\alpha_r(i) = \sum_{0 \lneq \epsilon \leq 1} \pi_{r-\epsilon}(i) b_i(O(r - \epsilon \to r))$$

$$\beta_r(i) = \sum_{0 \lneq \epsilon \leq 1} \sum_{j=1}^{N} a_{ij} b_j(O(r \to r + \epsilon)) \beta_{r+\epsilon}(j)$$

The probability of observing O is found from:

$$\Pr[O_t = O \text{ for some } t] = \sum_{i=1}^{N} \alpha_M(i) \quad (3)$$

2.3. Viterbi algorithm

The Viterbi algorithm computes the probability of the maximum path. This is done recursively by computing the maximum probability of reaching a position r and being in state i.

$$\delta_r(i) = \max_{\substack{t, x_1,\ldots,x_t, i_1,\ldots,i_{t-1} \\ x_1 \& \ldots \& x_t = O(0 \to r)}} \Pr\begin{bmatrix} x_1 & x_2 & \ldots & x_{t-1} & x_t \\ i_1 & i_2 & \ldots & i_{t-1} & i \end{bmatrix}$$

Define $\tilde{\pi}_r(i)$ to be the maximum probability of being in state i after being in position r. Then:

$$\tilde{\pi}_0(i) = \pi_i$$

$$\tilde{\pi}_r(i) = \max_j \delta_r(j) a_{ji}$$

Let $\tilde{\pi}_r(i) = 0$ if any coordinate of r is negative. Then the recursive expression for δ is:

$$\delta_r(i) = \max_{0 \lneq \epsilon \leq 1} \tilde{\pi}_{r-\epsilon}(i) b_i(O(r - \epsilon \to r))$$

The state sequence is derived from the computation of

δ. Let $\psi_r(i)$ record some pair (j, ϵ) that achieves the maximum in $\delta_r(i)$ and $\tilde{\pi}_r(i)$, and if the maximum occurs with $r=\epsilon$ then take $j=0$. To reconstruct the state sequence, let $T'=M_1+M_2$ and:

$$r_{T'}=M$$

$$q^*_{T'}=\underset{i}{\operatorname{argmax}}[\delta_M(i)]$$

Then for $t=T'-1, \ldots$ and until $q^*_t=0$, let:

$$(q^*_t, \epsilon^*_{t+1})=\psi_{r_{t+1}}(q^*_{t+1})$$

$$r_t=r_{t+1}-\epsilon^*_{t+1}$$

$$x^*_{t+1}=O(r_t \to r_{t+1})$$

At the end of the algorithm, both $q^*_t=0$ (undefined state) and $r_t=0$ and with this value of t let $T=T'-t$ and relabel the sequences x^*, q^* so that they begin at $t=1$ and end at $t=T$.

2.4. Training

In order to train we need to estimate the probability of a particular output and a particular transition from each state in the model, given the observed sequences. Let:

$$\xi^1_r(i,\ \epsilon)=\Pr[O_t=O(0 \to r),\ X_t=O(r-\epsilon \to r),\ q_t=i \text{ for some } t \le t' | O_{t'}=O \text{ for some } t']$$

Using $\pi_r(i)$ from (2):

$$\xi^1_r(i,\ \epsilon)=\pi_{r-\epsilon}(i)b_i(O(r-\epsilon \to r))\beta_r(i)/\Pr[O] \quad (4)$$

where the denominator is $\Pr[O_t=O \text{ for some } t]$ computed, for example, using (3). We also need:

$$\xi^2_r(i,\ j)=\Pr[O_t=O(0 \to r),\ q_t=i,\ q_{t+1}=j,\ \text{for some } t \le t' | O_{t'}=O \text{ for some } t']$$

which is found to be:

$$\xi^2_r(i,\ j)=\sum_{0 \lneq \epsilon \le 1} \alpha_r(i)a_{ij}b_j(O(r \to r+\epsilon))\beta_{r+\epsilon}(j)/\Pr[O] \quad (5)$$

Using expressions (4) and (5), the training formulas are:

$$\bar{\pi}_i \propto \sum_{0 \lneq \epsilon \le 1} \xi^1_\epsilon(i,\ \epsilon)$$

$$\bar{b}_i(x) \propto \sum_{\substack{0 \lneq \epsilon \le r \le M \\ O(r-\epsilon \to r)=x}} \xi^1_r(i,\ \epsilon)$$

$$\bar{a}_{ij} \propto \sum_{0 \lneq r \le M} \xi^2_r(i,\ j)$$

where the proportionality signs are used to indicate that the estimates are to be normalized to define probabilities.

When there are multiple sequence pairs observed in training, each gives rise to $\xi^1_r(i, \epsilon)$ and $\xi^2_r(i, j)$. These are added together in the reestimation and the totals are normalized to define probabilities.

In programing HMM algorithms, some form of scaling is used to prevent floating point underflow in the forward–backward algorithm. Scaling does not conveniently apply to these algorithms, and an alternative solution is to use a nonstandard floating point representation which allows a very large exponent.

3. Gene synonyms

Identifying and tagging gene names in scientific literature is a prerequisite for computer understanding of genetics and molecular biology text (Tanabe and Wilbur, 2002). One aspect of this problem is associating expressions referring to genes that are spelled, punctuated, or phrased differently. This was addressed in (Krauthammer et al., 2000) by translating names to false DNA sequences and making use of BLAST, a fast partial matching search algorithm. In this example, we treat names as sequences in their original alphabet and train a Markov model for sequence alignment to 'align' and score potential synonyms. We expect this approach to succeed by capitalizing on the fact that some characters are less critical, particularly punctation, to the meaning of a gene name phrase, as was analyzed systematically in (Cohen et al., 2002).

3.1. Training

Out of a list of approximately 42 000 gene names, we found that 3754 of these names occurred in from 100 to 10 000 MEDLINE documents. We began a process for each gene name of identifying closely related names in the MEDLINE database. If ph denotes the gene name, let X_{ph} denote the set of MEDLINE documents containing ph. Then we may denote the set of MEDLINE documents not containing ph by $MED-X_{ph}$. We applied naive Bayesian learning with words and MeSH terms as features to learn the difference between X_{ph} and $MED-X_{ph}$. This learning was then applied to score $MED-X_{ph}$ and the high scoring documents formed a set Y_{ph} in which we presumed the gene denoted by ph would likely be discussed but under a different name. We then applied the ABGene tagger (Tanabe and Wilbur, 2002) to all the documents in Y_{ph} to produce a set of potential gene names GN_{ph}. This set of terms contains generally much more than gene names closely related to ph. We finally applied a relatively naive matching algorithm to extract from GN_{ph} those names likely to be closely related to ph. Since this was an imperfect process, human review and correction was applied as a final step to produce a subset of GN_{ph}

which are gene names closely related to ph. We denote this final set by GR_{ph}. Since this is labor intensive, at the time of this writing, we have processed 29 gene names from the set of 3754. On average, it was found that over the set of 29 gene names, GR_{ph} was 0.8% of GN_{ph}.

From the 29 original gene names, 20 were selected at random, ph_i $i = 1, \ldots, 20$, and a training set was created that consisted in up to 300 random pairs of phrases sampled from each GR_{ph_i}. For example one of the 20 gene names is 'adrenergic receptor'. For this gene name $GR_{adrenergic\ receptor}$ contained 1273 names. Some of the 300 pairs that were randomly chosen for this gene name group are:

adipocyte beta-adrenergic receptors
human alpha 1 adrenoceptors

7000 beta-adrenergic receptors
smooth muscle alpha-1 adrenoceptors

cardiac beta-adrenergic receptors
beta 2-like adrenoceptors

Another gene name in the set of 20 is 'ccr4'. Here GR_{ccr4} contains only three names so only three pairs can be produced and no random selection is necessary. All three pairs are included in the training set. The result of this process is a training set of 3737 pairs of related gene names.

The training set was used to train three different models. The first model has only one state (a degenerate case in which only the outputs of the single state are trained and there are no state transitions). The remaining models have three states, and in both of these state 2 allows only gaps in the first string, and state 3 allows only gaps in the second string. The difference between these two models is that in one model (the 'pure' model), state 1 allows only matches to occur, whereas in the other model (the 'mixed' model), state 1 allows both matches and gaps. None of the models allow mismatches. All of these models are further constrained to be symmetric. In all models, state 1 satisfies $b_1(x, y) = b_1(y, x)$ where x, y are any letters or *. In the three state models, the symmetry requires identical probabilities when the outputs are interchanged along with interchanging states 2 and 3. That is, in addition to the symmetry in state 1, $b_2(x, y) = b_3(y, x)$ and $a_{12} = a_{13}$, $a_{21} = a_{31}$, $a_{22} = a_{33}$, and $a_{23} = a_{32}$.

3.2. Results

The log (base 10) of the probability of the entire set of 3737 synonym pairs achieved after training was $-250\,655$ for the one state model, $-225\,530$ for the three state pure model, and $-224\,093$ for the three state mixed model. The trained state transition probabilites are shown in Fig. 1 and the output probabilities are summarized in Table 1. As expected, the probability is higher in the three state models, and the added flexibility of the mixed three state model gives it an advantage over the pure model.

Our objective is not to learn the peculiarities of particular gene names or classes of names, but to learn over a large sample of name pairs what is not important in a gene name as opposed to what signals a difference in meaning. As expected, we found that the models learned that some characters are more important than others when comparing synonyms. For example, in the one state model the probability of a gapped 'e' is 0.7134 times the probability of a matching 'e' pair. Thus if an 'e' is missing from one phrase the probability of alignment is decreased by a factor of 0.7134. Compare this with the space character which is 1.1617 times more likely to occur as a gap than a match (together, the 'e' and space are the most frequently occuring characters, appearing in 18.9% of all pairs).

An example of the difference in the performance between the one and three state models is shown in the alignment of 'phospholipase c-coupled bombesin receptors' and 'subtype 2 bombesin receptors' illustrated in Table 2. The three state model is able to identify words inserted in one or the other phrase, and these are accurately grouped together. The one state model produces a meaningless alignment of the word 'subtype' with a gapped version of the word 'phospholipase'.

These trained models can be used to compute the probability of observing a given pair of phrases. This probability can then be used to construct a score measuring the relatedness of two phrases. This approach allowed us to carry out a retrieval experiment testing how well we could identify related gene/protein names. We used the nine sets GR_{ph} that were excluded from training. For each of these ph, we paired ph with every element $x \in GN_{ph}$ and computed the probability that the model would produce the pair $(ph,\ x)$. We also computed the probability that the model would produce (null, x). This allowed us to write:

$$\text{score}(x) = \log_{10} \Pr(ph,\ x) - \log_{10} \Pr(\text{null},\ x)$$

The scoring produces a ranking of the names in GN_{ph} and we can compute the precision and recall where GR_{ph} are the true positives. Results over all nine sets were averaged to produce the precisions at 11 different recall points, and the 11 point average precision as a summary value (Witten et al., 1999) in Table 3.

In this data it is clearly seen that the three state models far outperform the one state model, as one might expect. The mixed model performs slightly better than the pure model.

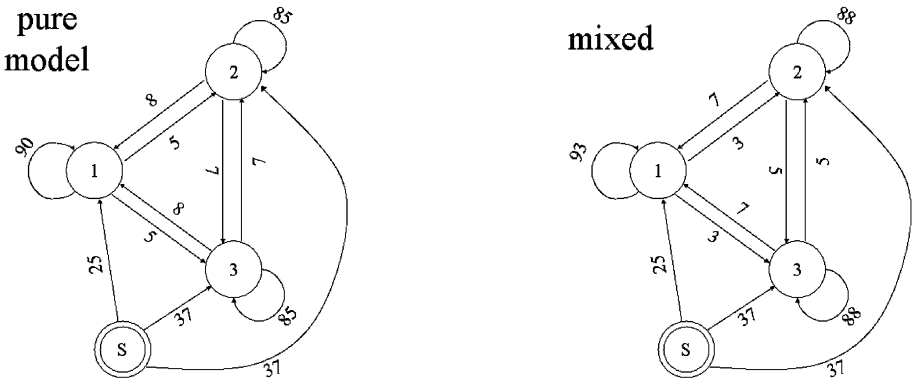

Fig. 1. State transitions and probabilities (× 100) of the two 3 state models trained on 3737 gene synonym pairs.

Table 1
Output probabilities of classes of output (match, mismatch, and gap) for each state of three models trained on 3737 gene synonym pairs

Model	State	Match	Mismatch	Gap in first	Gap in second
One state	1	0.36	0	0.32	0.32
Three state (pure)	1	1.0	0	0	0
	2	0	0	1.0	0
	3	0	0	0	1.0
Three state (mixed)	1	0.92	0	0.039	0.039
	2	0	0	1.0	0
	3	0	0	0	1.0

4. Sequence homologies

Sequence alignment is a technique used in molecular biology to perform research on related biological sequences. The standard algorithm for globally aligning and scoring sequences is the Needleman–Wunsch algorithm (Needleman and Wunsch, 1970) or the more efficient Gotoh algorithm (Gotoh, 1982). The quality of alignments and scores produced by the algorithm depends critically on cost parameters associated with matches, mismatches, and gaps in the alignment. Our approach gives a new method for estimating these parameters, i.e. the probabilities $b(x, y)$, and further extends the alignment model to allow multiple parameter sets (corresponding to states) selected by a hidden Markov process, and enables learning of these extended parameters. We will illustrate this by taking a training set of homologous protein sequence pairs from human and mouse.

4.1. Training

The training set consisted of homologous human and mouse proteins that were obtained by the following

Table 2
The optimal alignments of the gene synonyms 'phospholipase c-coupled bombesin receptors' and 'subtype 2 bombesin receptors' produced by one state and three state (mixed) models trained on 3737 gene synonym pairs

model	prob	alignment
1 state	-86.3	phos****pholipase c-coupled* bombesin receptors 1111111111111111111111111111111111111111111111111 ***subtyp*******e *********2 bombesin receptors
3 state (mixed)	-74.9	phospholipase c-coupled********* bombesin receptors 333333333333333333333222222222111111111111111111 **********************subtype 2 bombesin receptors

The prob column is the log (base 10) of the probability of the alignment in that model. The sequence of states is shown between each pair of aligned sequences.

Table 3
Precision achieved in identifying synonyms from nine gene synonym sets for three models trained on 3737 gene synonym pairs and for recall levels from 0 to 100%

Recall	One state	Three state (pure)	Three state (mixed)
0	0.526741	1	0.977778
10	0.508618	0.890567	0.917662
20	0.463762	0.867323	0.893462
30	0.400175	0.79764	0.831689
40	0.396255	0.746139	0.82642
50	0.391586	0.699388	0.717144
60	0.37287	0.69152	0.703653
70	0.360286	0.671416	0.700752
80	0.34832	0.559857	0.581367
90	0.228984	0.550942	0.561543
100	0.174817	0.495818	0.495418
Average	0.37931	0.724601	0.746081

Precision at 0 recall is simply the highest precision seen at any point in the ranking (Witten et al., 1999).

procedure. First, a list of 400 frequently searched genes was obtained from Genecards[1]. For each of these genes, LocusLink[2] was used to identify a LocusId of the human gene. The HomoloGene database[3] was then used to find the LocusId of homologous mouse (Mus musculus) genes. Given the LocusId of homologous genes, the corresponding proteins were found using the RefSeq database[4]. This resulted in 145 homologous protein sequences which were downloaded using Entrez[5].

These homologs were then aligned for preliminary evaluation using untrained probabilities that give the probability of a match that is ten times the probability of a mismatch or a gap ($b(x, x) = 0.016129$ and $b(x, y) = 0.001629$ for $x \neq y$)[6]. To reduce the computer memory required to execute the training algorithms, those homologs that had an alignment length of 2000 or more were removed from the set, leaving 138 homologs. Finally, to help ensure the quality of the homologs in the training set, each homolog was scored by taking the logarithm of the probability of the optimal alignment (untrained) divided by the length of the optimal alignment. A cutoff value of -2 was arbitrarily chosen with a further limitation of 500 in the length of the optimal alignment which resulted in a training set of 49 homologs. As a consequence, these 49 were more closely related than the remaining 89.

In order to evaluate the effect of multiple states, four different models were trained using the training set of 49 homologs, corresponding to one, two, three or four states in the underlying model. Unlike the gene synonym problem, we do not ask for symmetry in these models, rather we want to take advantage of specific differences in the sequences to improve the specifies-specific alignment. However, we do give states special meaning to help guide the interpretation of the results. The one state model permits any output (match, mismatch, or gap). The two state model permits a match or mismatch output from the first state (no gaps) and only gaps in the second state. In the three and four state models, the first state permits any output (match, mismatch, or gap), and the last state permits any match or mismatch. In the three state model, state 2 only permits gaps in either sequence. The four state model has two states that allow gaps, state 2 only permits outputs that gap the human protein, i.e. (*, x) and state 3 only permits outputs that gap the mouse protein, i.e. (x, *). Other than these restrictions, the probabilities were not constrained during training.

We also compared trained parameters with published mutation matrices. For this evaluation we trained a two state model on the full set of 138 homologs (those with optimal alignments not exceeding 2000 in length). The model was constrained similarly to the two state model described above except that it was also required to be symmetric, $b_i(x, y) = b_i(y, x)$. This model corresponds to the method of aligning sequences using a 20×20 mutation matrix together with a model for gap penalties. The output probabilities of state 1 in this model correspond precisely to a 20×20 matrix, while state 2 defines the penalties for single gaps and the Markov process models the overall gap penalty in the transition probabilities.

4.2. Results

The log (base 10) of the probability of the entire set of 49 homologs achieved after training of four models is shown in Table 4. This table shows that the overall performance of the models with two or more states is superior to the one state model on both the 49 training

Table 4
Performance of an untrained model and four models trained on 49 human/mouse protein homolog pairs

Model	Training total	Testing total
Untrained	−31 984	−11 7789
One state	−25 221	−109 189
Two state	−24 855	−98 467
Three state	−24 861	−98 540
Four state	−24 766	−98 159

The totals are the sum of all log (base 10) probabilities of the homologs in each set (49 homologs in the training set and 89 homologs in the test set).

[1] http://www.bioinformatics.weizmann.ac.il/cards.
[2] http://www.ncbi.nlm.nih.gov/LocusLink.
[3] http://www.ncbi.nlm.nih.gov/HomoloGene.
[4] ftp://www.ftp.ncbi.nih.gov/refseq/LocusLink/loc2ref.
[5] http://www.ncbi.nlm.nih.gov/Entrez.
[6] This is equivalent to a classic Needleman–Wunsch alignment with a cost of 1.79 for matches and 2.79 for nonmatches and gaps.

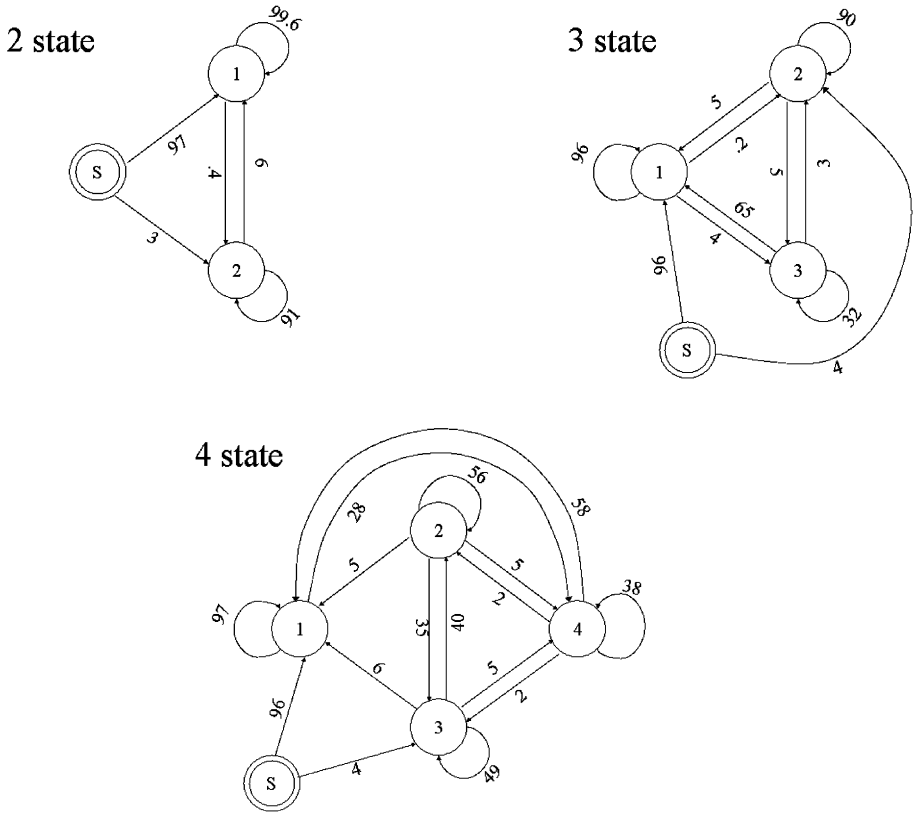

Fig. 2. State transitions and probabilities ($\times$ 100) of the 2, 3 and 4 state models trained on 49 human/mouse protein homolog pairs.

homologs and the remaining 89 homologs. The trained state transition probabilites are shown in Fig. 2 and the output probabilities are summarized in Table 5.

A comparison is made between the logarithm of the output probabilities (from state 1) of the symmetric two state model trained on 138 homologs with published mutation matrices. Following (Tomii and Kanehisa, 1996), we computed the correlation coefficients between the the log probabilities and the mutation matrices made available by those authors, 56 out of 83 of which were symmetric 20×20 matrices. The correlations were performed over all 210 entries. Any correlation observed that is greater (in absolute value) than 0.135 is statistically significant with $P < 0.05$. The average of the absolute value of correlation was found to be 0.4164 and 54 out of 56 were statistically significant. For example, the overall correlation with the Dayhoff PAM250 matrix was 0.5681, and the three BLOSUM series matrices had mean correlation of 0.4860. Generally, the correlation with matrices obtained from observed amino acid exchanges were better than those obtained from other methods, such as using physico-

Table 5
Output probabilities of classes of output (match, mismatch, and gap) for each state of an untrained model and four models trained on 49 human/mouse protein homolog pairs

Model	State	Match	Mismatch	Gap (human)	Gap (mouse)
Untrained	1	0.32	0.61	0.03	0.03
One state	1	0.91	0.08	0.008	0.005
Two state	1	0.93	0.07	0	0
	2	0	0	0.526	0.474
Three state	1	0.97	0.03	0.0002	0.0002
	2	0	0	0.533	0.467
	3	0.19	0.81	0	0
Four state	1	0.96	0.04	0.002	0.0001
	2	0	0	1.0	0
	3	0	0	0	1.0
	4	0.18	0.82	0	0

chemical properties of amino acids, as might be expected since our data is derived from homologies.

5. Discussion

The model presented provides two new benefits to modeling paired sequences. First, it extends the classical Needleman–Wunsch type algorithm to allow multiple mutation matrices selected by a hidden Markov process. And second, it provides a way for estimating the parameters of the model given a training set of paired sequences. No actual alignments need be specified as a starting point, only pairs which are to be aligned. We have described two very different applications, and there are many other possibilities. This algorithm is extendable to multiple sequence alignment and training, just as the Needleman–Wunsch type algorithm has been extended to align multiple sequences.

A practical limitation of the approach is the time and memory required to carry out the algorithms, especially when many iterations are required to achieve convergence. Two factors mitigate this limitation. First, in the applications presented here we found two to six iterations adequate for convergence. Second, the increasing speed of computers is making such calculations more feasible. In our evaluation we used a Dell PowerEdge 2550 computer with dual 1.26 GHz Pentium III processors and 2 Gbyte RAM, running the LINUX 2.4.17 operating system, and the time per iteration ranged from a few minutes for the one state models up to about an hour for the four state model.

The running time for a single training iteration with a single observed pair is proportional to the product of the number of states, N, and the lengths of the observed sequences, M_1 and M_2. But of course actual running time depends on the speed of the computer. On the other hand, the computer memory required can be reliably predicted. For each single training iteration with a single observed pair, the space required for each α, β, δ, is $(M_1+1)(M_2+1)N$ numbers, ψ requires $3(M_1+1)(M_2+1)N$ numbers, ξ^1 requires $4(M_1+1)(M_2+1)N$ numbers, and ξ^2 requires $(M_1+1)(M_2+1)N^2$. A floating point number is sufficient for the δ and ξ arrays, while a single byte may suffice to store ψ. However, since a large exponent implementation is required in computing α and β these numbers will require a floating point representation of a mantissa together with a numeric (integer) exponent. Assuming 8 bytes for floating point numbers and 4 bytes for integer numbers, the total space requirement in bytes is:

$$(80 + 8 \cdot N)(M_1 + 1)(M_2 + 1)N$$

During training and evaluation model probabilities require storage of $N^2+2N+NS^2$ floating point numbers where N is the number of states and S is the size of the alphabet. Normally, this space requirement is small compared with the space requirement needed for the training algorithms on each observation.

The gene synonym and protein homolog data sets developed for this study and an implementation of the algorithms in a C++ library are available for download[7].

References

Cohen, K.B., Dolbey, A.E., Acquaah-Mensah, G.K., Hunter, L., 2002. Proceedings of the 40th ACL Workshop on NLP in the Biomedical Domain, pp. 14–20.
Durbin, R., Eddy, S., Krogh, A., Mitchison, G., 1998. Biological Sequence Analysis. Cambridge University Press.
Gotoh, O., 1982. J. Mol. Biol. 162, 705–708.
Jurafsky, E., Martin, J.H., 2000. Speech and Language Processing. Prentice Hall, New Jersey.
Krauthammer, M., Rzhetsky, A., Morozov, P., Friedman, C., 2000. Gene 259 (1–2), 245–252.
Krogh, A., Brown, M., Mian, I., Sjolander, K., Haussler, D., 1994. J. Mol. Biol. 235, 1501–1531.
Needleman, S.B., Wunsch, C.D., 1970. J. Mol. Biol. 48, 443–453.
Rabiner, L.R., 1989. Proc. IEEE 77 (2), 257–286.
Tanabe, L., Wilbur, J., 2002. Bioinformatics 18, 1124–1132.
Tomii, K., Kanehisa, M., 1996. Protein Eng. 9 (1), 27–36.
Witten, I.H., Moffat, A., Bell, T.C., 1999. Managing Gigabytes. Morgan Kaufmann, San Francisco, pp. 188–190.

[7] ftp://www.ftp.ncbi.nlm.nih.gov/pub/lsmith.