# Evaluating Regularizers for Estimating Distributions of Amino Acids

Kevin Karplus

Board of Studies in Computer Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064

karplus@cse.ucsc.edu
(408)459-4250
fax: (408)459-4829

## Abstract

This paper makes a quantitative comparison of different methods, called *regularizers*, for estimating the distribution of amino acids in a specific context, given a very small sample of amino acids from that distribution. The regularizers considered here are *zero-offsets*, *pseudocounts*, *substitution matrices* (with several variants), and *Dirichlet mixture regularizers*.

Each regularizer is evaluated based on how well it estimates the distributions of the columns of a multiple alignment—specifically, the expected encoding cost per amino acid using the regularizer and all possible samples from each column.

In general, pseudocounts give the lowest encoding costs for samples of size zero, substitution matrices give the lowest encoding costs for samples of size one, and Dirichlet mixtures give the lowest for larger samples. One of the substitution matrix variants, which added pseudocounts and scaled counts, does almost as well as the best known Dirichlet mixtures, but with a lower computation cost.

**Keywords:** regularizers, entropy, encoding cost, pseudocounts, Gribskov average score, substitution matrices, data-dependent pseudocounts, Dirichlet mixture priors

## 1 Why estimate amino acid distributions?

Most search and comparison algorithms for proteins need to estimate the probabilities of the twenty amino acids in a given context. This probability is often expressed indirectly as a *score* for each of the amino acids, with positive scores for expected amino acids and negative scores for unexpected ones.

As Altschul pointed out [1], any alignment-scoring system is really making an assertion about the probability of the test sequences given the reference sequence. The score for an alignment is the sum of the scores for individual matched positions, plus the costs for insertions and deletions. For each match position, there are twenty scores—one for each of the possible amino acids in the test sequence. Each match score can be interpreted as the logarithm of the ratio of two estimated probabilities: the probability of the test amino acid given the amino acid in the reference sequence and the probability of the test amino acid in the background distribution. If we define $\hat{P}_t(i)$ as the estimated probability of amino acid $i$ in position $x$ and $\hat{P}_0(i)$ as the estimated background probability in any position, then the score for $i$ in column $t$ is $\log_b(\hat{P}_t(i)/\hat{P}_0(i))$ for some arbitrary logarithmic base $b$ [1].

Any method for estimating the probabilities $\hat{P}_t(i)$ and $\hat{P}_0(i)$ defines a match-scoring system. Rather than looking at the final scoring system, this paper will concentrate on methods that can be used for estimating the probabilities themselves.

In more sophisticated models than single sequence alignments, such as multiple alignments, profiles [7], and hidden Markov models [14, 3], we may have more than one reference sequence in our training set. Each position in such a model defines a context for which we need to estimate the probabilities of the twenty amino acids. In this paper, $s$ refers to a sample of amino acids from a column and $s(i)$ to the number of times that amino acid $i$ appears in that sample. Our problem, then, is to compute the estimated probabilities $\hat{P}_s(i)$ for the context from which sample $s$ was taken, given only the twenty numbers $s(i)$.

For alignment and search problems, we usually add scores from many positions, and so fairly small improvements in computing individual match scores can add up to significant overall differences. For example, the small differences between the PAM and BLOSUM scoring matrices have been shown to make a significant difference in the quality of search results [9].

The differences between regularizers are often fairly small; this paper attempts to quantify these small differences for several regularizers. Section 2 explains the measure used to quantify the tests; Section 3 explains the notion of posterior counts; Section 4 describes the data used for training and testing; and Section 5 presents the different methods and quantitative comparisons of them.

## 2 Quantitative measure for regularizers

The traditional method used in computational biology to demonstrate the superiority of one technique to another is to compare them on a biologically interesting search or alignment problem. Many of the regularizers in Section 5 have been validated in this way [9, 4, 18]. This sort of anecdotal evidence is valuable for establishing the usefulness of techniques in real biological problems, but is very difficult to quantify.

In this paper, regularizers are compared quantitatively on the problem of encoding the columns of multiple alignments. This generic problem has some attractive features:

- Large data sets of multiple alignments are available for training.
- Many techniques that use regularizers also produce multiple alignments, and regularizers that produce good encodings should be good regularizers for these algorithms.
- By using trusted alignments, we can have fairly high confidence that each amino acid distribution we see is for amino acids from a single biological context, and not just an artifact of a particular search or alignment algorithm.

The encoding cost (sometimes called *conditional entropy*) is a good measure of the variation among sequences of the multiple alignment. Since entropy is additive, the encoding cost for independent columns can be added to get the encoding cost for entire sequences, and strict significance tests can be applied by looking at the difference in encoding cost between a hypothesized model and a null model [16].

Each column $t$ of a multiple alignment will give us a (possibly weighted) count of amino acids, $F_t(i)$. If we write the sum of all the counts for a column as $|F_t|$, we can estimate the probability of each amino acid in the column as $\hat{P}_t(i) = F_t(i)/|F_t|$. This is known as the *maximum-likelihood estimate* of the probabilities. Note: throughout this paper the notation $|y|$ will mean $\sum_{\text{amino acid } i} y(i)$ for any vector $y$.

To evaluate the regularizers, we want to see how accurately they predict the true probabilities of amino acids, given a sample. Unfortunately, true probabilities are never available with finite data sets. To avoid this problem, we will take a small sample of amino acids from the column, apply a regularizer to it, and see how well the regularizer estimates the maximum-likelihood probabilities for the whole column.

Let's use $s(i)$ to be the number of occurrences of amino acid $i$ in the sample. The estimated probability of amino acid $i$ given the sample $s$ will be written as $\hat{P}_s(i)$, and the *Shannon entropy* or *encoding cost* of amino acid $i$ given the sample is $-\log_2 \hat{P}_s(i)$. The average encoding cost for column $t$ given sample $s$ is the weighted average over all amino acids in the column of the encoding for that amino acid:

$$H_s(t) = -\sum_i \frac{F_t(i)}{|F_t|} \log_2 \hat{P}_s(i) \ .$$

The better the estimation $\hat{P}_s(i)$ is of $\hat{P}_t(i)$, the lower the encoding cost $H_s(t)$ will be. The lowest possible value would be obtained if the estimate were exact:

$$H_{\min}(t) = -\sum_i \frac{F_t(i)}{|F_t|} \log_2 \frac{F_t(i)}{|F_t|} \ .$$

To make a fair comparison of regularizers, we should not look at a single sample $s$, but at the expected value when a sample of size $k$ is chosen at random:

$$H_k(t) = \sum_{\text{sample } s, |s|=k} P(s|t) H_s(t) \ .$$

The weighting for each of the encoding costs $H_s(t)$ is the probability of obtaining that particular sample from that column. If the samples of size $|s|$ are drawn by independent selection with replacement from the density $\hat{P}_t$, then the probability of each sample can be computed from the counts $F_t$:

$$\begin{aligned} P(s|t) &= |s|! \prod_i \hat{P}_t(i)^{s(i)}/s(i)! \\ &= |s|! |F_t|^{-|s|} \prod_i F_t(i)^{s(i)}/s(i)! \ . \end{aligned}$$

We can do a weighted average of the encoding costs over all columns to get the expected cost per amino acid for a given sample size:

$$H_k = \frac{\sum_{\text{column } t} H_k(t)}{\sum_{\text{column } t} |F_t|}$$

If we precompute the total count $T = \sum_{\text{column } t} |F_t|$, and *summary frequencies* for each sample

$$T_s(i) = \sum_{\text{column } t} P(s|t) F_t(i) \ ,$$

then we can rewrite the computation as

$$H_k = -\frac{1}{T} \sum_{\text{sample } s, |s|=k} \sum_i T_s(i) \log_2 \hat{P}_s(i) \ .$$

The average encoding cost $H_k$ would be minimized if $\hat{P}_s(i) = T_s(i)/|T_s|$, giving us a lower bound on how well a regularizer can do for samples of size $k$. Table 1 shows this minimum average encoding cost for the columns

| sample size<br>$\lvert s\rvert$ | encoding cost in bits<br>$H_{\lvert s\rvert}$ | relative encoding cost in bits<br>$H_{\lvert s\rvert-1} - H_{\lvert s\rvert}$ |
|---|---|---|
| 0 | 4.19666 | |
| 1 | 2.78084 | 1.41582 |
| 2 | 2.38691 | 0.39393 |
| 3 | 2.16913 | 0.21778 |
| 4 | 2.02703 | 0.14210 |
| 5 | 1.92380 | 0.10323 |
| full | 1.32961 | |

Table 1: Encoding cost of columns from the weighted BLOCKS database, given that a sample of $|s|$ amino acids is known. The encoding cost is a lower bound on the encoding cost for any regularizer. The last row (labeled "full") is the encoding cost if we know the distribution for each column of the alignment exactly, not just a sample from the column. The relative encoding cost is the information gain from seeing one more amino acid.

of the BLOCKS multiple alignment [8] (with sequence weights explained in Section 4), given that we have sampled $|s|$ amino acids from each column.

The last row of the table is the average encoding cost for the columns if we use the full knowledge of the probabilities for the column $\hat{P}_t$, rather than just a random sample. This is the best we can hope to do with any method that treats the columns independently. It is probably not obtainable with any finite sample size, but we can approach it if we use information other than just a sample of amino acids to identify the column. The relative entropy in the last column of Table 1 measures how much information we have gained by seeing one more amino acid.

One disadvantage of the encoding cost computation used in this paper is the cost of pre-computing the $T_s(i)$ values and computing the $\hat{P}_s(i)$ values for each of the possible samples. The number of distinct samples to be examined is $\binom{20+|s|-1}{|s|}$, which grows exponentially with $|s|$, but remains manageable for $|s| \leq 5$ (42,504 distinct samples for $|s| = 5$).

## 3 Posterior Counts

Section 2 introduced the maximum-likelihood method for estimating probabilities from counts, $\hat{P}_s(i) = s(i)/|s|$. Maximum likelihood is asymptotically optimal as $|s| \to \infty$, but performs very badly for small sample sizes, since the encoding cost for any amino acid not seen in the sample is $-\log_2 0 = \infty$. To avoid this infinitely high cost, we will constrain regularizers to provide non-zero estimates for all probabilities: $0 < \hat{P}_s(i)$.

Regularizers can be viewed as making an adjustment to the sample counts, $s$, to produce *posterior counts*, $X_s$, from which we estimate the probability:

$$\hat{P}_s(i) = \frac{X_s(i)}{|X_s|} .$$

To get legal estimated probabilities and avoid infinite costs, the primary constraint is $X_s(i) > 0$.

Note: there will be several different formulas given for computing $X_s$, corresponding to different regularizers. The symbols $X_s(i) \leftarrow$ will be used for defining the different methods. The notations $P_0(i)$ and $\hat{P}_0(i)$ refer to the background probabilities and their estimates (that is, the probabilities given a sample of size zero).

Once we have decided that the goal is to minimize the average encoding cost of the columns, and chosen a method to try, we can try to optimize the parameters of the method, using Newton's method to find parameter values at which all the first derivatives of the encoding cost are zero and all the second derivatives are positive (other gradient descent algorithms could also be used).

We can compute the derivatives of the encoding cost $H_k$ with respect to any parameter fairly easily from $X_s$ and its derivatives. For many of the methods, the second derivative of $X_s$ is 0 for all the parameters, further simplifying the optimization. For a more complete discussion of optimization, see [13].

## 4 Experimental method

The regularizers are compared by computing the average entropy $H_k$ of a multiple alignment given a regularizer. The multiple alignments chosen are from the BLOCKS database [8]. The sequences are weighted using a slight variant of the Henikoffs' position-specific weighting scheme [10], as implemented by Kimmen Sjölander. Her weighting scheme is proportional to the Henikoffs' position-specific weights, but instead of having the weights sum to 1.0 for each block, they sum to the number of sequences in the block, so that blocks containing more sequences have more influence than blocks with only a few sequences.

The weighting scheme attempts to reduce the sampling bias in the database, by reducing the weight of sequences that are similar to others, and increasing the weight of outliers. Other weighting schemes have been proposed for this purpose (for example, tree distances [20] or weighting for pairs of alignments [2]); I chose the position-specific one rather arbitrarily for its ease of computation.

## 5 Results for training and testing on full database

This section reports the average encoding costs obtained for different sample sizes and different regularizers. For each regularizer and sample size, the *excess entropy* is reported, that is, the difference between the

| $\|s\|$ | excess entropy | | | |
|---|---|---|---|---|
| | | optimized for | | |
| | | $\|s\| = 1$ | $\|s\| = 2$ | $\|s\| = 0, 1, 2, 3$ |
| | $z = 1$ | $z = 0.04851$ | $z = 0.05420$ | $z = 0.05260$ |
| 0 | 0.12527 | 0.12527 | 0.12527 | 0.12527 |
| 1 | 1.07961 | 0.20482 | 0.20677 | 0.20585 |
| 2 | 1.17080 | 0.18636 | 0.18457 | 0.18470 |
| 3 | 1.16489 | 0.16843 | 0.16587 | 0.16626 |
| 4 | 1.13144 | 0.15311 | 0.15063 | 0.15105 |
| 5 | 1.09164 | 0.14203 | 0.13989 | 0.14026 |
| full | 0.84541 | 0.08013 | 0.08884 | 0.08653 |

Table 2: Excess entropy for the zero-offset regularizers. The popular "add-one" regularizer is clearly a poor choice for the BLOCKS database.

average encoding cost per column using the regularizer and the average encoding cost per column using the best theoretically possible optimizer. For the entire database, the best possible encoding costs are reported in Table 1.

The last row of each of Tables 2 through 8 reports the excess entropy if the full column $F_t$ is given to the regularizer, rather than a sample $s$. Since the entropy for the column is minimized if the $X_t(i)$ values exactly match the observed counts $F_t(i)$, this measures how much the regularizer distorts the data. Because some of the columns have few counts, it is not the same as letting $|s| \to \infty$, but offers a more realistic idea of what can be expected with large sample sizes.

## 5.1 Zero-offset

The simplest method for ensuring that no probability is estimated as zero is to add a small, fixed, positive *zero-offset* to each count to generate the posterior counts:

$$X_s(i) \leftarrow s(i) + z \ .$$

For large sample sizes, the zero-offset has little effect on the probability estimation, and $\hat{P}_s(i) \to P_s(i)$ as $|s| \to \infty$.

For $|s| = 0$, the estimated distribution will be flat ($\hat{P}_0(i) = 1/\text{alphabet size} = 0.05$), which is generally a poor approximation to the amino acid distribution in a context about which nothing is known yet.

It is fairly traditional to use $z = 1$ when nothing is known about the distributions being approximated, but this value is much too large for highly conserved regions like the BLOCKS database—the optimal value is much closer to 0.05. Table 2 presents the excess entropy for three optimized regularizers, as well as the popular "add-one" regularizer.

| $\|s\|$ | excess entropy optimized for $\|s\| =$ | | | | |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 0,1,2,3 |
| 0 | 0.00000 | 0.01910 | 0.02848 | 0.03459 | 0.00610 |
| 1 | 0.14221 | 0.13384 | 0.13643 | 0.13925 | 0.13644 |
| 2 | 0.14499 | 0.13681 | 0.13455 | 0.13497 | 0.13720 |
| 3 | 0.13838 | 0.13127 | 0.12792 | 0.12757 | 0.13097 |
| 4 | 0.13006 | 0.12397 | 0.12060 | 0.12004 | 0.12350 |
| 5 | 0.12369 | 0.11845 | 0.11543 | 0.11484 | 0.11804 |
| full | 0.07966 | 0.07913 | 0.08767 | 0.09129 | 0.08521 |

Table 3: Excess entropy for pseudocount regularizers.

## 5.2 Pseudocounts

Pseudocount methods are a slight variant on the zero-offset, intended to produce more reasonable distributions when $|s| = 0$. Instead of adding a constant zero-offset, a different positive constant is added for each amino acid:

$$X_s(i) \leftarrow s(i) + z(i) \ .$$

These zero-offsets are referred to as *pseudocounts*, since they are used in a way equivalent to having counted amino acids.

Again, as $|s| \to \infty$ the pseudocounts have diminishing influence on the probability estimate and $\hat{P}_s(i) \to P_s(i)$. For $|s| = 0$, we can get $\hat{P}_0(i) = P_0(i)$, by setting $z(i) = aP_0(i)$, for any positive constant $a$. This setting of the pseudocounts has been referred to as *background pseudocounts* [15] or the *Bayesian prediction method* [18] (for the Bayesian interpretation of pseudocounts, see [13]). For the BLOCKS database and $|s| > 0$, the optimal value of $a$ is near 1.0.

For non-empty samples, the pseudocounts that minimize the encoding cost of Section 2 are not necessarily multiples of $P_0(i)$ (see [13] for details).

The pseudocounts were optimized for $|s| = 0$ through $|s| = 3$, both separately, and minimizing the average entropy for all four sample sizes combined. The excess entropy for different pseudocounts is given in Table 3. The pseudocount regularizers do much better than the zero-offset regularizers for $|s| = 0$ and $|s| = 1$, but already by $|s| = 5$, the difference is only 0.025 bits per column.

## 5.3 Gribskov average score

The *Gribskov profile* [7] method or *average-score method* [18] computes the weighted average of scores from a *score matrix* $M$. There are several standard scoring matrices in use, most notably the Dayhoff matrices [6] and the BLOSUM matrices [9], which were originally created for aligning one sequence with another ($|s| = 1$).

The scores are best interpreted as the logarithm of the ratio of the probability of the amino acid in the context to the background probability [1]:

$$\text{score} = \log \hat{P}_s(i)/P_0(i) \ .$$

The averaging of the score matrices is intended to create a new score. With the interpretation of scores given above, and assuming natural logarithms are used, the posterior counts are

$$X_s(i) \leftarrow P_0(i) e^{\frac{\sum_j M_{i,j} s(j)}{|s|}} \ .$$

We can avoid recording the extra parameters $P_0(i)$ by redefining the score matrix slightly. If we let $M'_{i,j} = M_{i,j} + \ln P_0(i)$, then

$$X_s(i) \leftarrow e^{\frac{\sum_j M'_{i,j} s(j)}{|s|}} \ .$$

The BLOSUM substitution matrices provide a score matrix

$$M_{i,j} = \log\left(\frac{P(i,j)}{P_0(i)P_0(j)}\right)$$

for matching amino acid $i$ and amino acid $j$, where $P(i,j)$ is the probability of $i$ and $j$ appearing as an ordered pair in any column of a correct alignment. Let's take natural logarithms in creating the score matrix (to match the exponential in the computation of $X_s(i)$). If we use $j$ to name the sample consisting of a single amino acid $j$, then

$$\hat{P}_j(i) = X_j(i) = \frac{P(i,j)}{P_0(j)} \ .$$

This is the optimal value for $\hat{P}_j$, and so the Gribskov method is optimal for $|s| = 1$ (with a properly chosen score matrix).

Although the Gribskov method is optimal for $|s| = 1$, it does not perform well at the extremes. For $|s| = 0$, it predicts a completely flat distribution (just as zero-offset methods do). As $|s| \to \infty$, the Gribskov method does not approach a maximum-likelihood estimate for $P_s(i)$.

We can get much better performance for $|s| > 1$ by optimizing the score matrix, but the Gribskov method does not generalize to other values of $|s|$ as well the substitution matrix method described in Section 5.4.

Four score matrices were tested: the BLOSUM62 matrix (more precisely, $\ln P(i,j)/P_0(j)$ for the BLOSUM62 data), the log-odds matrix with scores $\ln P(i,j)/P_0(j)$ for the test data, a matrix optimized for $|s| = 2$, and one optimized for $|s| = 0, 1, 2, 3$. The excess entropies are presented in Table 4.

| | excess entropy | | | |
|---|---|---|---|---|
| | BLOSUM62 | log-odds | optimized for $\|s\| =$ | |
| $\|s\|$ | | | 2 | 0,1,2,3 |
| 0 | 0.12527 | 0.12527 | 0.12527 | 0.12527 |
| 1 | 0.13294 | 0.00000 | 0.19046 | 0.08408 |
| 2 | 0.41066 | 0.13311 | 0.01694 | 0.03136 |
| 3 | 0.59404 | 0.27749 | 0.08442 | 0.12809 |
| 4 | 0.71962 | 0.38475 | 0.15471 | 0.21261 |
| 5 | 0.81315 | 0.46765 | 0.21601 | 0.28223 |
| full | 1.37003 | 0.98464 | 0.65103 | 0.74889 |

Table 4: Excess entropy for Gribskov average-score regularizers.

## 5.4 Substitution matrices

A substitution matrix computes the posterior counts as a linear combination of the counts:

$$X_s(i) \leftarrow \sum_j M_{i,j} s(j) \ .$$

This method is similar to the Gribskov average-score method of Section 5.3, with one major difference—the matrix $M$ is not a logarithmic score matrix.

Note that for $|s| = 0$, all the sample counts $s(j)$ are zero, and so the posterior counts $X_0(i)$ are also zero. This violates the constraints on posterior counts, and so some other method of deriving posterior counts is needed for $|s| = 0$. For experiments reported in this paper, all-zero count vectors are replaced by all-one count vectors ($s(j) = 1$ and $X_0(i) = \sum_j M_{i,j}$). This is equivalent to adding an infinitesimal zero-offset to the count vectors before multiplying by the substitution matrix $M$.

Substitution matrices, like score matrices, are designed for use in sequence-sequence alignment, where the sample consists of exactly one amino acid ($|s| = 1$). Let $P_j$ be the distribution we expect in a column in which amino acid $j$ has been seen (the *relatedness odds ratio* which has been widely studied; see, for example [12]). Then we can get optimal behavior for $|s| = 1$ by setting $M_{i,j} = a_j P_j(i)$, for arbitrary positive constants $a_j$.

Furthermore, if we set $a_j = a P_0(j)$, then $X_0(i) = \sum_j a P_0(j) P_j(i) = a P_0(i)$, and we get optimal estimation for $|s| = 0$ ($\hat{P}_0(i) = P_0(i)$) as well as $|s| = 1$. Neither the log-odds matrix ($a_j = 1$) nor this frequency matrix approach works well for larger sample sizes.

For large values of $|s|$, the substitution matrix does not guarantee that the estimated distribution approaches the true distribution, unless the count vector $s$ happens to be an eigenvector of the matrix.

Four substitution matrices were tested: the frequency matrix from which the BLOSUM62 scoring matrix was derived, a frequency matrix computed from the weighted BLOCKS database, a substitution matrix optimized for $|s| = 2$, and one optimized for $|s| = 0, 1, 2, 3$. Table 5 presents the excess entropies.

The pure frequency matrix is optimal for $|s| = 0$ and $|s| = 1$, but degrades badly for larger samples, and is worse than pseudocounts for $|s| = 3$. The BLOSUM62 matrix does not do well for any sample size greater than

| | excess entropy | | | |
|---|---|---|---|---|
| | BLOSUM62 | frequency | optimized for $\|s\| =$ | |
| $\|s\|$ | | matrix | 2 | 0,1,2,3 |
| 0 | 0.00369 | 0.00000 | 0.05348 | 0.05270 |
| 1 | 0.13294 | 0.00000 | 0.05723 | 0.03647 |
| 2 | 0.35455 | 0.08581 | 0.02495 | 0.02708 |
| 3 | 0.50493 | 0.17251 | 0.05577 | 0.06792 |
| 4 | 0.61172 | 0.24275 | 0.09300 | 0.11082 |
| 5 | 0.69341 | 0.30091 | 0.12933 | 0.15083 |
| full | 1.20369 | 0.71452 | 0.44748 | 0.48373 |

Table 5: Excess entropy for substitution matrix regularizers.

zero, probably because of the difference in weighting schemes used for building the matrix and for testing.

Optimizing the substitution matrix can preserve its superiority over pseudocounts up to $|s| = 4$, but as the sample size increases, the pseudocounts approach the optimum regularizer, while substitution matrices get farther from the optimum.

## 5.5 Substitution matrices plus extra terms

In an attempt to avoid the rather ad hoc approach for handling $|s| = 0$ with substitution methods, I tried a method which combines substitution matrices and pseudocount methods:

$$X_s(i) \leftarrow z(i) + \sum_j M_{i,j} s(j) \ .$$

If one thinks of a substitution matrix as a mutation model, then the pseudocounts represent a mutation or substitution that does not depend on what is currently in the sequence. For doing single alignments, where there is exactly one $s(i)$ that is non-zero, one could obtain the same effect by adding the pseudocounts to each column of the substitution matrix, but for other sample sizes, the separation of residue-specific and background substitutions turns out to be quite useful.

If $z(i)$ is set to $aP_0(i)$ for a very small positive number $a$, then the method is essentially identical to the pure substitution matrix method. If $M$ is set to be the identity matrix, then the method is identical to the pure pseudocount method. In practice, the optimal matrix is closer to the identity matrix than the simple substitution matrix is, but still has significant off-diagonal elements.

As with substitution matrices, substitution matrices plus pseudocounts do not converge to the optimal distribution as $|s| \to \infty$. They do a little better than pure substitution matrices, since the matrix is closer to being an identity matrix.

Adding pseudocounts makes substitution matrices work better for $|s| = 0$, but they still do not converge to the maximum-likelihood estimate as $|s| \to \infty$. This problem can be solved by adding one more term to the posterior counts, proportional to the counts and growing faster than the vector $Ms$ does. One easy way to accomplish this is to add the counts scaled by their sum:

$$X_s(i) \leftarrow |s| s(i) + z(i) + \sum_j M_{i,j} s(j) \ .$$

This substitution-matrix method is a slight generalization of the data-dependent pseudocount method [18]. The data-dependent method sets

$$X_s(i) \leftarrow s(i) + \frac{\sum_j B P_0(i) e^{\lambda A_{i,j}} s(j)}{|s|} \ ,$$

for arbitrary parameter $B$, "natural-scale" parameter $\lambda$, and a substitution matrix $A$. Scaling this by $|s|$ and absorbing the constants and exponentiation into the matrix gives us

$$X_s(i) \leftarrow |s| s(i) + \sum_j M_{i,j} s(j) \ ,$$

which is identical to the method here, if the pseudocounts $z(i)$ are all zero. However, the construction of the matrices in [18] is optimized for single-sequence alignment ($|s| = 1$) and may be far from optimal for other sample sizes.

Claverie proposed a similar method [5]—his method is equivalent to setting $z(i) = 0$ and scaling the $s(i)$ by $\max(\sqrt{|s|}, |s|/20)$, instead of $|s|$. It might be interesting to try other scaling functions, besides $|s|$ or $\sqrt{|s|}$; any positive function such that $f(|s|) \to \infty$ as $|s| \to \infty$ would give the correct convergence to the maximum-likelihood estimate. Lacking any theoretical justification for choosing the scaling function, I took the simplest one: $|s|$.

Adding pseudocounts, scaled counts, or both to the substitution matrices improves their performance significantly. Table 6 presents the excess entropies for these regularizers. The full method, using scaled counts and pseudocounts as well as the substitution matrix, has the best results of any of the methods mentioned so far.

Note that adding pseudocounts is not equivalent to any change in the substitution matrix and makes a noticeable improvement in the excess entropy, probably justifying the 5% increase in the number of parameters.

## 5.6 Dirichlet mixtures

The Dirichlet mixture method introduced in [4] has similarities to the pseudocount methods, but is somewhat more complex. They have been used quite successfully by several researchers [4, 18, 11]. The results

| | excess entropy | | | | | |
|---|---|---|---|---|---|---|
| | subst+pseudo | | subst+scaled | | subst+pseudo+scaled | |
| | optimized for $\vert s\vert =$ | | optimized for $\vert s\vert =$ | | optimized for $\vert s\vert =$ | |
| $\vert s\vert$ | 2 | 0,1,2,3 | 2 | 0,1,2,3 | 2 | 0,1,2,3 |
| 0 | 0.02555 | 0.00012 | 1.00651 | 0.00099 | 0.43012 | 0.00000 |
| 1 | 0.02670 | 0.01080 | 0.01970 | 0.00734 | 0.02960 | 0.00080 |
| 2 | 0.02498 | 0.02595 | 0.02502 | 0.03105 | 0.02496 | 0.02509 |
| 3 | 0.04969 | 0.04743 | 0.04099 | 0.04823 | 0.04157 | 0.03975 |
| 4 | 0.07834 | 0.06937 | 0.05093 | 0.05753 | 0.05210 | 0.04849 |
| 5 | 0.10718 | 0.09152 | 0.05833 | 0.06407 | 0.05973 | 0.05548 |
| full | 0.38692 | 0.32624 | 0.07968 | 0.07789 | 0.07492 | 0.09645 |

Table 6: Excess entropy for substitution matrix regularizers with pseudocounts and pseudocounts plus scaled counts.

here show that Dirichlet mixtures are quantitatively superior to all the other regularizers examined, and that there is not much room for improvement to better regularizers.

One way to view the posterior counts of Dirichlet mixtures is as a linear combination of pseudocount regularizers, where the weights on the combination vary from one sample to another, but the underlying regularizers are fixed. Each pseudocount regularizer is referred to as a *component* of the mixture. The weights for the components are the product of two numbers—a prior weight $q_c$ called the *mixture coefficient* and a weight that is proportional to the likelihood of the sample given the component.

Each pseudocount regularizer defines a Dirichlet density function ($\rho_1$ through $\rho_k$) on the possible distributions of amino acids, with $\rho_c$ characterized by the pseudocounts $z_c(i)$. We need to introduce some notation—the Gamma and Beta functions. The Gamma function is the continuous generalization of the integer factorial function $\Gamma(n+1) = n!$ and the Beta function is a generalization of the binomial coefficients:

$$\mathrm{B}(a) = \frac{\prod_i \Gamma(a(i))}{\Gamma(\sum_i a(i))} .$$

With this notation, we can define

$$X_s(i) \leftarrow \sum_{1\leq c\leq k} q_c \frac{\mathrm{B}(z_c+s)}{\mathrm{B}(z_c)} (z_c(i) + s(i)) ,$$

where $z_c + s$ should be interpreted as the componentwise sum of the two vectors. The derivation of this formula using Bayesian statistics can be found in [13].

Because each of the pseudocount regularizers converges to the correct estimate as $|s| \to \infty$, the Dirichlet mixture will also have the correct behavior in the limit. For $|s| = 0$, the Beta functions cancel, and we have

$$X_0(i) \leftarrow \sum_{1\leq c\leq k} q_c z_c(i) ,$$

which can easily be made to fit the background distribution.

Dirichlet mixtures are clearly the luxury choice among regularizers. The need for computing Gamma functions in order to evaluate the regularizer makes them much more expensive to use than any of the other regularizers reviewed here. However, the excess entropy results in Tables 7 and 8 show that the mixtures perform better than any other regularizer test, and may well be worth the extra computational cost in creating a profile or hidden Markov model.

The regularizers in the table (except for the 9-component one) were created by fitting a single component to the data, or by adding components to a previously created mixture, optimizing after each addition for $|s| = 1, 2$. The components were selected using the greedy strategy described in [13]. The 1-component mixture is just a set of pseudocounts, and so performs almost identically to the pseudocounts optimized for $|s| = 1$ or $|s| = 2$.

The 9-component mixture was optimized by Kimmen Sjölander to produce a good Bayesian prior for the count vectors from the BLOCKS database with all sequences given equal weight [4]. Sjölander's 9-component mixture is the best we have for $|s| = 5$, but it does fairly poorly for $|s| = 0, 1, 2$.

The overall best regularizer is the 21-component Dirichlet mixture, which gets within 0.027 bits of the best possible regularizer for sample sizes up to 5, and probably never takes more than 0.09 bits more than the optimum regularizer.

## 6 Results for separate training and testing

One possible objection to the tests in Section 5 is that the same data is used for training the regularizers and for testing them.

The tests were repeated using separate training and test sets. The BLOCKS database was divided into three disjoint sets, with about 10% of the blocks in set 10a, 10% in 10b, and the remaining 80% in 80c. Regularizers were created separately for each of the three sets, and

|  | excess entropy for $n$ components |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
|  | 1 | 3 | 4 | 6 | 7 | 9 | 10 | 10 |
| $\|s\|$ | 1 | 1+2 | 1+3 | 1+2+3 | 1+2+4 | 9 | 1+2+3+4 | 1+3+6 |
| 0 | 0.02230 | 0.02488 | 0.03134 | 0.02516 | 0.02694 | 0.06123 | 0.02385 | 0.01774 |
| 1 | 0.13437 | 0.05235 | 0.04026 | 0.02336 | 0.01972 | 0.05336 | 0.01115 | 0.00661 |
| 2 | 0.13524 | 0.07353 | 0.05311 | 0.03275 | 0.02678 | 0.02402 | 0.02290 | 0.01610 |
| 3 | 0.12930 | 0.08302 | 0.05933 | 0.03960 | 0.03301 | 0.01970 | 0.03127 | 0.02520 |
| 4 | 0.12206 | 0.08657 | 0.06200 | 0.04367 | 0.03715 | 0.02083 | 0.03620 | 0.03105 |
| 5 | 0.11676 | 0.08886 | 0.06455 | 0.04762 | 0.04119 | 0.02455 | 0.04066 | 0.03624 |
| full | 0.08308 | 0.07986 | 0.08234 | 0.08365 | 0.08837 | 0.10274 | 0.08607 | 0.08992 |

Table 7: Excess entropy for small Dirichlet mixtures regularizers optimized for $|s| = 1, 2$. The mixtures were built by adding new components to a previous mixture, except for for the nine-component mixture, which was provided by Kimmen Sjölander.

|  | excess entropy for $n$ components |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
|  | 15 | 15 | 20 | 21 | 28 | 31 | 35 |
| $\|s\|$ | 1+2+3+4+5 | 1+2+4+8 | 1+3+6+10 | 1+2+3+4+5+6 | 1+2+3+4+5+6+7 | 1+2+4+8+16 | 1+3+6+10+15 |
| 0 | 0.01989 | 0.01040 | 0.01111 | 0.00883 | 0.00832 | 0.00786 | 0.00812 |
| 1 | 0.00192 | 0.00227 | 0.00169 | 0.00115 | 0.00470 | 0.00198 | 0.00578 |
| 2 | 0.01137 | 0.01002 | 0.01003 | 0.00757 | 0.01750 | 0.00764 | 0.02100 |
| 3 | 0.01987 | 0.01958 | 0.01957 | 0.01471 | 0.02740 | 0.01776 | 0.03863 |
| 4 | 0.02608 | 0.02613 | 0.02653 | 0.02051 | 0.03380 | 0.02479 | 0.04903 |
| 5 | 0.03186 | 0.03199 | 0.03286 | 0.02636 | 0.03943 | 0.03092 | 0.05650 |
| full | 0.08603 | 0.09155 | 0.08715 | 0.08589 | 0.09357 | 0.09474 | 0.10174 |

Table 8: Excess entropy for larger Dirichlet mixtures regularizers optimized for $|s| = 1, 2$. The mixtures were built by adding new components to a previous mixture, with the history of the additions shown in the name. The 21-component mixture 1+2+3+4+5+6 is the best overall regularizer for the BLOCKS database.

tested on the other two. The ordering of the methods produced by these tests was almost identical to the ordering produced by the self-test presented in Section 5, but the results are too voluminous to present here.

This separate train-test evaluation lends extra confidence to the comparative evaluation of the regularizers, and some assurance that the good regularizers will generalize to similar multiple alignments.

## 7 Conclusions and future research

For applications that can afford the computing cost, Dirichlet mixture regularizers are clearly the best choice. In fact, they are so close to the theoretical optimum for regularizers, that there doesn't seem to be much point in looking for better regularizers. Other evaluations of regularizers, based on searches in biological contexts, have also found Dirichlet mixtures to be superior [18, 11], validating the more information-theoretic approach taken here.

For applications in which there is little data to train a regularizer, pseudocounts are probably the best choice, as they perform reasonably well with few parameters. Dirichlet mixtures and substitution matrices have comparable numbers of parameters and so require comparable amounts of training data. If the regularizers do not need to be re-evaluated frequently, then Dirichlet mixtures are the preferred choice.

Although most applications (such as training hidden Markov models or building profiles from multiple alignments) do not require frequent evaluation of regularizers, there are some applications (such as Gibbs sampling) that require recomputing the regularizers inside an inner loop. For these applications, the substitution matrix plus pseudocounts plus scaled counts is probably the best choice, as it has only about 0.03 bits more excess entropy than the Dirichlet mixtures, but does not require evaluating Gamma functions.

One weakness of the empirical analysis done in this report is that all the data was taken from the BLOCKS database, which contains only highly conserved blocks. While this leads us to have high confidence in the alignment, it also means that the regularizers do not have to do much work. The appropriate regularizers for more variable columns may look somewhat different, though one would expect the pseudocount and substitution-matrix methods to degrade more than the Dirichlet mixtures, which naturally handle high variability.

To get significantly better performance than a Dirichlet mixture regularizer, we need to incorporate more information than just the sample of amino acids seen in the context. There are two ways to do this: one uses more information about the column (such as solvent accessibility or secondary structure) and the other uses more information about the sequence (such as a phylogenetic tree relating it to other sequences).

Using extra information about a column could improve the performance of a regularizer up to the "full" row shown in Table 1, but no more, since the full row assumes that the extra information uniquely identifies the column. There is about 0.6 bits that could be gained by using such information (relative to a sample size of 5), far more than difference between the best regularizer and a crude zero-offset regularizer.

Incorporating sequence-specific information may provide even larger gains than using column-specific information. Based on preliminary work at UCSC, there may be a full bit per column to be gained by taking into account phylogenetic relationships among sequences in a multiple alignment.

Another way to use sequence-specific information would be to use modified regularizers for residues that are in contact, adjusting the probabilities for one amino acid based on what is present in the contacting position.

A longer version of this paper, including derivations for Dirichlet mixture regularizers and results for feature-based techniques [19, 17] is available as [13].

## Acknowledgements

Thanks to Leslie Grate, who wrote an early version of the program used to test regularizers; Kimmen Sjölander for providing the 9-component Dirichlet mixture and comments on this paper; Michael Brown for providing an early implementation of the Dirichlet mixture regularizer; the Henikoffs for providing the BLOCKS database, pre-publication drafts of their papers and comments on early drafts of this paper; and everyone in the compbio group at UCSC for advice and support.

## References

[1] S. F. Altschul. Amino acid substitution matrices from an information theoretic perspective. *JMB*, 219:555–565, 1991.

[2] S. F. Altschul, R. J. Carroll, and D. J. Lipman. Weights for data related by a tree. *JMB*, 207:647–653, 1989.

[3] P. Baldi, Y. Chauvin, T. Hunkapillar, and M. McClure. Hidden Markov models of biological primary sequence information. *PNAS*, 91:1059–1063, 1994.

[4] M. P. Brown, R. Hughey, A. Krogh, I. S. Mian, K. Sjölander, and D. Haussler. Using Dirichlet mixture priors to derive hidden Markov models for protein families. In L. Hunter, D. Searls, and J. Shavlik, editors, *ISMB-93*, pages 47–55, Menlo Park, CA, July 1993. AAAI/MIT Press.

[5] J.-M. Claverie. Some useful statistical properties of position-weight matrices. *Computers and Chemistry*, 18(3):287–294, 1994.

[6] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. In *Atlas of Protein Sequence and Structure*, chapter 22, pages 345–358. National Biomedical Research Foundation, Washington, D. C., 1978.

[7] M. Gribskov, A. D. McLachlan, and D. Eisenberg. Profile analysis: Detection of distantly related proteins. *PNAS*, 84:4355–4358, July 1987.

[8] S. Henikoff and J. G. Henikoff. Automated assembly of protein blocks for database searching. *NAR*, 19(23):6565–6572, 1991.

[9] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *PNAS*, 89:10915–10919, Nov. 1992.

[10] S. Henikoff and J. G. Henikoff. Position-based sequence weights. *JMB*, 243(4):574–578, Nov. 1994.

[11] S. Henikoff and J. G. Henikoff. Personal communication, Jan. 1995.

[12] D. T. Jones, W. R. Taylor, and J. M. Thornton. The rapid generation of mutation data matrices from protein sequences. *CABIOS*, 8(3):275–282, 1992.

[13] K. Karplus. Regularizers for estimating distributions of amino acids from small samples. Technical Report UCSC-CRL-95-11, University of California, Santa Cruz, Mar. 1995. URL ftp://ftp.cse.ucsc.edu/pub/tr/ucsc-crl-95-11.ps.Z.

[14] A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *JMB*, 235:1501–1531, Feb. 1994.

[15] C. E. Lawrence, S. Altschul, M. Boguski, J. Liu, A. Neuwald, and J. Wootton. Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science*, 262:208–214, 1993.

[16] A. Milosavljević. Discovering sequence similarity by the algorithmic significance method. In *ISMB-93*, pages 284–291, Menlo Park, 1993.

[17] R. F. Smith and T. F. Smith. Automatic generation of primary sequence patterns form sets of related protein sequences. *PNAS*, 87:118–122, Jan. 1990.

[18] R. L. Tatusov, S. F. Altschul, and E. V. Koonin. Detection of conserved segments in proteins: Iterative scanning of sequence databases with alignment blocks. *PNAS*, 91:12091–12095, Dec. 1994.

[19] W. R. Taylor. The classification of amino acid conservation. *Journal of Theoretical Biology*, 119:205–218, 1986.

[20] J. D. Thompson, D. G. Higgins, and T. J. Gibson. Improved sensitivity of profile searches through the use of sequence weights and gap excision. *CABIOS*, 10(1):19–29, 1994.