

# Contents

<b>6 Profiles and profile hidden Markov models</b>	<b>1</b>
Profiles	2
Profile HMM algorithms	3
Estimating probabilities from counts	3
Derivation of maximum likelihood estimation	3
Using prior information	5
The Dirichlet distribution	5
Maximum a posteriori (MAP) estimation	6
Mean posterior (MP) estimation	7
The Laplace “plus-one” rule, revisited	7
Pseudocounts, revisited	8
Sampling from a Dirichlet	8
Estimating a Dirichlet from data	9
Mixture Dirichlet priors	9
Supplemental reading	10



## Chapter 6

# Profiles and profile hidden Markov models

**Again, the notes this week are an outline of what I'm covering, rather than complete text. For detailed information on the topics covered here, see chapter 5 (profile HMMs) in the Durbin book.**

We've now seen all the basic machinery we need to tackle a particularly important problem in sequence analysis: the use of sequence *profiles*.

We're given a *multiple sequence alignment* of a number of homologous sequences. Generally, in that alignment, there will be columns that are highly conserved and columns that are less conserved; and there will be places which are tolerant of insertions and deletions, and places that are not. These correspond to the conserved features of the protein's three-dimensional structure and its function. Pairwise alignment methods do not generally use position-specific information, since position-specific conservation only becomes apparent in multiple sequence alignments. We want to extend the position-independent scoring systems of pairwise alignment methods to position-specific scoring systems; we want to be able to compile a multiple sequence alignment into some sort of *consensus model*, so we can search for similar sequences.

We've already seen weight matrices as position-specific models of multiple alignment information, but they only worked for *ungapped* multiple alignments. We've also already seen pairwise alignment by dynamic programming, which could handle insertions and deletions, but we've only seen position-independent scoring systems.

Extending those DP algorithms to use position-specific scores is easy. The problem, is in knowing *how to determine good score parameters*. If I have an alignment with 100 columns in it (a typical folded protein structural domain is about 100-200 residues big), a position specific scoring system will need at least 20 residue scores per position, plus (if we're doing the usual affine gap penalties) at least two gap scores for gap-open and gap-extend at each position. That's 4400 parameters, a big increase from the 192 we needed for pairwise alignment. It'd be good to have some recourse to a theory that tells us how to set all these parameters, rather than trying to set them all by some *ad hoc* means.

In the last couple of chapters, we've seen a survey of formal stochastic regular grammars and finite state automata, and a more in depth introduction to a particularly useful class of stochastic model, hidden Markov models (HMMs). The idea I've been hammering on is that you now have all the tools to draw state diagrams of HMMs that fit a particular problem of interest, and convert that model to code that implements the standard Viterbi and/or Forward/Backward algorithms. We'll now see a major example of applying HMMs to a biological sequence analysis problem - using HMMs to formally deal with the profile analysis problem, which leads to models that are called *profile HMMs*.

We'll get a little more practical in this chapter, too. As with most things, implementing HMMs from the theory is straightforward at first; but you quickly start finding details that matter. We'll talk about both the overall strategy of profile HMMs, as well as some of the more relevant implementation details in one of the widely available profile-HMM software packages, my HMMER software.

## Profiles

- Review: weight matrices of ungapped alignments (R. Staden; G. Stormo; S. Henikoff ; others)
  - Position-specific scores for each residue in each column.
  - General form: arbitrary scores (including perceptrons)
  - Probabilistic weight matrices and maximum likelihood formalisms
  - Major assumptions: we ignore interactions between positions, and we ignore the phylogenetic history of the sequences.
  - BLOCKS: database of ungapped protein motifs (S. and J. Henikoff) [8, 9]
  - MOST: iterative motif search (R. Tatusov et. al) [13]
- Viewing weight matrices as HMMs : a match-only HMM.
- Fully gapped alignment models: the architecture of profiles and profile HMMs.
  - Historical background on profiles (Gribskov; Taylor; Barton).
  - Original Krogh/Haussler profile HMM architecture.
    - \* how to deal with insertions
    - \* how to deal with deletions
  - Current profile HMM architectures (HMMER's Plan 7 as an example) <http://hmmer.wustl.edu>
    - \*  $D \rightarrow I$  and  $I \rightarrow D$  transitions removed
    - \* modeling complete sequences (HMMER N and C states)
    - \* modeling multidomain proteins (HMMER J state)
    - \* modeling local alignments (entry/exit probabilities)

## Profile HMM algorithms

- Building a profile HMM from an alignment
  - getting around the HMM’s independence assumption:
    - \* ”effective sequence number”
    - \* ad hoc sequence weighting
      - The Gerstein/Sonnhammer/Chothia algorithm
      - The Voronoi algorithm
  - choosing an architecture = choosing which columns are consensus
    - \* Krogh’s simple rule
    - \* Maximum a posteriori construction - sketch of the algorithm.
    - \* Contrast to the general problem of building HMM architectures from data; model merging, etc.
  - Converting counts to probabilities - mixture Dirichlet priors
  - Choosing ”unlearned” (data-independent, algorithm-dependent) parameters
  - Sketch of model-dependent weighting strategies: maximum discrimination
- Scoring target sequences
  - Drawbacks of log likelihood scores include composition dependence and length dependence.
  - Composition dependence: remedied by log-odds scores
  - Length dependence: somewhat remedied by Z-scores (UCSC)
  - Difficulties with reporting E-values; Karlin/Altschul theory does not hold well for profile HMM scores. Solved by empirical fitting to an extreme value distribution;
  - Difficulties with inhomogeneity of nonhomologous biological sequences; use of composition filters, rescoring (UCSC), and the HMMER ”null2” model.

## Estimating probabilities from counts

Estimation of “optimal” parameters from observed counts is just another application of Bayes.

### Derivation of maximum likelihood estimation

We’re given a count vector  $\vec{c}$ , and we’re trying to find optimal probability parameters  $\vec{p}$ . We know that  $(\vec{c}|\vec{p})$  is the *multinomial distribution*:

$$P(\vec{c}|\vec{p}) = \frac{|\vec{c}|!}{\prod_i c_i!} \prod_i p_i^{c_i}$$

We'd view this as a likelihood, because we're really interested in finding the best model  $\vec{p}$  given the data  $\vec{c}$  (hence, in optimizing the posterior  $P(\vec{p}|\vec{c})$ ). If we assume that different parameter choices are *a priori* equiprobable, we optimize the posterior  $P(\vec{p}|\vec{c})$  by maximizing the likelihood  $P(\vec{c}|\vec{p})$ .

OK, then, what are the parameters  $\vec{p}$  that maximize this likelihood equation, given a particular count vector  $\vec{p}$ ? This is an optimization problem:

$$f(\vec{p}) = \frac{|\vec{c}|!}{\prod_i c_i!} \prod_i p_i^{c_i}$$

Now, clearly this  $f(\vec{p})$  function is differentiable with respect to our free parameters  $p_i$ :

$$\nabla f(\vec{p}) = \nabla \sum_i c_i \log p_i = \sum_i \frac{c_i}{p_i} \frac{\delta}{\delta p_i}$$

and as anyone who passed high school calculus knows, all we have to do to find an extremum is set the gradient to zero and solve for the  $p_i$ ... which gives us  $p_i = \infty$ . Oops. Yes, thinking about it, of course the best way to maximize  $\prod_i p_i^{c_i}$  is to set  $p_i = \infty$  – if the  $p_i$  were *unconstrained*. But the  $p_i$  are not unconstrained, they're probabilities; we also have to deal with the constraint that  $\sum_i p_i = 1.0$ . *Constrained optimization problems* are common in probabilistic modeling, because of how often our parameters are probabilities constrained to sum to one.

A very useful technique for solving simple constrained optimization problems is the technique of *Lagrange multipliers*, and it applies very well here.

Let's say I have a function  $f(\vec{x})$  that I'm trying to optimize with respect to some parameters  $\vec{x}$ , and also a constraint equation on the parameters,  $g(\vec{x}) = 0$ . Then:

$$\nabla f(\vec{x}) = \lambda \nabla g(\vec{x})$$

where the  $\lambda$  is a constant multiplier (a "Lagrange multiplier"). An explanation of why this is true is beyond my scope; most any calculus textbook includes both a graphical and a mathematical rationale for the method.

Our constraint equation  $g()$  will be  $\sum_i p_i - 1.0 = 0$ ; the gradient is trivial, just  $\sum_i 1 \frac{\delta}{\delta p_i}$ . So we get:

$$\sum_i \frac{c_i}{p_i} \frac{\delta}{\delta p_i} = \lambda \sum_i 1 \frac{\delta}{\delta p_i}$$

Solving this for each  $p_i$  gives:

$$p_i = \frac{c_i}{\lambda}$$

and to get  $\lambda$ , we substitute the above into the constraint equation; this gives us  $\sum_j \frac{c_j}{\lambda} = 1.0$ , so  $\lambda = \sum_j c_j$ . Thus:

$$p_i = \frac{c_i}{\sum_j c_j} = f_i$$

and so, the observed frequency of event  $i$  is the maximum likelihood estimator for  $p_i$ .

Lagrange multipliers are not the only way to solve constrained optimization problems. We'll see some other tricks soon.

### Using prior information

The thing is, different choices of parameters are *not* equiprobable, so we probably *shouldn't* be using maximum likelihood estimators if we can help it. We usually have prior (extrinsic) information about what parameters will be reasonable, besides the intrinsic information about the directly observed count data. For instance, if we're talking about a column of aligned residues in a protein multiple alignment, I *a priori* prefer to see parameter vectors that favor chemically and structurally similar amino acids.

In order to develop a principled way of combining the observed count data  $\vec{c}$  with our prior knowledge, we cast it as a problem for Bayes; the probability of a parameter vector  $\vec{p}$  is dependent on both  $\vec{c}$  and some kind of parameterized prior model,  $\theta$ :

$$P(\vec{p}|\vec{c}, \theta) = \frac{P(\vec{c}|\vec{p}, \theta)P(\vec{p}|\theta)}{P(\vec{c}|\theta)} \quad (6.1)$$

$$= \frac{P(\vec{c}|\vec{p})P(\vec{p}|\theta)}{P(\vec{c}|\theta)} \quad (6.2)$$

$$(6.3)$$

(We can drop  $\theta$  from  $P(\vec{c}|\vec{p})$  because the count vector depends only on the probability vector.)

So, to use prior information, we'll need to be able to specify a prior probability density  $P(\vec{p}|\theta)$  over the possible parameter vectors  $\vec{p}$ . (This is a funny beast: a probability density over probability vectors. We'll discuss it more when we talk about sampling from it.) There are various such densities we can choose from, but if we want mathematically tractable equations, we'll want to choose a *conjugate prior*: a prior that has a similar functional form as our the functional form of the likelihood, so that they play well together. The conjugate prior for the multinomial distribution is the *Dirichlet distribution*.

### The Dirichlet distribution

The *Dirichlet distribution* is:

$$P(\vec{p}|\vec{\alpha}) \propto \prod_i p_i^{\alpha_i - 1}$$

Normalizing this into a probability distribution involves a multidimensional integration over all probability vectors  $\vec{p}$ :

$$P(\vec{p}|\vec{\alpha}) = \frac{\prod_i p_i^{\alpha_i-1}}{\int_{\vec{p}} \prod_i p_i^{\alpha_i-1} d\vec{p}}$$

Solving this integral gives:

$$P(\vec{p}|\vec{\alpha}) = \frac{\Gamma(|\vec{\alpha}|)}{\prod_i \Gamma(\alpha_i)} \prod_i p_i^{\alpha_i-1}$$

The gamma function,  $\Gamma(x)$ , is the real-number counterpart of the factorial function for integers: for integers  $x$ ,  $x! = \Gamma(x + 1)$ . Thus, notice that the functional form of the multinomial distribution for  $P(\vec{c}|\vec{p})$  and the Dirichlet distribution for  $P(\vec{p}|\vec{\alpha})$  are essentially identical, except that  $\vec{c}$  is a discrete count vector and  $\vec{p}$  is a continuous probability vector.

Coding this up requires getting your hands on a numerical routine for calculating  $\Gamma(x)$ ... actually,  $\log \Gamma(x)$ , since like the factorial  $x!$ ,  $\Gamma(x)$  blows up fast. *Numerical Recipes in C* includes a copyrighted `GAMMLN()` routine [12] along with some explanation of how it works. The C code provided in class (the `DIRICHLET.C` module) includes a freely available `GAMMLN()` implementation from John Spouge at the NCBI, with no explanation of how it works.

The  $\log(P(\vec{p}|\vec{\alpha}))$  calculation is implemented in the `DIRICHLET` C module as `dirichlet.c:Dchlet_logp_probs()`.

## Maximum a posteriori (MAP) estimation

Back to maximizing our probability parameters, given both data and a prior:

$$P(\vec{p}|\vec{c}, \theta) \propto P(\vec{c}|\vec{p})P(\vec{p}|\theta)$$

If we maximize the posterior probability of our parameters (e.g. including a prior on the parameters), rather than simply the likelihood, we're doing *maximum a posteriori* (MAP) parameter estimation instead of ML estimation. Since we've got equations for both the multinomial and the Dirichlet now, we can obtain:

$$P(\vec{p}|\vec{c}, \theta) \propto \frac{|\vec{c}|!}{\prod_i c_i!} \prod_i p_i^{c_i} \frac{\Gamma(|\vec{\alpha}|)}{\prod_i \Gamma(\alpha_i)} \prod_i p_i^{\alpha_i-1}$$

where the normalization terms containing factorials and gamma functions are constants, independent of  $\vec{p}$ , so:

$$P(\vec{p}|\vec{c}, \theta) \propto \prod_i p_i^{c_i+\alpha_i-1}$$

It's straightforward to do a constrained optimization on this objective function using Lagrange multipliers, just the way we did it to ML, which yields:

$$p_i = \frac{c_i + \alpha_i - 1}{\sum_j [c_j + \alpha_j - 1]}$$

The MAP estimator is a little dangerous; if we have  $\alpha_i < 1$  and zero counts, we obtain negative “probabilities”. In practice, we prefer mean posterior estimates, as described below.

### Mean posterior (MP) estimation

An attractive alternative to finding parameters that *maximize* the likelihood or the posterior is to find *expected values*  $\hat{p}_i$  given the count data and a Dirichlet prior:

$$\hat{p}_i = \int_{\vec{p}} p_i P(\vec{p} | \vec{c}, \vec{\alpha}) d\vec{p}$$

Well, that’s a nasty looking integral. We do know (from above) that:

$$P(\vec{p} | \vec{c}, \vec{\alpha}) \propto \prod_i p_i^{c_i + \alpha_i - 1}$$

but normalizing that is essentially the same nasty integral. And, in fact, it’s the same nasty integral that we need to do to get the normalization factor for the multinomial distribution. So, eventually I’ll write it down and introduce it when I introduce the multinomial - for now, I’ll punt, and say that it’s a known result, and the derivation is in Sjolander’s 1996 paper; see for example her Lemma 2, in the paper’s appendix. Let’s leave it for now as “it is then straightforward to obtain”:

$$P(\vec{p} | \vec{c}, \vec{\alpha}) = \frac{\Gamma(|\vec{\alpha}| + |\vec{c}|)}{\prod_i \Gamma(\alpha_i + c_i)} \prod_i p_i^{c_i + \alpha_i - 1}$$

(Check this against the multinomial, using the relation  $x! = \Gamma(x + 1)$  for integer  $x$  - it’s the same functional form.)

From there, solving  $\int_{\vec{p}} p_i P(\vec{p} | \vec{c}, \vec{\alpha}) d\vec{p}$  uses the same techniques (for the derivation, which will eventually be here but currently isn’t, see Sjolander’s section 3.2.1) and (amazingly) simplifies wonderfully, yielding:

$$\hat{p}_i = \frac{c_i + \alpha_i}{|\vec{c}| + |\vec{\alpha}|}$$

### The Laplace “plus-one” rule, revisited

The simple Laplace plus-one rule is mean posterior estimation using a uniform prior over  $\vec{p}$ : the plus-one implies that we’re imposing a Dirichlet prior with parameters  $\alpha_i = 1$ . Notice that if I set all  $\alpha_i = 1$  for my Dirichlet, then  $p_i^{\alpha_i - 1} = 1$  for any  $p_i$ , so all  $P(\vec{p} | \vec{\alpha})$  are equiprobable.

(I suppose we could also say that the plus-one rule is MAP estimation using a Dirichlet with parameters  $\alpha_i = 2$ , but Laplace in fact derived his rule by assuming a uniform prior and doing mean posterior estimation.)

## Pseudocounts, revisited

The Dirichlet lets us inject a little more information into parameter estimation than a uniform prior. The expected  $p_i$ 's, given only the Dirichlet and no observed data, are:

$$\hat{p}_i = \frac{\alpha_i}{|\vec{\alpha}|}$$

which is to say that the  $\alpha_i$ 's are proportional to the *a priori* expected frequencies of the events  $i$ ; the Dirichlet conveys information about these expected frequencies. If we know that glycine is three times more abundant than tryptophan, then we would tend to use an amino acid Dirichlet prior in which  $\alpha_G = 3\alpha_W$ .

Also, the sum of the  $\alpha_i$  terms is inversely proportional to the variance of the Dirichlet. Larger  $\alpha_i$ 's give a peakier, tighter distribution around the expected  $\hat{p}_i$ 's.

This all means that we can view pseudocount methods as mean posterior estimation using Dirichlet priors, too. If I specify an *a priori* expected mean  $f_i$  for my events, and an arbitrary weight  $W$  that controls how much weight I give pseudocounts versus the real counts, I would add  $W f_i$  pseudocounts to each  $c_i$ ; I can interpret the  $W f_i$ 's as the  $\alpha_i$  parameters of a Dirichlet.

## Sampling from a Dirichlet

A good way to get an intuition about what Dirichlets are doing is to sample from them. The routine `dirichlet.c:SampleDirichlet()` in the **DIRICHLET** C module samples a probability vector  $\vec{p}$  from the Dirichlet density specified by given parameters  $\vec{\alpha}$ . The algorithm is:

**Algorithm 1:** Sampling a probability vector from a Dirichlet density

```
SAMPLE_DIRICHLET( $\vec{\alpha}$ , K)
(1)   for  $i \leftarrow 1$  to  $K$ 
(2)      $p_i = \text{SAMPLE\_GAMMA}(\alpha_i)$ 
(3)   NORMALIZE( $\vec{p}$ )
```

That's easy enough, but now we need a routine for sampling a value from a gamma density, and that's not so easy. An implementation is provided in the routine `dirichlet.c:sample_gamma()`, based on pp. 133-134 of Knuth's *Seminumerical Algorithms* [10]. Explaining how and why it works is beyond my scope here.<sup>1</sup>

It's hard to visualize  $K$ -dimensional probability vector spaces with  $K - 1$  free parameters, in general, but it's easy to do coins: the 2-dimensional probability space for heads and tails,  $p_H$  and  $p_T$ , has only one free parameter. To familiarize yourself with the Dirichlet, write a little routine that samples from a Dirichlet for different choices of  $\alpha_H$  and  $\alpha_T$ . In particular, use  $\alpha_H = \alpha_T = 1$  (a uniform prior),  $\alpha_H = \alpha_T = 5$  (a mild expectation of fair-ish coins),  $\alpha_H = \alpha_T = 5000$  (strong expectation of fair-ish coins), and  $\alpha_H < 1$  and  $\alpha_T < 1$  (remember that  $\alpha$ 's must be  $> 0$ ).

<sup>1</sup>At least for now - it would be nice to use this as an excuse to explain rejection sampling.

## Estimating a Dirichlet from data

Given a data set of  $M$  count vectors  $\vec{c}^m$ , we want to find a maximum likelihood Dirichlet prior; e.g. parameters  $\alpha_i$  that maximize:

$$\log P(\text{data}|\vec{\alpha}) = \sum_m \log P(\vec{c}^m|\vec{\alpha})$$

This is just a sum of the likelihoods of individual count vectors, so let's drop the subscript and worry about one of them at a time:

$$P(\vec{c}|\vec{\alpha}) = \int_{\vec{p}} P(\vec{c}|\vec{p})P(\vec{p}|\vec{\alpha})d\vec{p}$$

Solving that integral (again, the same way we've been doing it) gives:

$$P(\vec{c}|\vec{\alpha}) = \frac{\Gamma(|\vec{c}|+1)\Gamma(|\vec{\alpha}|)}{\Gamma(|\vec{c}+\vec{\alpha}|)} \prod_i \frac{\Gamma(c_i+\alpha_i)}{\Gamma(c_i+1)\Gamma(\alpha_i)}$$

Computationally and numerically it's best to solve that in log space:

$$\log P(\vec{c}|\vec{\alpha}) = \log \Gamma(|\vec{c}|+1) + \log \Gamma(|\vec{\alpha}|) - \log \Gamma(|\vec{c}+\vec{\alpha}|) + \sum_i \log \Gamma(c_i+\alpha_i) - \sum_i \log \Gamma(c_i+1) - \sum_i \log \Gamma(\alpha_i)$$

That calculation is implemented in the **DIRICHLET C** module as `dirichlet.c:Dchlet_logp_counts()`.

## Mixture Dirichlet priors

A *mixture density* is just a sum of more than one density. To make sure that probabilities sum to one, we have to specify *mixture coefficients*  $p_q$  for the probability of each component  $q$  in the mixture.

A mixture Dirichlet  $\theta$  therefore consists of two or more components  $q$ , each of which is a Dirichlet density parameterized by  $\vec{\alpha}^q$ , and which is used with probability  $p_q$ . The resulting density is a weighted sum of the individual component densities:

$$P(\vec{p}|\theta) = \sum_q P(\vec{p}|\vec{\alpha}^q)P(q|\theta)$$

Note from the way I wrote this that it just falls out of probability theory: we're marginalizing over the components  $q$  to get  $P(\vec{p}|\theta)$ .  $P(q|\theta)$  is the mixture coefficient  $p_q$ , of course.

## Sampling from a mixture Dirichlet

Easy: sample a  $q$  from the distribution  $p_q$ ; then sample  $\vec{p}$  from Dirichlet component  $\vec{\alpha}^q$  using **SAMPLE\_DIRICHLET**.

**Estimating a mixture Dirichlet from data**

$$P(\vec{c}|\theta) = \sum_q P(\vec{c}|\vec{\alpha}^q)P(q|\theta)$$

**Supplemental reading**

A variety of more or less *ad hoc* profile methods were in use before profile HMMs [14, 7, 1]. Michael Gribskov coined the name “profile”, and has written good reviews of the field [6, 5, 4]. Profile HMMs were first introduced by Anders Krogh, David Haussler, and colleagues [11]. I wrote two reviews of the early profile HMM literature [2, 3].

# Bibliography

- [1] Geoffrey J. Barton. Protein multiple sequence alignment and flexible pattern matching. *Meth. Enzymol.*, 183:403–427, 1990.
- [2] Sean R. Eddy. Hidden Markov models. *Curr. Opin. Struct. Biol.*, 6:361–365, 1996.
- [3] Sean R. Eddy. Profile hidden Markov models. *Bioinformatics*, 14:755–763, 1998.
- [4] M. Gribskov and S. Veretnik. Identification of sequence pattern with profile analysis. *Meth. Enzymol.*, 26:198–212, 1996.
- [5] Michael Gribskov. Profile analysis. In A. M. Griffin and H. G. Griffin, editors, *Methods in Molecular Biology: Computer Analysis of Sequence Data Part II*, pages 247–267. Humana Press, Totowa NJ, 1994.
- [6] Michael Gribskov, Roland Luthy, and David Eisenberg. Profile analysis. *Meth. Enzymol.*, 183:146–159, 1990.
- [7] Michael Gribskov, Andrew D. McLachlan, and David Eisenberg. Profile analysis: Detection of distantly related proteins. *Proc. Natl. Acad. Sci. USA*, 84:4355–4358, 1987.
- [8] Steven Henikoff and Jorja G. Henikoff. Automated assembly of protein blocks for database searching. *Nucl. Acids Res.*, 19:6565–6572, 1991.
- [9] Steven Henikoff and Jorja G. Henikoff. Protein family classification based on searching a database of blocks. *Genomics*, 19:97–107, 1994.
- [10] Donald E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison–Wesley, Reading, Massachusetts, second edition, 1981.
- [11] Anders Krogh, Michael Brown, I. Saira Mian, Kimmen Sjolander, and David Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *J. Mol. Biol.*, 235:1501–1531, 1994.
- [12] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 1988.

- [13] R. L. Tatusov, S. F. Altschul, and E. V. Koonin. Detection of conserved segments in proteins: Iterative scanning of sequence databases with alignment blocks. *Proc. Natl. Acad. Sci. USA*, 91:12091–12095, 1994.
- [14] William Ramsay Taylor. Identification of protein sequence homology by consensus template alignment. *J. Mol. Biol.*, 188:233–258, 1986.