Applications of Nonlinear System Identification to Protein Structural Prediction

by

Robert David

B.Sc., Applied Mathematics (1998)

McGill University

Submitted to the Department of Mechanical Engineering in Partial Fulfillment of the Requirements for the Degree of Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2000

© 2000 Massachusetts Institute of Technology
All rights reserved

Signature of Author

Department of Mechanical Engineering
May 5, 2000

Certified by

Ian W. Hunter
Professor of Mechanical Engineering
Thesis Supervisor

Accepted by

Ain A. Sonin
Chairman, Department Committee on Graduate Students
Applications of Nonlinear System Identification to Protein Structural Prediction

by
Robert David

Submitted to the Department of Mechanical Engineering on May 5, 2000
in Partial Fulfillment of the Requirements for the Degree of Master of Science in Mechanical Engineering

Abstract

The prediction of structural properties of proteins based on their amino acid sequence has been a very active area of biology research for the past three decades. In this thesis nonlinear system identification techniques are applied to three problems concerning the prediction of aspects of protein structure: classification into families, 1-dimensional structure prediction, and identification of ATP binding sites. The amino acid sequence is treated as a time series input to a nonlinear system, with the structural property of interest as the output. Published hydrophobicity values are frequently employed for numerical representation of the residues. Parallel cascade identification (PCI) and Fast Orthogonal Algorithm (FOA) are used to model the systems.

Earlier results showing accurate classification by PCI of sequences into two or three families are confirmed on large test sets. Extensions to the algorithm are implemented and it is combined with an existing hidden Markov model for improved results. Preliminary tests are carried out using PCI for prediction of the radial distance of a residue from the protein center, and the secondary structure. Various simple methods and FOA are compared with the performance, in earlier work, of a neural network and a statistical method for identifying ATP binding sites. More accurate predictors are obtained.

Thesis Supervisor: Ian W. Hunter
Title: Professor of Mechanical Engineering and Professor of BioEngineering
Acknowledgments

I thank my advisor, Prof. Ian Hunter, for the opportunity to work in his extraordinary laboratory and for his perpetual enthusiasm and support. I thank Prof. Michael Korenberg of Queen’s University for the use of his computer programs, for his many clever suggestions, and for his support. Thanks to Colin Brenan for supervising my earlier work in the lab.

I thank the many members of the lab who make it a friendly place to work in – Patrick Anquetil, Matthew Graham, Beth Sebern, Peter Madden, James Tangorra, John Madden, Luke Sosnowski, Bryan Crane, and others. I also appreciate the efforts of Max Berniker, Justin Won, Brandon Rohrer, and their colleagues, who work hard to maintain a thoroughly professional atmosphere on the mezzanine.

I thank Yanqing for her friendship and advice. Finally, I thank my family and out-of-town friends for reminding me about life outside MIT and for time enjoyed on relaxing vacations. Speaking of which...
# Table of Contents

Abstract .......................................................................................................................... 3

Acknowledgments ........................................................................................................... 4

List of Figures ................................................................................................................. 7

List of Tables .................................................................................................................. 8

Chapter 1. Introduction ................................................................................................ 9
  1.1 Overview .............................................................................................................. 9
  1.2 Linear System Identification ............................................................................. 9
  1.3 Parallel Cascade Identification ......................................................................... 11
  1.4 Fast Orthogonal Algorithm ............................................................................. 12

Chapter 2. Protein Classification .............................................................................. 13
  2.1 Background ......................................................................................................... 13
  2.2 Protein Classification Using Parallel Cascade Identification ....................... 14
  2.3 Further Modifications to the Classification Algorithm ......................... 16
  2.4 Combination with a Hidden Markov Model ............................................ 20

Chapter 3. Protein Folding ......................................................................................... 23
  3.1 Background ......................................................................................................... 23
  3.2 Tertiary Structure .............................................................................................. 23
  3.3 Secondary Structure ......................................................................................... 28

Chapter 4. ATP Binding Sites ........................................................................... 29
  4.1 Background ......................................................................................................... 29
  4.2 Discernment of Binding P-loops from Non-binding P-loops ....................... 29
  4.3 Neural Networks .............................................................................................. 33
  4.4 Larger Test Set .................................................................................................. 34

Chapter 5. Conclusion ................................................................................................ 36
  5.1 Summary ............................................................................................................. 36
  5.2 Future Work ....................................................................................................... 36
List of Figures

Figure 1. Output means from the parallel cascade model that were farther from zero indicated greater confidence.............................................................................................................. 20

Figure 2. Model output (dotted line) and actual radial coordinate (solid line) for 1mbc and four other sequences, using model trained on 1mbc ........................................................................ 24

Figure 3. Models for prediction of atomic coordinates were generally less accurate on longer sequences........................................................................................................................................ 26

Figure 4. Improvement in network results with voting, at a learning rate of 0.3. .............. 33

Figure 5. Output values used as confidence levels of individual neural networks............. 33
List of Tables

Table 1. Summary of results obtained by various modifications of the (original) PCI program used in [19]................................................................................................................. 16

Table 2. Testing of PCI protein classification over large data sets available on the Internet........................................................................................................................................ 16

Table 3. Improvements in classification accuracy using a logic scheme, in percentage points.......................................................................................................................... 17

Table 4. PCI classification accuracy using various hydrophobicity scales to specify the sequence inputs........................................................................................................ 18

Table 5. 5-digit coding for amino acids listed in order of decreasing hydrophobicity according to the Rose scale................................................................................................... 19

Table 6. Results using the state input scheme. .................................................................................................................. 19

Table 7. Combination of PCI with SAM improved on the solo results of either method. ................................................................................................................................. 21

Table 8. Results of combined schemes between PCI (with state input) and SAM........ 22

Table 9. Similarity between 1mbc and four preliminary test sequences for structure prediction. ..................................................................................................................... 24

Table 10. Different hydrophobicity scales used as inputs for structure models. ........ 26

Table 11. Parameter values for prediction of radial coordinate........................................ 26

Table 12. Glycine residues occur more commonly in the [AG] positions of P-loops that bind ATP. .................................................................................................................. 30

Table 13. Results for jack-knife method ............................................................................. 32
Chapter 1. Introduction

1.1 Overview

One of the most important challenges in modern molecular biology is the protein folding problem – that is, the precise determination of the structure and/or function of a protein based solely on the sequence of amino acids comprising it. Such data are becoming available at a rapid pace due to large-scale DNA sequencing projects underway for a variety of organisms. This thesis describes the application of nonlinear system identification techniques to three subsidiary problems of the protein folding problem. In all cases it is desired to find some information about a protein using knowledge of only its primary structure.

The first problem studied was the classification of proteins into structure/function groups; this can be regarded as protein folding on the coarsest scale. The second problem was 1-dimensional structure prediction; the radial distance of a residue from the protein’s center, and the secondary structure at each residue, were examined. Finally, the third problem concerned the identification of ATP binding sites on proteins. More details about the biology of each topic will be given in the first sections of subsequent chapters. The main code used to implement the algorithms is shown in Appendix B.

The principal mathematical tools applied to these problems are described below, preceded by a brief introduction to system identification.

1.2 Linear System Identification

System identification concerns the derivation of a mathematical relationship between a given input function and a given output function. It is useful in situations where the system being studied is not well-known or is too complex for a physical model to be solved.

In linear system identification the output $y(i)$, $i = 0...T$, is assumed to be a linear function of the input $x(i)$ with a memory length $R + 1$. Here we assume an input with zero mean for simplicity. In discrete time and for a stationary process,

$$y(i) = \sum_{j=0}^{R} h(j)x(i-j),$$  \hspace{1cm} (1)
where \( h(i) \) is the impulse response function, so named because for an input of \( \delta(i) \) it is the output. In matrix form this can be written as

\[
\begin{pmatrix}
  y(R) \\
y(R + 1) \\
\vdots \\
y(T)
\end{pmatrix} =
\begin{pmatrix}
x(0) & x(1) & \cdots & x(R) \\
x(1) & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \vdots \\
x(T - R) & \cdots & \cdots & x(T)
\end{pmatrix}
\begin{pmatrix}
h(R) \\
h(R - 1) \\
\vdots \\
h(0)
\end{pmatrix}.
\]

(2)

To facilitate the solution for \( h(i) \) from a given \( x(i) \) and \( y(i) \) we multiply both sides of Equation 2 on the left by

\[
\begin{pmatrix}
x(0) & x(1) & \cdots & x(T - R) \\
x(1) & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \vdots \\
x(R) & \cdots & \cdots & x(T)
\end{pmatrix}
\]

(equivalent to convolving both sides of Equation 1 on the left with \( x(i) \)) to obtain

\[
\begin{pmatrix}
C_{xy}(R) \\
C_{xy}(R - 1) \\
\vdots \\
C_{xy}(0)
\end{pmatrix} =
\begin{pmatrix}
C_{xx}(0) & C_{xx}(1) & \cdots & C_{xx}(R) \\
C_{xx}(1) & C_{xx}(0) & \ddots & \vdots \\
\vdots & \ddots & \ddots & \vdots \\
C_{xx}(R) & C_{xx}(R - 1) & \cdots & C_{xx}(0)
\end{pmatrix}
\begin{pmatrix}
h(R) \\
h(R - 1) \\
\vdots \\
h(0)
\end{pmatrix},
\]

where \( C_{xx}(j) \) are elements of the unbiased estimator of the input autocorrelation function

\[
C_{xx}(j) = \frac{1}{T - R + 1} \sum_{i=R}^{T} x(i - j)x(i),
\]

and \( C_{xy}(j) \) are the elements of the unbiased estimator of the input-output crosscorrelation function

\[
C_{xy}(j) = \frac{1}{T - R + 1} \sum_{i=R}^{T} x(i - j)y(i).
\]

The former form a Toeplitz matrix allowing quick inversion to solve for \( h(i) \) [21]. The unbiased estimators, rather than the biased estimators, which have factors of \( 1 / (T - j + 1) \) instead of \( 1 / (T - R + 1) \) before the sums, are a technical necessity to ensure convergence of the algorithm [26].

In the more general Wiener (LN) system the linear (L) element in Equation 1 is followed by a static nonlinear (N) element, which may be a polynomial of degree \( d \):

---

1 This relation will be only approximately true for a finite sampling of a single realization of the random process, which is assumed to be ergodic.
\[ z(i) = \sum_{j=0}^{d} a_j y(i)^j, \]  

where \( z(i) \) is now the system output. Bussgang's theorem [10] states that for a Gaussian white input the procedure given above may still be used to obtain the impulse response function.

1.3 Parallel Cascade Identification

A different model for a system, called a Volterra series, is as follows:

\[ y(t) = k_0 + \sum_{m=1}^{M} V_m, \]

where

\[ V_m = \sum_{j_1, \ldots, j_m=0}^{R} k_m(j_1, \ldots, j_m) \cdot x(i - j_1) \ldots x(i - j_m). \]

Here \( k_m \) is the \( m^{th} \) order Volterra kernel and \( M \) is the order of the Volterra series. Direct calculation of the Volterra kernels for a given system is time-consuming; however, it has been shown [26] that a finite-order Volterra series can be re-cast as a specific series of LN cascades. Parallel cascade identification (PCI), developed by Korenberg [26], is an algorithm that exploits this to quickly estimate the Volterra kernels of a system from given input-output data.

In PCI, the impulse response function of each L element may be either a first-order crosscorrelation of the input and output signals, or a slice of a second- or third-order crosscorrelation. For example, a slice of a third-order crosscorrelation is

\[ h(i) = \frac{1}{T - R + 1} \sum_{i=R}^{T} y(i) x(i - j) x(i - r_1) x(i - r_2), \]

where \( r_1 \) and \( r_2 \) are fixed integers between 0 and \( R \). Each cascade is formed using a randomly selected impulse response together with the polynomial coefficients \( a_j \) of the \( N \) element (Equation 3), which are found by a least-squares fit to the given output.

In this way the first cascade is built to give the best approximation to the given output; the error is then treated as the output of a second cascade with the same input, and so on, until a desired accuracy is obtained. The final model, from which the Volterra
kernels may be extracted, is the sum of the cascades. A proof of convergence and a more lengthy discussion of the algorithm are given in [26].

1.4 Fast Orthogonal Algorithm

A functional expansion of a system (such as Equation 4) may be generalized by considering the output signal to depend on past outputs as well as the current and past inputs. Thus we may consider

\[ y(i) = \sum_{n=0}^{N} a_n p_n(i), \]

with

\[ p_n(i) = \prod_{j} x(i - j) \prod_{k} y(i - k), \]

where \( j \) ranges over a (possibly empty) subset of \( 0...R_1 \) and \( k \) ranges over a (possibly empty) subset of \( 1...R_2 \). In this work equal memory lengths \( R_1 = R_2 \) were used. Equation 5 is known as a nonlinear difference equation; when the \( p_n(i) \) each contain only a single term it is called an auto-regressive moving average (ARMA). Since the past outputs are themselves functions of past inputs, this representation can provide a more succinct approximation to a system than a functional expansion (i.e. fewer terms are needed). For a given input \( x(i) \) and output \( y(i) \), and a choice of terms \( p_n(i) \), Fast Orthogonal Algorithm (FOA) [25] is a way to quickly find the values of the coefficients \( a_n \) which give the best model.

The problem is decoupled by orthogonalizing the \( p_n(i) \) over the data record by the Gram-Schmidt process, to produce \( w_n(i) \). The model can then be written as

\[ y(i) = \sum_{n=0}^{N} g_n w_n(i), \]

where the \( g_n \) are best-fit constants from which the \( a_n \) of Equation 5 may be recovered. Since calculation of the \( g_n \) requires only certain time-averages of the \( w_n(i) \), the process can be dramatically sped by avoiding explicit computation of the \( w_n(i) \). More details are available in [25].
Chapter 2. Protein Classification

2.1 Background

Proteins are classified by structure/function into dozens of families. While environmental factors must play a role in the function of a protein, it is expected that the sequence of amino acids that comprises the protein should be instrumental in determining the 3-dimensional folded structure (this is sometimes referred to as Anfinsen’s principle [2]). Thus a system with the amino acid sequence as input and the classification of the protein as output can be conceived. An accurate model of this system could classify proteins based on their amino acid sequences, serving as interpreter for data generated by, for example, the Human Genome Project.

Techniques that have been applied to automatic classification of proteins include homology, hidden Markov models (HMM’s), neural networks [47], and others. In [28], Korenberg et al. proposed PCI as an alternative method.

Homologous classification is based on the similarity of two sequences, one from a known family and the other from an unknown family, in terms of identical residues. Technically the term “homologous” refers to two sequences that are related by evolution [17]. The critical step in homology methods is the proper alignment of the sequences against each other. The standard algorithm for this is Smith-Waterman [45], which forms the basis of the BLAST [1] and FASTA [36] search tools. The PROSITE database [6], focussing on local homology, is a compilation of short subsequences (called motifs) that are characteristic of certain protein families and no others. Homology has the advantages of being direct and accurate, although for new sequences that are not similar to known proteins it cannot be useful. Such sequences are estimated to comprise at least half of newly-discovered proteins [40].

The use of generalized motifs (e.g. containing secondary structure information [7]), led to the emergence of HMM’s as a tool for protein classification [29]. The type of HMM used, called a profile HMM, assigns a position-dependent probability to the occurrence of each amino acid at each location in a sequence, based on training set data. Probabilities are also assigned to insertions and deletions of single amino acids relative to the main pattern. A model is built in this way for each family of proteins; given a new
sequence, the probability of each model generating that sequence can be calculated, and
the sequence is then classified according to the highest probability. Two examples of
HMM's are the Sequence Alignment and Modeling system (SAM) [20,29] and HMMER
[16]. The advantages of PCI relative to HMM's are its lack of a need for a large training
set, and its incorporation of memory, which allows the identity of each amino acid to
influence the identities of those surrounding it. However, with sufficient training data, or
for distinguishing between many families, PCI proved inferior to SAM (Section 2.4).

Neural networks are collections of inter-connected nodes, each of which receives
an input, processes it according to some (usually simple) equation, and passes it on.
Connection weights between nodes are optimized using feedback, effecting a learning
process. Neural networks are modeled after the functioning of the brain. A concrete
example will be given later in conjunction with identification of ATP binding sites
(Section 4.2).

2.2 Protein Classification Using Parallel Cascade Identification

For protein classification using PCI, the input amino acids needed to be specified
numerically; hydrophobicity scales were chosen for this purpose. It is generally believed
[24] that the dominant force in protein folding is the relative hydrophobicity of the amino
acids; the more hydrophobic sections of the protein are packed into the center. The
hydrophobicity of a single amino acid is not a well-defined property and thus it is not
generally agreed upon how to measure it. Dozens of hydrophobicity scales have been
published based on both experimental and theoretical results, many of them in marked
disagreement [12]. The aim of most experimental measurements is to determine the free
energy lost by a residue on being transferred from the cellular environment to the interior
of a protein. The pioneers in this area were Nozaki and Tanford in 1971 [35]. Most
theoretically derived scales are based on X-ray crystallographic data of amino acid
positions in folded proteins. One example is the scale of Rose et al. [39], who calculated
the accessible surface area lost by a residue during the folding process. The Rose scale
was used extensively in this thesis; numerous other scales, all found in Table 3 of [12]
except where otherwise noted, were used as well.
The automatic classification program, provided in BASIC by Prof. Michael J. Korenberg of Queen’s University in Canada, took the form of a 2-way classifier; that is, given an input sequence it classified it in one of two possible classes. The training routine was given an input of two concatenated sequences, one from each family to be distinguished, and an output of -1 at each position along the first sequence, and 1 at each position along the second sequence. The program used PCI to find a model mapping the given input onto the given output. The transition region at the start of the second sequence, where memory from the second sequence overlapped the first sequence, was excluded from the model. Test sequences were then fed into the model and the outputs examined. If the output mean was less than zero the sequence was classified in the first family, and if it was greater than zero the sequence was classified in the second family. The impulse response in Equation 1 could not be formed for sequences that were shorter than the memory length, and they are excluded from all results given below. Such sequences typically constituted less than 5% of the total.

In a pilot study, Korenberg et al. [28] tested the program on a total of 253 sequences from the globin, calcium-binding, and kinase families of proteins. The models were trained on a single sequence from each family (Protein Data Bank identifiers 1hds, 1scp, and 1pfk, respectively). Certain parameters (the confidence level at which cascades were accepted into the model and the polynomial degree in the N element) were set separately for each classifier using a verification set of 34 sequences. First, these results were extended in this thesis by making various modifications to the program. 3-way classifiers were built as combinations of 2-way classifiers; multi- and single-domain training sequences were assayed; and a fourth family of proteins (aminotransferase) was added. Results are summarized in Table 1. As well, the memory length and total number of cascades in the model, held constant in [28], were adjusted to optimal values using the verification set. This latter change led to an improvement in accuracy on the test set of 4% for 2-way classifications and 8% for 3-way classifications.

A control experiment was performed to check whether the problem was non-trivial. The distance between a test sequence and a training sequence was computed using the sum of squares of the differences in Rose hydrophobicity. All possible alignments (with no insertions or deletions) between two sequences of different lengths
were used, and the minimum distance was kept. With this primitive method the prediction accuracy for 2-way classifications was 53%, only slightly better than the expected score of a random classifier, and far below that of PCI.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Original Value</th>
<th>Value</th>
<th>Overall Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>--</td>
<td>--</td>
<td>79%</td>
</tr>
<tr>
<td>Multiplicity of Classification</td>
<td>2-way</td>
<td>3-way</td>
<td>69%</td>
</tr>
<tr>
<td># Domains in Trainer</td>
<td>2</td>
<td>1</td>
<td>61%</td>
</tr>
<tr>
<td># Classes</td>
<td>3</td>
<td>4</td>
<td>75%</td>
</tr>
</tbody>
</table>

*Table 1.* Summary of results obtained by various modifications of the (original) PCI program used in [28].

In order to take advantage of large databases of protein sequences available on the Internet, the classification programs were translated into J++ 6.0 (Visual Java), which supports Web access. Sequences from the web sites of the NCBI [33] and the SwissProt Protein Database [5] were gathered. Results, shown in Table 2, were generally consistent with those obtained on the original test set (Table 1); variation of results between test sets can be attributed to biases in the test sets’ compositions.

<table>
<thead>
<tr>
<th>Test set</th>
<th>NCBI</th>
<th>SwissProt</th>
<th>SwissProt/TrEMBL</th>
<th>PROSITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>3606</td>
<td>2446</td>
<td>8657</td>
<td>2123</td>
</tr>
<tr>
<td>2-way accuracy (%)</td>
<td>75</td>
<td>83</td>
<td>77</td>
<td>75</td>
</tr>
<tr>
<td>3-way accuracy (%)</td>
<td>59</td>
<td>76</td>
<td>67</td>
<td>73</td>
</tr>
</tbody>
</table>

*Table 2.* Testing of PCI protein classification over large data sets available on the Internet. All sets (except PROSITE) were extracted using keyword searches. There was likely significant overlap between different sets. Size = total number of sequences in all 3 families. TrEMBL = less reliable [4] supplement to SwissProt; PROSITE = sequences from SwissProt used to evaluate PROSITE patterns.

### 2.3 Further Modifications to the Classification Algorithm

Many additional modifications were made to the algorithm in order to improve its accuracy. Those that were successful, and some of those that were not, are described below.

Logical combinations of classifiers built using different training sequences were constructed. For example, one classifier could be trusted unless another classifier gave a certain result, in which case the former was overruled. Using these types of arrangements, the accuracy of both 2-way and 3-way classification was raised (Table 3).
However, a larger verification set was required for these models in order to set effective logic schemes.

<table>
<thead>
<tr>
<th>Test set</th>
<th>SwissProt</th>
<th>SwissProt 3-way</th>
<th>PROSITE</th>
<th>PROSITE 3-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy improvement</td>
<td>2</td>
<td>5</td>
<td>11</td>
<td>4</td>
</tr>
</tbody>
</table>

*Table 3.* Improvements in classification accuracy using a logic scheme, in percentage points.

A second input was added in the form of a different hydrophobicity profile – the reasoning being that hydrophobicity scales measured using different methodologies represent different physical properties of the amino acids; therefore a second input may have provided additional relevant information to the model. Increased accuracy was not achieved, however only a very limited study was made along this angle. The main difficulty was that a second hydrophobicity scale, producing results as good as those obtained with the Rose scale, was needed but could not be found (see below); otherwise the second input tended to decrease the quality of the model.

To increase the information available to the model, additional sequences from each family were concatenated in the training file. In particular, sequences that were misclassified by the original model were used. The training program avoided all transition regions. However this did not prove useful, suggesting that a single model was not capable of capturing the entire range of variation of each family (supporting the use of the logic schemes outlined above).

It was noted as well, that since in this system the output was constant, anticipation (i.e. negative values of $i$ in Equation 1) was unnecessary, being equivalent to lengthening the memory. Another observation was made concerning the proportion of first-, second-, and third-order crosscorrelations incorporated into the model: the accuracy of the original model, which randomly selected the order of crosscorrelation, could be nearly reproduced using only second-order crosscorrelations (but not only first- or third-order). This suggested that the dominant forces in determining the general fold of the protein are interactions between two amino acids, rather than one or three.

Another parameter that was varied was the seed for the random number generator. These numbers were used to decide the order (first-, second-, or third-) and (possibly)
slice of crosscorrelation to include in each L element. The verification set was used to select the best seed out of five that were tried.

Various hydrophobicity scales were assayed as inputs to the algorithm, with the four parameters mentioned above being optimized using the verification set. No scale was found that produced classifications as good as those using Rose. Results are summarized in Table 4.

<table>
<thead>
<tr>
<th>Hydrophobicity Scale</th>
<th>Rose</th>
<th>Rose1</th>
<th>Rose2</th>
<th>FP</th>
<th>Eisenberg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Accuracy (%)</td>
<td>83</td>
<td>84</td>
<td>82</td>
<td>70</td>
<td>73</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hydrophobicity Scale</th>
<th>Chothia</th>
<th>Cornette</th>
<th>KD</th>
<th>Manavalan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Accuracy (%)</td>
<td>72</td>
<td>78</td>
<td>59</td>
<td>74</td>
</tr>
</tbody>
</table>

Table 4. PCI classification accuracy using various hydrophobicity scales to specify the sequence inputs. Rose1 = Rose scale to 3 (rather than 2) decimal places using data from the original source [39]; Rose2 = Rose scale adjusted to zero mean; FP = Fauchère-Pliska; Cornette = TOTFT scale of [12]; KD = Kyte-Doolittle scale shown on p.113 of [31]; Manavalan from [32].

It was noted that there were two main weaknesses in the application of PCI to protein classification: the necessity of numerically representing amino acids using hydrophobicity scales (here HMM’s have the advantage of specifying the amino acids simply as different states), and the use of somewhat arbitrary human-determined protein families as static outputs. A system developed by Korenberg [27] for simulating state inputs to PCI was used to address the first weakness. Each of the 20 possible amino acids was represented by a 5-digit sequence, containing either two 1’s or two −1’s, and three 0’s, in some order. Thus there were $2 \times \binom{5}{2} = 20$ possible 5-digit sequences, one for each amino acid. The residues were ordered by increasing hydrophobicity according to the Rose scale. The codes were then assigned in a logical way, with the 10 most hydrophobic residues being coded with 1’s and the 10 most hydrophilic residues coded with −1’s (Table 5).

As well, relative mean square error was now used as the criterion for classification. Thus,

$$Q = \frac{(y - d_i)^2}{(t_i - d_i)^2}$$

was computed, where $y$ is the model output for the test sequence, $d_i$ is the desired static output of −1 or 1 for class $i$, $t_i$ is the model output for the training sequence from class $i$,
and the overbar represents averaging over the sequence length. If \( Q \) was smaller for \( d = -1 \) than for \( d = 1 \) the sequence was classified in the first class, and vice versa.

<table>
<thead>
<tr>
<th>Amino Acid</th>
<th>Code</th>
<th>Amino Acid</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>1,1,0,0,0</td>
<td>g</td>
<td>0,0,0,-1,-1</td>
</tr>
<tr>
<td>f</td>
<td>1,0,1,0,0</td>
<td>t</td>
<td>0,0,-1,0,-1</td>
</tr>
<tr>
<td>i</td>
<td>1,0,0,1,0</td>
<td>s</td>
<td>0,0,-1,-1,0</td>
</tr>
<tr>
<td>v</td>
<td>1,0,0,0,1</td>
<td>r</td>
<td>0,-1,0,0,-1</td>
</tr>
<tr>
<td>l</td>
<td>0,1,1,0,0</td>
<td>p</td>
<td>0,-1,0,-1,0</td>
</tr>
<tr>
<td>w</td>
<td>0,1,0,1,0</td>
<td>n</td>
<td>0,-1,-1,0,0</td>
</tr>
<tr>
<td>m</td>
<td>0,1,0,0,1</td>
<td>d</td>
<td>-1,0,0,0,-1</td>
</tr>
<tr>
<td>h</td>
<td>0,0,1,1,0</td>
<td>q</td>
<td>-1,0,0,-1,0</td>
</tr>
<tr>
<td>y</td>
<td>0,0,1,0,1</td>
<td>e</td>
<td>-1,0,-1,0,0</td>
</tr>
<tr>
<td>a</td>
<td>0,0,0,1,1</td>
<td>k</td>
<td>-1,-1,0,0,0</td>
</tr>
</tbody>
</table>

Table 5. 5-digit coding for amino acids listed in order of decreasing hydrophobicity (c \( \geq f \geq ... \geq k \)) according to the Rose scale.

With these two changes, results improved by about 2% over previous classifications (Table 6). However, the major advantage of the state input method was that the verification set was no longer needed in order to set model parameters for each classification; parameters were found that were effective for all three classifiers.

<table>
<thead>
<tr>
<th>Test set</th>
<th>Original</th>
<th>SwissProt</th>
<th>NCBI2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy (%)</td>
<td>85</td>
<td>83</td>
<td>79</td>
</tr>
</tbody>
</table>

Table 6. Results using the state input scheme. NCBI2 = set of 16144 (1056 globin, 1864 calcium-binding, 13264 kinase) sequences from a broader search, compared with that in Table 2, of the NCBI database.

Up to this point, all PCI models were homogenous over the length of the sequence. However, it is reasonable to expect that amino acids near the ends of the sequence may influence the structure or function of the protein in a different way from those nearer the center. Moreover it was observed that PCI model errors (deviations from the ideal output value) often varied considerably over the length of the sequence, and that profile HMM’s, which produce quite accurate classifications (Section 2.4), generate position-dependent models. Therefore PCI models were built that were a conglomerate of three separate submodels, each covering one third of the sequence’s length. However, perhaps due to overfitting of the training sequence, results with these models were slightly worse than with regular models.
3-way classification accuracy of PCI was markedly lower than 2-way accuracy (Tables 1,2). An attempt was made to perform 3-way classifications using the average outputs of the models. Let \( g \) (i.e. globin score) be the sum of the output means of the globin vs. calcium-binding and globin vs. kinase classifiers for a particular test sequence; define \( c \) and \( k \) analogously for calcium-binding and kinase proteins, respectively. Sequences were classified according to the maximum of \(|g|, |c|, \text{ and } |k|\). Although this method was effective in a different application of PCI [15], here it was not.

In another approach to 3-way classification, a single 3-way classifier was built, with outputs of \(-1, 0, \text{ or } 1\) to characterize the three possible families. While a similar (4-way) classifier had again been successfully implemented in [15], here the models did not produce accurate results.

Finally, the following scheme was attempted: one classifier was built to distinguish globins from non-globins, another to distinguish calcium-binding sequences from non-calcium binding, etc. Then an unknown sequence was classified according to the best output mean amongst the three models. Results improved for globins and kinases but were very poor for calcium-binding sequences.

### 2.4 Combination with a Hidden Markov Model

The classifications produced in [28] were compared with the performance of SAM, the Web-accessible HMM by Hughey and Krogh [20,29]. The same training and test sequences were used as for PCI. It was found that for 2- and 3-way classifications on the original test set SAM’s accuracy was higher than PCI’s (88% and 86% compared with 80% and 69% respectively) but that the two methods tended to err on different sequences, raising the possibility of a combined method. Such a method would make use of some measure of the confidence of a classification by either algorithm. In the case of SAM it was found that the incorrect classifications were primarily amongst those based on the smallest differences in alignment scores (the NLL-NULL column of the hmmscore output file). For PCI, the mean of the output, but not its standard deviation, served as a good confidence level (Figure 1).
Since SAM had higher accuracy, PCI classifications (using hydrophobicity inputs) were substituted for the lowest confidence SAM classifications. These were identified using a threshold (of 1) on the alignment score. Experiments with a verification set showed selection using a threshold to be better than selection using some percentage of the least confident classifications. The choice of threshold caused, on average, 60% of the SAM classifications to be substituted by PCI classifications. The substitution increased the accuracy of classification on a large test set (Table 7) [27].

<table>
<thead>
<tr>
<th>Accuracy (%)</th>
<th>PCI</th>
<th>SAM</th>
<th>SAM w/PCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-way</td>
<td>76</td>
<td>79</td>
<td>83</td>
</tr>
<tr>
<td>3-way</td>
<td>65</td>
<td>73</td>
<td>74</td>
</tr>
</tbody>
</table>

Table 7. Combination of PCI with SAM improved on the solo results of either method. 34 training sequences were used and the test set was NCBI2 (see Table 6).

The improvement for 3-way classification was only slight. A different combination was tried: for marginal classifications of SAM, rather than substituting relatively inaccurate PCI 3-way classifications, only a 2-way PCI classifier was consulted to decide between the two highest-scoring families according to SAM. Surprisingly this did not raise the accuracy.

The confidence of PCI classifications was then incorporated into the scheme. Thus, a PCI classification would only be substituted for one of SAM's when its output mean had absolute value above a certain threshold (Figure 1). However, no improvement
over the previous scheme was found using a verification set to test thresholds of 0.1, 0.2, 0.3, and 0.4.

As described above, PCI with state inputs required only one sequence from each family for training. SAM models were built using exactly these sequences and combined with PCI in the manner just related. Since in this case PCI alone performed better than SAM alone, an analogous scheme in which SAM corrected PCI results was also implemented, with the average PCI output used to identify low confidence classifications. However, the original method of combination gave better results (Table 8). The improvement in accuracy on the more representative NCBI2 test set mirrored results using hydrophobicity inputs (Table 7).

<table>
<thead>
<tr>
<th>Test set</th>
<th>SAM</th>
<th>PCI</th>
<th>SAM w/PCI</th>
<th>PCI w/SAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwissProt 2-way</td>
<td>90</td>
<td>83</td>
<td>90</td>
<td>88</td>
</tr>
<tr>
<td>SwissProt 3-way</td>
<td>86</td>
<td>73</td>
<td>83</td>
<td>--</td>
</tr>
<tr>
<td>NCBI2 2-way</td>
<td>75</td>
<td>79</td>
<td>82</td>
<td>79</td>
</tr>
<tr>
<td>NCBI2 3-way</td>
<td>69</td>
<td>68</td>
<td>72</td>
<td>--</td>
</tr>
</tbody>
</table>

Table 8. Results of combined schemes between PCI (with state input) and SAM. Only 3 training sequences (1 from each family) were used for both methods. The numbers are the average accuracy in %. PCI w/SAM = SAM classifications substituted for low confidence PCI classifications. PCI w/SAM was not implemented for 3-way classifications due to the inferior results for 2-way classifications compared to SAM w/PCI.
Chapter 3. Protein Folding

3.1 Background

Prediction of the folded structure of a given sequence of amino acids is in general an unsolved problem, although significant strides have been made in the last few decades, primarily aided by the increase in the number of experimentally solved structures. Currently a total of about 200,000 protein sequences are known while only about 10,000 3D structures have been mapped experimentally, mainly using X-ray crystallography [8]. Homology modeling is a viable technique for sequences sharing >30% of their residues with sequences that have known structures [40]. In 1998, the SWISS-MODEL homology algorithm generated about 65,000 models of proteins in the 3DCrunch experiment [37]. The reliability of these models is highly dependent on the availability of homologues with known structure. Other methods attempt to find the conformation that minimizes the structure’s potential energy: fold recognition methods (e.g. [13]) thread the sequence onto known folds from a library, while ab initio predictors (e.g. [48]) use molecular dynamics to calculate the optimal conformation from scratch. The latter method has been used only for short stretches of polypeptides due to the extremely high computational cost; however, IBM’s Blue Gene project [22] aims to build a petaflop computer to fully fold an average-length protein with ab initio methods by 2004.

Since there is greater variety in sequences than in structures, a basic problem is to understand how two widely differing sequences can produce a similar structure [40]. Such pairs of sequences are called remote homologues. While progress has been made toward the recognition of remote homologues [40], the biannual CASP [11] competitions for structure prediction have highlighted the inability of current methods to penetrate the “midnight zone” of sequences with no close homologues.

3.2 Tertiary Structure

The use of classifications, which are not inherently numerical, as the system output was mentioned above (Section 2.3) as a weakness of the PCI approach. With this in mind, a new output was proposed: the 3-dimensional coordinates of the amino acids in the folded structure. This output had the advantages of being continuous, non-subjective,
and dynamic (in space, along the residue), and thus more natural as a time series output of a system identification problem. The coordinates used were those from the Protein Data Bank (PDB), currently at the RCSB web site [8]. As a simplification, only the coordinates of the \( C^\alpha \) atom, which lies on the backbone, were considered, as is widely practiced (e.g. [34]).

For simplicity only one coordinate was used: the distance of the amino acid from the protein center. This was chosen because it should correlate well with the input hydrophobicity; on the other hand, since many proteins are far from spherical, highly accurate models using this measure could not be expected. As a preliminary experiment, one protein (PDB identifier 1mbc) was chosen at random and a model trained on its data until a root mean square deviation (rmsd) of less than 0.2 nm was reached. The model was then tested on four other sequences of varying identity with 1mbc (Table 9). Three of the sequences had similar structure to 1mbc and one (1bla) did not. The model output, shown in Figure 2, showed qualitative agreement with the actual structure for these five cases.

<table>
<thead>
<tr>
<th>Protein</th>
<th>1mbs</th>
<th>1hs</th>
<th>1hl</th>
<th>1bla</th>
</tr>
</thead>
<tbody>
<tr>
<td>% identity with 1mbc</td>
<td>85</td>
<td>66</td>
<td>15</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table 9.** Similarity between 1mbc and four preliminary test sequences for structure prediction. % identity (i.e. common residues) was calculated for the best alignment using the online program Combinatorial Extension of the Optimal Path [44].

A set of 89 PDB entries, with no pair of sequences sharing \( \geq 25\% \) of their residues, was found in [49]. 8 of the sequences had incomplete records in the database. 5 of the complete sequences were chosen at random (3b5c, 8dfr, 8rnt, 1fba, and 7aat) and concatenated to form a single training file. Another randomly selected 5 complete sequences (4enl, 3chy, 1hsb, 1gmp, and 2ctc) served as the verification set, and were used to find optimal values for memory length, number of cascades in the model, threshold for accepting cascades into the model (see [26]), and polynomial degree of the \( N \) element.

As well, several different hydrophobicity scales were evaluated as inputs, again using the verification set. As can be seen in Table 10, the Rose scale provided the most accurate models. A “hydrophobicity” scale of pseudo-random numbers was also used as
a control; it produced surprisingly good results, indicating that the hydrophobicity scales did not in fact correlate closely with radial distance.

**Figure 2.** Model output (dotted line) and actual radial coordinate (solid line) for 1mbc and four other sequences, using a model trained on 1mbc.
Table 10. Different hydrophobicity scales used as inputs for structure models. Rmsd's in nm.

For the Rose scale, the optimal parameters found using the verification set are shown in Table 11. The rmsd of this model on the test set was 0.608 nm. When the incomplete sequences were excluded from the results the rmsd became 0.599 nm.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Memory</th>
<th>Anticipation</th>
<th># cascades</th>
<th>Threshold</th>
<th>Poly deg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>12</td>
<td>12</td>
<td>20</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 11. Parameter values for prediction of radial coordinate. Anticipation length was always held equal to memory length. Poly deg = degree of polynomial in N element.

45 of the sequences in the entire set were single-chain proteins (monomers) while the remainder were individual chains from multi-chain proteins (multimers). When a model was similarly trained using only monomers in the training and verification sets, no appreciable difference was found between its accuracy on monomers (0.565 nm) and multimers (0.569 nm). This was in keeping with the accepted belief that inter-chain contacts have little effect on conformation; however it also highlighted the sensitivity of results to the sequences chosen for the training set.

While the number of disulfide bridges (unaccounted for in the models) did not correlate with model accuracy, sequence length did, with longer sequences being modeled less accurately (Figure 3), perhaps due to the presence of very long range interactions.

A series of modifications, some of which are described below, were made to attempt to drive down the rmsd.

It was observed that higher polynomial degree models tended to capture high frequency modulation of the output (i.e. secondary structure of the protein), however they were too volatile. Two heuristic smoothing approaches were taken to correct this: (a) limiting jumps from one coordinate to the next and (b) eliminating large deviations with respect to a model generated with a lower-degree polynomial; however neither produced overall improvement.
Another scheme was then attempted: two models with 2\textsuperscript{nd}-degree polynomials in the N element were arranged in series, i.e. the output of the first was the input of the second. The models were intended to emulate the secondary and tertiary structure, respectively. This did not prove successful, producing models that were too smooth and inaccurate. The smoothness was the result of the second model having input and output with low correlation at non-zero lags, due to the fit from the first model. This caused the crosscorrelations used in the second model’s L element (Equation 3) to act as windows. A potential remedy, which was not explored, could be to clump the input to the second model in order to simulate tertiary folding taking place on a larger scale, or to restrict input to the first model to a local window of residues, as is commonly practiced in secondary structure predictors (e.g. [41]).

Since the sequence is constrained to be in one piece, and this constraint is not explicitly present in the model, the input was changed to differential coordinates. That is, the difference between the current and previous coordinates was substituted for the current coordinate itself. This too was unsuccessful, perhaps because the model could no longer differentiate between clusters of low- and high-hydrophobicity residues.
3.3 Secondary Structure

Many 3D structure prediction algorithms utilize a secondary structure predictor as a stepping stone. Currently, the best secondary structure prediction algorithms achieve 3-state (α-helix, β-strand, other) accuracy around 75% [23]. An attempt was made to predict secondary structure using PCI. 8-state secondary structure according to the DSSP classification was downloaded from the RCSB web site, and translated to 3-state form according to the mapping given in [42].

Two approaches were taken. The first was to build a single model that would output −1 for an α-helix, 1 for a β-sheet, and 0 for a loop. The second was to build one model that would output −1 for an α-helix and 1 otherwise, and similar models for β-sheets and loops, and then to combine the three to form 3-state predictions. The latter method gave better results on a verification set, with 2-state accuracy of about 70% and 3-state accuracy of about 53%. Nevertheless this was far below the performance of the best existing methods.
Chapter 4. ATP Binding Sites

4.1 Background

A wide variety of proteins from disparate families bind adenosine 5'-triphosphate (ATP), the primary transporter of energy in the cell. The majority of such proteins contain an 8-residue binding site called a P-loop (or Walker A site [46]) at which the ATP molecule is attached. The P-loop is normally found between an α-helix and a β-strand. P-loops may be fingerprinted by the sequence G-X-X-X-X-G-K-[ST], where X represents any amino acid and [ST] represents either serine or threonine. The PROSITE [6] profile contains [AG] rather than G at the beginning. However, both of these patterns identify many false positives when a large database is scanned for their occurrence [43].

4.2 Discernment of Binding P-loops from Non-binding P-loops

Hirst and Sternberg [19] reported using both a neural network and a statistical method to discriminate P-loops that bind ATP from those that do not. They used the amino acid sequence in a 17-residue neighborhood of the P-loop as input to each algorithm. Their rate of success was 78% with the neural network and 84% with the statistical method, using a jack-knife (leave-one-out) test.

The two-layer feed-forward neural network of Hirst and Sternberg used the following model:

\[ y = \sum_{i,j} w_{ij} \cdot \delta_{ij}, \]

where \( \delta_{ij} \) is 1 if residue \( i \) is present at position \( j \) in the fragment, and 0 otherwise; and \( w_{ij} \) are weights. The desired value of the output \( y \) was positive for an ATP-binding sequence and negative for a non-binding sequence. The optimal values of the weights were learned using approximately 30 iterations of proportional feedback control.

Hirst and Sternberg’s statistical method was an unspecified algorithm that made use of the PAM250 matrix by Dayhoff [14]. This matrix is a tabulation of the estimated likelihoods of all possible amino acid point substitutions during the evolution of a protein.
Hirst and Sternberg's test set consisted of 349 sequences (197 binding and 152 non-binding) available in 1991 from the SwissProt database (release 14). These sequences were retrieved from the SwissProt database in 1999 (release 38), with the exception of 8, which either could not be found, did not contain P-loops, or were duplicates (some sequences are updated when newer data are published). Thus the set used here contained a total of 341 sequences, divided into 192 which bind ATP and 149 which do not.

A fragment of length 17 was again found to be optimal for methods used in this work, which are described below. Hirst and Sternberg did not include the conserved [AG], G, K, or [ST] positions of the P-loop in their inputs. However the identity of the residue at the [AG] position appears to bear significantly on whether or not the protein binds ATP (Table 12). Despite this it was found that the inclusion of these residues had little effect on results in this section; for consistency, results reported below are for the same fragments as those used by Hirst and Sternberg.

<table>
<thead>
<tr>
<th>[AG] position</th>
<th># Binding</th>
<th># Non-binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>34</td>
<td>77</td>
</tr>
<tr>
<td>G</td>
<td>158</td>
<td>72</td>
</tr>
</tbody>
</table>

Table 12. Glycine residues occur more commonly in the [AG] positions of P-loops that bind ATP.

A PCI classifier was built to distinguish binding from non-binding sequences and was tested via multiple cross-validation; 5 pairs of sequences (1 binding and 1 non-binding) were removed from the test set and used to train models, then the remainder of the sequences were tested with the resulting 5 models and the majority vote taken. Accuracy was low at only 70%. Increasing the number of trained models and/or using state input provided no improvement.

For a simpler approach, a jack-knife test was then set up and the nearest neighbor method was used to predict whether each sequence bound ATP. A number of distance measures were separately tested for the purpose of identifying the nearest neighbor: the difference in residue-by-residue Rose hydrophobicity measured with a norm (L1 and L2 were tried); the correlation between Rose hydrophobicities; the identity (number of shared residues); the identity with allowance for certain amino acid substitutions that are
known to generally have little effect on protein function [9]; two substitution matrices, the PAM250 and the more recently proposed BLOSUM62 [18]; and finally, the mean square error of a 2nd-order FOA model mapping one sequence onto the other. For FOA models, both Rose input with memory length 2 and state input with memory length 5 were tried. Longer memory lengths would have introduced more parameters than data into the models. FOA was selected over PCI because of its speed and because the short fragments used for input necessitated short memory lengths.

As an alternative to the nearest neighbor method, sequences could instead be classified using any of these measures by calculating the average distance to each class.

Finally, considering the residues as independent random variables, a decision rule was formulated for determining, given a test sequence, to which class it most likely belongs:

\[
\prod_{i=1}^{L} \frac{n_{i,b}}{n_{i,nb}} \geq \left( \frac{n_b}{n_{nb}} \right)^{L-1},
\]

where \( L \) is the fragment length, \( n_{i,b} \) is the number of training sequences in the binding class which have the same amino acid at position \( i \) as the test sequence, \( n_b \) is the total number of binding sequences in the training set, and \( n_{i,nb} \) and \( n_{nb} \) are defined similarly for non-binding sequences (derivation in Appendix A). The test sequence was classified as binding if and only if the \( \geq \) relation held. This method is a (very degenerate) profile HMM with no insertions or deletions.

Results were on a par with those of Hirst and Sternberg [19]. A summary is provided in Table 13. Performance using average distance was inferior to nearest neighbor due to the presence of close homologues in the test set, which were easily detected by the latter method. Lower accuracy for non-binding sequences, especially when using average distance, indicated more variation amongst sequences in the non-binding set than in the binding set. Specifically, non-binding membrane proteins\( ^3 \) were often incorrectly classified using average distance. Membrane proteins constituted about 15% of the test set; they are exposed to a different cellular environment than soluble

---

\(^2\) A BASIC program implementing FOA was provided by Prof. Korenberg.

\(^3\) Only proteins which were clearly identified as residing in the membrane by the SwissProt database were counted as such; since there is no explicit annotation for this feature, some uncertainty was involved in compiling the list.
proteins, which uniquely affects their structure. In [41] secondary structure prediction using neural networks was similarly lower for 4 chains of a membrane protein.

<table>
<thead>
<tr>
<th>Distance Measure</th>
<th>Average distance</th>
<th>Nearest neighbor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Binding</td>
<td>Non-binding</td>
</tr>
<tr>
<td>L1</td>
<td>90</td>
<td>53</td>
</tr>
<tr>
<td>L2</td>
<td>91</td>
<td>52</td>
</tr>
<tr>
<td>Correlation</td>
<td>90</td>
<td>52</td>
</tr>
<tr>
<td>Identity</td>
<td>92</td>
<td>54</td>
</tr>
<tr>
<td>ID w/subs.</td>
<td>92</td>
<td>47</td>
</tr>
<tr>
<td>PAM250</td>
<td>94</td>
<td>44</td>
</tr>
<tr>
<td>BLOSUM62</td>
<td>91</td>
<td>55</td>
</tr>
<tr>
<td>FOA Rose</td>
<td>81</td>
<td>48</td>
</tr>
<tr>
<td>FOA state</td>
<td>84</td>
<td>64</td>
</tr>
<tr>
<td>Probability</td>
<td>79</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 13. Results for jack-knife method. The numbers represent percentage correct. ID w/subs. is identity measure with substitutions allowed.

If the results of the nearest neighbor method with the L1 measure were ordered starting with the shortest nearest neighbor distances (i.e. the most confident predictions), the first half were 99% accurate as a whole. None of the methods were found to give improved results over the remaining set.

Similar results to those using Rose hydrophobicity were obtained using the solvent exposed area [9] or the turn preference [3, Table I] of the residues. FOA’s poor performance as compared with the simple L1 measure may have been due to its inability to capture position-dependent effects. The residue weights of Hirst and Sternberg’s neural network showed strong position-dependence.

Hirst and Sternberg used their statistical method to score each sequence against the ATP-binding set; those scoring above a certain cutoff were then classified as binding. The cutoff was chosen (after-the-fact) such that it maximized the number of correct predictions. This is not strictly a fair test, but for comparison’s sake it was implemented here as well. Using the PAM250 matrix the accuracy was 85%, using the BLOSUM62 matrix it was 86%, and using the L1 norm it was 81%. Thus direct use of the PAM250 matrix produced accuracy slightly higher than the statistical method of Hirst and Sternberg.
Since hydrophobicity is a relative measure, a shift term may be added uniformly to each hydrophobicity in one sequence before comparing it with another, in order to compare relative hydrophobicity only. In the case of the $L_1$ norm the optimal shift is that which sets the median of the pairwise differences in hydrophobicity to zero$^4$. When this term was added results degraded slightly, perhaps because the conserved glycine and lysine residues in the P-loop provide a fixed reference for the hydrophobicities of the surrounding residues.

### 4.3 Neural Networks

Hirst and Sternberg's neural network was reproduced as closely as possible according to the specifications given in [19]. Its prediction accuracy was found to average 75.4\% with a standard deviation of 2.3\% over 50 trials, slightly lower than their report. This may have been due to small differences in the test set and/or different stopping criteria for the network training. Inclusion of the [AG] and [ST] residues increased accuracy to 76.5\% ± 1.6\%, despite the fact that the number of parameters was now greater than the number of training sequences (a similar result is discussed in [30]). The subset of membrane proteins scored about 5\% lower than the average, but for consistency was included in what follows. Increasing the learning rate from 0.05 (Hirst and Sternberg's value) to 0.3 improved average accuracy to 83.0\% ± 1.0\%. By running multiple neural networks and taking the majority vote prediction accuracy was improved to above 85\% (Figure 4). Neural network outputs could be used as confidence levels, as shown in Figure 5; therefore a more accurate voting procedure was obtained by adding the outputs of the voters (Figure 4).

A similar improvement could be realized by maintaining the lower learning rate and increasing the number of voters to 45; in this case the individual voters were less accurate, being trapped in local minima close to their random starting positions, but voting was more effective because of the diversity of the voters.

Initializing the network weights to probabilities calculated similarly to the values in Equation 6 did not result in an improvement in accuracy.

$^4$ If the majority of the residues in the first sequence have greater hydrophobicities than their counterparts in the second sequence, then the distance between the two can be reduced by shifting the first sequence down, and vice versa.
Figure 4. Improvement in network results with voting, at a learning rate of 0.3. Data from multiple trials; expected standard deviation of each point (not shown for clarity) was about 0.15%.

Figure 5. Output values used as confidence levels of individual neural networks. Data was averaged over 5 trials.

4.4 Larger Test Set

A current test set of sequences containing P-loops was obtained from the SwissProt database using the ScanProsite tool [6]. There were 1061 binding and 1458
non-binding sequences, classified according to SwissProt annotation\(^5\). The new classifications were 92% consistent with those of Hirst and Sternberg over their test set.

The nearest neighbor method with L\(_1\) measure identified 90.2% of the fragments correctly in the new test set. The BLOSUM matrix nearest neighbor classification was correct for 90.8%. Higher accuracy was to be expected since there was a greater likelihood of finding a homologue to a given sequence in the larger test set. The replica of Hirst and Sternberg’s neural network was on average 77.3% accurate. The extra data allowed the addition of a hidden layer to the network, however this did not improve its performance. This served as evidence of the unimportance of inter-residue effects in the P-loop. This is consistent with the findings of Qian and Sejnowski [38] on using neural networks to predict secondary structure.

Voting did not improve neural network results on the large test set. This may have been because there were now far more data than parameters in the system, so networks always converged to the same optimal solution. Previously, the existence of homologous sequences in the smaller test set lowered the amount of meaningful data present, creating an under-determined problem with multiple solutions which could serve as voters.

\(^5\) Sequences with the line entry FT NP_BIND ... ATP/GTP were classified as binding, unless followed by (POTENTIAL). Sequences with records not containing the words ATP or GTP were classified as non-binding. All others were discarded as ambiguous.
Chapter 5. Conclusion

5.1 Summary

A study was made of the application of nonlinear system identification algorithms to three problems in protein structure prediction. While the protein folding problem has generated a vast literature in which it has been addressed with a host of mathematical procedures, system identification techniques have seldom been employed.

The work on protein classification, described in Chapter 2, was a validation and extension of preliminary work carried out by Korenberg et al. [28]. Applications of system identification methods to structural and binding site prediction, covered in Chapters 3 and 4 respectively, constituted original contributions. Chapter 4 also contains modifications to Hirst and Sternberg’s methods [19].

Varying degrees of success were achieved in the problems investigated. For protein classification, preliminary PCI results were confirmed using large test sets, and performance was enhanced with the use of logic schemes and state input. Results were generated which in some cases rivaled those of a state-of-the-art hidden Markov model. Combination of the two methods proved beneficial. The application of PCI to 1-dimensional prediction of protein structure showed some promise but was pursued to only a very rudimentary stage. Lastly, while system identification methods provided no advantage for identifying ATP binding sites, simple procedures were developed for the task that out-performed methods from the literature, and improvements were made to a neural network approach. The best methods were then re-tested on more current data.

5.2 Future Work

Suggested future work on protein classification includes an extension of results to more than three families, and testing of Korenberg’s idea [27] of concatenating state input codes from multiple inputs. As well, a natural way to introduce position-dependence to the models is desirable. The algorithm should be applied to a relevant classification problem in which little training data is available and critically compared with other methods. It would also be interesting to open the black box and discover how
PCI distinguishes different classes from each other. Perhaps these extracted rules may be combined with a position-dependent approach.

A more comprehensive evaluation of the potential of the methods used for 1-dimensional structure prediction should be made; for example, a less primitive measure than the residue’s radial distance, such as the actual degree of burial computed from solved coordinates, should be used. Another possible improvement would be the use of state inputs rather than hydrophobicity profiles. Furthermore, ways for integrating the comparatively simple system identification tools into some more sophisticated and tailored methods may be explored, since protein folding is a highly complex process.

Results for prediction of ATP binding sites should be weighed against those of other existing alignment tools (such as BLAST) and/or secondary structure predictors; if they compare favorably, similar methods may be applied to other problems such as prediction of phosphorylation sites or DNA binding sites.
Appendix A. Decision Rule Derivation

The decision rule that minimizes the probability of error is to declare a sequence $S$ of length $L$ as binding if and only if

$$
P(b \mid S) \geq P(nb \mid S)
\iff P(S \mid b)P(b) \geq P(S \mid nb)P(nb)
$$

where $b$ stands for binding and $nb$ for non-binding. If the residue identities are independent of each other then

$$
P(S \mid b) = \prod_{i=1}^{L} P(i \mid b) \quad \text{and} \quad P(S \mid nb) = \prod_{i=1}^{L} P(i \mid nb),
$$

where the products are over the residues along the length of the sequence. Using

$$
P(i \mid b) = \frac{n_{i,b}}{n_b} \quad \text{and} \quad P(i \mid nb) = \frac{n_{i,nb}}{n_{nb}},
$$

$$
P(b) = \frac{n_b}{n_b + n_{nb}} \quad \text{and} \quad P(nb) = \frac{n_{nb}}{n_b + n_{nb}},
$$

(notation from Section 4.2) we substitute Equations 8, 9, and 10 back into Equation 7 to get the decision rule

$$
\prod_{i=1}^{L} \frac{n_{i}}{n_{i,nb}} - \left( \frac{n_b}{n_{nb}} \right)^{L-1}.
$$
Appendix B. Code

The principal algorithmic functions are shown; some functions and variables referred to in the code below are omitted.

- The following was the top-level function controlling data collection from various sources and testing for protein classification (Sections 2.2 and 2.3):

```java
public void button1_click(Object source, Event e)
{ int numclass = 1;

    // Initializations
    record = "";
    richEdit1.setText("");
    edit1.setText("");
    start_time = new Date();
    JT.show();
    // note: the program must be restarted each time for this function to work
    // properly; I don't know why yet
    if (checkBox8.getChecked()) // read classifications from record file
        ReadRecordFile();
    if (checkBox5.getChecked()) DoLogic();
    if (checkBox7.getChecked()) ThreeWay();
    return;
}
if (checkBox2.getChecked()) PrepareRecordFile();
use_avg_output = checkBox10.getChecked();
threshold = checkBox11.getChecked();
if (threshold) thresh = Double.valueOf(editl3.getText().doubleValue();
ternary = checkBox12.getChecked();
jack_knife = checkBox14.getChecked();
if (!siteselected)
    AddText("Please select the source for the sequences from the drop-down list.");
else if (radioButton2.getChecked() & !scaleselected)
    AddText("Please select the hydrophobicity scale to be used.");
else if (comboBox1.getSelectedItem().equals("NCBI UID's"))
    { 
        JT.edit3.setText(editll.getText());
        filebutton(edit6.getText());
    }
else if (comboBox1.getSelectedItem().equals("Swiss-Prot"))
    { 
        if (comboBox2.getSelectedItem().equals("Rose & Chothia"))
            AddText("Rose & Chothia not available for Swiss-Prot sequences.");
        return;
    }
Ian4 SP = new Ian4();
//SP.show();
// transfer settings to SP: a different form handles this file format
SP.edit6.setText(edit6.getText());
SP.edit8.setText(edit8.getText());
SP.radioButton1.set_checked(radioButton1.getChecked());
SP.radioButton2.set_checked(radioButton2.getChecked());
SP.radioButton4.set-checked(true); // get letters, translated by javatext
JT.comboBox1.setSelectedItem(comboBox2.getSelectedItem()); // set hydr scale
JT.edit11.setText(edit11.getText());
Jt.edit3.setText(editl2.getText());
if (checkBox4.getChecked())
    { SP.edit6.setText(edit9.getText());
        filebutton(edit9.getText());
        JT.edit3.setText(editl2.getText());
    }
39
// SwissProt with feature information, or Hirst ATP binding sites
else if (comboBox1.getSelectedItem().equals("Swiss-Prot FT"))
{ if (jack_knife)
    JackKnife();
else
{
    clean LemonFresh = new clean(); // extract seqs 1-by-1 from list file
    LemonFresh.show();
}

for (int j=0; j<2; j++) // 2nd list file
{ nto = nw = nsh = nerr = t_nw = t_nsh = 0; // t_ = using threshold
  if (j == 1 && !checkBox4.getChecked()) break; // 2nd list file
  if (j == 0) LemonFresh.edit1.setText(edit6.getText()); // filename
  else if (j == 1) LemonFresh.edit2.setText(edit9.getText());
  LemonFresh.edit2.setText("Protein/temp.dat"); // sequence file
  LemonFresh.edit3.setText("1"); // number of sequences
  LemonFresh.radio1Button1.setChecked(true);
  LemonFresh.checkBox1.set(boolean); // select from top of file
  LemonFresh.checkBox5.set(boolean); // test file format
  LemonFresh.button1_click(null, null);
  nw = nsh = nto = 0; // # wrong, short, total
  jt.comboBox1.setSelectedItem(comboBox2.getSelectedItem());
  if (j == 0) jt.edit3.setText(edit11.getText()); // ideal value
  else if (j == 1) jt.edit3.setText(edit12.getText());
  Analysis();

  // shortcut once all sequences have been read on 1st run
  LemonFresh.checkBox6.set(boolean);
  for (int i=1; i<LemonFresh.numseqs; i++)
  { LemonFresh.edit6.setText("" + i);
    LemonFresh.button2_click(null, null);
    Analysis();
  }
  LemonFresh.checkBox6.set(false); // reset
  AddText(str);
  if (threshold) AddText(t_str);
}
}

// Protein DataBank file format, www.rcsb.org
else if (comboBox1.getSelectedItem().equals("Files (pdb)"))
{ jt.edit1.setText(edit6.getText()); // jt is the testing form that uses PCI
  jt.edit3.setText(edit11.getText());
  jt.rich1Button1.setText(""");
  jt.checkBox3.set(checkBox9.getChecked()); // anticipation
  jt.checkBox4.set(checkBox12.getChecked()); // ternary classifier
  jt.comboBox1.setSelectedItem(comboBox2.getSelectedItem()); // hydr scale
  jt.show();
  if (checkBox6.getChecked()) // multiple classifiers
  { numclass = (new Integer(edit10.getText())).intValue();
    for (int i=0; i<numclass; i++)
    { if (i == 0) jt.gobutton(); // call the javatest form
      else jt.gobutton("paralin" + i + ".dat", "parapoly" + i + ".dat");
      if (threshold)
      { for (int j=0; j<jt.long return value.length(); j++)
        { // insert 's' for uncertain classifications
          if (Math.abs(jt.avgoutput[j]) < thresh)
            jt.long return value = jt.long return value.substring(0, j) + 's' +
            jt.long return value.substring(j+1, jt.long return value.length());
        }
        if (checkBox2.getChecked()) record += jt.long return value + '\n';
      } if (checkBox3.getChecked()) // logic matrix
        { index = jt.long return value.length();
...
for (int j=0; j<index; j++)
{ logic[i][j] = jt.long_return_value.charAt(j);
 if (use_avg_output) avg_outputs[i][j] = jt.avg_output[j];
}

if (checkBox5.getChecked()) DoLogic();
else if (checkBox7.getChecked()) ThreeWay();
if (checkBox4.getChecked()) // 2nd list file
{ jt.edit1.setText(edit9.getText());
 jt.edit3.setText(edit12.getText());
 for (int i=0; i<numclass; i++)
 { if (i == 0)jt.gobutton(); // call the javatest form
  else jt.gobutton("paralin" + i + ".dat", "parapoly" + i + ".dat");
  if (threshold)
  { for (int j=0; j<jt.long_return_value.length(); j++)
     if (Math.abs(jt.avg_output[j]) < thresh)
       { logic[i][j] = jt.long_return_value.substring(0,j) + 's' +
         jt.long_return_value.substring(j+1, jt.long_return_value.length());
      }
   }
  }
  if (checkBox5.getChecked()) DoLogic();
  else if (checkBox7.getChecked()) ThreeWay();
}
else if (comboBox1.getSelectedItem().equals("NCBI Fasta"))
{ jt.edit1.setText(edit11.getText()); // ideal value
  jt.comboBox1.setSelectedItem(comboBox2.getSelectedItem()); // hydrophobicity scale
  Fasta(edit6.getText());
  if (checkBox4.getChecked()) // 2nd list file
  { jt.edit3.setText(edit12.getText());
    Fasta(edit9.getText());
  }
}
if (checkBox2.getChecked()) // record file selected
{ try
  { RecordOut.write(record);
    RecordOut.close();
  }
  catch (IOException exc) { AddText("RecordOut.write: "+ exc.toString());
  }
}

Date end_time = new Date();
double total_time = (end_time.getTime() - start_time.getTime())/1000.0;
edit4.setText("total_time = " + total_time);

• This function kept track of protein classification results (Sections 2.2 and 2.3):

public void Analysis()
{ int numclass = 1;
  char result = 'e';
  FileWriter Score = null;

  if (comboBox2.getSelectedItem().equals("Rose & Chothia")) // 2 inputs
    jt.checkBox1.setchecked(true);
  else
    jt.checkBox1.setchecked(false);
  if (checkBox6.getChecked()) // multiple classifiers
    numclass = (new Integer(edit10.getText()).intValue());

  for (int j=0; j<index; j++)
  { logic[i][j] = jt.long_return_value.charAt(j);
    if (use_avg_output) avg_outputs[i][j] = jt.avg_output[j];
  }
  }
The following function computed protein classification results based on one example of a logical combination of classifiers (Section 2.3):

```java
public void DoLogic()
{
    int not_tall = 0, class1 = 0, class2 = 0; // short is a keyword
    char combo[] = new char[MAX];
    // CK scheme of 6/1/99
    for (int i=0; i<index; i++)
    { if (logic[1][i] == 'o' || logic[3][i] == 'o') combo[i] = 'o';
    }
}```
else if (logic[2][i] == 'x') combo[i] = 'o';
else combo[i] = logic[0][i];
}
for (int i=0; i<=index; i++)
{ if (combo[i] == 'o') class1++;
  else if (combo[i] == 'x') class2++;
  else if (combo[i] == 's') not_tall++;
  if (checkBox2.getChecked()) record += combo[i];
} if (checkBox2.getChecked()) record += '\n';
AddText("Logic results:");
AddText((index+1) + " total");
AddText(class1 + " in class #1, " + class2 + " in class #2");
AddText(not_tall + " too short to classify" + '\n');

- This function produced 3-way classification using three 2-way classifiers (Sections 2.2 and 2.3):

```java
// 3-way classification, assumes classifiers are ordered 1 v. 2, 1 v. 3, 2 v. 3
public void ThreeWay()
{
  int class1 = 0, class2 = 0, class3 = 0, undet = 0;
  double g, c, k;
  String s = "";
  for (int i=0; i<index; i++)
  { // 3-way classification by logical combination of 2-way classifiers
    if (!use_avg_output)
    { if (logic[0][i] == 'o' && logic[1][i] == 'o')
      { class1++;
        s += '1';
      }
    else if (logic[0][i] == 'x' && logic[1][i] == 'o')
      { class2++;
        s += '2';
      }
    else if (logic[1][i] == 'x' && logic[2][i] == 'x')
      { class3++;
        s += '3';
      }
    else
      { undet++; // undetermined
        s += 'O';
      }
    }
  }
  // 3-way classification using the average outputs of the 2-way classifiers
  g = avg_outputs[0][i] - avg_outputs[1][i];
  c = avg_outputs[0][i] - avg_outputs[2][i];
  k = avg_outputs[1][i] + avg_outputs[2][i];
  if (Math.max(Math.max(g, c), k) == g)
  { class1++;
    s += '1';
  }
  else if (Math.max(c, k) == c)
  { class2++;
    s += '2';
  }
  else
  { class3++;
    s += '3';
  }
  AddText("" + Math.max(Math.max(g, c), k));
  record += s;
  AddText("3-way classification\n");
  AddText("Class 1: " + class1);
```

43
This function compiled protein classification results from the output file of SAM and combined them with stored results of PCI (Section 2.4):

```java
private void button1_click(Object source, Event e)
{
    int class1 = 0, class2 = 0, class3 = 0;
    int mpct, mclass1 = 0, mclass2 = 0, mclass3 = 0;
    int pclass1 = 0, pclass2 = 0, pclass3 = 0;
    double mthresh;
    FileWriter RecordFile = null;
    boolean record = false;
    boolean threeway = checkBox2.getChecked();
    boolean SAMcorrectsPCI = checkBox5.getChecked();
    double diffs[]][ = new double[3][MAX], margins[] = new double[MAX];
    char classes[] = new char[MAX];
    int mins[];
    PCI = checkBox3.getChecked();
    binary_correct = checkBox4.getChecked();
    mthresh = (new Double(edit6.getText()).doubleValue;
    for (int i=0; i<MAX; i++)
    { scores[0][i] = scores[1][i] = scores[2][i] = 0;
        seqids[i] = "";
        classes[i] = 'e';
        PCIclasses[0][i] = PCIclasses[1][i] = PCIclasses[2][i] = 'e';
    }
    richEdit1.setText(""");
    try
    {
        Reader1 = new FileReader(edit1.getText());
        Reader2 = new FileReader(edit2.getText());
        if (threeway) Reader3 = new FileReader(edit3.getText());
        record = checkBox1.getChecked();
        if (record) RecordFile = new FileWriter("record-hmm.txt");
        HammerTime(Reader1, 0); // reads the output file from SAM
        HammerTime(Reader2, 1);
        if (threeway) HammerTime(Reader3, 2);
        if (PCI) ReadRecordFile();
        for (int i=0; i<numseqs; i++) // diffs used to identify marginal SAM results
        { diffs[0][i] = Math.abs(scores[0][i] - scores[1][i]);
            if (threeway)
            { diffs[1][i] = Math.abs(scores[0][i] - scores[2][i]);
                diffs[2][i] = Math.abs(scores[1][i] - scores[2][i]);
            }
            if (scores[0][i] == 0 || scores[1][i] == 0 || (threeway && scores[2][i] == 0))
            AddText("bad: " + i);
        }
    int lynyrd = 0;
    // classify as the minimum score (all scores are negative)
    // ties assigned to the later class
    for (int i=0; i<numseqs; i++)
    { if (threeway)
        { if (scores[0][i] < scores[1][i])
            { if (scores[0][i] < scores[2][i])
                class1++;
                if (record) RecordFile.write("a");
                margins[i] = Math.min(diffs[0][i], diffs[1][i]);
                if (margins[i] < mthresh) mclass1++;
                classes[i] = 'a';
            }
        }
    }
    }
```
```java
else
    class3++;
    if (record) RecordFile.write("c");
    margins[i] = Math.min(diffs[1][i], diffs[2][i]);
    if (margins[i] < mthresh) mclass3++;
    classes[i] = 'c';
} else if (scores[1][i] < scores[2][i])
    class2++;
    if (record) RecordFile.write("b");
    margins[i] = Math.min(diffs[0][i], diffs[2][i]);
    if (margins[i] < mthresh) mclass2++;
    classes[i] = 'b';
} else
    class3++;
    if (record) RecordFile.write("c");
    margins[i] = Math.min(diffs[1][i], diffs[2][i]);
    if (margins[i] < mthresh) mclass3++;
    classes[i] = 'c';

// replace marginal classification with PCI when PCI gives result
if (margins[i] < mthresh && PCI)
    if (!binary_correct && PCIclasses[0][i] == 's')
        classes[i] = PCIclasses[0][i];
    else if (binary_correct) // avoid error for 3 evenly spaced results
        if (margins[i] == diffs[0][i] && classes[i] == 'c' &&
            PCIclasses[0][i] == 's')
            classes[i] = PCIclasses[0][i];
        else if (margins[i] == diffs[1][i] && classes[i] == 'b' &&
            PCIclasses[1][i] == 's')
            classes[i] = PCIclasses[1][i];
        else if (margins[i] == diffs[2][i] && classes[i] == 'a' &&
            PCIclasses[2][i] == 's')
            classes[i] = PCIclasses[2][i];
    } // double marginality: substitute PCI 3-way classification
if (binary_correct && Math.max(diffs[0][i], Math.max(diffs[1][i],
    diffs[2][i])) < mthresh && PCIclasses[0][i] == 's' &&
    PCIclasses[1][i] == 's')
    if (PCIclasses[0][i] == PCIclasses[1][i]) classes[i] = 'a'; // must be
    else if (PCIclasses[0][i] == PCIclasses[2][i]) classes[i] = 'b';
    else if (PCIclasses[1][i] == PCIclasses[2][i]) classes[i] = 'c';
} else // 2-way classification
    if (!SAMcorrectsPCI)
        if (scores[0][i] < scores[1][i])
            class1++;
            if (record) RecordFile.write("a");
            classes[i] = 'a';
            if (diffs[0][i] < mthresh) mclass1++; // below threshold
        else
            class2++;
            if (record) RecordFile.write("b");
            classes[i] = 'b';
            if (diffs[0][i] < mthresh) mclass2++;
    } else // replace w/PCI
        if (diffs[0][i] < mthresh && PCI && PCIclasses[0][i] == 's')
            classes[i] = PCIclasses[0][i];
    else // SAM corrects PCI: s represents marginal or short PCI result
        if (PCIclasses[0][i] == 's') classes[i] = PCIclasses[0][i];
        else // substitute SAM classification
```
if (scores[0][i] < scores[1][i]) classes[i] = 'a';
else classes[i] = 'b';
}
}

if (radioButton2.getChecked())    // marginal percentage, rather than threshold
{
    mpct = (int)numseqs*(new Integer(edit7.getText())).intValue/100;
    mins = new int[mpct];
    Sort(margins, mins, 0, mpct);
    mclass1 = mclass2 = mclass3 = 0;
    for (int i=0; i<mpct; i++)
    {
        if (classes[mins[i]] == 'a') mclass1++;
        else if (classes[mins[i]] == 'b') mclass2++;
        else if (classes[mins[i]] == 'c')
        {  mclass3++;
        }
    }
}

if (PCI)
{
    for (int i=0; i<numseqs; i++)
    {
        if (classes[i] == 'a') pclass1++;
        else if (classes[i] == 'b') pclass2++;
        else if (classes[i] == 'c') pclass3++;
        else AddText("Classes array has bad element at " + i + ": " + classes[i]);
    }
}

AddText("HMM classification results: ");
AddText(class1 + " (" + mclass1 + " marginal) in Class #1");
AddText(class2 + " (" + mclass2 + " marginal) in Class #2");
if (threeway) AddText(class3 + " (" + mclass3 + " marginal) in Class #3");
if (PCI)    // PCI classifications
{
    AddText("Replacing marginal results with PCI:
    ");
    AddText(pclass1 + " in Class #1");
    AddText(pclass2 + " in Class #2");
    if (threeway) AddText(pclass3 + " in Class #3");
}
AddText("U Can't Touch This!
");
catch (Exception exc) { AddText("button1_click: " + exc.toString()); }
finally
{
    try
    {
        Reader1.close();
        Reader2.close();
        if (threeway) Reader3.close();
        if (record) RecordFile.close();
    }
    catch (IOException exc) { AddText("closing files: " + exc.toString()); }
}

• The following function handled scoring for 1-dimensional structure prediction
  including several experimental modifications (Sections 3.2 and 3.3):

public void AtomEgoyan(String sl, double z[], int tl, int rr, int qq, String b)
{
    FileReader ActualReader = null;    // read in the actual coordinates from a file
    FileWriter OutputFile = null;
    String suffix = "-";
    boolean diff_coords = checkBox3.getChecked();
    boolean in_eq_out = checkBox11.getChecked();
    boolean relative_MSE = checkBox14.getChecked();
    boolean sarah = checkBox15.getChecked();
    int n = 0, ne = 0, n1 = 0, n1r = 0, n2 = 0, n2r = 0, n3 = 0, n3r = 0;
    int state_nlr = 0, staten2r = 0,
    state n3r = 0;
    double rmse1 = 0, rmse2 = 0, rmse3 = 0;
if (relative_MSE) {
    rmsel = Double.valueOf(edit9.getText()).doubleValue();
    rmse2 = Double.valueOf(edit10.getText()).doubleValue();
    rmse3 = Double.valueOf(edit11.getText()).doubleValue();
}

if (diff_coords) suffix = "-d";
// warning that the second poly is coming next time in paralin4.dat, or the beta
// strands in paralin-ssb.dat: this function is called in between building 2 models
if (second_poly || (sl.equals("paralin-ssa.dat") && !two_state_sec_str)) {
    z_backup = new double[tl+1];
    for (int i=0; i<=tl; i++)
        z_backup[i] = z[i];
    if (second_poly)
        checkBox6.setChecked(false);
    gobutton("paralin4.dat", "parapoly4.dat");
}
return;
else if (sl.equals("paralin-ssb.dat") && alpha_beta_other) // 2nd of 3 visits
    z_backup2 = new double[tl+1];
    for (int i=0; i<=tl; i++)
        z_backup2[i] = z[i];
    return;
}

try {
    // assign filenames to the actual data file and the output file
    if (sl.equals("paralin.dat")) // x coordinate
        ActualReader = new FileReader(b.substring(0, 4) + suffix + "x");
    OutputFile = new FileWriter("output.dat");
    } else if (sl.equals("paralin1.dat")) // y coordinate
            ActualReader = new FileReader(b.substring(0, 4) + suffix + "y");
    OutputFile = new FileWriter("output1.dat");
    } else if (sl.equals("paralin2.dat")) // z coordinate
            ActualReader = new FileReader(b.substring(0, 4) + suffix + "z");
    OutputFile = new FileWriter("output2.dat");
    } else if (sl.equals("paralin3.dat")) // r coordinate
            if (secondary_structure)
            ActualReader = new FileReader("Protein\" + b.substring(0, 4) + suffix + "ss");
            else if (in_eq_out) ActualReader = new FileReader("Protein\" + b);
            else if (classification) ActualReader = null;
            else ActualReader = new FileReader("Protein\" + b.substring(0, 4) + suffix + "r");
            if (!two_two) OutputFile = new FileWriter("output3.dat");
            else OutputFile = new FileWriter("Protein\output3.dat");
    } else if (sl.equals("paralin4.dat"))
        ActualReader = new FileReader(b.substring(0, 4) + suffix + "r");
    OutputFile = new FileWriter("output4.dat");
    } else if (sl.equals("paralin5.dat"))
        ActualReader = new FileReader("Protein\" + b); // will be output3.dat
    OutputFile = new FileWriter("output5.dat");
    } // alpha vs. non-alpha
else if (sl.equals("paralin-ssa.dat") && two_state_sec_str)
    ActualReader = new FileReader("Protein\" + b.substring(0,4) + suffix + "ssa");
    OutputFile = new FileWriter("output3.dat");

    } // beta vs. non-beta
else if (sl.equals("paralin-ssb.dat") && two_state_sec_str)
    ActualReader = new FileReader("Protein\" + b.substring(0,4) + suffix + "ssb");
    OutputFile = new FileWriter("output3.dat");

    } // loop vs. non-loop
else if (sl.equals("paralin-ssc.dat") && two_state_sec_str)
    ActualReader = new FileReader("Protein\" + b.substring(0,4) + suffix + "ssc");
OutputFile = new FileWriter("output3.dat");
if (sl.equals("paralin-sab.dat"))
  (ActualReader = new FileReader("Protein\" + b.substring(0,4) + suffix + "ss");
OutputFile = new FileWriter("output3.dat");
} // secondary structure 2-binary (for 3-way)
else if (sl.equals("paralin-ssb.dat"))
  ActualReader = new FileReader("Protein\" + b.substring(0,4) + suffix + "ss");
OutputFile = new FileWriter("output3.dat");
} // secondary structure 3-binary
else if (sl.equals("paralin-ssc.dat"))
  ActualReader = new FileReader("Protein\" + b.substring(0,4) + suffix + "ss");
OutputFile = new FileWriter("output3.dat");

double z_act[] = new double[tl+l], rms = 0, z_act_avg = 0, z_avg = 0;
rmsd = 0; // root mean square deviation, the error measurement
rmse = 0; // for 2-state secondary structure
if (classification) ReadInt(ActualReader); // discard count at top of file
for (int k=0; k<=tl; k++)
  if (in_eq_out)
    z_act[k] = ReadDouble(ActualReader); // input = output
else if (classification) z_act[k] = 1;
else
  ReadDouble(ActualReader); // discard the hydrophobicity
  z_act[k] = ReadDouble(ActualReader);
}
if (diff_coords) // differential coordinates
  if (k == rr-1) z[k] = z_act[k];
else if (k >= rr && k <= tl-qq)
  { z[k] += z[k-1];
    z_act[k] += z_act[k-1];
    z_avg += z[k];
    z_act_avg += z_act[k];
  }
}
if (limit_jumps) // smoothing: should use only with non-differential coords
  if (k > rr && k < tl-qq) // first and last coordinates can jump from 0
    { if (z[k] - z[k-1] > 4.0) z[k] = z[k-1] + 4.0;
      else if (z[k] - z[k-1] < -4.0) z[k] = z[k-1] - 4.0;
    }
}
if (sl.equals("paralin4.dat")) // second poly
  if (Math.abs(z[k] - z_backup[k]) < 5) z[k] = z_backup[k];
if (secondary_structure)
  if (Math.abs(z[k] - z_backup[k]) < 5) z[k] = z_backup[k];
}
if (classification) ReadInt(ActualReader); // discard count at top of file
for (int k=0; k<=tl; k++)
  if (in_eq_out)
    z_act[k] = ReadDouble(ActualReader); // input = output
else if (classification) z_act[k] = 1;
else
  ReadDouble(ActualReader); // discard the hydrophobicity
  z_act[k] = ReadDouble(ActualReader);
}
if (diff_coords) // differential coordinates
  if (k == rr-1) z[k] = z_act[k];
else if (k >= rr && k <= tl-qq)
  { z[k] += z[k-1];
    z_act[k] += z_act[k-1];
    z_avg += z[k];
    z_act_avg += z_act[k];
  }
}
if (limit_jumps) // smoothing: should use only with non-differential coords
  if (k > rr && k < tl-qq) // first and last coordinates can jump from 0
    { if (z[k] - z[k-1] > 4.0) z[k] = z[k-1] + 4.0;
      else if (z[k] - z[k-1] < -4.0) z[k] = z[k-1] - 4.0;
    }
}
if (classification) ReadInt(ActualReader); // discard count at top of file
for (int k=0; k<=tl; k++)
  if (in_eq_out)
    z_act[k] = ReadDouble(ActualReader); // input = output
else if (classification) z_act[k] = 1;
else
  ReadDouble(ActualReader); // discard the hydrophobicity
  z_act[k] = ReadDouble(ActualReader);
}
if (diff_coords) // differential coordinates
  if (k == rr-1) z[k] = z_act[k];
else if (k >= rr && k <= tl-qq)
  { z[k] += z[k-1];
    z_act[k] += z_act[k-1];
    z_avg += z[k];
    z_act_avg += z_act[k];
  }
}
if (limit_jumps) // smoothing: should use only with non-differential coords
  if (k > rr && k < tl-qq) // first and last coordinates can jump from 0
    { if (z[k] - z[k-1] > 4.0) z[k] = z[k-1] + 4.0;
      else if (z[k] - z[k-1] < -4.0) z[k] = z[k-1] - 4.0;
    }
}
if (classification) ReadInt(ActualReader); // discard count at top of file
for (int k=0; k<=tl; k++)
  if (in_eq_out)
    z_act[k] = ReadDouble(ActualReader); // input = output
else if (classification) z_act[k] = 1;
else
  ReadDouble(ActualReader); // discard the hydrophobicity
  z_act[k] = ReadDouble(ActualReader);
}
if (diff_coords) // differential coordinates
  if (k == rr-1) z[k] = z_act[k];
else if (k >= rr && k <= tl-qq)
  { z[k] += z[k-1];
    z_act[k] += z_act[k-1];
    z_avg += z[k];
    z_act_avg += z_act[k];
  }
}
if (limit_jumps) // smoothing: should use only with non-differential coords
  if (k > rr && k < tl-qq) // first and last coordinates can jump from 0
    { if (z[k] - z[k-1] > 4.0) z[k] = z[k-1] + 4.0;
      else if (z[k] - z[k-1] < -4.0) z[k] = z[k-1] - 4.0;
    }
}
if (classification) ReadInt(ActualReader); // discard count at top of file
for (int k=0; k<=tl; k++)
  if (in_eq_out)
    z_act[k] = ReadDouble(ActualReader); // input = output
else if (classification) z_act[k] = 1;
else
  ReadDouble(ActualReader); // discard the hydrophobicity
  z_act[k] = ReadDouble(ActualReader);
}
if (diff_coords) // differential coordinates
  if (k == rr-1) z[k] = z_act[k];
else if (k >= rr && k <= tl-qq)
  { z[k] += z[k-1];
    z_act[k] += z_act[k-1];
    z_avg += z[k];
    z_act_avg += z_act[k];
  }
}
if (limit_jumps) // smoothing: should use only with non-differential coords
  if (k > rr && k < tl-qq) // first and last coordinates can jump from 0
    { if (z[k] - z[k-1] > 4.0) z[k] = z[k-1] + 4.0;
      else if (z[k] - z[k-1] < -4.0) z[k] = z[k-1] - 4.0;
    }
}
if (z[k] < 1) rmse += (1 - z[k])*(1 - z[k]);
} n++;
}

else if (sl.equals("paralin-ssb.dat")) // separate models for alpha & beta
{ if (z_act[k] == -1) // alpha
  { if (z_backup[k] + z[k] < -cutoff) n1r++;
    if (z_backup[k] < 0 && z[k] < 0) state_n1r++;
    n1++;
  }
  else if (z_act[k] == 1) // beta
  { if (z[k] + z_backup[k] > cutoff) n2r++;
    if (z_backup[k] > 0 && z[k] > 0) state_n2r++;
    n2++;
  }
  else // other (loop)
  { if ((z_backup[k] + z[k] <= cutoff) &&
      (z_backup[k] + z[k] >= -cutoff)) n3r++;
    if ((z_backup[k] >= 0 && z[k] <= 0) ||
        (z_backup[k] <= 0 && z[k] >= 0)) state_n3r++; n3++;
  }
}

// alpha-beta-other: pick largest of three scores
else if (sl.equals("paralin-ssc.dat"))
{ z_backup[k] /= (1-rmse1);
  z_backup2[k] /= (1-rmse2);
  z[k] /= (1-rmse3);
  if (z_act[k] == -1) // alpha
    { if (z_backup[k] < -z_backup2[k] && z_backup[k] < z[k]) n1r++;
       n1++;
    }
  else if (z_act[k] == 1) // beta
    { if (-z_backup[k] < z_backup2[k] && z_backup2[k] > -z[k]) n2r++;
       n2++;
    }
  else if (z_act[k] == 0) // other
    { if (z[k] < z_backup[k] && z[k] < -z_backup2[k]) n3r++;
       n3++;
    }
}

else // -ss files, which classify 3-way with one model
{ if (z_act[k] == -1)
  { n1++;
    if (z[k] < -cutoff)
      { n1r++;
        nr++;
      }
  }
  else if (z_act[k] == 1)
  { n2++;
    if (z[k] > cutoff)
      { n2r++;
        nr++;
      }
  }
  else if (z_act[k] == 0)
  { n3++;
    if (z[k] <= cutoff && z[k] >= -cutoff)
      { n3r++;
        nr++;
      }
  }
}
}

49
if (smooth)
{
    int window = Integer.valueOf(edit7.getText()).intValue();
    double local_avg;
    for (int m=0; m<t1; m+=window)
    {
        local_avg = 0;
        for (int nn=0; nn<window && m+nn<=t1; nn++)
            local_avg += z_act[m+nn];
        local_avg /= window;
        for (int nn=0; nn<window && m+nn<=t1; nn++)
            z_act[m+nn] = local_avg;
    }
}

if (checkBox3.getChecked()) // differential coordinates
{
    z_avg /= (t1-rr-qq+1);
    z_act_avg /= (t1-rr-qq+1);
}

// calculate rmsd
for (int k=0; k<=t1; k++)
{
    if (k >= rr && k <= t1-qq)
    {
        if (diff_coords) z[k] += z_act[k] - z_avg; // see lab book 7.22.99
        // non-differential r coordinates cannot be negative
        if (!diff_coords && !secondary_structure && sl.equals("paralin3.dat")
            || sl.equals("paralin4.dat") && z[k] < 0)
            z[k] = 0;
        rmsd += (z[k] - z_act[k]) * (z[k] - z_act[k]);
        rms += z_act[k] * z_act[k];
    }
    // store the model output and actual output for comparison in e.g. MATLAB
    if (k >= rr && k <= t1-qq)
    {
        if (alpha_beta_other)
            OutputFile.write(z_backup[k] + " " + z_backup2[k] + " " + z[k] + " " +
                             z_act[k] + "\n");
        else if (!sl.equals("paralin-ssb.dat") || two_state_sec_str)
            OutputFile.write(z[k] + " " + z_act[k] + "\n");
        else
            OutputFile.write((z_backup[k] + z[k]) + " " + z_act[k] + "\n");
    }
}

if (!classification) ActualReader.close();
OutputFile.close();

if (two_two) // beware the Hooded Fang!
{
    FileWriter ListFile = new FileWriter("Protein\" + edit1.getText());
    ListFile.write("output3.dat\n");
    ListFile.close();
    checkBox7.setchecked(false);
    gobutton("paralin5.dat", "parapoly5.dat");
    return;
}

rmsd /= (t1 - rr - qq + 1); // normalization
rmsd = Math.sqrt(rmsd);
rms /= (t1 - rr - qq + 1);
rms = Math.sqrt(rms);
rmse /= t1; // used with 2-state secondary structure
rmse = Math.sqrt(rmse);

AddText(b);
if (secondary_structure)
{
    if (two_state_sec_str)
    {
        pct = 100*(double)nr/n;
        AddText("Accuracy of 2-state secondary structure prediction = " +
                Round(pct, 1) + ";
        AddText("RMSE is " + Round(rmse, 2));
    }
    else
    {
        pct = 100*(double)(n1r + n2r + n3r) / (n1 + n2 + n3);
    }
}
state_pct = 100*(double)(state_n1r + state_n2r + state_n3r) /
(n1 + n2 + n3);
AddText("Accuracy of 3-state secondary structure prediction = " + Round(pct, 1) + "+%");
AddText("Alpha helices: " + Round(100*(double)nlr/nl, 1) + "% " +
"Beta strands: " + Round(100*(double)n2r/n2, 1) + "% " +
"Others: " + Round(100*(double)n3r/n3, 1) + "%");
AddText("Accuracy by state scoring is " + Round(state_pct, 1) + "%";
}
else
AddText("Root mean square deviation is " + rmsd);
AddText("The root mean square coordinate is " + rms);
catch (IOException exc) { AddText("AtomEgoyan: " + exc.toString()); }

The following function implemented the nearest neighbor methods using the jackknife test for identification of ATP-binding P-loops (Section 4.2):

```java
private void button13_click(Object source, Event e)
{| String[] seqs = new String[3000], seqnames = new String[3000];
  int len, class1r = 0, class2r = 0, closest = 0, wclosest = 0, vote1, vote2;
  int streak, numtrue, numfalse, best_numtrue, best_numfalse, max_MSE = 0;
  int class1mw = 0, class2mw = 0;
  int nmr = Integer.valueOf(edit86.getText().intValue();
  int[] closests = new int[numnns];
  int[][] freq1 = new int[100][21], freq2 = new int[100][21]; // for Bayesian
  double[] min_dists = new double[num_nns];
  double[][] hydr = new double[3000][100], curr_diffs = new double[3000][100];
  double[][] diffsl = null, diffs2 = null; // for arms
  double distl, dist2, d = 0, min_dist, dist = 0, thresh = 0, mean1 = 0;
  double mean2 = 0, std1 = 0, std2 = 0, med = 0, wdist = 0, wmin_dist;
  double old_hydr = 0;
  double[] ATPscore = new double[3000], pairwise_dists = new double[100];
  double[] random_hydr = new double[21];
  boolean[] ATPrank = new boolean[3000]; // for ATPonly
  boolean avg_dist = checkBox10.getChecked(), short_dist = checkBox11.getChecked();
  boolean combined = checkBox15.getChecked();
  boolean L1 = checkBox12.getChecked(), L2 = checkBox13.getChecked();
  boolean correlation = checkBox18.getChecked();
  boolean Wiener = checkBox20.getChecked(), identity = checkBox22.getChecked();
  boolean Bayesian = checkBox27.getChecked();
  boolean matrix = checkBox29.getChecked() || checkBox30.getChecked();
  boolean threshold = checkBox14.getChecked(), ATPonly = checkBox17.getChecked();
  boolean median_shift = checkBox19.getChecked(), Wiener_L1 = checkBox21.getChecked();
  boolean extend_memory = checkBox23.getChecked(), arms = checkBox24.getChecked();
  boolean anticipation = checkBox25.getChecked(), L1_Wiener = checkBox26.getChecked();
  boolean mark_non_homs = checkBox38.getChecked();
  boolean display_wrong = checkBox39.getChecked();
  boolean multiple_nns = (num_nns > 1);
  boolean marginal = false, current, use_sarah = false, longer_mem = false;
  Date start_time = new Date();
  if (threshold) thresh = Double.valueOf(edit37.getText()).doubleValue();
  if (!identity || matrix) use_sarah = comboBox2.getSelectedItem().toString().equals("Sarah"); // avoid error
  if (extend_memory) max_MSE = Integer.valueOf(edit38.getText().intValue());
  count1 = Count2 = 0;
  try
  { FileReader Class1 = new FileReader(edit35.getText());
    FileReader Class2 = new FileReader(edit36.getText());
    // get sequences from first class
    eof = false;
```
while (true)
    ReadString(Class1);
    if (eof) break;
    seqnames[count1] = ReadString(Class1);
    ReadString(Class1);
    seqs[count1++] = ReadString(Class1);
}

// get sequences from second class
EOF = false;
while (true)
    ReadString(Class2);
    if (EOF) break;
    seqnames[count1+count2] = ReadString(Class2);
    ReadString(Class2);
    seqs[count1+(count2++)] = ReadString(Class2);
}

len = seqs[0].length() - 2;  // omit '//' at end
if (use_sarah) len *= 5;
if (Wiener || Wiener_Ll || Ll_Wiener)
    FastOrtAlg.edit6.setText(Integer.toString(len));
non_homologues = new boolean[count1+count2];

if (!identity && !matrix)
    if (comboBox2.getSelectedItem().equals("Random"))
        for (int i=0; i<21; i++)
            random_hyd[i] = Math.random();

    // translate to hydrophobicities
    for (int i=0; i<count1+count2; i++)
        for (int j=0; j<len; j++)
            if (use_sarah)
                hyd[i][j] = Sarah(seqs[i].charAt(j/5), j%5);
            else if (comboBox2.getSelectedItem().equals("Solvent exposed area"))
                hyd[i][j] = SEA(seqs[i].charAt(j));
            else if (comboBox2.getSelectedItem().equals("Turn preference"))
                hyd[i][j] = Turn(seqs[i].charAt(j));
            else if (comboBox2.getSelectedItem().equals("Random"))
                hyd[i][j] = random_hyd[AAtoNumber(seqs[i].charAt(j))];
            else
                hyd[i][j] = Rose(seqs[i].charAt(j));

    // y(S, x|S, x+a|S ⇒ y+a|S
    if (arms)
        diffs1 = new double[count1][count1][len];
        diffa2 = new double[count2][count2][len];
        for (int i=0; i<count1; i++)
            for (int j=0; j<count1; j++)
                for (int k=0; k<len; k++)
                    diffs1[i][j][k] = hyd[i][k] - hyd[j][k];
        for (int i=0; i<count2; i++)
            for (int j=0; j<count2; j++)
                for (int k=0; k<len; k++)
                    diffa2[i][j][k] = hyd[count1+i][k] - hyd[count1+j][k];

    // calculate table of frequencies at each position: see lab book 1.5.00
    if (Bayesian && !matrix)
        for (int i=0; i<count1+count2; i++)
            for (int j=0; j<len; j++)
                if (i < count1) freqs1[j][k]++;
                else freqs2[j][k]++;
// for each sequence to be tested
for (int i=0; i<count1+count2; i++)
{
    dist1 = dist2 = 0;
    wmin_dist = min_dist = Double.MAX_VALUE;
    closest = -1;
    if (multiple_nns)
    {
        for (int j=0; j<num_nns; j++)
        {
            min_dist[i][j] = Double.MAX_VALUE;
            closests[j] = -1;
        }
    }
    if (Wiener & longer_mem)
    {
        FastOrtAlg.edit5.setText("" +
            (Integer.valueOf(FastOrtAlg.edit5.getText()).intValue() + 1));
    }
    if (correlation)
    {
        mean1 = std1 = 0;
        for (int k=0; k<len; k++)
        {
            mean1 += hydr[i][k];
            std1 += hydr[i][k]*hydr[i][k];
        }
        mean1 /= len;
        std1 = std1/len - mean1*mean1;
    }
    if (Bayesian & !matrix)  // see lab book 4.20.00
    {
        double prob = 1;
        for (int j=0; j<len; j++)
        {
            int aa = AAtoNumber(seqs[i].charAt(j));
            if (aa == 20) continue;  // 'x': unknown residue
            double f1 = freqs1[j][aa];
            double f2 = freqs2[j][aa];
            if (i < count1) f1--;
            else f2--;
            prob *= f1/f2;
        }
        prob = Math.pow(prob, 1.0/(len-1));
        double cutoff;
        if (i < count1) cutoff = (double)(count1-1)/count2;
        else cutoff = (double)count1/(count2-1);
        if (prob >= cutoff & i < count1) class1r++;
        else if (prob < cutoff & i >= count1) class2r++;
        else if (display_wrong) AddText2(seqnames[i]);
    }
    if (ATPonly) ATPscore[i] = -prob;
    if (prob >= cutoff)
    {
        scores[i] = 1 - prob;  // smaller scores[i] => higher confidence
        nearest_neighbours[i] = 0;  // dummy for first class
    }
    else
    {
        scores[i] = prob;
        nearest_neighbours[i] = count1+1;
    }
    continue;
}
// if (Bayesian)
if (Bayesian & matrix)  // only coding BLOSUM for now: unfinished
{
    double p = 0;
    for (int j=0; j<len; j++)
    {
        double p1 = 0, p2 = 0;
        for (int k=0; k<count1; k++)
        {  
            p1 += Math.pow(2, BLOSUM[AAtoNumber(seqs[i].charAt(j))
                [AAtoNumber(seqs[k].charAt(j))]] / 2);
            for (int k=count1; k<count1+count2; k++)
            {  
                p2 += Math.pow(2, BLOSUM[AAtoNumber(seqs[i].charAt(j))
                    [AAtoNumber(seqs[k].charAt(j))]] / 2);
            }
        }
        p = (p1+1)/count1 / ((p1+1)/count1 + (p2+1)/count2);
    }
    p /= len;
if (p>0.5 && i<count1) class1r++;
else if (p<0.5 && i>=count1) class2r++;
else if (display_wrong) AddText2(seqnames[i]);
continue;
}

// compare it with each other sequence
for (int j=0; j<count1+count2; j++)
{ if (j == i) continue; // but not itself
dist = 0;

if (correlation)
{ mean2 = std2 = 0;
  for (int k=0; k<len; k++)
  { mean2 += hydr[j][k];
    std2 += hydr[j][k]*hydr[j][k];
  }
  mean2 /= len;
  std2 = std2/len - mean2*mean2;
}

// shift one sequence to optimize the L1 distance: use median of pairwise
// distances
if (median_shift)
{ for (int k=0; k<len; k++)
  pairwise_dists[k] = hydr[i][k] - hydr[j][k];
  med = Median(pairwise_dists, len);
}

// loop through the length of the sequences
for (int k=0; k<len; k++)
{ if (L1 || Wiener_L1 || L1_Wiener)
  { if (median_shift) d = Math.abs(hydr[i][k] - hydr[j][k] - med);
    else d = Math.abs(hydr[i][k] - hydr[j][k]);
  }
  else if (L2)
  d = (hydr[i][k] - hydr[j][k])*(hydr[i][k] - hydr[j][k]);
  else if (correlation) // higher correlation = less distance
  d = -(hydr[i][k] - mean1)*(hydr[j][k] - mean2)/(std1*std2);

if (avg_dist || combined)
{ if (j < count1) dist1 += d;
  else dist2 += d;
  }

if (short_dist || combined)
  dist += d;
if (arms)
curr_diffs[j][k] = hydr[i][k] - hydr[j][k];
} // for(k)

// shift output sequence to simulate anticipation for FOA
if (anticipation)
{ // len to make undoing easier: this code is for state input
  for (int k=len+4; k>4; k--)
    hydr[j][k] = hydr[j][k-5];
}

// Wiener_L1 = Wiener with L1 on threshold, only implemented for
// shortest distance, L1_Wiener = opposite
if (Wiener || Wiener_L1 || L1_Wiener)
{ if (Wiener_L1 || L1_Wiener)
  wdist = FastOrtAlg.FastOrt(hydr[i], hydr[j], len);
  else
    dist = FastOrtAlg.FastOrt(hydr[i], hydr[j], len);
  if (Double.isNaN(dist)) AddText2(i + " + j + " dist = NaN!");

if (avg_dist)
{ if (j < count1) dist1 += dist;
  else dist2 += dist;
}
if (anticipation) // shift back
{ for (int k=0; k<len; k++)
    hydr[j][k] = hydr[j][k+5]; // this code is for state input
}

if (identity)
{ dist = IdentityScore(seqs[i], seqs[j], len); // takes state, not hydrophobicity
  if (avg_dist)
  { if (j < counti) dist1 += dist;
    else dist2 += dist;
  }
}

if (matrix)
{ dist = MatrixScore(seqs[i], seqs[j], len); // takes state, not hydrophobicity
  if (avg_dist)
  { if (j < counti) dist1 += dist;
    else dist2 += dist;
  }
}

if ((short_dist || combined) && !multiple_nns)
{ if (dist < min_dist)
  { min_dist = dist;
    closest = j;
  }
  if ((Wiener_L1 || L1*Wiener) && wdist < wmin_dist)
  { wmin_dist = wdist;
    wclosest = j;
  }
}

if (multiple_nns)
{ for (int k=0; k<numnns; k++)
  { if (dist < min_dists[k]) // min_dists[0] is the smallest
    { for (int L=num_nns-1; L>k; L++)
      { if (L > k) break;
        min_dists[L] = min_dists[L-1];
        closests[L] = closests[L-1];
      }
      min_dists[k] = dist;
      closests[k] = j;
      break;
    }
  }
}

// go to longer memory when over threshold
if (Wiener && extend_memory && longer_mem)
{ FastOrtAlg.edit5.setText("" +
    (Integer.valueOf(FastOrtAlg.edit5.getText()).intValue() - 1));
  longer_mem = false;
}
else if (Wiener && extend_memory && short_dist && min_dist > max_MSE)
{ longer_mem = true;
  i--;
  continue;
}

// use arms when over threshold
if (arms && threshold && short_dist && min_dist > thresh)
{ double min_diff1 = Double.MAX_VALUE, min_diff2 = Double.MAX_VALUE;
  for (int j1=0; j1<count1; j1++)
    for (int j2=0; j2<count1; j2++)
      { if (j1 == i || j2 == i) continue;
        for (int j=0; j<count1; j++)
          if (hydr[i][j] != hydr[j][j])
            min_diff1 = Math.min(min_diff1, dist(i, j1) + dist(j1, j2))
                - (dist(i, j) + dist(j, i) + dist(j, j2));
  }
  for (int j1=0; j1<count1; j1++)
    for (int j2=0; j2<count1; j2++)
      { if (j1 == i || j2 == i) continue;
        for (int j=0; j<count1; j++)
          if (hydr[i][j] != hydr[j][j])
            min_diff2 = Math.min(min_diff2, dist(i, j1) + dist(j1, j2))
                - (dist(i, j) + dist(j, i) + dist(j, j2));
  }
  if (min_diff1 < min_diff2)
    min_diff = min_diff1;
  else
    min_diff = min_diff2;
}
else if (arms && threshold && short_dist)
{ double min_diff1 = Double.MAX_VALUE, min_diff2 = Double.MAX_VALUE;
  for (int j1=0; j1<count1; j1++)
    for (int j2=0; j2<count1; j2++)
      { if (j1 == i || j2 == i) continue;
        for (int j=0; j<count1; j++)
          if (hydr[i][j] != hydr[j][j])
            min_diff1 = Math.min(min_diff1, dist(i, j1) + dist(j1, j2))
                - (dist(i, j) + dist(j, i) + dist(j, j2));
  }
  for (int j1=0; j1<count1; j1++)
    for (int j2=0; j2<count1; j2++)
      { if (j1 == i || j2 == i) continue;
        for (int j=0; j<count1; j++)
          if (hydr[i][j] != hydr[j][j])
            min_diff2 = Math.min(min_diff2, dist(i, j1) + dist(j1, j2))
                - (dist(i, j) + dist(j, i) + dist(j, j2));
  }
  if (min_diff1 < min_diff2)
    min_diff = min_diff1;
  else
    min_diff = min_diff2;
}
else if (arms && threshold)
{ double min_diff = Double.MAX_VALUE;
  for (int j1=0; j1<count1; j1++)
    for (int j2=0; j2<count1; j2++)
      { if (j1 == i || j2 == i) continue;
        for (int j=0; j<count1; j++)
          if (hydr[i][j] != hydr[j][j])
            min_diff = Math.min(min_diff, dist(i, j1) + dist(j1, j2))
                - (dist(i, j) + dist(j, i) + dist(j, j2));
  }
  if (min_diff < min_diff)
    min_diff = min_diff;
}
else
{ double min_diff = Double.MAX_VALUE;
  for (int j1=0; j1<count1; j1++)
    for (int j2=0; j2<count1; j2++)
      { if (j1 == i || j2 == i) continue;
        for (int j=0; j<count1; j++)
          if (hydr[i][j] != hydr[j][j])
            min_diff = Math.min(min_diff, dist(i, j1) + dist(j1, j2))
                - (dist(i, j) + dist(j, i) + dist(j, j2));
  }
  if (min_diff < min_diff)
    min_diff = min_diff;
}
```java
if (j == i) continue;
d = 0;
for (int k=0; k<len; k++)
    d += Math.abs(diffsl[j1][j2][k] - curr_diffs[j][k]);
if (d < min_diff1)
    min_diff1 = d;
} } for (int j1=0; j1<count2; j1++)
for (int j2=0; j2<count2; j2++)
{ if (j1+count1 == i || j2+count1 == i) continue;
    for (int j=0; j<count2; j++)
    { if (j == i) continue;
        d = 0;
        for (int k=0; k<len; k++)
            d += Math.abs(diffs2[j1][j2][k] - curr_diffs[count+j][k]);
        if (d < min_diff2)
            min_diff2 = d;
    }
}
if (min_diff1 < min_diff2) closest = 0; // classify as binding
// classify as non-binding: if equal use NN prediction
else if (min_diff2 < min_diff1) closest = count1+i;
}
// apply threshold: non-marginal case
if (Wiener_Ll && wmin_dist < thresh) closest = wclosest;
// go to Wiener for marginal Ll
if (Ll_Wiener && min_dist > thresh) closest = wclosest;

// calculate score based on average distance
if ((avg_dist && !ATPonly) || (combined && min_dist > thresh))
{ if (i < count1)
    { dist1 /= count1 - 1;
        dist2 /= count2;
    }
else
    { dist1 /= count1;
        dist2 /= count2 - 1;
    }
if (i<count1 && dist1<=dist2) classlr++;
else if (i>count1 && dist2<dist1) class2r++;
else if (display_wrong) AddText2(seqnames[i]);
if (threshold)
{ if (Math.abs(dist1-dist2) < thresh)
    { if (i<count1 && dist1<=dist2) class1mr++;
        else if (i<count1 && dist2<dist1) class1mw++;
        else if (i>count1 && dist2<dist1) class2mr++;
            else if (i>count1 && dist2>dist1) class2mw++;
    }
}
}
```

// calculate score based on nearest neighbour
else if ((short_dist || combined) && !multiple_nns)
{ if (closest == -1) AddText2("Error: closest is -1 for i = " + i);
    else if (i<count1 && closest<count1) class1r++;
    else if (i>count1 && closest<count1) class1++;
    else if (i>count1 && closest>count1) class2r++;
    else if (display_wrong) AddText2(seqnames[i]);
    if (threshold && !Wiener_Ll && !Ll_Wiener)
{ if (min_dist > thresh)
    { if (i<count1 && closest<count1) class1mr++;
        else if (i<count1 && closest>count1) class1mw++;
        else if (i>count1 && closest<count1) class2mr++;
                else if (i>count1 && closest>count1) class2mw++;
    // mark non-homologues for reference of other functions
        non_homologues[i] = true;
    }
    }
    // calculate distance based on several nearest neighbours
```
else if (multiple_nns)
{ vote1 = vote2 = 0;
  for (int j=0; j<num_nns; j++)
  { if (closests[j] < count1) vote1++;
    else vote2++;
  }
  if (vote1 > vote2 && i < count1) class1r++;
  else if (vote2 > vote1 && i >= count1) class2r++;
}
else if (ATPonly) // grade each sequence against only ATP, as Hirst did
{ ATPscore[i] = dist1;
  scores[i] = min_dist; // nearest neighbour scores in global vector for other fcns
  nearestneighbours[i] = closest;
} // for(i)

// determine the optimal dividing line for the one-model scoring
if (ATPonly)
{ Sort(ATPscore, ATPrank, count1+count2, count1);
  current = ATPrank[0];
  streak = 1;
  best_numfalse = Integer.MAX_VALUE;
  best_numtrue = 0;
  if (ATPrank[0])
  { numtrue = 1;
    numfalse = 0;
  }
  else
  { numtrue = 0;
    numfalse = 1;
  }
  for (int j=1; j<count1+count2; j++)
  { if (ATPrank[j] != current) // at the end of a streak
    { // mistakes at a cutoff = count1 - numtrue + numfalse
      if (numfalse-numtrue < best_numfalse-best_numtrue)
      { best_numtrue = numtrue;
        best_numfalse = numfalse;
      }
      current = ATPrank[j];
      streak = 0;
    }
    if (ATPrank[j]) numtrue++; // true means it is binding
    else numfalse++;
    streak++;
  }
  AddText2("Best cutoff has " + (count1-best_numtrue) + " out of " +
  count1 + " ATP wrong and " + best_numfalse + " out of " +
  count2 + " non-ATP wrong.\n");
} // if (ATPonly)
Class1.close();
Class2.close();
if (threshold)
{ AddText2("Class #1 total: " + count1 + "; right " + class1r + " (" +
  class1mr + " marginal), wrong " + (count1-class1r) + " (" +
  class1mw + " marginal)\n");
  AddText2("Class #2 total: " + count2 + "; right " + class2r + " (" +
  class2mr + " marginal), wrong " + (count2-class2r) + " (" +
  class2mw + " marginal)\n");
}
else if (!ATPonly)
{ AddText2(class1r + " out of " + count1 +
  in Class #1 were classified correctly.\n");
  AddText2(class2r + " out of " + count2 +
  in Class #2 were classified correctly.\n");
}

Date end_time = new Date();
double time = (end_time.getTime() - start_time.getTime()) / 1000.0;
AddText2("Run time = " + Round(time, 1) + " seconds.\n");
checkBox43.setEnabled(true);
}

catch (IOException exc) { AddText2("Jack-knife: " + exc.toString()); }
}
References


4. A. Bairoch. Personal communication.


22. IBM. “Blue Gene to tackle protein folding grand challenge.”


40. B. Rost, S. O'Donoghue. "Sisyphus and protein structure prediction." 
*Bioinformatics* **13**: 345-56 (1997); 


http://www.embl-heidelberg.de/~rost/Papers/98revSecStr.html.


