

UCSF

UC San Francisco Electronic Theses and Dissertations

Title

A segment based approach to protein secondary structure prediction

Permalink

<https://escholarship.org/uc/item/8pd9z1kc>

Author

Cohen, Bruce I

Publication Date

1991

Peer reviewed|Thesis/dissertation

A Segment Based Approach to Protein Secondary Structure Prediction

by

Bruce Ira Cohen

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Medical Information Science

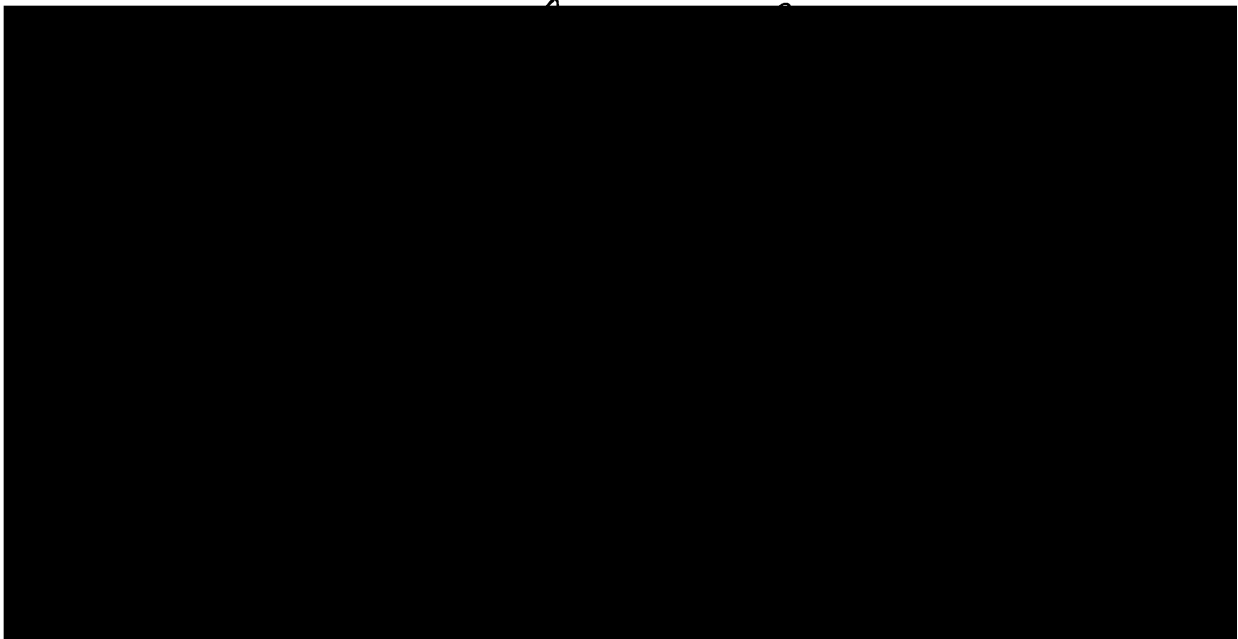
in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA

San Francisco



To Gale

Acknowledgements

I thank my research committee, Professors Robert Langridge, Irwin (Tack) Kuntz, and Fred Cohen for their time, advice, and guidance. Though I arrived at UCSF with many years of computer experience, Bob Langridge opened my eyes to an array of new (for me) ways to use computers. I appreciate the opportunity he gave me to be a graduate student in the UCSF Computer Graphics Laboratory. Tack Kuntz has been a wonderful teacher in both the lecture hall and in his office. Discussions with him have challenged me to sharpen my ideas. Most of the work contained in this thesis grew out of regular conversations with Fred Cohen. I have learned much about protein structure and about approaches to difficult problems from these sessions.

The computer and network facilities of the UCSF Computer Graphics Laboratory are very impressive. The hardware and software, which facilitated my work, would not be available without the dedicated staff. In particular, I wish to acknowledge: Tom Ferrin, who manages CGL; Willa Crowell for her help with various administrative tasks; Conrad Huang, who has kindly responded to my questions about hardware and software, and has even taken time to fix the $\frac{56}{100}\%$ which were not pure *user error*; and Teri Klein for among other things, introducing me to CGL.

I have also been fortunate to work with the members of Fred Cohen's research group. I have learned a great deal from them and have enjoyed working with them. I thank Don Kneller, Chris Ring, and John Troyer for sharing our "spacious" office alcove. Don Morris has been very helpful in discussions on scoring issues.

My major collaborator has been Scott Presnell. His mastery of both the biochemistry and computing aspects of protein structure has been invaluable. It has also been a bonus that he cares about people and has joined me in exploring local maxima (on bicycles). Thanks for being a great colleague.

My orals committee was composed of Robert Langridge, Bob Abarbanel, Fred Cohen, Tom Ferrin, and John Starkweather.

Chapter 2 of this thesis is a reprint of the material as it will appear in: Bruce I. Cohen, Scott R. Presnell, and Fred E. Cohen; **Pattern Based Approaches to Protein Structure Prediction; *Methods in Enzymology* (1991) 202, (*in press*)**. Similarly, Chapter 3 of this thesis is a reprint of some the material as it appears in: Bruce I. Cohen, Scott R. Presnell, Macdonald Morris, Robert Langridge, and Fred E. Cohen; *Proceedings of the 24th Hawaii International Conference on System Sciences* (1991) pp 574-584. The coauthors listed in these publications directed and supervised the research which forms the basis for the two chapters.

I thank the NIH Division of Research Resources (RR-1081) and the DARPA University Research Initiative (N00014-86-K-0757) for providing the facilities and support.

Last (but not least) I thank my wife, Gale Mondry, and our children, Eli, Paula, Samuel, and Barry, for wonderful support, understanding, and shared joy.

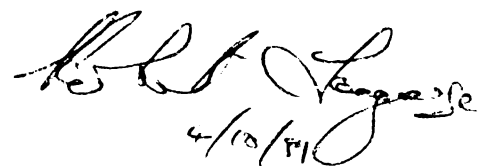
A Segment Based Approach to Secondary Structure Prediction

by
Bruce Ira Cohen

Abstract

An outstanding problem in molecular biology is the "protein chain folding" problem. One ultimate goal is to predict the three dimensional structure (the tertiary structure) of a protein from the amino acid sequence (the primary structure). An intermediate problem is to go from primary structure to secondary structure, identifying helix and/or strand subsequences within the amino acid sequence. Secondary structure provides a low resolution representation of protein structure. This thesis addresses using computers to make and evaluate secondary structure predictions.

Pattern based secondary structure prediction, which is used to mark turns, is reviewed in Chapter 2. Chapter 3 introduces a new language (*A Language for the Prediction of Protein Substructures* [ALPPS]) which allows an investigator to divide a protein sequence into segments and explore segments with both patterns and metapattern (patterns of patterns) in a hierarchal manner. Chapters 4 and 5 explore the problem of scoring secondary structure predictions on α/α class proteins. Standard residue based scoring and enhancements are described in Chapter 4. A proposal for feature based scoring is introduced in Chapter 5. The intent of feature based scoring is to evaluate a prediction as an approximation of the "known" secondary structure. An application of ALPPS development on α/α class proteins and testing using some of the scoring ideas is presented in Chapter 6.



4/10/81

Table of Contents

Chapter 1- Introduction	1
Chapter 2- Pattern Based Approaches to Protein Structure Prediction	7
2.1. Introduction	7
2.2. Theory and Methods	8
2.2.1. Definitions	8
2.2.2. Turn Prediction	10
2.2.3. Hierarchical Organization	11
2.2.4. Pattern Language	13
2.2.5. α-helices	17
2.2.6. Implementation	20
2.3. Results and Discussion	21
2.3.1. Turn Predictions	21
2.3.2. Helix Identification	26
2.4. Conclusion	29
Chapter 3- A Language for the Prediction of Protein Substructures	30
3.1. Introduction	30
3.2. A Language for the Prediction of Protein Substructures	31
3.2.1. ALPPS Language Description	33
3.2.2. An Example of ALPPS Processing	35
3.2.3. PLANS Patterns and Segment Boundaries	35
3.2.4. Segment Manipulation Functions	39
3.2.5. Region Definition Functions	41
3.2.6. ALPPS Interpretations	41
3.3. Discussion and Conclusion	45
3.3.1. Software Environment	45
3.3.2. Future Work	45
Chapter 4- Scoring Secondary Structure Predictions on α/α Proteins	47
4.1. Introduction	47
4.1.1. A Brief History of Evaluating Secondary Structure Predictions	
.....	48
4.2. Methods & Theory	50
4.2.1. Standard Residue-Based Scoring	50
4.2.1.1. Q — the Fraction of Correctly Predicted Residues	51

4.2.1.2. The Correlation Coefficient as a Scoring Measure	52
4.2.2. Adjusting for Inexact Capping: Trimming	53
4.2.3. Observed Secondary Structure of Proteins with Known Structures	57
4.2.4. Data Set of 20 α/α Proteins	57
4.2.5. Prediction Methods	58
4.2.6. Aggregate Scores and Summary Statistics	59
4.3. Results	59
4.4. Discussion	91
4.4.1. Secondary Structure Assignments	91
4.4.1.1. Differences in Secondary Structure Assignments	91
4.4.1.2. Secondary Structure Assignments and Trimming	94
4.4.2. Scoring Predictions	95
4.4.2.1. Residue Scores	95
4.4.2.2. Looking at the Feature Diagrams	96
4.5. Conclusions and Future Directions	97
Chapter 5- A Proposal for Feature-Based Scoring	99
5.1. Introduction	99
5.2. Registration and Analysis Methods	101
5.2.1. Taylor's Structure Percentage	102
5.2.2. Alternative Registrations	102
5.3. Pseudo-String Editing	105
5.3.1. String Edit Distances	106
5.3.2. Pseudo-String Objects	110
5.3.3. Pseudo-String Operations	111
5.3.4. Pseudo-String Operational Costs	114
5.4. Conclusions	116
6- A Segment Based Approach to Protein Secondary Structure Prediction	
6.1. Introduction	117
6.2. Methods	122
6.3. Results and Discussion	139
6.4. Conclusions	147
Chapter 7- Conclusions	153
References	156
Appendix A - Common Lisp Source Code for <i>def-alpps</i> Macro	164
Appendix B - Common Lisp Source Code for <i>Trimmed</i> Scoring	213
Appendix C - ALPPS and PLANS Patterns for α/α Proteins	225

List of Tables

Table II-1: Domain Size and Link Length.....	12
Table II-2: Symbols used in pattern specifications.....	15
Table II-3: Turns Patterns for α/α Class Proteins	23
Table II-4: Results on Turn Predictions.....	24
Table II-5: Helix Residue Results.....	27
Table IV-1: Proteins Used for Scoring Comparisons	58
Table IV-2: Aggregate Scores for All 20 Proteins	81
Table IV-3: Assignment Comparison Summary Statistics on Q Scores	82
Table IV-4: Assignment Comparisons Ranking by Standard Q	83
Table IV-5: Assignment Comparisons Ranking by Trimmed Q	84
Table IV-6: Assignment Comparisons Ranking by Trimmed \pm Q	85
Table IV-7: Summary Statistics on Residue Based Q Scores	86
Table IV-8: GOR Summary Statistics without 2tmv.....	87
Table IV-9: Ranking of Predictions by Standard Q	88
Table IV-10: Ranking of Predictions by Trimmed Q	89
Table IV-11: Ranking of Predictions by Trimmed \pm Q	90
Table V-1: String Edit Scores	109
Table VI-1: Proteins Used for Pattern Development and Analysis.....	123
Table VI-2: Feature Scores.....	139
Table VI-3: Region scores	145
Table VI-4: Residue by residue scores	148

List of Figures

Figure 2-1: Turns and Hydrophobic Core.....	11
Figure 2-2: Helix Components	18
Figure 2-3: Marking Turns	22
Figure 2-4: Overpredicted Turn.....	26
Figure 2-5: Underpredicted Helix Cap	28
Figure 3-1: An Example of ALPPS Processing	34
Figure 3-2: Boundary Crossings.....	38
Figure 3-3: ALPPS Blocks	43
Figure 3-4: ALPPS Regions	43
Figure 3-5: Secondary Structure Annotation	44
Figure 3-6: Hierarchy Interpretations	46
Figure 4-1: Feature Diagrams.....	48
Figure 4-2: Capping Notation	53
Figure 4-3: Sample of <i>Trimming</i>	54
Figure 4-4: Trimming Examples	55
Figure 4-5: Individual Sequence Results.....	60
Figure 4-6: Run-on Helices.....	92
Figure 5-1: Residue Scoring Ambiguity.....	100
Figure 5-2: Taylor's Structure Percentage.....	103
Figure 5-3: Alternative Registrations	104
Figure 5-4: Feature Diagrams for Editing	106

Figure 5-5: String Edit Examples	107
Figure 5-6: Distance Requirements	108
Figure 5-7: Pseudo-String Editing Operations.....	112
Figure 5-8: Nonreverse Pseudo-String Editing Operations	113
Figure 5-9: PSOPs and Adjacent PSOs	114
Figure 6-1: Turn Prediction Reviewed	128
Figure 6-2-(a): Helical Core — Hydrophobic Patch	129
Figure 6-2-(b): Helical Core — Charged Pair	130
Figure 6-2-(c): Helical Core — Putative Helix-Helix Interaction Site	131
Figure 6-3-(a): N-Cap Examples — Specific Residues	132
Figure 6-3-(b): N-Cap Examples — Acidic and Large Hydrophobic Residues.....	133
Figure 6-3-(c): N-Cap Examples — Terminating a Putative Hydrophobic Patch	134
Figure 6-4: A Working Example of an ALPPS Prediction	137
Figure 6-5: Over-predicted Turns	141
Figure 6-6: Under-predicted Turns	142
Figure 6-7: Sample ALPPS Result	149

Chapter 1

Introduction

The computer has become an integral tool for many molecular biologists. Like the microscope, the computer allows scientists to "see" new vistas which were previously inaccessible. *Computational biology* is becoming recognized as an important discipline. Molecular biologists and biochemists who devote most of their time to more traditional laboratory work can benefit from computational results.

The area of protein structure prediction offers an example of the potential benefits of computational biology. Two groups, one based at Scripps and the other in Germany, were independently studying peptides, seeking potential epitopes which would produce immunization against foot-and-mouth disease virus (FMDV). Both groups based their work on the observation that FMDV viral protein 1 (VP1) has some immunizing activity. The primary sequence of VP1 was known from DNA sequencing. The Scripps group synthesized 7 peptides with sequences corresponding to 7 subsequences of the primary structure of VP1 and tested the antibody responses in rabbits and guinea pigs. They determined that the peptide corresponding to residues 141-160 elicit high levels of antibody (Bittle *et al.*, 1982).

Unlike the Scripps group which tested 7 different peptides, the German group used computational biology to select only one, and reached a similar result with a peptide corresponding to residues 144-159 (Pfaff *et al.*, 1982). They reasoned that the antibody combining site would be on a stable section of the surface of VP1. Noting that helices have local hydrogen bonds which provide stability, they attempted to use secondary structure prediction methods on the primary sequence of VP1 to look for helices. To

meet the requirement that the epitope be on the surface, they limited the potential peptide to predicted helices with well separated hydrophilic and hydrophobic sides. Only one amphipathic helix was predicted, and subsequently the peptide based on this subsequence produced the immune response.

As demonstrated in the FMDV work, secondary structure prediction can be a valuable tool. Secondary structure prediction is also used as a step in various methods of generating tertiary structure predictions (Cohen and Kuntz, 1989; Crawford, Niermann, and Kirschner, 1987).

The widespread use of protein structure prediction to reduce (not replace) time spent in wet labs has yet to arrive. Protein structure prediction is still primarily in the hands of scientists who are developing rather than only using prediction techniques.¹ Schulz (1988) (as well as others) observes that secondary structure prediction has a limited success rate when simply looking at local sequence information. The inclusion of long-range information is necessary to improve secondary structure prediction. Work remains to be done in improving prediction techniques in this area and then making them available to a larger user community.

This thesis addresses the problem of improving secondary structure prediction for globular proteins. *Segment based* secondary structure prediction offers a new approach to predicting secondary structure features and new evaluation methods which are based on features rather than residue counts.

The goal of secondary structure prediction is to take a sequence composed from an

¹ In the FMDV example, computational biologists, W. Kabsch and C. Sander are acknowledged collaborators in the prediction of antigenic sites.

alphabet of 20 amino acids and "read" this character string to be a "paragraph" composed of super-secondary structure "sentences." In turn, the sentences are composed of "words" in the form of strands, helices, and turns. We have a *Rosetta Stone* — the Brookhaven Protein Databank (PDB) (Bernstein *et al.*, 1977) — which contains about 200 examples of protein paragraphs. Secondary structure prediction methods are developed by studying some examples from the PDB and then testing the methods on proteins which were not included in the original development set. It should be noted that some secondary structure prediction methods are based on a purely statistical evaluation of PDB examples, while others (including the work in this thesis) are grounded upon biophysical principles.

In deciphering the Hieroglyphics of the Rosetta Stone, Champollion began by finding the proper names which allowed a mapping between the phonetics of many Hieroglyphic characters and corresponding Greek characters. The recognition that oval rings (cartouches) encircled the proper names facilitated the phonetics mapping (Budge, 1950). The positions of the proper names within sentences also contributed to determining the grammar. This bootstrap approach began with a partial solution to the problem. Then the added clues from the partial solution contributed to a more complete understanding of Hieroglyphics and the ancient Egyptian language.

Similarly, a bootstrap procedure can be used in secondary structure prediction. Local sequence patterns combined with some longer range information derived from the tertiary structure class (α/α , α/β , β/β , Levitt and Chothia, 1976) gives accurate predictions of turns — places between regular secondary structure feature (*i.e.*, β -strands and α -helices). Chapter 2 reviews this work and describes the use of augmented regular

expressions to express sequence patterns.

Our segment based approach builds on the success of the turn predictions. An attempt is made to consider segments in terms of the local secondary structure. For example, assuming that all turns have been marked at one or more residues within the turn, the segment composed of residues running between two turn markings (residues which have been marked as turn residues), should contain one or no regular secondary structure feature. The size of these segments will vary, but they make sense as logical units of a protein sequence. Other methods use subsequences (*e.g.*, windows in neural net methods), but these are not meaningful in terms of the underlying putative secondary structure. As detailed in Chapter 3, different types of segments are used to construct a secondary structure prediction. *A Language for the Prediction of Protein Substructures* (ALPPS) is introduced which allows investigators to:

- specify different types of segments,
- explore and characterize segments with both local sequence patterns and metapattern (patterns of patterns), and
- look at patterns of characterized segments,

all in a hierarchal manner.

In evaluating secondary structure predictions, comparisons have generally been made based on four tallies (true positive, true negative, false positive, false negative) for each residue in the sequence. As demonstrated in Chapter 4, this type of evaluation does not necessarily portray the degree to which a prediction approximates the observed secondary structure. The helices of 20 α/α proteins from the PDB are determined by 3 assignment methods and helix predictions are obtained from 4 prediction methods. By representing secondary structures in juxtaposed *feature diagrams* — scaled boxes and lines representing helices and turns — differences between predictions and assignments

based on known tertiary structure can be observed. Chapter 4 also gives some insight into the differences between the results of applying different secondary structure assignment methods to the same α/α protein.

Chapter 5 continues the theme of evaluating secondary structure predictions by looking at predictions as approximations of the "known" structure. For *feature based scoring*, we must first define "acceptable approximation" in the context of the intended use of the prediction. After reviewing one existing attempt at feature based scoring, a proposal is sketched for a method, *pseudo-string edit distance*, which looks at the types of changes (*e.g.*, insert a turn into the D helix) which would transform an abstraction of the prediction into an abstraction (of the same resolution) of the observed secondary structure.

The integration of concepts contained in the previous chapters find application in Chapter 6. A complete cycle of segment based helix analysis and prediction on α/α proteins is presented. The chapter shows that the segment based approach achieves better results than earlier prediction methods (Chou-Fasman, GOR), but the results are generally not as good as recent neural net results. On the other hand, the segment based approach needs to be seen as an overall philosophy which contributes to advancing the value of secondary structure prediction. This and future directions are included in the concluding Chapter 7.

A note on organization: Two of the chapters (2 and 3) are manuscripts of published [or in press] articles and are reprinted with permission of the respective publishers.² A third chapter (6) is essentially the manuscript of a paper which is being submitted for publication. Taken as parts of this thesis, these three chapters contain some redundant material which is necessary to allow them to stand as independent works. References for all chapters are integrated in one list after Chapter 7. Two appendices list Common LISP code for the `def-alpps` macro, which implements the features of ALPPS, and for the *trimming* variation of residue based scoring introduced in Chapter 4. A final appendix contains the ALPPS and PLANS patterns used to produce the results of Chapter 6.

² Chapter 2 will appear as

Bruce I. Cohen, Scott R. Presnell, and Fred E. Cohen; **Pattern Based Approaches to Protein Structure Prediction**; *Methods in Enzymology* (1991) 202, (*in press*).

Chapter 3 appears as part of

Bruce I. Cohen, Scott R. Presnell, Macdonald Morris, Robert Langridge, and Fred E. Cohen; **Pattern Recognition and Protein Structure Prediction** in *Proceedings of the 24th Hawaii International Conference on System Sciences* (1991) pp 574-584.

A slightly different version of Chapter 6 will be submitted to *Biochemistry* as
Scott R. Presnell, Bruce I. Cohen, and Fred E. Cohen; **A Segment Based Approach to Protein Secondary Structure Prediction**.

Chapter 2

Pattern Based Approaches to Protein Structure Prediction[†]

2.1. Introduction

In the appropriate milieu, polypeptide chains spontaneously assemble into unique tertiary structures guided by their amino acid sequence (Anfinsen *et al.*, 1961). While numerous experiments suggest the existence of a folding code, explicit specification of sequence based folding rules has proven difficult (Schultz, 1988). The goal of our research is to develop a set of sequence-structure correlates that can be used to predict secondary structure from protein primary sequence. This chapter begins by discussing a series of principles which form a foundation for structure prediction. Next we sketch an algorithm for finding turns and describe some of the requirements for a pattern language which facilitates the identification of sequence-structure correlates. We will then present the pattern language itself and finally offer examples of patterns which can be used to recognize turns or loops and α -helices.

A pattern language must allow for the specification of exact residue-by-residue matches and simultaneously offer flexibility and generalizability. We have developed a convenient computer interface for the development of patterns that recognize protein sub-structures. While efforts to completely automate the development of reliable sequence-structure correlates have failed in our hands, we believe that structural principles can be translated by an individual into a pattern formalism and that refinement of

[†] © 1991 Academic Press. Reprinted, with permission, from:
Bruce I. Cohen, Scott R. Presnell, and Fred E. Cohen; **Pattern Based Approaches to Protein Structure Prediction**; *Methods in Enzymology* (1991) **202**, (in press).

these initial patterns can lead to useful algorithms for predicting secondary structure.

2.2. Theory and Methods

2.2.1. Definitions

There are a number of different approaches to assigning secondary structure to proteins of known three dimensional structure. Although crystallographers' assignments are readily available in Protein Databank files (Bernstein *et al.*, 1977), they are subjective. A researcher looking at the structure on a computer graphics screen might reasonably make an alternative assignment. Attempts have been made to devise algorithms (*e.g.*, Levitt and Greer, 1977; Kabsch and Sander 1983; Richards and Kundrot, 1988) which would provide objective assignments based on:

- distribution of backbone dihedral angles;
- hydrogen bonding patterns; and/or
- interatomic distances of α -carbons.

Each of these methods have strengths and weaknesses. An algorithm that does well in assigning α -helices may not do as well for β -structure. For example, hydrogen bonding patterns tend to underassign β -structure when compared to many subjective assessments. Backbone dihedral angles ϕ and ψ give very local information which may not be sufficient to adequately define actual (as opposed to ideal) secondary structure. The point here is not to champion any one algorithm, but rather to underscore the importance of examining the underlying secondary structure assignment when evaluating the results of secondary structure predictions.

A fundamental assumption in this work is that the structural class ($\alpha/\alpha, \alpha/\beta, \beta/\beta$) of the protein in question is known (Levitt and Chothia, 1976). Some clues for accurate

secondary structure prediction (*e.g.*, *link length*) are dependent upon structural class (Cohen *et al.*, 1986). Currently, experimental data (*e.g.*, circular dichroism, Manavalan and Johnson, 1983), compositional analysis (Sheridan *et al.*, 1985), and/or homology with known structures can yield some structural class information. In the worst case, results from the three (α/α , α/β , β/β) secondary structure predictions could be used to guide inclusion in one of the three classes.

The protein domains of known tertiary structure are used both as sources for the development of hypothetical sequence-structure correlates and as specimens to examine for potential correlates and as sequences for testing the theories. To eliminate potential bias, a clear division is made between sequences used to develop sequence-structure correlates, and sequences used for testing. Generally applicable sequence-structure correlates might be discovered without this separation, but it is also possible for candidate sequence-structure correlates to appear very strong in the development environment and yet prove to be of no value outside of the development sequences. The development set is studied exhaustively and aids in the creation and refinement of sequence patterns. The test set is sequestered from the investigator's view and evaluated only in a blind manner to test the generality of concepts proposed through an analysis of the development set.

Although the alphabet of amino acids which make up the protein sequences contain 20 characters, there are many ways to group *similar* amino acids for the purpose of exploring sequence-structure correlates (Schulz and Schirmer, 1979; Smith and Smith, 1990; Karlin *et al.*, 1989). Examples (*e.g.*, groupings by hydrophobicity, charge, or size) can be found in the case of helical core recognition. The pattern language must be able to accommodate such groupings.

2.2.2. Turn Prediction

Typically, turns³ are solvent accessible, evenly distributed through the sequence and dominated by hydrophilic residues. Rose has shown that the total number of turns in a globular protein varies linearly with the molecular weight of the chain (Rose, 1978). We have refined this relationship by focusing on isolated protein domains and separating the structural subclasses (*e.g.*, α/α , α/β , β/β).

Consider a simplified model of a domain of a globular protein as a chain subtended by a spherical boundary following a path from one side to the other and back again (see Figure 2-1). The radius of the bounding sphere is

$$r = (3 \times 110n / 0.602 \times 4\pi\bar{v})^{1/3}$$

where 110 is the average molecular weight of an amino acid, n is the number of residues, and \bar{v} is the partial specific volume (0.75 g/cm^3) (Cohen *et al.*, 1986). The solvent accessible turns should occupy most of the volume within 4\AA of the boundary. In this model, the length of the sequences linking the turns are a function of the number of residues in a domain. Average domain size varies from protein class to class. These are compiled in Table II-1. Since the link length varies as the $1/3$ power of the domain size, extreme precision is not required. The separation between consecutive turns can then be calculated from this information and the pitch associated with α -helices and β -strands. Since the secondary structure composition in α/β proteins is approximately $1/3$ β -strand and $2/3$ α -helix, a weighted average pitch is used. Protein class imparts a clear distinction in locating turns in this formalism. For example, a weakly hydrophilic region might be labeled a turn in a β/β protein, whereas this same sequence in an α/α protein could be

³ We define turns as segments of the polypeptide chain which join secondary structure units. β -turns and irregular coiled regions are unified under this heading.

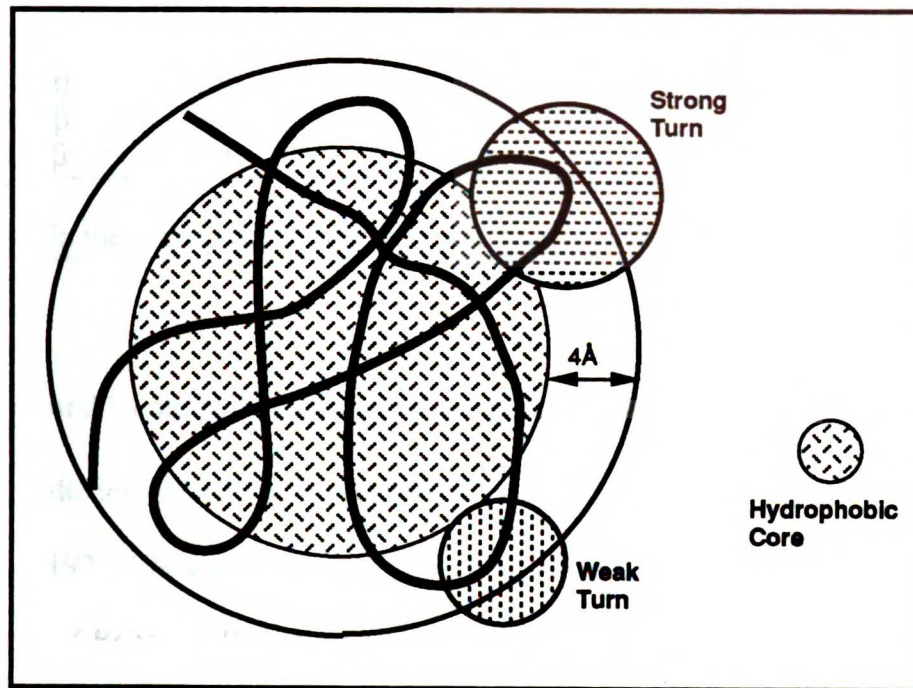


Figure 2-1: Turns and Hydrophobic Core

A globular protein is represented by a chain (bold line) traversing a sphere. The solvent accessible turns occupy most of the volume within 4\AA of the boundary. Weak turn sequence patterns (*e.g.*, areas with some hydrophilic residues) may be good turn indicators when they are appropriately spaced from strong turn sequence patterns (areas dominated by hydrophilic residues).

assigned to a part of an α -helix.

2.2.3. Hierarchical Organization

Several physical principles are used to guide the development of patterns for locating turns. These include: local maxima in hydrophilicity; putative secondary structure identification and avoidance; special backbone dihedral angle restrictions on proline; and weakly hydrophilic regions appropriately spaced from well-defined turns. This also

Table II-1: Domain Size and Link Length

Class	Domain Size (residues)	Radius (Å)	Pitch (Å/residue)	Link Length (residues)
α/α	150	20.6	1.5	22 [†]
α/β	200	22.7	2.25	16
β/β	100	18.0	3.0	9

[†] In the current set of patterns a link length of 26 residues is used.

suggests a natural hierarchy of patterns reflecting the relative merits of each concept.

Turn prediction algorithms developed previously (Lewis, Momany, and Scheraga, 1971; Kuntz, 1972; Chou and Fasman, 1974; Chou and Fasman, 1977; Garnier, Osguthorpe, and Robson, 1978) have relied ultimately on the calculation of one local parameter to specify turn likelihood. A cutoff is applied to sort turns from non-turns. Unfortunately, no one cutoff value is adequate. Typically, there exists a cutoff value that partitions turns from non-turns which is specific (few false positives) but not sensitive (many false negatives) and a second cutoff value which is sensitive but not specific (Cohen and Kuntz, 1989). The observation of Kabsch and Sander (Kabsch and Sander, 1984) that sequentially identical pentapeptides adopt dramatically different conformations in different proteins demonstrates that local sequence information alone is not sufficient for determining structure. To overcome this problem, we have combined the local sequence features with their global spacing to improve turn prediction accuracy. In practice, "strong" turns are located and neighboring sections of the chain are *masked*

from further consideration. The extent of the mask varies with protein class.

2.2.4. Pattern Language

The language used to specify sequence patterns is called PLANS, Pattern Language for Amino and Nucleic acidS (Abarbanel, 1984). As the name implies, this language could be used for investigating nucleic acid sequences (*e.g.*, RNA secondary structure prediction).

The simplest patterns associate a label with a particular sequence. For example, *aspartyl-protease-site* could be the label associated with the sequence aspartic acid, threonine, glycine (Pearl and Taylor, 1987).

aspartyl-protease-site: "DTG"

The aspartyl protease site could be generalized through the use of a *logical OR* represented symbolically with square brackets [].

asp-site: "D[TS]G"

Here, serine may replace threonine in the sequence.

More general patterns are created through the concept of *regular expressions* (Lewis and Papadimitriou, 1981); a concept used in *grep*, a UNIX (Ritchie and Thompson, 1974) pattern matching utility. PLANS expressions are composed of sequences (*e.g.*, "DTG"), ORs (*e.g.*, "[TS]"), and *quasi-closures* (repetition *e.g.*, "S"%1,3% - meaning 1, 2, or 3 serines). In addition to regular expressions, PLANS allows complex patterns to be built from less complex patterns using logical operations (*and, or, not*). The pattern *asp-site-complex* demonstrates the use of a boolean operation on two simple patterns. Note that *asp-site* and *asp-site-complex* are functionally

equivalent.

asp-site-complex: "DTG" OR "DSG"

In addition to the usual one letter code for amino acids, other special characters are used and summarized in Table II-2. A particularly useful special character is '.', which represents any residue. Thus

charge-pair: "[DE]...[KR]"

would identify a pair of oppositely charged residues spaced by four along the sequence.

This pattern is based on the concept of a stabilizing charge pair in an α -helix (Kim and Baldwin, 1984). The pattern

charge-pair-2: "[DE]..[KR]",

which has a spacing of only three residues, represents the same biophysical concept. The pattern

gen-charge-pair: charge-pair OR charge-pair-2

is a generalization which could also be specified using the repeat notation,

gen-charge-pair-repeat: "[DE].%2,3%[KR]".

Table II-2: Symbols used in pattern specifications

Special Symbol	Meaning
^	beginning of sequence
\$	end of sequence
*	ZERO or more repeats of the proceeding symbol, equivalent to %0,*
%n,m%	between n and m repeats of previous symbol or pattern specification; m may be '*' to indicate n or more repetitions
[]	logical OR of symbols in brackets, as in [ABC]
-	"through", used in [...] to indicate a range of values, e.a. [A-CG-K] means [ABCGHIJK].
{m,u}	spreading of previous symbol that hits at position i, to all sites between i+m and i+u.
()	used for grouping of characters for repetitions or logical combinations of pattern expressions involving AND, OR, or NOT.
group	group(N,patspec) finds contiguous matches of 'patspec' at least N in length. For example, group(5,"A") finds 5 or more contiguous A's.
density	density(op,N,D,pat-spec) finds regions where <i>op</i> (one of >, <, =, != or <>, <=, >=) operator on N of D matches for pat-spec is found. For example, density(>,2,5,"[AT]") finds areas of length at least 5 continuing 2 or more out of 5, "A" or "T" matches.

Every pattern has associated markers pointing to those locations where the pattern succeeds in matching part of a sequence. Although the matching subsequence may be necessarily greater than one residue (*e.g.*, any subsequence which matches *aspartyl-protease-site* must be three residues long), the PLANS marker is placed by default on the first and only the first residue.

*
...SYDTGC...

Markers can be manipulated by the *spreading* feature. By attaching {x,y} to a pattern, marks are placed on all residues from n+x through n+y when the pattern first matches at position n. Thus

aspartyl-protease-site-spread: "DTG"{0,2}

will be marked:

...SYDTGC...

since the mark is to be spread one to include an additional two residues after the starting point. The term *pattern match* refers to the placement of the marks.

Complex patterns are built from other patterns using logical operators which use these marks. Logical operators allow the requirement of more than one pattern match at a given site (*AND*), at least one match at a given site (*OR*), or the absence of a match (*NOT*). For example, a helix could be specified as

helix: (*hydrophobic-patch* AND NOT *many-pro*)

where *hydrophobic-patch* is a complex pattern identifying amphipathic sequences having periodicity consistent with helical geometry, and *many-pro* represents two or more pro-lines within a window of eight residues. This pattern,

many-pro: density(\geq ,2,8,"P"){0,8},
 also demonstrates the *density* operator. Any one of six relational operators (*i.e.*, \geq , \leq , $>$, $<$, $=$, \neq) can be used with the density operator. In this way, simple patterns based on biophysical principles can be used to rapidly assemble a hierarchical list of complex statements about the location of substructures in protein sequences.

2.2.5. α -helices

Continuing with a hierarchical approach, turn prediction forms a starting point for predicting α -helices and β -strands. They serve as markers which tokenize a sequence into segments or blocks containing at most one unit of secondary structure. In this chapter we restrict our attention to α -helices in all helical proteins. In this regime, the challenge is to distinguish blocks which contain α -helices from those including aperiodic structures.

Previous attempts at predicting the location of α -helices along a sequence have treated helices as homogeneous entities. Some attempts have been made to exploit the amphipathicity of helices (Gribskov, McLachlan, and Eisenberg, 1987; Schiffer and Edmundson, 1967). Others have distinguished the middle of the helix from its ends (Chou and Fasman, 1978; Argos and Palau, 1982; Richardson and Richardson, 1988). We set out to link these concepts in an attempt to improve predictive accuracy (see Figure 2-2).

α -helices are commonly observed in globular proteins. Typically, the backbone dihedral angles cluster around $\phi = -47^\circ$, $\psi = -57^\circ$ producing backbone C=O(i) to N-H(i+4) hydrogen bonds with 3.6 residues per turn and a pitch of 1.5 Å per residue. Dis-

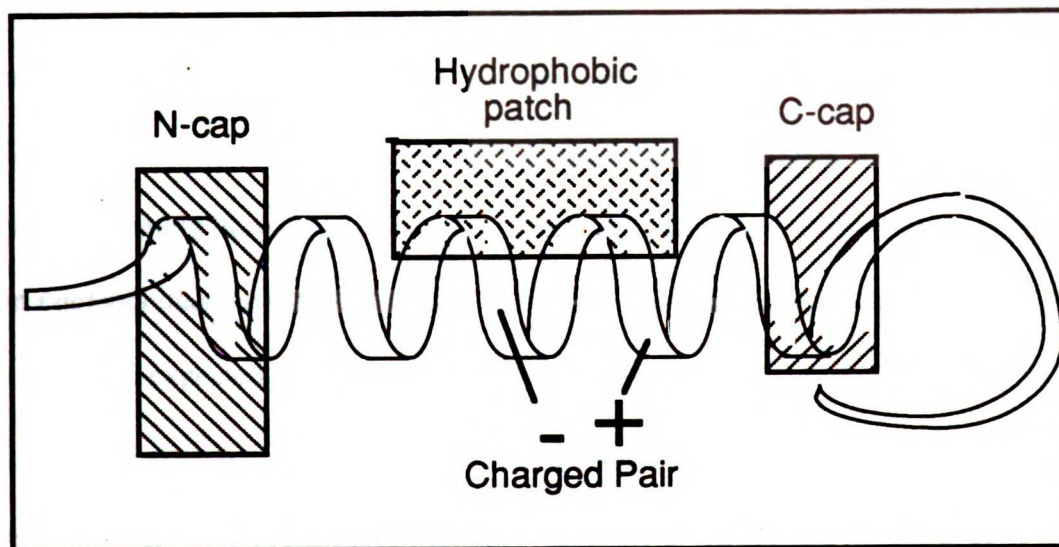


Figure 2-2: Helix Components

A helix can be considered as a helical core sandwiched between an N- and C-cap. Each component has a set of characterizing patterns. For example, the helix core commonly contains a hydrophobic patch (pattern H1) and/or a charge pair (pattern charge-pair).

tortions are observed including bifurcated backbone hydrogen bonds (Parry, 1982), bends in the helix axis (Barlow and Thornton, 1988), and changes in periodicity near the helix boundaries (*i.e.*, 3_{10} helix formation) (Schellman, 1980; Kabsch and Sander, 1983a). The packing of α -helices creates a seamless interface with geometric properties dictated by the periodicity and bulk of the α -helix. These may be seen in continuous repeats in fibrous proteins (*e.g.*, tropomyosin (McLachlan and Stewart, 1975; Parry, 1982)) or in a variety of arrangements in globular proteins (Chothia, Levitt, and Richardson, 1977;

Richmond and Richards, 1978). Typically, the helix-helix interface is dominated by hydrophobic residues which cluster with a predictable distribution along the chain (Chothia, Levitt, and Richardson, 1977; Richmond and Richards, 1978; Cohen, Sternberg, and Taylor, 1982). The hydrophobic packing interface is often complemented by a hydrophilic region on the opposite side of the helix. This amphipathic arrangement depends upon the specific location of the helix within the folded protein. While most helices contribute residues to the solvent interface and hydrophobic core, some are almost completely buried (*e.g.*, the J helix, residues 222-244, in thymidylate synthase from *Lactobacillus casei* (Hardy *et al.*, 1987)). In these less common cases, amphipathicity is neither expected nor observed.

Regular expression patterns have been constructed to recognize hydrophobic residues clustering in a manner compatible with α -helices. One such expression is

H1: $\phi.. \phi\phi.. \phi$
 where ϕ : [AVILMCKFWY].

Additional patterns which reflect the plausible alternative arrangements of hydrophobic and hydrophilic residues which reflect the concepts embodied in *H1* have been constructed. Composite patterns can be developed which recognize sequences containing hydrophobic patches and hydrophilic stripes juxtaposed in a manner consistent with an amphiphilic α -helix.

The identification of sequences which cap the N- and C-terminal ends of α -helices has been a subject of recent interest in the literature (Argos and Palau, 1982; Richardson and Richardson, 1988; Presta and Rose, 1988). It is clear from this work that the intrinsic conformational properties of certain amino acids are well suited to the geometry of helix termination. Some residue biases have been attributed to the helix macrodipole which

creates the equivalent of $+1/2e$ at the N-terminus and $-1/2e$ at the C-terminus of the helix axis (Hol, Duijnen, and Berendsen, 1979). Unfortunately, these residue tendencies are not specific for the caps of α -helices. However, in the appropriate sequence and structure context, residue capping tendencies are relevant. For example, the N-terminal cap of an α -helix tends to be in phase with the hydrophobic face of the helix. Thus, the N-cap is usually pointing inward toward the rest of the molecule creating a smooth segue between a particular helix and its predecessor. In practice, this takes the form

NZ: [GDNST][DEKPNQRS].. $\phi\phi$.. ϕ

where glycine, aspartate, asparagine, serine, and threonine are useful N-cap residues and aspartate, glutamine, lysine, proline, asparagine, glutamine, arginine, and serine disrupt the hydrophobic patch.

C-caps can be defined by analogy to N-caps. For example,

CZ: ϕ .. $\phi\phi$..[DEKPNQRS][GKNH]

describes a helix C-cap. Glycine and asparagine can adopt backbone dihedral angles which favor left handed α -helices thereby breaking the usual helical periodicity. Lysine and histidine can interact with the negative pole of the helix dipole when the side chain lies along the helix axis. This interaction requires a break in the helical repeat.

2.2.6. Implementation

The program *Match* (Abarbanel, 1984), written in the C programming language, is available on request from the authors. Once installed, users do not need to be concerned about the underlying intricacies of the software, only the concepts of the pattern matching language.

The computer language LISP is used as a development tool for building both pattern matching languages and user environments for developing sequence structure correlates. LISP is available in both interpreted and compiled versions which offer different balance points in the trade-off between run-time speed and ease of program changes. Working in a UNIX environment also allows stable, mature portions of the software to be transferred to the C computer language, and linked to the underlying LISP system.

In addition to being easily modified in an interpreted environment, LISP's list data structure readily lends itself to modelling residue sequences and secondary structure assignment sequences. Object constructions available in Common LISP are used to build instances which house sequences and PLANS matches for a set of proteins and an associated set of patterns. A mouse and window based application, *Match-Point*, is being developed on Sun workstations as an interactive software tool for exploring potential structure sequence correlates.

2.3. Results and Discussion

2.3.1. Turn Predictions

The α/α turn patterns are compiled in Table II-3. The 5% decrease in prediction accuracy from the development set (89%) to the test set (84%) suggests that the possibility that some patterns recognize specific features of the development set instead of general principles of turn formation. An example of turns located along a protein sequence is shown in Figure 2-3. The prediction accuracy was based on the total number of turns correctly predicted. Multiple identifications of a single continuous turn had no impact on

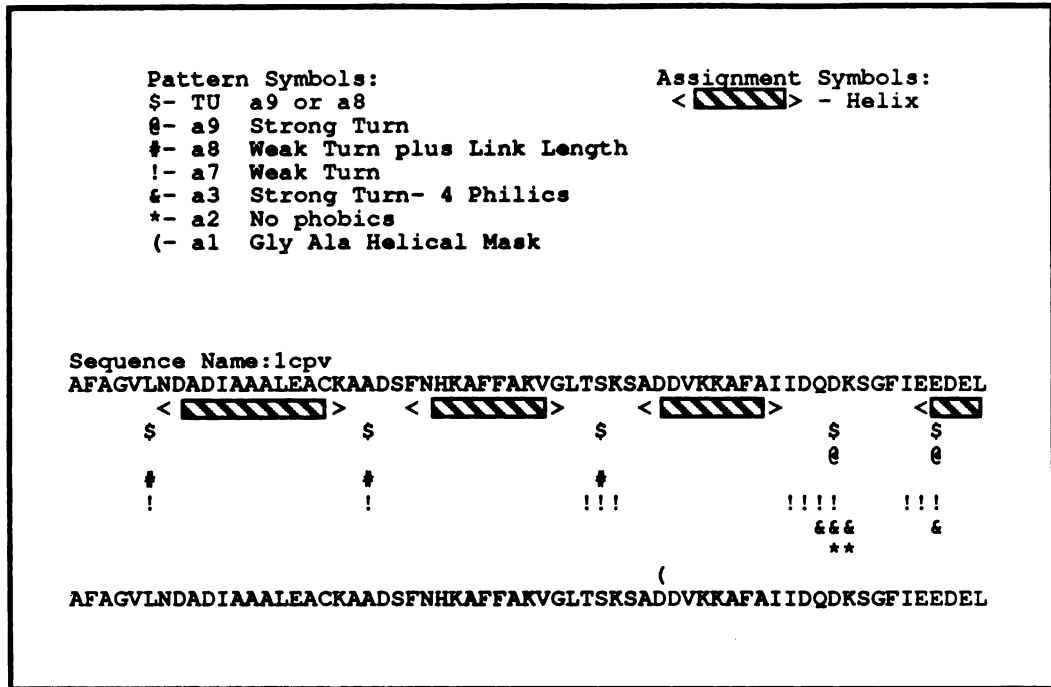


Figure 2-3: Marking Turns

The final turn pattern, TU, and its constituent subpatterns are demonstrated. Notice the cascade from the subpatterns to the final pattern.

the prediction score. Correctly identified turns (true positives) and overpredictions (false positives) are presented in Table II-4.

The turn algorithm has been applied to Interleukin-2 (IL-2, Cohen *et al.*, 1986a) and Human Growth Hormone (HGH, Cohen and Kuntz, 1987) sequences prior to the elucidation of their crystal structures (Brandhuber *et al.*, 1987; Abdel-Meguid *et al.*, 1987). In IL-2, all turns were correctly located. In HGH, turns were predicted which consistently shortened the crystallographically observed α/α -helices by 4 to 8 residues. For IL-2, the domain contains 132 residues while HGH contains 191 amino acids. We

Table II-3: Turns Patterns for α/α Class Proteins

Name	Pattern ;;; comment
g1	<p>"[ACFIKLM TVWY][ADEHKNQRST]%2,2[AG][ADEHKNQRST]%2,2 [ACFIKLM TVWY]" {4,4} ;;; Cluster of Hydrophobics bounded by hydrophilics (helix-helix interactions)</p>
g2	<p>"[ACFIKLM TVWY]%2,2[ADEGHKNQRST][ADEHKNQRST][AG] [ADEHKNQRST]%2,2.[ACFIKLM TVWY]" {5,5} ;;; Cluster of Hydrophobics bounded by hydrophilics (helix-helix interactions)</p>
g3	<p>"[ACFIKLM TVWY].[ADEHKNQRST]%2,2[AG] [ADEHKNQRST]%2,2[ACFIKLM TVWY]%2,2" {5,5} ;;; Cluster of Hydrophobics bounded by hydrophilics (helix-helix interactions)</p>
ma	<p>(density(>=,3,9,"A")) {5,5} ;;; Too many alanines in one area - bad helix-helix interactions</p>
ga	<p>(density(>=,2,9,"G")) {5,5} ;;; Too many glycines in one area - bad helix-helix interactions</p>
gs	<p>((g1 or g2 or g3) and (not ma) and (not ga)) ;;; A Gly-Ala type helical site without too many Alas</p>
a1	<p>(gs) ;;; Primary mask for helical secondary structure</p>
a2	<p>(density(=,0,5,alpha_strong_phobic)) {2,2} ;;; Strong turn denoted by the absence of strong hydrophobic residues</p>
a3	<p>(density(=,4,4,alpha_philic)) {1,1} ;;; Strong Turn of four hydrophilic residues in sequence</p>
HP	<p>("[VLI AWYKFCT][VLAIWKYFCT]P[VLAIWYFCT]" {2,2} and (not density(>=,1,5, "[NQRS]")) {1,1}) ;;; Potential situation of a proline in helical (hydrophobic) region</p>
a5	<p>("P" {-1,-1} and (not a3 {-13,0}) and (not HP {-1,-1})) ;;; Turn denoted by a proline that is not in a hydrophobic environment</p>

Table II-3 (Continued): Turns Patterns for α/α Class Proteins

Name	Pattern ;;; comment
alpha_philic	"[DEGKPNQRS]" ;;; Hydrophilic residues
alpha_strong_phobic	"[ACFILMVW]" ;;; Strong hydrophobic residues
ap	(not density(>=,2,3,alpha_philic)) {-1,-1} ;;; Regions unlikely to be good turns
a7	((not ap{-1,1}) and (not a1{-2,2})) ;;; Possible regions for weak turns, distant from the strongest turns
a8	(group(7,a7) and (not a9{-13,13})) ;;; Merged weak turns (up to seven hits) and no other turn indications
a9	(group(7,a3) or group(7,a5)) ;;; Grouped possible turn sites
TU	(a8 or a9) ;;; Final consensus turn pattern

Table II-4: Results on Turn Predictions

Set	True Positives	Overpredictions
Learning Set 47 turns from 8 α/α proteins	42 (89%)	3 (6%)
Test Set 51 turns from 9 α/α proteins	43 (84%)	4 (8%)

initially assumed that the average domain size for α/α proteins would be 150 amino acids. This effectively limited the length of HGH helices to at most 22 residues. Following the calculation described in Table II-1, an α/α domain of 191 amino acids should contain 25 residue helices; the average length of a α -helix in HGH is indeed, 26 residues. This suggests that it may be advantageous to recalculate the anticipated link length for a sequence prior to making a secondary structure prediction.⁴

Within an error range of four residues, the strong turn patterns rarely generate false turn indications. Therefore we can depend on these patterns as landmarks in a sequence. However, the weaker turn prediction patterns are not as reliable an indicator of actual turns. These patterns are more dependent on additional signals, such as the expected periodic distance between turns (Table II-1). As a consequence, most of the over- or under-predictions (false positives and false negatives) are a result of the noise in these additional signals.

In a specific example of an overprediction (false positive), the first helix of 2ccy (Cytochrome c prime from *Rhodospirillum molischianum*) is broken by the presence of a weak turn indicator (Figure 2-4). The periodic distance used to mask erroneous weak turn indications is 26 residues; 13 residues on either side of the current pattern. In this particular instance, the weak turn indications come 15 residues from the nearest strong turn indicator, hence, the pattern is accepted as an authentic turn.

As an example of an underprediction (false negative), the final turn of 2lzm (Bacteriophage T4 Lysozyme [E.C.3.2.1.17] residue 157) is not recognized by the existing turn patterns. While there are indications of weak hydrophilicity near the position we

⁴ In the current set of patterns a link length of 26 residues is used.

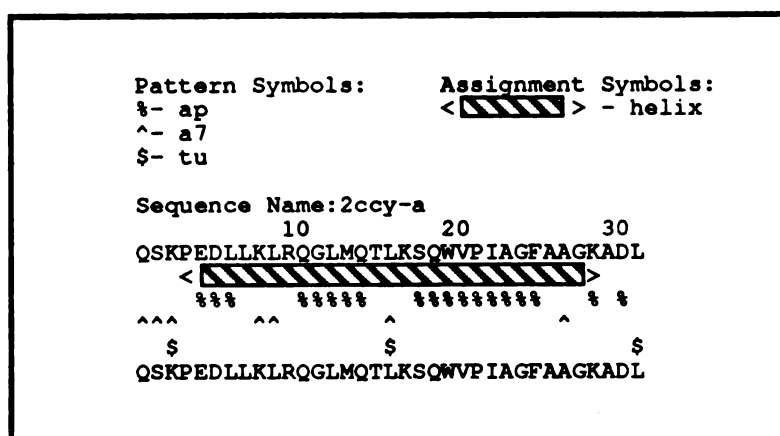


Figure 2-4: Overpredicted Turn

The pattern a7, represented by ^, denotes areas of weak hydrophilicity. The the marked region in the center of the helix is sufficiently far from neighboring turns to be predicted a turn.

would like to call a turn and that position is sufficiently distant from the last predicted turn, the indications are below the level considered dependable for predicting a weak turn.

2.3.2. Helix Identification

Preliminary results on the accuracy of pattern based prediction of helical features are collected in Table II-5. The patterns used to predict helix core regions are well

Table II-5: Helix Residue Results

	Percent Accuracy	
	Development Set	Test Set
Helix Core	87%	82%
N-Cap	75%	65%
C-Cap	74%	58%

defined, providing a predictive accuracy of 87% for the development set and 82% for the test set. Again, the 5% decrease in prediction accuracy is expected and presumably indicates the recognition of specific features in the development set proteins instead of the general principles of helix core formation. The majority of over-predictions (false positives) for helix cores stem from a displacement or extension of the predicted helical core, either beginning too soon, or ending too late in the sequence. Currently, it is difficult to associate this character with a specific structure or sequence phenomenon. Helices characterized by short runs are often under-reported (false negatives). Helices with a strong hydrophilic character are also problematic. Prediction of the core region of these kinds of helices is generally dependent on the the recognition of oppositely charged residue pairs in spatial proximity to one another (a "charge-pair"), and putative tertiary helix-helix interaction sites (Cohen, Richmond, and Richards, 1979).

The predictive capability of the N- and C- cap patterns is about 10% lower than that of the helix core patterns. The decrease in predictive accuracy from the development to the test set for the cap patterns is currently greater (at 10% to 15%) than the decrease for the turn or helix core prediction. This indicates a higher tendency for the capping patterns to recognize specific features in the development set proteins, rather than general

```

Pattern Symbols:  Assignment Symbols:
@- Nt           <██████████> - helix

Sequence Name:lhds-b
80             90             100
LKGAFQAQLSGLHCNKLHVNPQNFRLLGNVLAL
  <██████████>           <██████████>
                        @

```

Figure 2-5: Underpredicted Helix Cap

The left most helix shows a situation where a helix N-cap could not be predicted using the current set of patterns.

principles of the amino acid sequences that initiate and terminate helices. While there is no structural feature that identifies over-prediction; under-prediction is characterized by the lack of crucial residues in the amino acid sequence near the site of the N- or C- cap. Typically, the capping patterns will have one of a class of residues commonly found at the terminus of a helix. The N-cap positions in helices are often one of the residues G, N, S, T, or D. The C-cap positions are usually G, K, H, or N. Further, proline is often a constituent of C-cap areas of sequence, appearing one or two residues after the cap posi-

tion. If a helix does not begin or end with one of these residues (*e.g.*, the N-terminal residue of the fifth helix in the β chain of hemoglobin, a phenylalanine, Figure 2-5), the likelihood of a correctly predicted helix cap is low. This accounts for most of the under-prediction of this helical feature.

2.4. Conclusion

Pattern based secondary structure prediction appears to be useful. An algorithm to predict the locations of turns in proteins as a function of protein class has been validated on a set of proteins excluded from the information used in pattern development. Turn prediction for protein structures solved subsequently reinforces the statistical validity of the development test set paradigm.

Turns form a useful division of the protein sequence into subsegments which contain no more than one piece of secondary structure. In the case of all helical proteins, it appears possible to consistently recognize helical core sequences. The exact boundaries of the helices have proved more problematic. Work is underway to provide a framework for meta-patterns, patterns of PLANS patterns (*e.g.*, a syntactic organization of N-cap, helix core, C-cap patterns) which could enhance the prediction accuracy. Additionally, studies on model helix systems (Hughson, Wright, and Baldwin, 1990; Bradley *et al.*, 1990) could help define the rules of helix initiation and termination.

Chapter 3

A Language for the Prediction of Protein Substructures[†]

3.1. Introduction

The rapid generation of DNA sequence data is creating an information explosion, but the determination of protein structure from protein sequence information is not keeping pace. There are two primary reasons for this gap. First, protein structures are now determined by laborious physical techniques. X-ray crystallography and nuclear magnetic resonance spectroscopy (NMR) routinely yield 1-2 structures per month (Blundell and Johnson, 1976; Wuthrich, 1986). This falls short of the sequencing rate by two to three orders of magnitude. Second, not all proteins are amenable to structural characterization by these physical techniques. X-ray crystallography requires crystals with diffraction ordered out to near atomic resolution, suitable heavy atom derivatives to aid in the solution of the phase problem, and three to five man years of work per protein. NMR studies require proteins of relatively low molecular weight (currently <20 kD), high solubility, and one to three man years. To keep up with the sequence information, new computational schemes which relate protein sequence to structure must be developed.

The pioneering experiments of Anfinsen and co-workers suggest that a protein sequence contains all of the information necessary to encode the three dimensional structure of a water soluble globular protein (Anfinsen *et al.*, 1961). Although the long term goal of predicting three dimensional structure from a protein sequence remains an open problem in molecular biology, the methods continue to advance. Improved secondary

[†] © 1991 IEEE. Reprinted, with permission, from:
Proceedings of the 24th Hawaii International Conference on System Sciences; Kauai, Hawaii,
January 1991; pp 574-584.

structure prediction can serve as a stepping stone for structure generation (Cohen and Kuntz, 1989). A concept of "structural homology," which can define relationships between proteins that lack strong sequence homology, is beginning to develop.

This paper addresses a pattern matching approach to secondary structure prediction. The work presented here is not intended to be a treatise on the linguistics of sequences. Other authors have covered this topic in greater detail (Searls, 1990). Instead, this work builds on previous efforts that used *regular expression* patterns to facilitate the identification of sequence-structure correlates (see Cohen *et al.*, 1986; Cohen, Presnell, and Cohen, 1991 [reprinted here as Chapter 2]) the work discussed below takes a *segment-oriented* approach to secondary structure prediction. This chapter introduces the details of a new specification language, A Language for the Prediction of Protein Substructures (*ALPPS*).

3.2. A Language for the Prediction of Protein Substructures

This section will discuss a specification language⁵ called *ALPPS*, A Language for the Prediction of Protein Substructures. The vocabulary of *ALPPS* is designed to mimic a hierarchal view of protein structure. Although the terms reflect descriptions of protein substructures, the names are free of the targeted substructure. This allows an abstract view of the problem which may be valuable in preventing premature commitments to one potential structural assignment. For example, we speak of *regions*, and not *putative-helices*. On the other hand, a region can be thought of as a segment which may approxi-

⁵ The term *specification language* is used here because *specification* is relevant to both people and computers. A goal of the language design is to provide a means for discussing a segment oriented approach among people in the protein research community. Additionally, the language serves as a description for computer aided analysis of protein sequences.

mate a piece of regular secondary structure. Each type of segment has a potential correspondence to some form of protein substructure. Each definition below will include that potential correspondence.

A *segment* is an arbitrary subdivision of a residue sequence. A *block* is the basic type of ALPPS segment. It is anticipated that a block generally will contain at most one piece of secondary structure, but this is not necessarily the case. A preliminary partitioning of a residue sequence into blocks, may produce blocks which ultimately should be split or merged in order to better meet the goal of one piece of secondary structure per block. A *frame* is a set of contiguous blocks.

A *region* is a segment, fully contained in a block, which may actually approximate a protein substructure. Each block can contain at most one region. A *target* - e.g., "helix" - is attached to each region definition. It is possible that this target merely eliminates some substructure - e.g., "non-turn." Region targets are used to develop a detailed secondary structure prediction.

Blocks and regions can be *hidden*. This means that they will not be considered for further processing until they are *exposed*. Initially, all blocks are exposed, but there may be a reason to hide certain blocks. For example, regions which emit strong "structural signals" can be masked to allow the reception of weaker signals from other blocks.

A final construction in the ALPPS hierarchy, *frame*, is discussed later in the section on future work. The hierarchy is:

- **frame-** a set of contiguous blocks
- **block-** the basic ALPPS segment
- **region-** the part of a block which may approximate a protein substructure

3.2.1. ALPPS Language Description

ALPPS is written in Common Lisp (Steele, 1984) and the notation has a Lisp feel. The Common Lisp use of keyword (rather than positional) arguments has been adopted. Anything written after a semi-colon is a comment.

The specification of residue level patterns is done with PLANS (Abarbanel, 1984), a pattern matching language that was developed and used successfully for identifying turns by this research group. PLANS patterns are used to initially partition a sequence into blocks, and can be used to characterize blocks. Regions can also be bracketed by PLANS patterns.

An ALPPS pattern may look like this:

```
(def-alpps example-name (:pat "TU")
  (hide-all-blocks) ; this is a comment
  (expose-blocks :pat "ch-pr"))
```

A Common Lisp macro, *def-alpps*, is used to define an ALPPS pattern. In this example **example-name** is the name of the ALPPS pattern. It is originally based on the PLANS pattern, "TU". After the two working block-lists (a "full block-list" and a "visible block-list") are created, **hide-all-blocks** will remove all blocks from the visible block-list. The full block-list holds all blocks, visible or hidden. The **expose-blocks** function will supplement the visible block list with any hidden blocks which contain the "ch-pr" PLANS pattern. A more detailed pattern with an explanation of ALPPS processing will be given later.

Sample Sequence: 4321B2468C8642D13579E97531

Goal: Letters divide this alphanumeric sequence into *blocks*. *Blocks* with a subsequence dense with odd numbers are worth exploring. Within these *blocks*, we are interested in *regions* which begin with a low number and end with a high number.

PLANS Patterns		
Name	Pattern	Symbol Comment
<i>alpha</i>	[A-Z]	^ any upper case letter, A through Z
<i>odd</i>	[13579]	
<i>high</i>	[5-9]	\$
<i>low</i>	[0-4]	>
<i>manyodd</i>	(density(>=,3,4,odd))	% at least 3 of 4 residues are <i>odd</i>

Step	Visible Sequence and Symbols	Comment
(def-alpps example (:pat "alpha") (hide-all-blocks) (expose-blocks :pat "manyodd") (make-regions :start-pat "low" :end-pat "high" :symbol "[")	4321B2468C8642D13579E97531	Original Sequence <i>alpha</i> markings
	B2468C D13579E 4321B C8642D E97531	Visible Blocks
		All Blocks are hidden
	D13579E E97531	Blocks with <i>manyodd</i> pattern are exposed
	D13579E E97531 >>\$\$\$ \$\$\$>>>	<i>low</i> and <i>high</i> markings on visible Blocks
D13579E E97531 [[[[[final visible Blocks and Region	

Figure 3-1: An Example of ALPPS Processing

3.2.2. An Example of ALPPS Processing

Figure 3-1 is an example of ALPPS processing on an arbitrary sequence. This sequence is used to focus on the processing and not on any particular protein substructure.

ALPPS begins with a partitioning of the sequence into blocks. A PLANS pattern (*alpha*) marks the block boundaries. At this point, blocks are visible. Since the PLANS pattern *anything*, can be found in all blocks, the visible block list is empty after the *hide-blocks* function. Blocks of potential interest are then revealed by the *expose-blocks* function. Finally regions are defined within these visible blocks.

3.2.3. PLANS Patterns and Segment Boundaries

Having looked at an example, a more detailed discussion of ALPPS will examine each of the functions and some of the issues involved in providing options for the functions. How should segments be defined with respect to PLANS matches? Should the sequences fed to PLANS extend beyond a segment boundary? These related questions need to be considered in defining the semantics of ALPPS.

The initial use of PLANS in an ALPPS procedure comes during the partitioning of a sequence into blocks. One option in this partitioning is a tolerance, the number of residues on either side of the PLANS mark which go with the opposite block. The tolerance is an overlap. For example (using PLANS patterns on Figure 3-1) with this sequence:

4321B2468C8653D13579E97531

```
(def-alpps no-tol (:pat "alpha"))
  yields
    B2468C      D13579E
4321B      C8653D      E97531.
```

While

```
(def-alpps tol-1 (:pat "alpha" :tol 1))
  yields
    1B2468C8    3D13579E9
4321B2    8C8653D1    9E97531.
```

One reason for the tolerance is that block boundaries, like the boundaries of protein secondary structure, may be fuzzy. Predicted turns may actually fall within a couple of residues of the beginning or end of a piece of secondary structure. Refinement of the boundary may come later. The current implementation of ALPPS has a minimum block overlap of one residue.

The second question is not as straightforward. PLANS patterns are used to characterize blocks and define regions within blocks. When applying PLANS patterns to a block, should residues which fall before or after a block boundary be examined? Consider the ALPPS procedures in Figure 3-2. Assuming that block boundaries cannot be crossed for purposes of evaluating PLANS patterns on blocks, three blocks are exposed in ALPPS procedure *tol-1* while only two are exposed in *no-tol*. The difference being that with the tolerated residues, the PLANS pattern is able to find more matches. In the third procedure, *no-tol-crossing*, the blocks do not include any tolerated residues, but because residues in adjacent blocks can be considered, the resulting exposed blocks are the same three as in *tol-1*.

Note that the PLANS match markings in *no-tol-crossing* are not identical to those in *tol-1*. This is important because it implies that the tolerance number alone would not be a way to control the boundary crossing. Moreover, one can imagine situations where

blocks should be defined with no tolerance (high confidence in turn placements) and **PLANS** patterns should look beyond block boundaries (amino terminus hydrophobic face **alignment**). In order to meet this need, the crossing flag (**:crossing**) can be turned on.

One final option for partitioning sequences into blocks is the minimum block size (**=min-size**). This number gives a minimum number of residues for each block.

<pre> (def-alpps tol-1 (:pat "alpha" :tol 1) (hide-all-blocks) (expose-blocks :pat "manyodd")) 3D13579E9 %%%%%%%%% 8C8653D1 9E97531. % %%%%%%%%% </pre>
<pre> (def-alpps no-tol (:pat "alpha") (hide-all-blocks) (expose-blocks :pat "manyodd")) D13579E %%%%%%%%% E97531. %%%%%%%%% </pre>
<pre> (def-alpps no-tol-crossing (:pat "alpha" :crossing t) (hide-all-blocks) (expose-blocks :pat "manyodd")) D13579E %%%%%%%%%%% C8653D E97531. %%%%%%%%% %%%%%%%%% </pre>

Figure 3-2: Boundary Crossings

3.2.4. Segment Manipulation Functions

The following ALPPS functions are used to manipulate the visible block-list.

```
(hide-blocks :pat pat
             :pat-count-min n
             :crossing bool
             :region-target target)
(hide-all-blocks)
(expose-blocks :pat pat
              :pat-count-min n
              :crossing bool
              :region-target target)
(expose-all-blocks)
```

A **hide-blocks** removes blocks from future consideration until they are retrieved by **expose-blocks**. The **expose-blocks** function appends blocks to the visible block list. **Blocks** can be hidden or exposed in two ways. If PLANS patterns are used, a minimum **pattern count** (**:pat-count-min**) and crossing flag (**:crossing**) - described in the previous **subsection** - are available to refine the specification. The two "all" variations, **hide-all-blocks** and **expose-all-blocks** are shorthand for specifying all blocks. An alternative **method** of specification involves using region targets (**:region-target**), a concept **described** in the subsection on regions.

```
(split-blocks :pat pat)
```

This function will create two blocks out of one visible block which matches *pat*. The **new** blocks are visible.

```
(cat-blocks :pat1 pat-1 :pat2 pat-2)
```

This function goes through the visible block-list and combines adjacent blocks that have *pat-1* in the left block, and *pat-2* in the right block.

Each of these functions has an additional set of options which can be used to manipulate the use of PLANS patterns to characterize segments. The minimum pattern count option (`:pat-count-min`) can be used to specify that more than one residue must be marked as matching the given PLANS pattern before ALPPS will act on the segment.

For example,

```

Sequence:
4321B2468C8653D13579E9842

(def-alpps no-tol (:pat "alpha")
  (hide-all-blocks)
  (expose-blocks :pat "odd"
    :pat-count-min 3)
  yields
      D13579E.
```

While

```

Sequence:
4321B2468C8653D13579E9842
(def-alpps no-tol (:pat "alpha")
  (hide-all-blocks)
  (expose-blocks :pat "odd"))
  yields
      D13579E
4321B      C8653D      E9842.
```

One example of using this option would be in characterizing segments with a PLANS pattern that tends to overpredict (false positives). By requiring multiple hits on the same segment overprediction may be minimized.

3.2.5. Region Definition Functions

```
(make-regions :start-pat pat
             :end-pat pat
             :mid-pat pat ;; optional
             :min-size n   ;; default 1
             :crossing bool ;; default false
             :color color  ;; optional
             :target target);; optional
```

This function walks through the visible block-list and builds regions in those blocks that meet the criteria but do not already contain a defined region. Regions begin with some starting pattern (:start-pat) and end with an ending pattern (:end-pat). A middle pattern (:mid-pat) is optional as is a minimum number of residues (:min-size) contained in a region. The crossing flag (:crossing) allows residues adjacent to the block boundaries to be considered in the application of the PLANS patterns. The color flag (:color) is used to generate label defined regions on PostScript output.

The target (:target) indicates the type of protein substructure which is suspected to be found by this region specification. Targets play two important roles in ALPPS to make a complete secondary structure prediction. They are used to make the final secondary structure prediction annotations on a residue basis. They can also be used to characterize blocks which work as a *frame*. Both of these roles will be discussed in more detail below.

3.2.6. ALPPS Interpretations

ALPPS can be used with the following strategy illustrated in Figures 3-3-3-5. Details on the patterns are given below in the section on α/α proteins. The PLANS pattern *TU*, which identifies turn residues, partitions a protein sequence into blocks. In Figure 3-3 ovals represent residues in a protein sequence. Those with solid black filling are

annotated by the PLANS pattern *TU* - marking turns. Each double-headed arrow represents the extent of a block. The target for a block is a segment which contains at most one piece of regular secondary structure (*i.e.*, a helix or strand). Figure 3-4 demonstrates the use of meta-patterns to focus on sub-structures. Finally, Figure 3-5 shows a complete residue-level annotation. Each residue is represented by a vertical pair of ovals with PLANS markings in the upper oval and secondary structure annotations below. In this way, ALPPS meta-patterns provide the ability to apply a hierarchical prediction framework on a sequence.

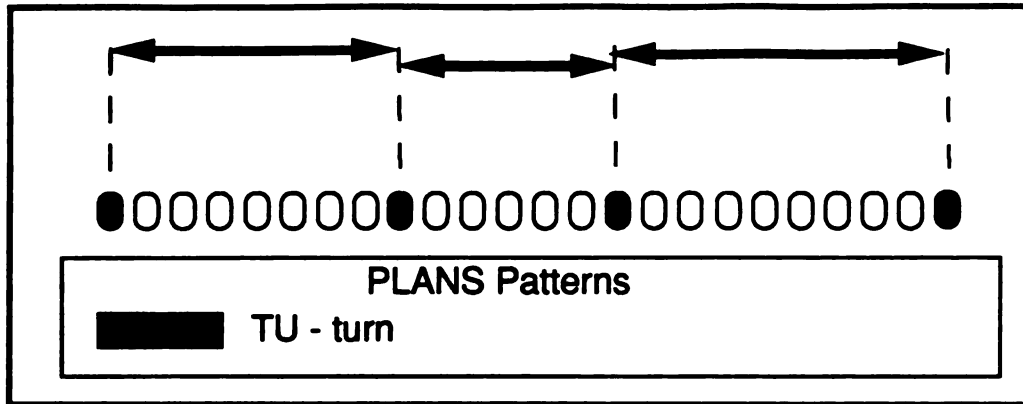


Figure 3-4: ALPPS Blocks

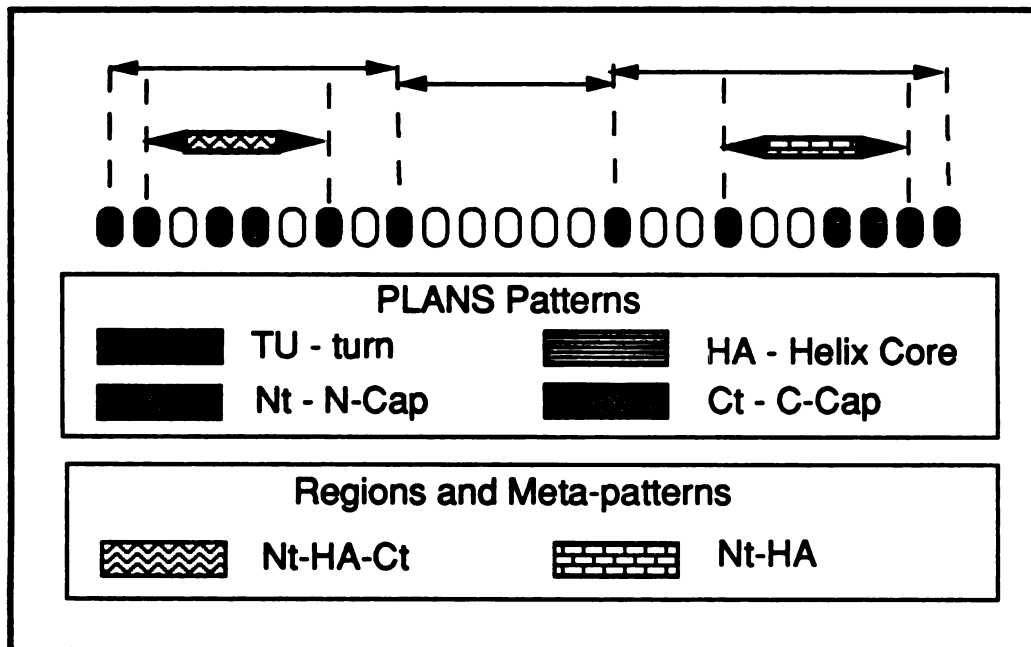


Figure 3-5: ALPPS Regions

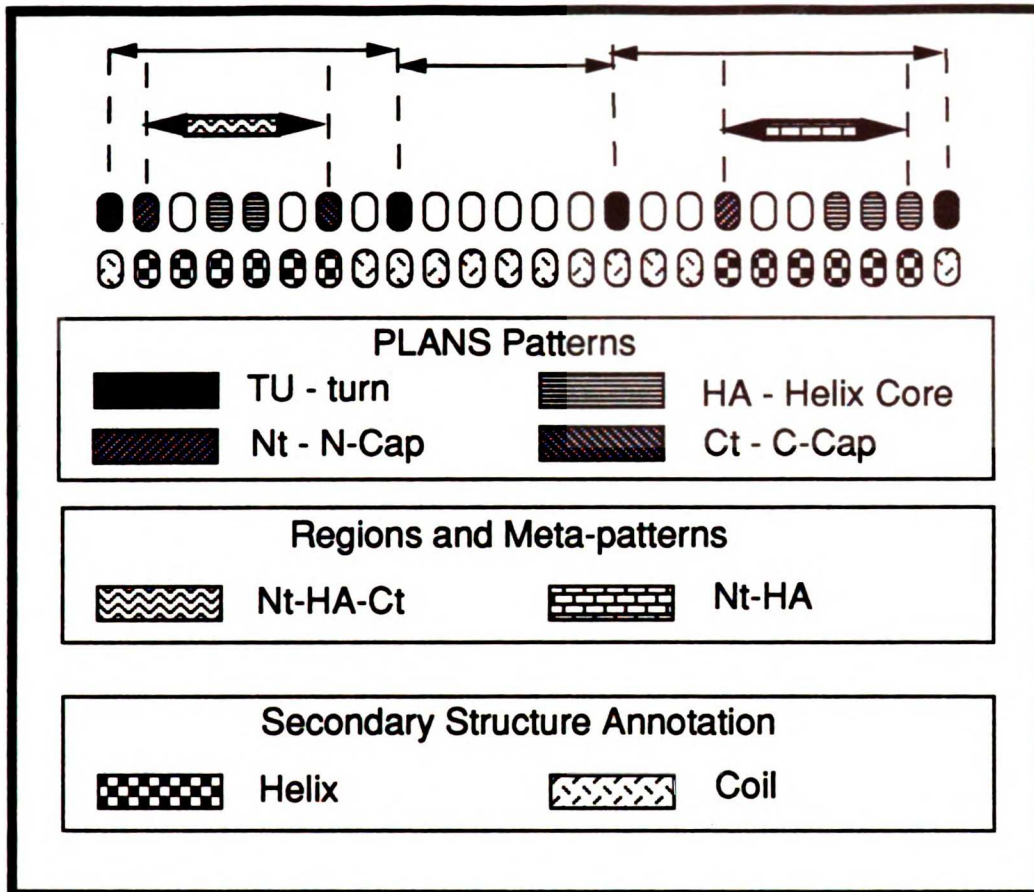


Figure 3-6: Secondary Structure Annotation

3.3. Discussion and Conclusion

3.3.1. Software Environment

ALPPS is currently running as part of a Lisp system called *Match-Set*. The user interface, *GNU Emacs* (Stallman, 1986), is valuable for the software developer and experienced users, but is difficult to learn for new users. An interactive, mouse and window based system called *Match-Point* was built as a prototype for developing PLANS patterns. There is a need to expand *Match-Point* from the prototype phase and introduce the ALPPS capabilities contained or planned for *Match-Set*.

3.3.2. Future Work

There are at least two areas of future development planned for ALPPS itself — frames and interpretations.

Making preliminary assignments within some segments can influence assignments in other segments. For example, in an α/β barrel, α and β structure generally are found in pairs. The concept of a *frame* needs to be more fully developed. A frame of four blocks showing α - β -?- β would lead to an expectation of finding a region with an α characterization in the middle block. In this case, weaker helical signals might be accepted. This type of computer-based reasoning is found in the use of "scripts" in natural language processing (Allen, 1987).

The interpretation of a ALPPS hierarchy — region, block, frame — shown in Figures 3-3-3-5 works well for annotating helices in α/α proteins. Other interpretations of the hierarchy may be better for different tasks. For example, rather than using turn markings to partition a sequence into blocks, one could use helical core markings. Under this

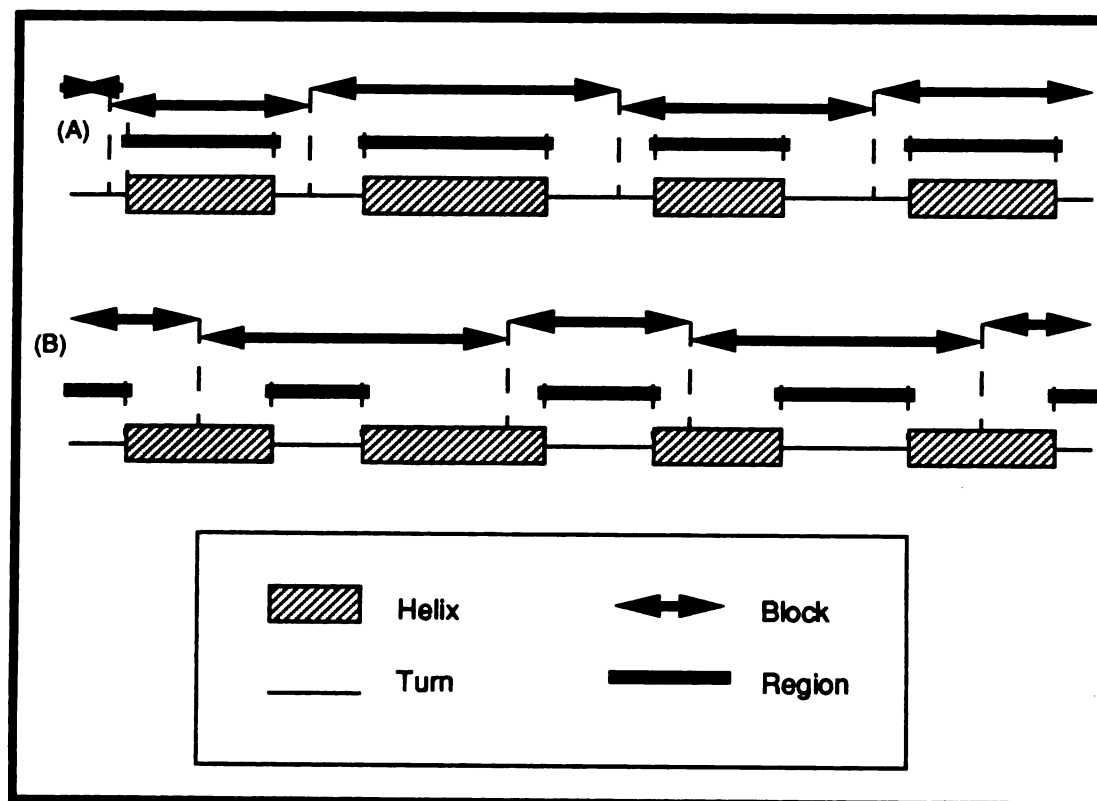


Figure 3-6: Hierarchy Interpretations

interpretation, turns might be the targets for regions. Figure 3-6 shows two interpretations which ultimately yield the same annotation. Interpretation (A) is the one described in Figures 3-3-3-5, while (B) is described in this paragraph.

Chapter 4

Scoring Secondary Structure Predictions on α/α Proteins

4.1. Introduction

Both users and developers of secondary structure prediction techniques need methods for evaluating the quality of a given secondary structure prediction. For users, this type of information is important for making a decision on which secondary structure prediction technique to use and how much confidence should be placed on a prediction. For developers, quality evaluations provide both a means for comparing different secondary structure prediction methods and a means for giving constructive feedback for improvement. This last point is especially important to methods based on machine learning. The general idea behind any evaluation or scoring scheme is to compare a predicted secondary structure to an observed secondary structure based on the known tertiary structure of a given protein.

After briefly reviewing the literature with respect to evaluating secondary structure predictions, this chapter looks at scoring secondary structure predictions on a set of 20 α/α proteins. Predicted helices are collected from four methods (Chou-Fasman; Garnier-Osguthorpe-Robson; neural nets; ALPPS). Three different secondary structure assignments (Kabsch-Sander; Richards-Kundrot; PDB) are used for determining the observed helices. In addition to comparing each of the four predictions to each of the three assignments for all 20 proteins, the secondary structure assignments themselves are compared to each other for a better understanding of how the assignments differ. A variation on residue based scoring which discounts residues at or near the terminals of helices (*trimming*) is introduced. The results of three scoring methods are reported. These

results focus on the relationship between the scores and the concept that predictions should be approximations to the observed secondary structures. One conclusion, that there is a need for methods of evaluation which supplement residue-based scoring, is the subject of the next chapter.

4.1.1. A Brief History of Evaluating Secondary Structure Predictions

Guzzo (1965) appears to be the first investigator to publish a secondary structure prediction. After looking for sequence-structure correlations (helical or non-helical) in the known structures of myoglobin and hemoglobin, Guzzo makes a prediction of the secondary structure of lysozyme. The evaluation of the prediction is basically descriptive. In addition to a table listing the beginning and ending residues of each predicted and observed helix, there is a figure similar to the feature diagrams (*e.g.*, Figure 4-1) used in this chapter. With only one prediction on one structure and no other predictions in the literature at that time, Guzzo's descriptive evaluation is a reasonable approach.

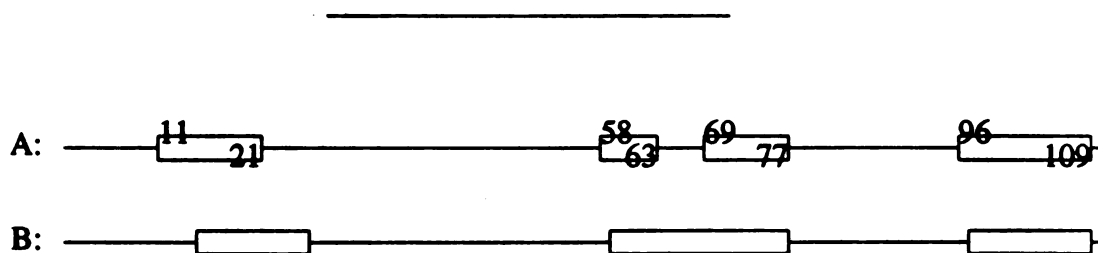


Figure 4-1: Feature Diagrams

This figure shows two sample *feature diagrams*. These two state feature diagrams use boxes to represent helices and lines to represent turns (non-helices). In diagram A, the number in the upper left hand corner of a given box is the position of the first residue (N-cap) of the represented helix. Similarly, the number in the lower right hand corner is the position of the last residue (C-cap). Diagram B does not include explicit capping positions, but both diagrams are drawn to scale.

By 1971, several secondary structure prediction methods were brewing. In the paper detailing their structure for cytochrome c, Dickerson *et al.* (1971) discuss about ten methods for predicting helices applied to the cytochrome c sequence. A table shows three observed helices (by residues numbers) and lists the predicted helices of 5 methods. The discussion is qualitative (*e.g.*, "a tendency to predict more helix than is actually present"). Similarly, just before publishing his structure for adenyl kinase, Schulz sent out the sequence to several groups working on secondary structure prediction. In a companion article in *Nature*, ten prediction methods are reported and compared (Schulz *et al.*, 1974). Here, Schulz includes residue counts of *predicted right* (true positives), *predicted wrong* (false positives), and *not predicted*. No attempt is made to summarize these residue tallies. The discussion includes counts of correctly identified helices.

“Two groups found nine, two groups found eight, and one group found seven out of the 10 helices. Only one of these five groups predicted a wrong piece of chain as being helical. The results in Fig. 1 indicate, however, that it is still difficult to find the correct starting and termination point of a helix.”

Shortly after the publication of Dickerson's paper on cytochrome c, Lewis and Scheraga (1971) revise their amino acid "helix-forming categories" secondary structure prediction method and apply it to 11 proteins of known structure. In doing so, they introduce a pair of quality indices which summarize residue tallies — the *overall percentage of correctly predicted conformations* and the *percentage of correctly predicted helical residues*. They note that "it is essential to consider both measures of correctness" in order to avoid the perfect score (*percentage of correctly predicted helical residues*) available by simply guessing that an entire protein is helical. This problem of dismissing random guessing is discussed by Matthews (1975) in his report comparing over 10 secondary structure prediction methods on T4 lysozyme. He devotes significant space to using

the correlation coefficient for summarizing the four tallies. Although Matthews only looked at lysozyme, he does note that the lysozyme predictions are much less successful than those presented in Schultz' paper on adenylate kinase a year earlier.

Argos *et al.* (1976) use and compare five secondary structure prediction methods on 40 known structures. This comparison would move away from the possible bias of a method doing particularly well on one structure, but not so well in general. Kabsch and Sander (1983) evaluate three (then) widely used methods on 62 proteins of known structure. They state that one important feature of their methodology is the use of an algorithm to achieve "objective and accurate assignment of secondary structure." Their algorithm (Kabsch and Sander, 1983a) removes the subjective variations caused by using the depositors' (crystallographers) secondary structure assignments found in the Brookhaven Protein Databank (PDB). Although the algorithm is *objective*, this chapter questions the appropriateness of the *accurate* billing.

4.2. Methods & Theory

4.2.1. Standard Residue-Based Scoring

Predictions are compared to "observed" secondary structures of proteins whose 3-dimensional structure is known. The standard method makes comparisons on residue-by-residue basis (Schulz and Schirmer, 1979). Four totals — true positives, true negatives, false positives, false negatives — are tallied for each type of predicted structure type — *e.g.*, helix. In the example below, τ represents a turn residue while A represents an α -helix residue.

observation: tttAAAAAAAAAAAAAtttt
 prediction: ttAAAAAAttAAAAAAtt

Here, the tallies are:

True Positives (<i>P</i>):	10
True Negatives (<i>N</i>):	4
False Positives (<i>p</i>):	3
False Negatives (<i>n</i>):	2

These four tallies can be summarized by a single number resulting from some arithmetic combination (see review in Chapter 6 of Schulz and Schirmer, 1979). While there are many possible quality indices, including the seven in Schulz and Schirmer, this chapter will only look at the two which are widely used in recent work.

4.2.1.1. *Q* — the Fraction of Correctly Predicted Residues

A commonly used quality index is the fraction of correctly predicted residues, which Schultz and Schirmer call Q_3 while others simply call it Q .

$$Q = \frac{\text{TruePositives} + \text{TrueNegatives}}{\text{TotalNumberResidues}}$$

or

$$Q = \frac{P+N}{P+N+p+n} \quad (1)$$

This is the fraction of residues which are correctly predicted. A range of 0 to 100 percent can be obtained when Q is converted to a percentage. Although other quality indices are available, Q will be sufficient to examine issues involved in defining the known secondary structure of a protein.

4.2.1.2. The Correlation Coefficient as a Scoring Measure

Another popular quality index is the correlation coefficient, C (Q_7 in Shultz and Schirmer):

$$C = \frac{(P \times N) - (p \times n)}{\sqrt{(N+p)(N+n)(P+p)(P+n)}} \quad (2)$$

The correlation coefficient indicates how a given prediction differs from a random guess.

The range of C is -1 to 1. A score of 0 says that the prediction is essentially the same as a random guess. Negative scores show a high proportion of false predictions while positive scores show a better proportion of true predictions. Matthews (1975) is credited with bringing the measure to secondary structure prediction, though the correlation coefficient itself, has been known to statisticians for many years.

A potential problem with the correlation coefficient is the fact that it gives no credit for correctly predicting the ratio of helix to coil residues. A random prediction can be based on any preselected ratio of helix to coil residues. Consider the following:

Let R be number of helical residues and g , the predicted number of helical residues.

$T = P + N + p + n$, is the number of residues in the sequence. Then the expected values

for the four residue tallies are:

$$P = R \times \frac{g}{T}$$

$$p = (T - R) \times \frac{g}{T}$$

$$N = (T - R) \times \frac{(T - g)}{T}$$

$$n = R \times \frac{(T - g)}{T}$$

In the cases where all assigned residues are [not] helical and all predicted residues are [not] non-helical (*i.e.*, $R=T$ and $R=0$) the denominator is zero, and C is not

defined. Otherwise, the denominator of C is always non-zero, so only the numerator needs to be evaluated.

$$P \times N = \frac{R \times g \times (T-R) \times (T-g)}{T^2}$$

$$p \times n = \frac{(T-R) \times g \times R \times (T-g)}{T^2}$$

$$= \frac{R \times g \times (T-R) \times (T-g)}{T^2}$$

$$P \times N - p \times n = 0.$$

Therefore the expected value of C is zero even if $g=H$.

4.2.2. Adjusting for Inexact Capping: Trimming

The difference between secondary structure assignment methods might relate to the difficulty in assigning the terminals of helices. Though the cores of helices are generally easy to assign, the ends — especially the exact capping residues are not.⁶ Figure 4-2 gives a notation system for discussing this issue.

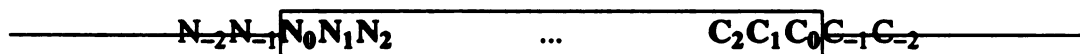


Figure 4-2: Capping Notation

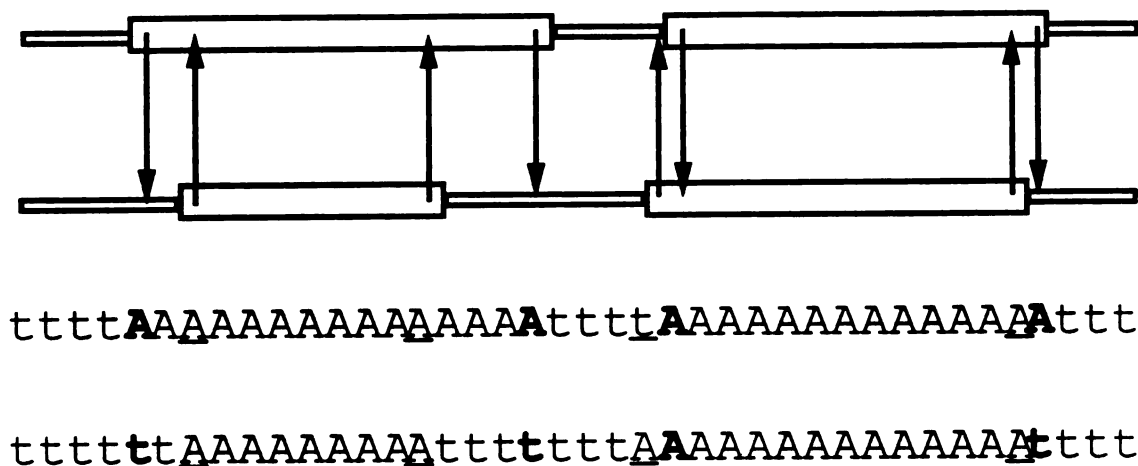
In this notation, the capping residue is noted by a zero subscript (*i.e.*, N_0 for the first residue and C_0 for the last). The positive subscripts on helical residues count the sequential distance from a given cap. Negative are used to mark residues which are close to a capping residue, but outside a given helix. In this example, the helical residues are in bold type.

Noting that the N_0 and C_0 residues are difficult to assign, it is reasonable to assume that these residues are also difficult to predict. In tallying scoring counts, these residues might be given discounted weights or even excluded from the tallies altogether. Given

⁶ A demonstration of this assertion is provided in the results section of this chapter.

that a secondary structure prediction provides a "low resolution" view of the protein's structure, an exact residue-by-residue result is not always required.

The *trimming* technique removes the residues which cap helices in either the assigned or predicted secondary structures. Figure 4-3 demonstrates the technique.



Method	TP	TN	FP	FN	Q
Standard	22	10	1	7	.80
Trimmed	18	10	0	4	.88

Figure 4-3: Sample of *Trimming*

This figure demonstrates trimming and shows the resulting changes in the scoring tallies and one representative (Q) quality index. In the feature diagrams, lines represent turns while boxes represent helices. The first feature diagram represents an observed secondary structure assignment while the second is a prediction. The rows of character strings made up of A and t are alternative representations of the assignment and prediction. Residues which are subject to trimming are shown in **bold** when they cap the assigned helices and by an underline when they cap the predicted helices. The trimming technique removes both types of capping residues from consideration in the tallies.

The examples in Figure 4-4 show how residue-based scoring is affected by trimming. Trimmed scores (in these examples, Q) can be higher, lower, or unchanged

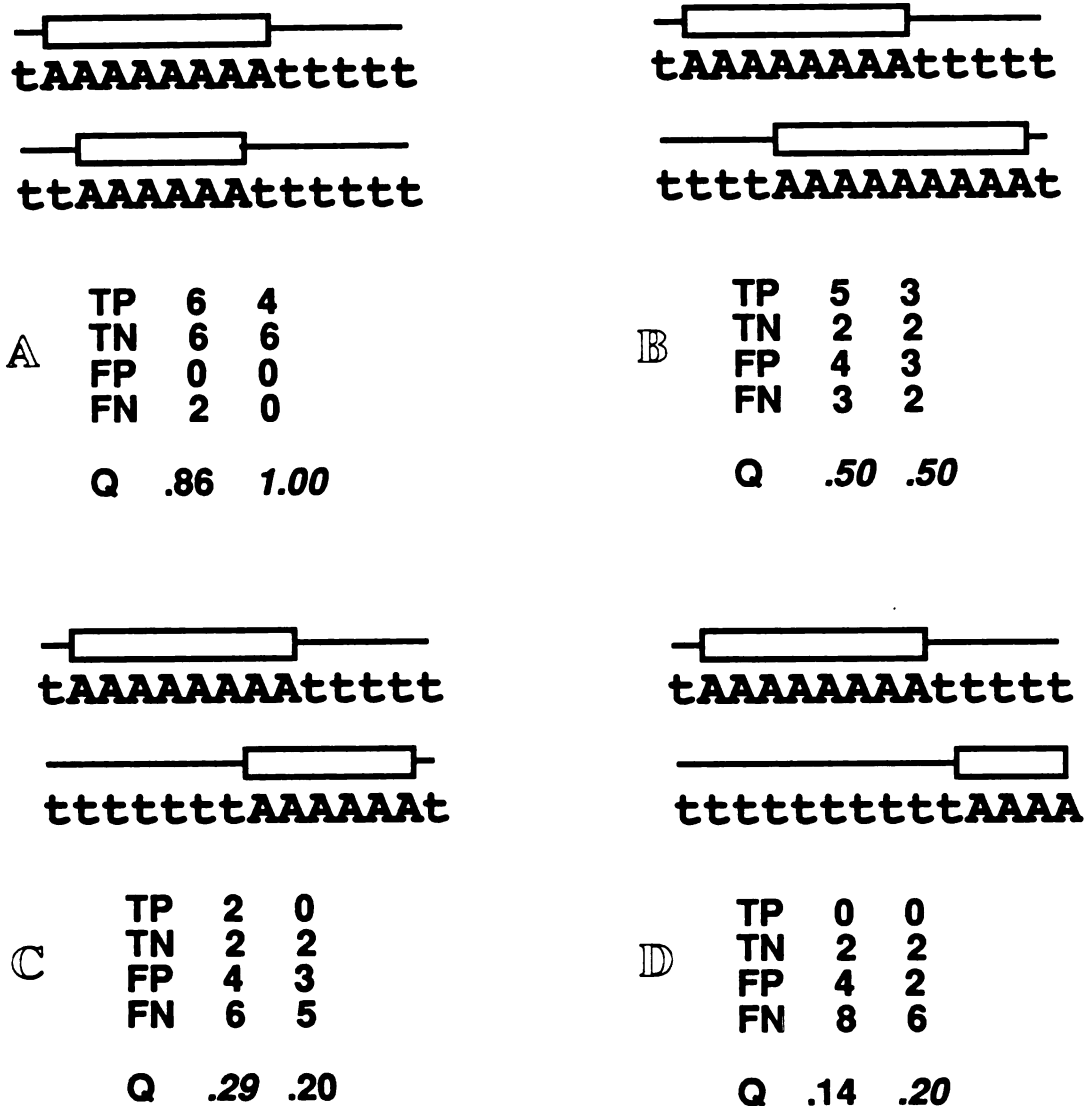


Figure 4-4: Trimming Examples

Four situations are shown to demonstrate the effect of not counting capping residues in tallies. In each example, the bottom feature diagram represents a prediction while the top feature diagram represents the observed structure. True positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) are given as both standard tallies (first column) and trimmed tallies (second column). Note that in example D, the trimmed tallies do not reflect the pairing requirement which is described in the text. Example D is the motivation for the pairing requirement.

depending on the relationship between the observed and predicted helices. The difference between the standard Q score and the trimmed Q score (ΔQ) is positive when the predicted helix comes close to matching the assigned helix. (See Figure 4-4 A.) A perfect Q score (1.00) is unchanged by a trimming adjustment. A moderate match (Figure 4-4 B) results in little or no change, while a poor match (*e.g.*, a one residue overlap as seen in Figure 4-4 C) results in a negative ΔQ . In the first three examples, the trimmed scores preserve and perhaps enhance the relative ordering of predictions. Unfortunately, the example D of Figure 4-4 shows a mismatch where ΔQ is positive. This suggests an additional rule for trimming. Capping residues should be eliminated from scoring consideration only after the sets of observed and predicted helices are paired with a minimum of 1 overlapping residue. This pairing is the same as the pairing used in Taylor's structure percentage scoring (discussed below).

The trimming algorithm can also include a level of tolerance. For example, residues at positions N_{-1} , N_0 , N_1 , C_1 , C_0 , and C_{-1} of both the predicted and observed helices would be removed from consideration. Again, non-overlapping assigned and predicted helices would result in unwarranted score improvements if a minimum overlap rule is not included. This form is called *extended trimming* or *trimming \pm* .

Various methods could be used to discount the predictions at the end of helices and focus on the helix cores. For example, some function could be used to weight the tallies from various residues around the N_0 and C_0 residues of both the observed and predicted helices. This type of weighing was implemented by my colleague, Don Morris (Morris, 1990). The results appeared to be no better than the results of the more simplistic trimming technique. Trimming offers a way of emphasizing the correct prediction of well

established helical residues in an easy and understandable manner.

4.2.3. Observed Secondary Structure of Proteins with Known Structures

Chapter 2 notes that assigning secondary structure to proteins of known tertiary structure is not an easy task. Expert assignments are subjective and may differ between experts. Algorithmic assignments do not always produce an assignment which is consistent with a consensus of subjective expert assignments.

One of the purposes of this chapter is to give a better portrait of the actual secondary structure assignment differences on α/α proteins. For α/α proteins the polypeptide chain we consider only two conformational states: helix and turn. All forms of helical structure (3_{10} , α , and π helix) are treated identically. Similarly, any region that interconnects regular secondary structure is considered a turn. Three assignment sets are used this chapter. The Brookhaven Protein Databank (PDB) (Bernstein *et al.*, 1977) assignments are generally available and serve as an example of subjective assignments. The method devised by Kabsch & Sander (1983) is widely used. The assignments produced by the more recent Richards & Kundrot (1988) algorithm appear to be closer to subjective assessments of helical structure, while retaining the consistency of an objective algorithm.

4.2.4. Data Set of 20 α/α Proteins

The 20 α/α proteins used in this scoring comparison are listed in Table IV-1. These are the same 20 proteins which form the test and development sets described in Chapter 6.

PDB	Protein	
156b	Cytochrome B562	(Lederer <i>et al.</i> , 1981)
1cc5	Cytochrome C5	(Carter <i>et al.</i> , 1985)
1ccr	Cytochrome C	(Ochi <i>et al.</i> , 1983)
1ecd	Erythrocrurin	(Steigemann and Weber, 1979)
1fdh	Human Fetal Hemoglobin (gamma chain)	(Frier and Perutz, 1977)
1hmq	Hemerythrin	(Stenkamp, Sieker, and Jensen, 1983)
1mbd	Myoglobin	(Phillips and Schoenborn, 1981)
2ccy	Cytochrome C prime	(Finzel <i>et al.</i> , 1985)
2cts	Citrate Synthase	(Remington, Wiegand, and Huber, 1982)
2cyp	Cytochrome c Peroxidase	(Finzel, Poulos, and Kraut, 1984)
2lh1	Leghemoglobin	(Arutyunyan <i>et al.</i> , 1980)
2lhb	Hemoglobin V	(Honzatko, Hendrickson, and Love, 1985)
2lzm	T4 Lysozyme	(Matthews, 1975)
2tmv	Tobacco Mosaic Virus Coat Protein	(Namba, Pattanayek, and Stubbs, 1989)
3c2c	Cytochrome C2	(Bhatia, 1981)
3cln	Calmodulin	(Babu, Bugg, and Cook, 1988)
3cpv	Parvalbumin B	(Moews and Kretsinger, 1975)
3hhb	Human Hemoglobin (alpha chain)	(Fermi <i>et al.</i> , 1984)
3icb	Vitamin D-dependent Calcium-binding Protein	(Szebenyi and Moffat, 1986)
3wrp	Trp Aporepressor	(Lawson <i>et al.</i> , 1988)

4.2.5. Prediction Methods

Four prediction methods — Chou-Fasman, GOR, neural nets, and ALPPS — are used. Chou-Fasman (1978) [CF] and Garnier, Osguthorpe, & Robson (1978) [GOR] were selected because they are well known and readily available on many computers. Neural nets [NN] are a recent tool for secondary structure prediction. The network and weights⁷ developed by Kneller, Cohen, and Langridge (1990) take advantage of training only on α/α proteins. ALPPS is the segment based method which we are currently developing.

⁷ When a protein was in the original neural net protein set, a "jackknife" weight set developed by excluding the given protein from the training set is available. All but 156b, 2cts, 2tmv, 3cln, and 3wrp were in the original set of proteins used in the neural network. Surprisingly, the neural net aggregate scores for these proteins exceeds the aggregate scores for the other 15.

4.2.6. Aggregate Scores and Summary Statistics

An *aggregate* score can be calculated for any residue based measure by taking the four tallies — true positives, true negatives, false positives, false negatives — over the set of 20 proteins. This essentially treats the 20 proteins as one long sequence. Summary statistics can be obtained by treating each the score on each of the 20 proteins as one entry. Not surprisingly, the aggregate score will not always be the same as the mean score since the scores on longer sequences get more weight in the aggregate score.

4.3. Results

For each of the twenty α/α proteins listed in Table IV-1, predictions are generated based on four methods (Chou-Fasman, GOR, neural nets, and ALPPS) and compared against helix assignments from three methods (Kabsch-Sander, Richards-Kundrot, and PDB). Comparisons are also made between pairs of assignment methods. The comparisons are scored by Q and C quality indices calculated on standard, trimmed, and trimmed \pm residue tallies.

The main set of results is found in the twenty pages of Figure 4-5. Each page contains 7 feature diagrams — one for each assignment or prediction — and a table containing the described Q and C scores.

Table IV-2 with the same format as each of the 20 tables in Figure 4-5 gives results based on aggregating the 20 sequences. The concept of trimming appears to be validated by the aggregate results shown in the last 3 columns. Table IV-3 gives summary statistics — mean, median, standard deviation, worst, best, and range — on each set of Q

scores on the three significant assignment-assignment pairings.⁸ Three assignment-assignment pairings and three tallying methods yield 9 sets of 20 (one for each protein) Q scores. The ranking of proteins by Q score within each set is shown in Tables IV-4, IV-6, and IV-6 for standard, trimmed, and trimmed \pm tallies.

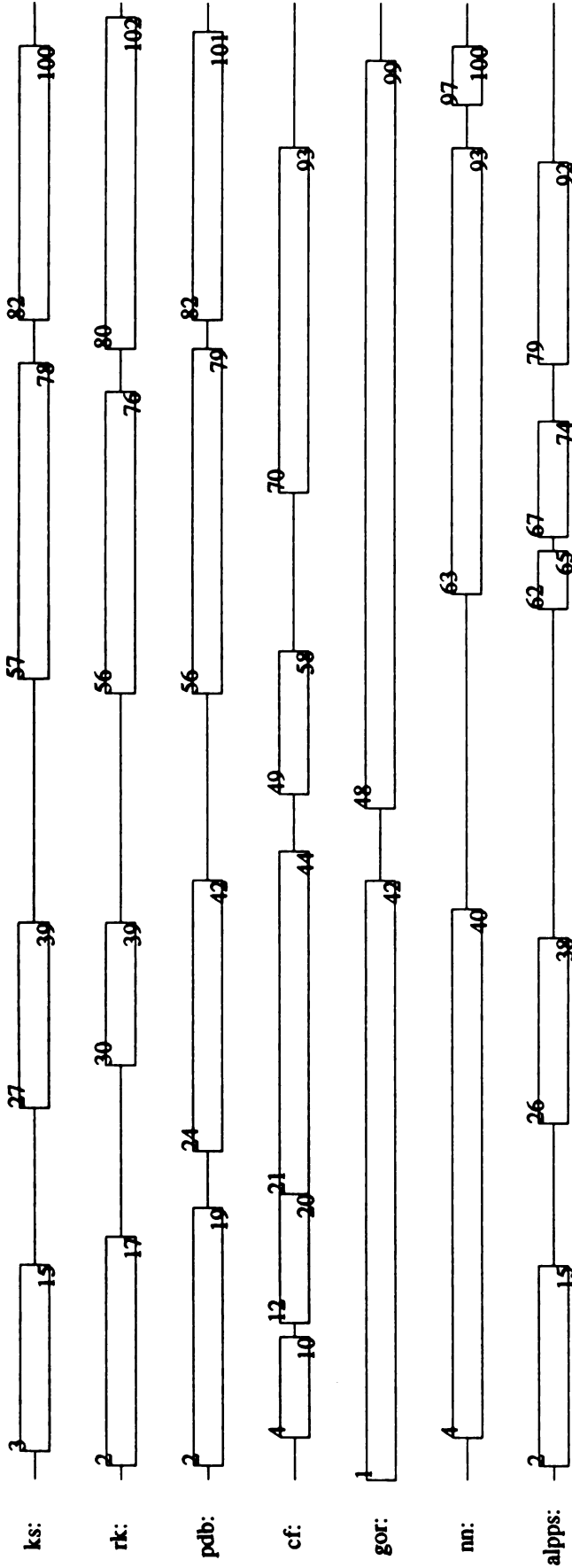
Similarly Table IV-7 gives summary statistics for the 12 prediction-assignment pairing sets and Tables IV-9 through IV-11 show the ranking of proteins by Q score withing each set.

Figure 4-5: Individual Sequence Results

The next 20 pages contain the results of running four secondary structure prediction methods and scoring the predictions against the secondary structure assignments based on three different methods. Each page contains the results for one of the 20 proteins. The assignments and predictions are displayed as feature diagrams. A table shows Q and C values generated from the 21 paired comparisons of a predicted or assigned secondary structure with an assigned secondary structure. Standard, trimmed, and trimmed \pm tallying methods are used and listed separately. The proteins are identified by the Protein Databank name given in Table IV-1. The prediction methods are Chou-Fasman (cf), Garnier-Osguthorpe-Robson (gor), neural nets (nn), and segment based (alpps). The assignment methods are Kabsch-Sander (ks), Richards-Kundrot (rk), and Protein Databank HELIX records (pdb).

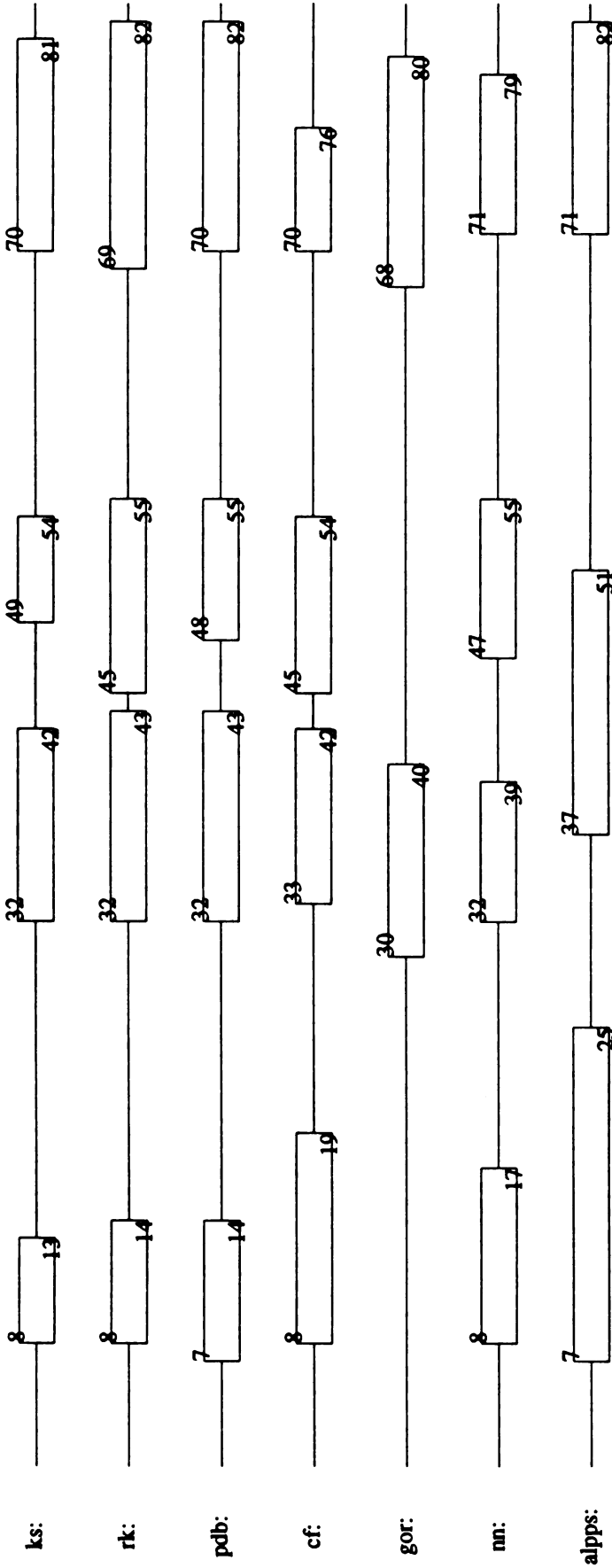
⁸ For convenience, the scoring tables of Figure 4-5 show 21 pairings even though 6 are redundant. Twelve come from pairing each prediction with each assignment. An additional three come from using an assignment as a prediction for scoring purposes. The symmetry of true and false tallies in both Q and C lead make predicted tallied against observed the same as observed tallied against predicted. Obviously an assignment scored against itself gives Q and C of 1.

Secondary Structure Assignments for 156b:



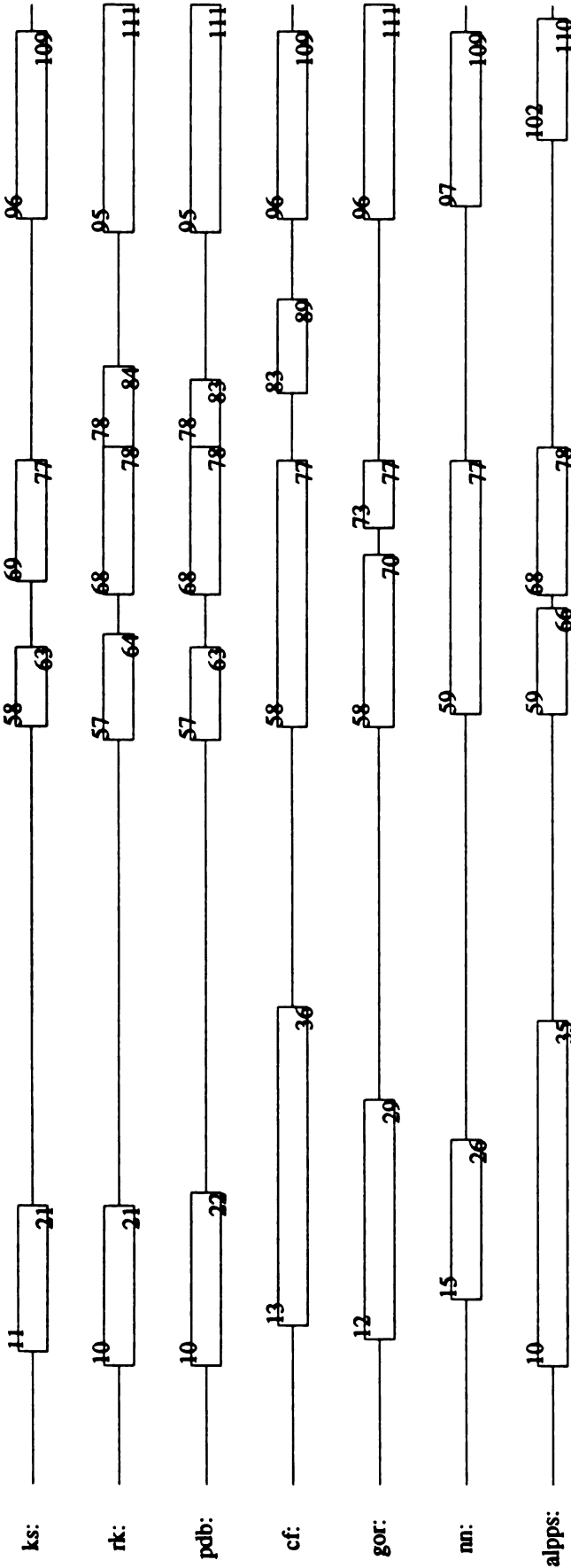
Observed	Prediction		Residue Based Scores- Q (C)							ks	rk	pdb
	Method	cf	gor	nn	alpps	ks	rk	pdb				
ks	standard	.54 (-.05)	.72 (.35)	.76 (.45)	.77 (.55)	1.00 (1.00)	.87 (.72)	.86 (.71)				
	trimmed	.52 (-.06)	.72 (.41)	.76 (.47)	.80 (.64)	1.00 (1.00)	.93 (.85)	.92 (.82)				
	trimmed±	.54 (-.06)	.78 (.42)	.79 (.56)	.87 (.77)	1.00 (1.00)	1.00 (1.00)	.98 (.96)				
rk	standard	.52 (-.15)	.71 (.23)	.71 (.32)	.74 (.50)	.87 (.72)	1.00 (1.00)	.83 (.61)				
	trimmed	.48 (-.17)	.71 (.28)	.71 (.35)	.77 (.57)	.93 (.85)	1.00 (1.00)	.88 (.72)				
	trimmed±	.49 (-.19)	.78 (.39)	.76 (.47)	.82 (.67)	1.00 (1.00)	1.00 (1.00)	.96 (.88)				
pdb	standard	.64 (.04)	.83 (.43)	.80 (.48)	.69 (.44)	.86 (.71)	.83 (.61)	1.00 (1.00)				
	trimmed	.64 (.08)	.85 (.49)	.81 (.53)	.69 (.46)	.92 (.82)	.88 (.72)	1.00 (1.00)				
	trimmed±	.68 (.11)	.91 (.57)	.89 (.71)	.77 (.60)	.98 (.96)	.96 (.88)	1.00 (1.00)				

Secondary Structure Assignments for 1cc5:



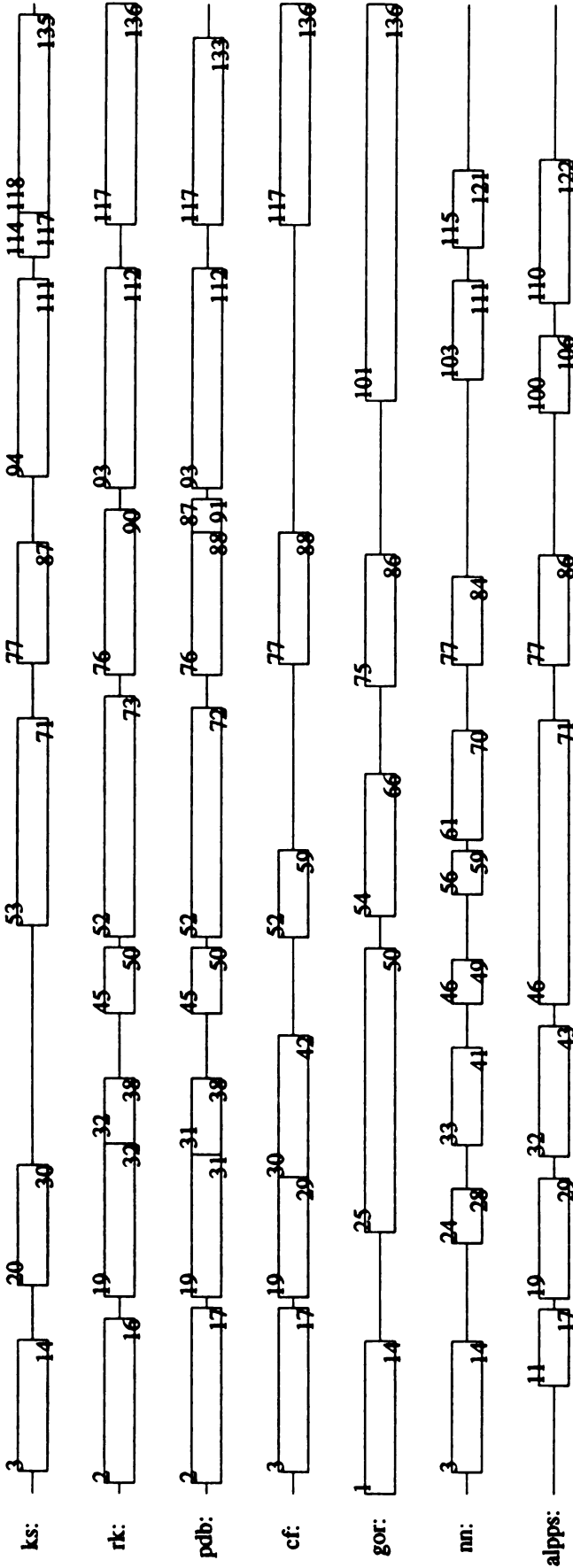
		Residue Based Scores- Q (C)									
Observed	Prediction	cf	gor	nn	alpps	ks	rk	pd			
	Method										
ks	standard	.81 (.61)	.77 (.53)	.84 (.68)	.65 (.32)	1.00 (1.00)	.89 (.80)	.93 (.86)			
	trimmed	.83 (.65)	.80 (.58)	.90 (.78)	.67 (.35)	1.00 (1.00)	.96 (.91)	1.00 (1.00)			
	trimmed±	.88 (.72)	.81 (.62)	.98 (.95)	.69 (.43)	1.00 (1.00)	.98 (.96)	1.00 (1.00)			
rk	standard	.82 (.65)	.69 (.44)	.83 (.68)	.71 (.42)	.89 (.80)	1.00 (1.00)	.94 (.88)			
	trimmed	.87 (.74)	.71 (.49)	.87 (.75)	.73 (.46)	.96 (.91)	1.00 (1.00)	.97 (.95)			
	trimmed±	.90 (.80)	.72 (.52)	.98 (.96)	.78 (.56)	.98 (.96)	1.00 (1.00)	1.00 (1.00)			
pdb	standard	.78 (.57)	.70 (.43)	.84 (.69)	.70 (.40)	.93 (.86)	.94 (.88)	1.00 (1.00)			
	trimmed	.84 (.67)	.72 (.48)	.90 (.79)	.70 (.42)	1.00 (1.00)	.97 (.95)	1.00 (1.00)			
	trimmed±	.90 (.78)	.75 (.54)	.98 (.95)	.74 (.50)	1.00 (1.00)	1.00 (1.00)	1.00 (1.00)			

Secondary Structure Assignments for 1ccr:



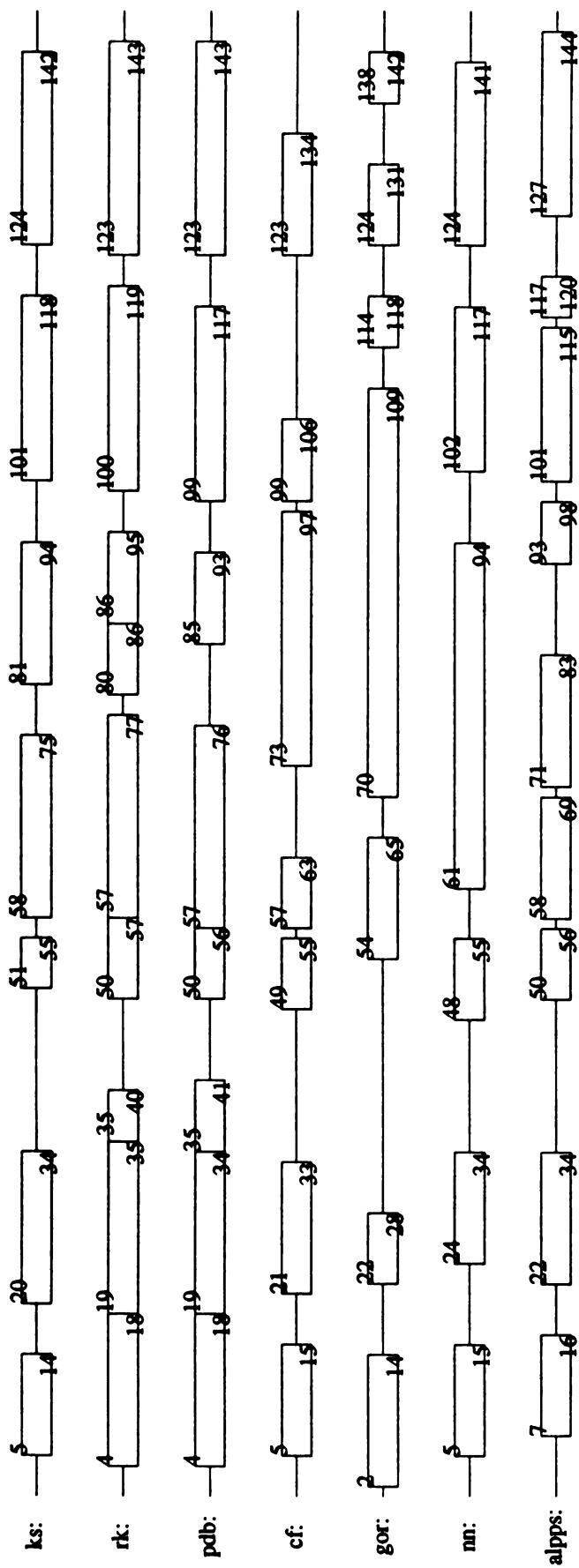
Observed	Residue Based Scores- Q (C)										
	Prediction Method	cf	gor	nn	alpps	ks	rk	pdb			
ks	standard	.74 (.56)	.84 (.69)	.86 (.70)	.75 (.51)	1.00 (1.00)	.87 (.77)	.88 (.79)			
	trimmed	.73 (.55)	.85 (.69)	.88 (.73)	.79 (.57)	1.00 (1.00)	.93 (.85)	.94 (.87)			
	trimmed†	.73 (.56)	.89 (.76)	.92 (.81)	.81 (.59)	1.00 (1.00)	.93 (.85)	.94 (.87)			
rk	standard	.69 (.38)	.78 (.57)	.77 (.54)	.71 (.42)	.87 (.77)	1.00 (1.00)	.97 (.95)			
	trimmed	.70 (.44)	.80 (.59)	.79 (.57)	.73 (.46)	.93 (.85)	1.00 (1.00)	1.00 (1.00)			
	trimmed†	.75 (.54)	.85 (.69)	.85 (.68)	.75 (.49)	.93 (.85)	1.00 (1.00)	1.00 (1.00)			
pdb	standard	.68 (.36)	.79 (.58)	.78 (.55)	.72 (.44)	.88 (.79)	.97 (.95)	1.00 (1.00)			
	trimmed	.69 (.42)	.81 (.61)	.80 (.59)	.73 (.46)	.94 (.87)	1.00 (1.00)	1.00 (1.00)			
	trimmed†	.74 (.52)	.87 (.72)	.87 (.71)	.76 (.49)	.94 (.87)	1.00 (1.00)	1.00 (1.00)			

Secondary Structure Assignments for 1ecd:



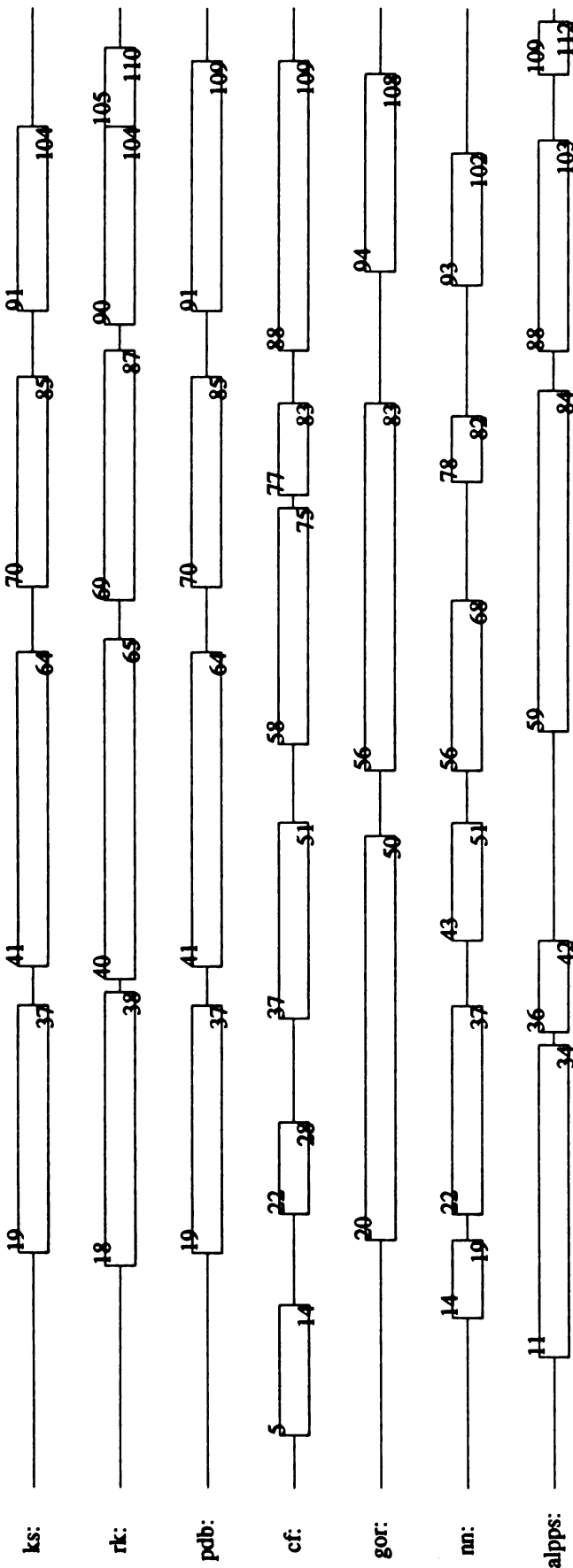
Observed	Prediction		Residue Based Scores- Q (C)									
	Method		cf	gor	nn	alpps	ks	rk	pdbs			
ks	standard		.62 (.19)	.66 (.18)	.62 (.27)	.58 (.07)	1.00 (1.00)	.77 (.43)	.76 (.39)			
	trimmed		.62 (.24)	.67 (.24)	.62 (.26)	.56 (.07)	1.00 (1.00)	.83 (.56)	.81 (.52)			
	trimmed±		.59 (.18)	.68 (.19)	.58 (.16)	.54 (.01)	1.00 (1.00)	.85 (.53)	.83 (.55)			
rk	standard		.64 (.24)	.70 (.07)	.56 (.17)	.60 (.02)	.77 (.43)	1.00 (1.00)	.96 (.82)			
	trimmed		.63 (.29)	.71 (.11)	.56 (.24)	.59 (.04)	.83 (.56)	1.00 (1.00)	.98 (.93)			
	trimmed±		.59 (.20)	.76 (-.13)	.51 (.13)	.59 (-.18)	.85 (.53)	1.00 (1.00)	.99 (.92)			
pdb	standard		.62 (.19)	.67 (-.01)	.57 (.21)	.63 (.11)	.76 (.39)	.96 (.82)	1.00 (1.00)			
	trimmed		.61 (.23)	.68 (.03)	.58 (.28)	.62 (.13)	.81 (.52)	.98 (.93)	1.00 (1.00)			
	trimmed±		.57 (.17)	.73 (-.15)	.58 (.29)	.64 (.06)	.83 (.55)	.99 (.92)	1.00 (1.00)			

Secondary Structure Assignments for 1fdh:



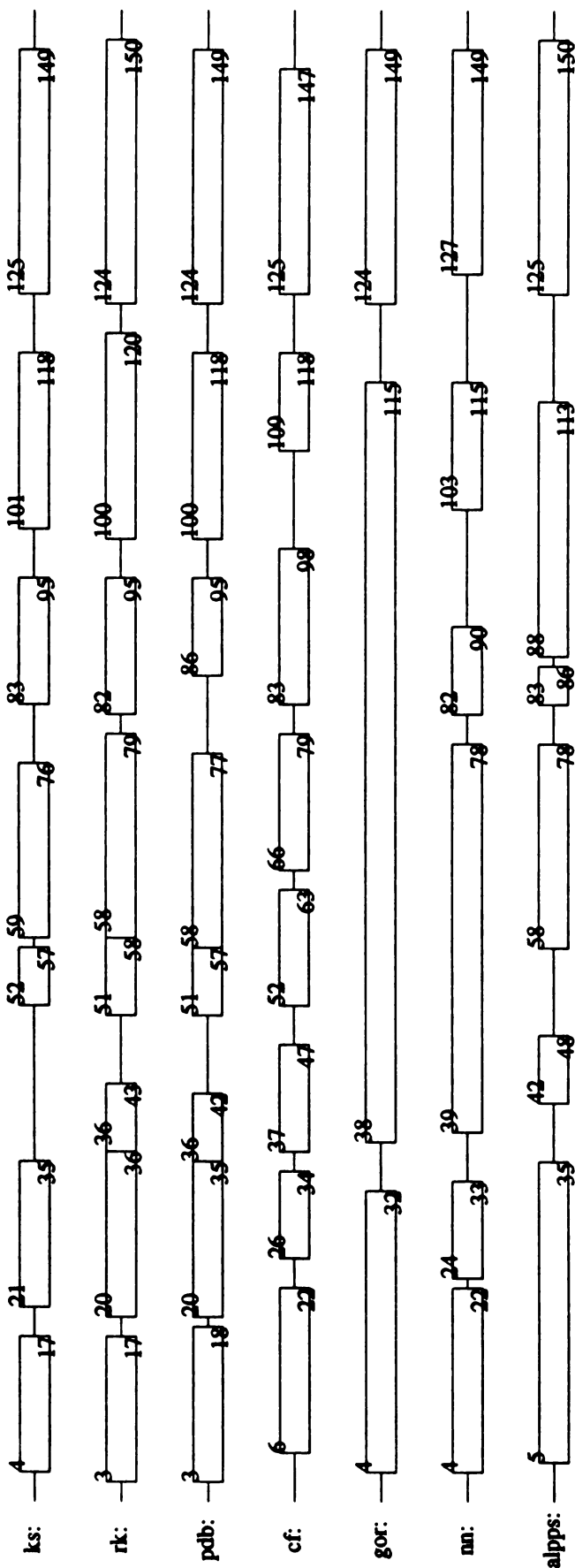
Observed	Residue Based Scores- Q (C)									
	Prediction Method	cf	gor	nn	alpps	ks	rk	pdh		
ks	standard	.69 (.35)	.72 (.39)	.87 (.70)	.76 (.45)	1.00 (1.00)	.84 (.64)	.81 (.56)		
	trimmed	.70 (.42)	.70 (.38)	.90 (.79)	.78 (.52)	1.00 (1.00)	.92 (.79)	.88 (.72)		
	trimmed†	.71 (.48)	.77 (.52)	.96 (.90)	.85 (.67)	1.00 (1.00)	.94 (.84)	.91 (.77)		
rk	standard	.65 (.28)	.67 (.26)	.78 (.48)	.74 (.36)	.84 (.64)	1.00 (1.00)	.92 (.75)		
	trimmed	.65 (.35)	.66 (.28)	.82 (.58)	.77 (.46)	.92 (.79)	1.00 (1.00)	.95 (.86)		
	trimmed†	.67 (.43)	.72 (.42)	.92 (.77)	.88 (.69)	.94 (.84)	1.00 (1.00)	.98 (.91)		
pdh	standard	.61 (.17)	.62 (.13)	.74 (.37)	.67 (.19)	.81 (.56)	.92 (.75)	1.00 (1.00)		
	trimmed	.60 (.20)	.59 (.13)	.77 (.45)	.69 (.27)	.88 (.72)	.95 (.86)	1.00 (1.00)		
	trimmed†	.60 (.20)	.62 (.18)	.83 (.54)	.78 (.39)	.91 (.77)	.98 (.91)	1.00 (1.00)		

Secondary Structure Assignments for 1hmq:



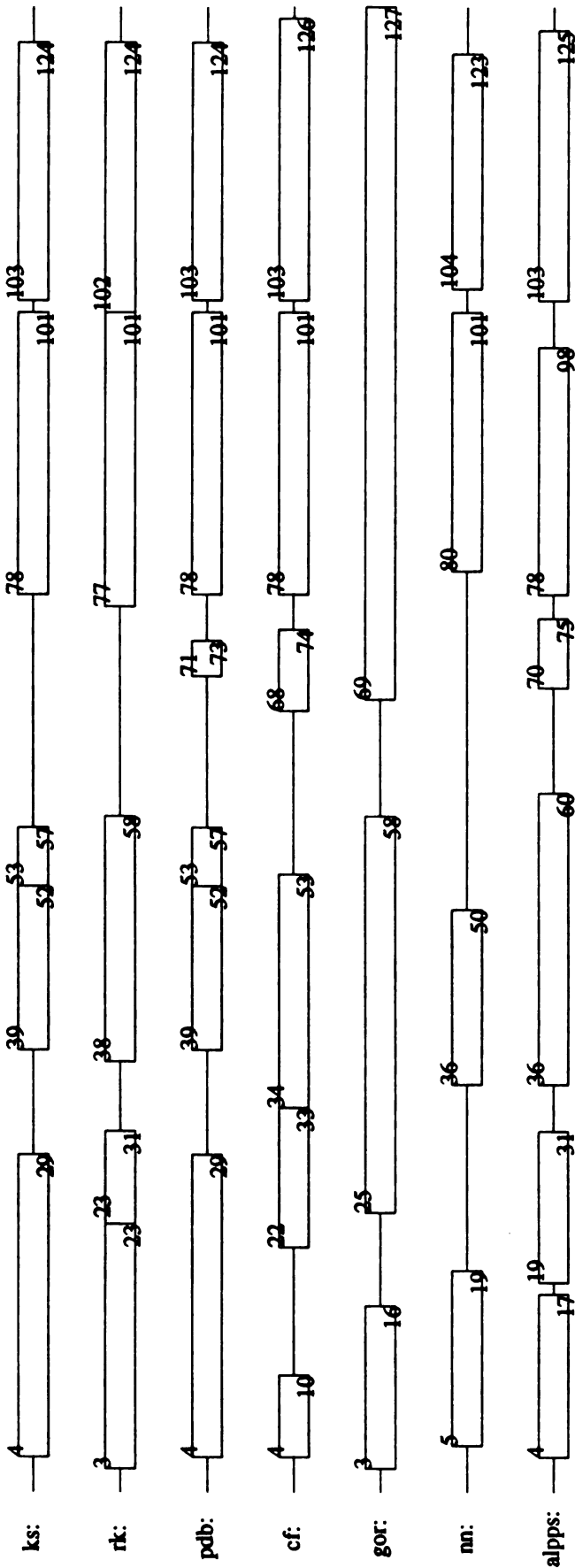
Observed	Residue Based Scores-Q (C)									
	Prediction Method	cf	gor	nn	alpps	ks	rk	pdb		
ks	standard	.59 (.08)	.80 (.55)	.72 (.44)	.63 (.17)	1.00 (1.00)	.88 (.74)	.96 (.91)		
	trimmed	.56 (.06)	.81 (.59)	.74 (.49)	.61 (.16)	1.00 (1.00)	.93 (.84)	.96 (.92)		
	trimmed±	.59 (.13)	.89 (.77)	.84 (.69)	.64 (.21)	1.00 (1.00)	.94 (.86)	.98 (.95)		
rk	standard	.65 (.10)	.81 (.58)	.63 (.28)	.65 (.12)	.88 (.74)	1.00 (1.00)	.92 (.82)		
	trimmed	.61 (.09)	.82 (.61)	.63 (.33)	.63 (.15)	.93 (.84)	1.00 (1.00)	.99 (.97)		
	trimmed±	.64 (.15)	.91 (.80)	.74 (.53)	.71 (.33)	.94 (.86)	1.00 (1.00)	1.00 (1.00)		
pdb	standard	.64 (.14)	.82 (.60)	.67 (.35)	.60 (.08)	.96 (.91)	.92 (.82)	1.00 (1.00)		
	trimmed	.61 (.13)	.84 (.65)	.68 (.40)	.57 (.06)	.96 (.92)	.99 (.97)	1.00 (1.00)		
	trimmed±	.64 (.19)	.91 (.80)	.77 (.56)	.63 (.18)	.98 (.95)	1.00 (1.00)	1.00 (1.00)		

Secondary Structure Assignments for 1mbd:



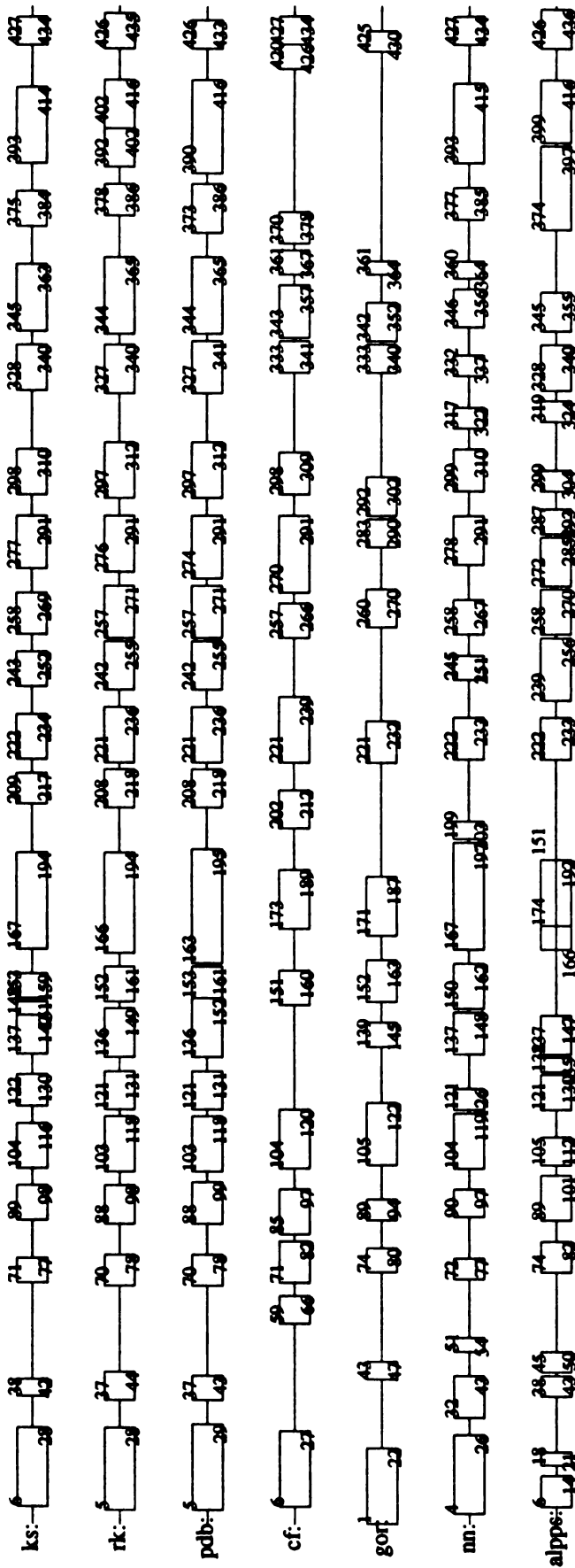
Observed	Residue Based Scores- Q (C)									
	Prediction Method	ks	rk	nn	gor	cf	alpps	ks	rk	pdb
ks	standard	.74 (.36)	.77 (.35)	.77 (.42)	.77 (.35)	.74 (.36)	.79 (.47)	1.00 (1.00)	.86 (.66)	.88 (.70)
	trimmed	.74 (.40)	.76 (.40)	.79 (.50)	.76 (.40)	.74 (.40)	.80 (.51)	1.00 (1.00)	.92 (.79)	.93 (.82)
	trimmed±	.80 (.47)	.81 (.49)	.86 (.62)	.81 (.49)	.80 (.47)	.83 (.55)	1.00 (1.00)	.94 (.79)	.94 (.84)
rk	standard	.77 (.32)	.82 (.27)	.78 (.34)	.82 (.27)	.77 (.32)	.76 (.27)	.86 (.66)	1.00 (1.00)	.93 (.77)
	trimmed	.79 (.41)	.83 (.32)	.80 (.40)	.83 (.32)	.79 (.41)	.78 (.32)	.92 (.79)	1.00 (1.00)	.96 (.88)
	trimmed±	.90 (.54)	.90 (.41)	.90 (.51)	.90 (.41)	.90 (.54)	.84 (.28)	.94 (.79)	1.00 (1.00)	.99 (.95)
pdb	standard	.76 (.34)	.80 (.33)	.77 (.36)	.80 (.33)	.76 (.34)	.75 (.30)	.88 (.70)	.93 (.77)	1.00 (1.00)
	trimmed	.79 (.44)	.81 (.39)	.79 (.44)	.81 (.39)	.79 (.44)	.77 (.37)	.93 (.82)	.96 (.88)	1.00 (1.00)
	trimmed±	.88 (.61)	.86 (.53)	.89 (.64)	.86 (.53)	.88 (.61)	.84 (.44)	.94 (.84)	.99 (.95)	1.00 (1.00)

Secondary Structure Assignments for 2ecy:



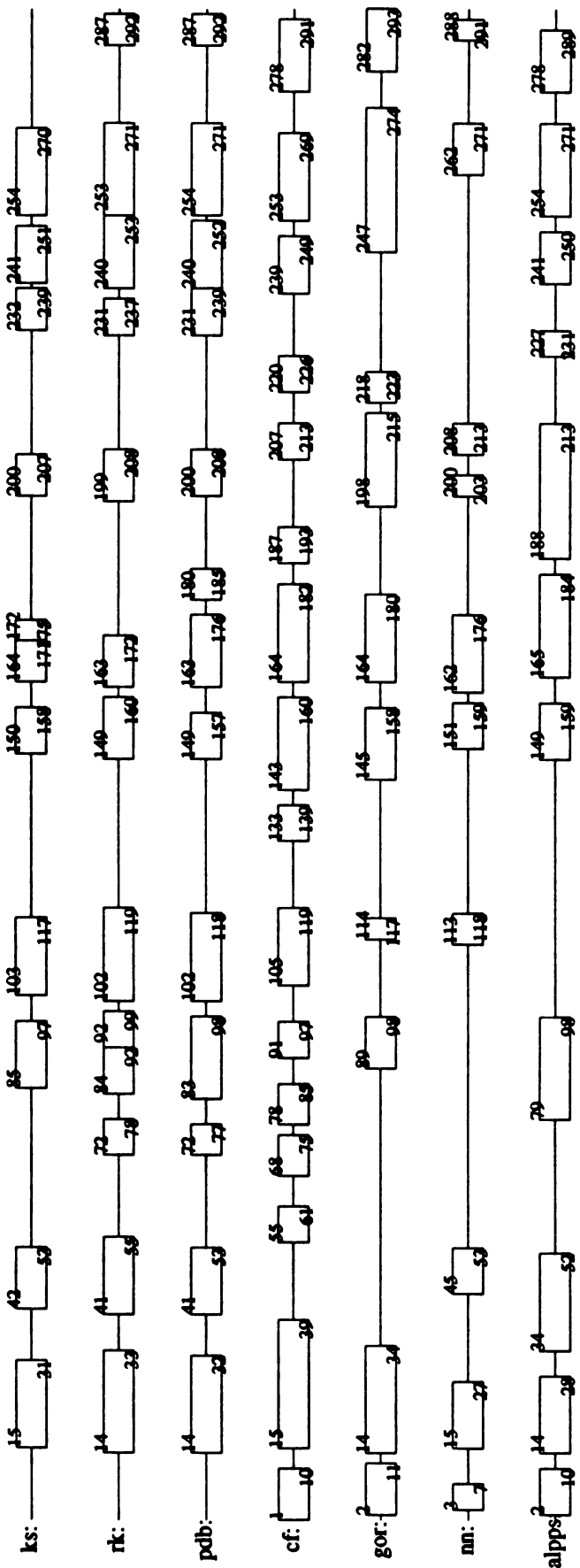
Observed	Prediction		Residue Based Scores- Q (C)									
	Method		cf	gor	nn	alpps	ks	rk	pdbs			
ks	standard		.74 (.34)	.75 (.30)	.80 (.61)	.85 (.61)	1.00 (1.00)	.94 (.86)	.98 (.94)			
	trimmed		.74 (.37)	.75 (.32)	.84 (.69)	.87 (.68)	1.00 (1.00)	.99 (.98)	.97 (.94)			
	trimmed±		.78 (.43)	.77 (.35)	.88 (.77)	.93 (.80)	1.00 (1.00)	1.00 (1.00)	.97 (.93)			
rk	standard		.73 (.28)	.80 (.38)	.76 (.55)	.86 (.58)	.94 (.86)	1.00 (1.00)	.92 (.79)			
	trimmed		.75 (.34)	.81 (.41)	.80 (.61)	.90 (.71)	.99 (.98)	1.00 (1.00)	.96 (.91)			
	trimmed±		.78 (.40)	.85 (.49)	.85 (.71)	.93 (.78)	1.00 (1.00)	1.00 (1.00)	.97 (.91)			
pdb	standard		.76 (.39)	.77 (.34)	.78 (.57)	.87 (.66)	.98 (.94)	.92 (.79)	1.00 (1.00)			
	trimmed		.78 (.45)	.77 (.36)	.81 (.64)	.92 (.77)	.97 (.94)	.96 (.91)	1.00 (1.00)			
	trimmed±		.84 (.56)	.81 (.42)	.85 (.71)	1.00 (1.00)	.97 (.93)	.97 (.91)	1.00 (1.00)			

Secondary Structure Assignments for 2cts:



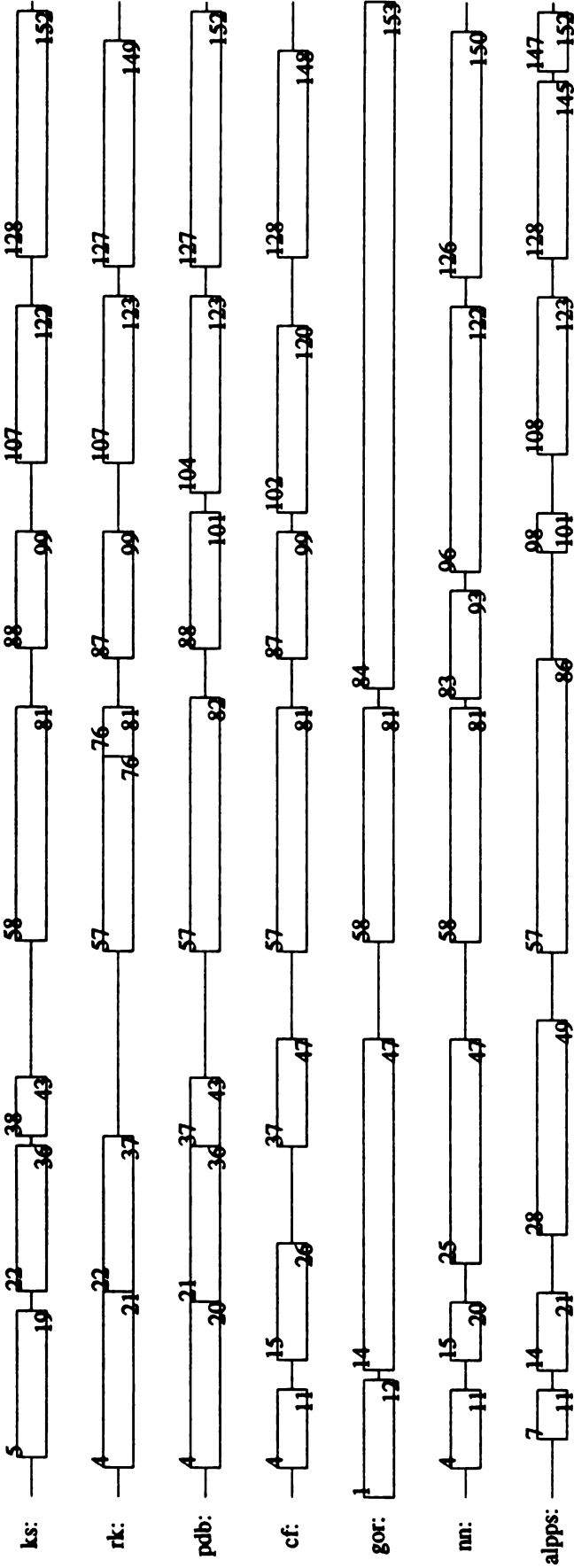
		Residue Based Scores- Q (C)						
Observed	Prediction	cf	gor	nn	alpps	ks	rk	pdb
	Method							
ks	standard	.64 (.27)	.61 (.28)	.82 (.63)	.72 (.42)	1.00 (1.00)	.89 (.77)	.87 (.75)
	trimmed	.64 (.30)	.62 (.31)	.86 (.71)	.74 (.46)	1.00 (1.00)	.96 (.92)	.95 (.90)
	trimmed±	.65 (.33)	.64 (.36)	.89 (.78)	.76 (.51)	1.00 (1.00)	1.00 (1.00)	1.00 (.99)
rk	standard	.61 (.22)	.59 (.31)	.77 (.52)	.71 (.36)	.89 (.77)	1.00 (1.00)	.95 (.89)
	trimmed	.62 (.26)	.60 (.34)	.82 (.64)	.73 (.43)	.96 (.92)	1.00 (1.00)	.97 (.94)
	trimmed±	.61 (.27)	.62 (.42)	.86 (.70)	.77 (.51)	1.00 (1.00)	1.00 (1.00)	.99 (.98)
pdb	standard	.61 (.23)	.56 (.26)	.75 (.48)	.72 (.40)	.87 (.75)	.95 (.89)	1.00 (1.00)
	trimmed	.61 (.27)	.56 (.28)	.79 (.57)	.74 (.46)	.95 (.90)	.97 (.94)	1.00 (1.00)
	trimmed±	.62 (.30)	.56 (.35)	.85 (.65)	.79 (.54)	1.00 (.99)	.99 (.98)	1.00 (1.00)

Secondary Structure Assignments for 2cyp:



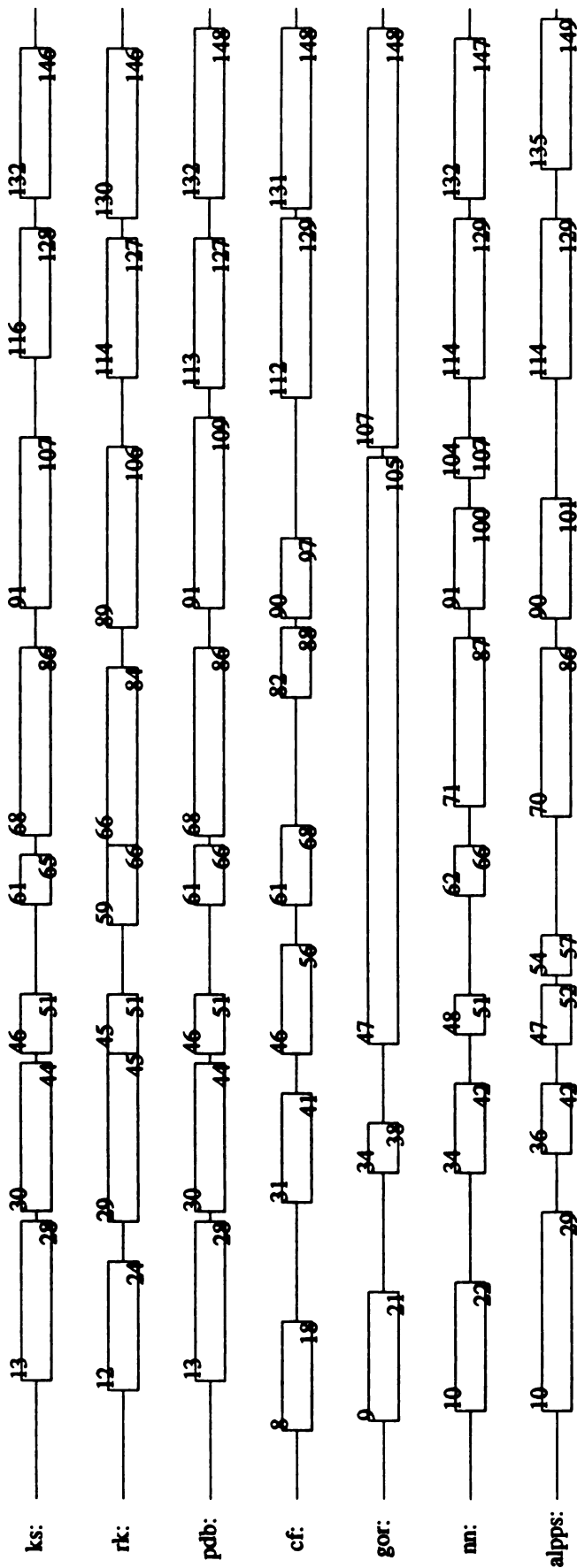
		Residue Based Scores- Q (C)								
Observed	Prediction	cf	gor	nn	alpps	ks	rk	rk	pdb	
	Method									
ks	standard	.53 (.12)	.66 (.32)	.72 (.41)	.66 (.34)	1.00 (1.00)	.86 (.74)	.88 (.78)		
	trimmed	.53 (.11)	.66 (.30)	.73 (.40)	.67 (.35)	1.00 (1.00)	.91 (.83)	.93 (.86)		
	trimmed±	.51 (.10)	.67 (.30)	.73 (.34)	.65 (.32)	1.00 (1.00)	.94 (.88)	.92 (.84)		
rk	standard	.55 (.09)	.64 (.29)	.65 (.36)	.62 (.23)	.86 (.74)	1.00 (1.00)	.92 (.84)		
	trimmed	.54 (.09)	.65 (.31)	.67 (.36)	.63 (.26)	.91 (.83)	1.00 (1.00)	.95 (.90)		
	trimmed±	.54 (.11)	.65 (.30)	.67 (.35)	.65 (.30)	.94 (.88)	1.00 (1.00)	.97 (.93)		
pdb	standard	.55 (.09)	.64 (.29)	.66 (.38)	.66 (.31)	.88 (.78)	.92 (.84)	1.00 (1.00)		
	trimmed	.55 (.11)	.65 (.30)	.66 (.38)	.67 (.34)	.93 (.86)	.95 (.90)	1.00 (1.00)		
	trimmed±	.55 (.13)	.66 (.33)	.66 (.35)	.69 (.38)	.92 (.84)	.97 (.93)	1.00 (1.00)		

Secondary Structure Assignments for 2lh1:



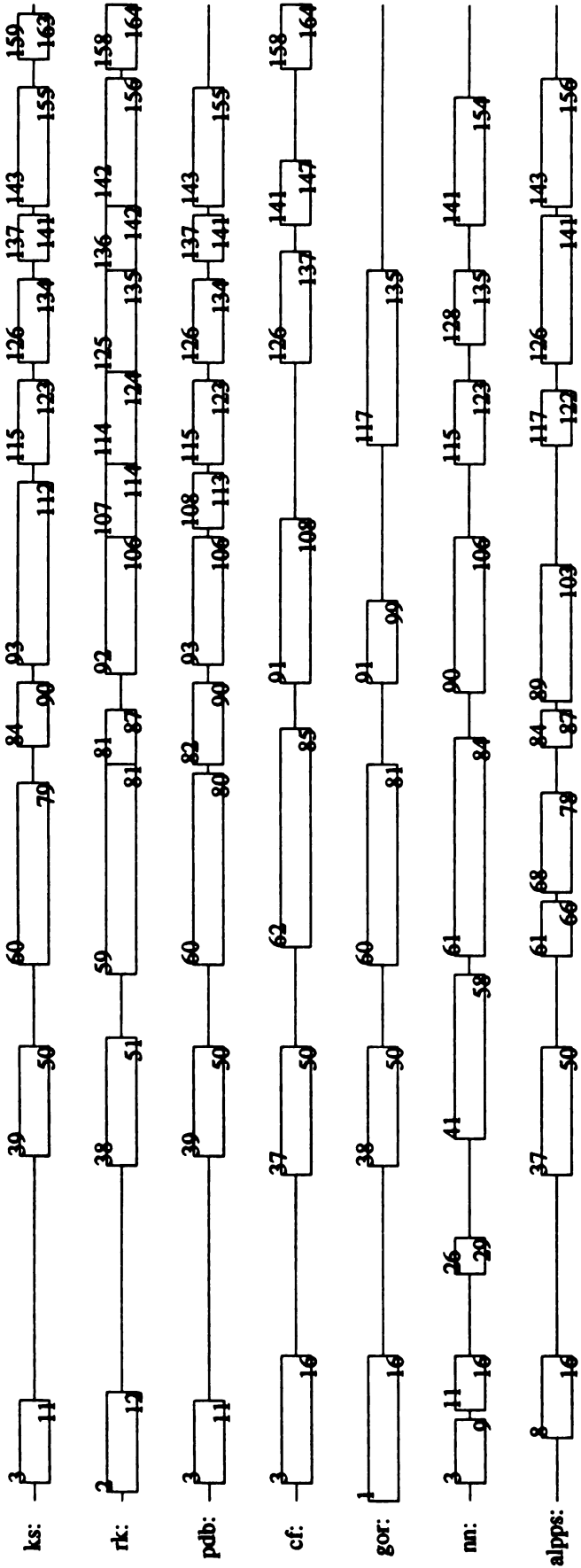
Observed	Prediction			cf	gor	nn	alpps	ks	rk	pdb
	Method	standard	trimmed							
ks		.78 (.44)	.81 (.54)	.87 (.65)	.81 (.46)	.81 (.48)	.88 (.59)	.74 (.34)	1.00 (1.00)	.89 (.71)
rk		.78 (.46)	.79 (.50)	.81 (.51)	.79 (.40)	.80 (.45)	.85 (.52)	.69 (.23)	.89 (.71)	1.00 (1.00)
pdb		.80 (.46)	.83 (.56)	.89 (.69)	.87 (.47)	.89 (.58)	.96 (.78)	.76 (.35)	.92 (.78)	.90 (.73)
		.81 (.54)	.87 (.65)	.89 (.69)	.81 (.46)	.81 (.48)	.88 (.59)	.76 (.43)	1.00 (1.00)	.93 (.84)
		.78 (.46)	.79 (.50)	.81 (.51)	.79 (.40)	.80 (.45)	.85 (.52)	.83 (.54)	1.00 (1.00)	.95 (.88)
		.80 (.46)	.83 (.56)	.89 (.69)	.87 (.47)	.89 (.58)	.96 (.78)	.70 (.29)	.93 (.84)	1.00 (1.00)
		.89 (.69)	.89 (.69)	.89 (.69)	.96 (.78)	.86 (.54)	.94 (.77)	.75 (.35)	.95 (.88)	1.00 (1.00)

Secondary Structure Assignments for 2libb:



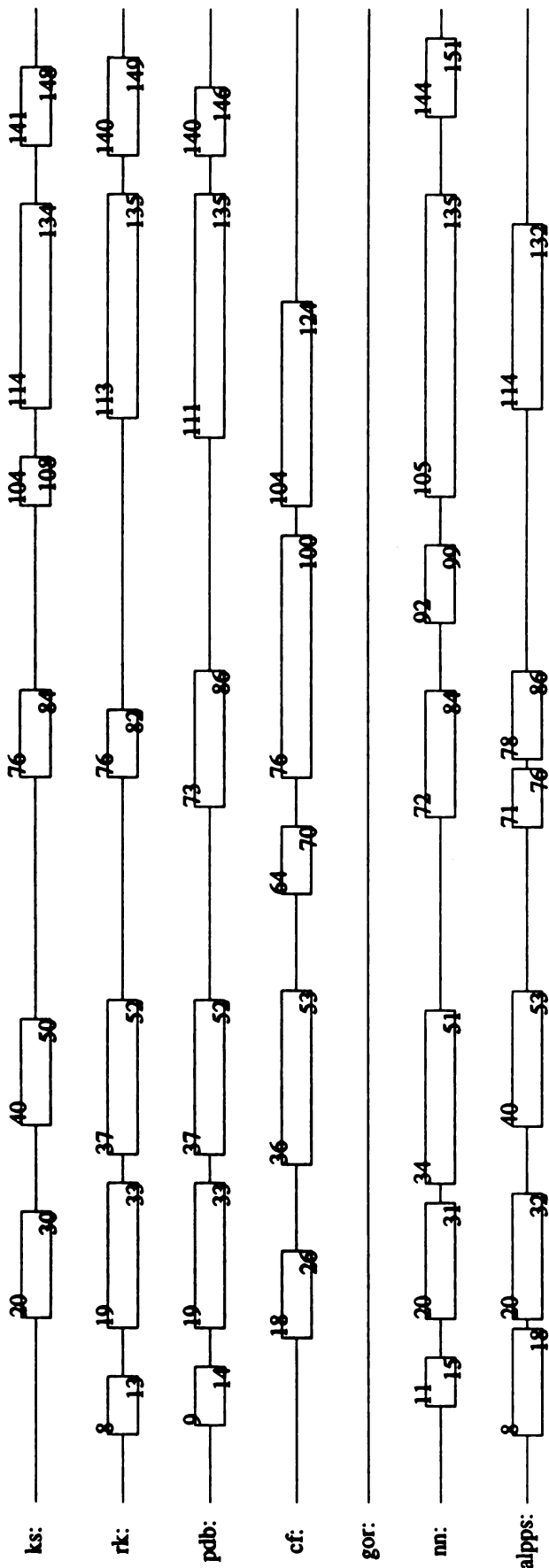
Observed	Residue Based Scores- Q (C)										
	Prediction Method	cf	gor	nn	alpps	ks	rk	pdbs			
ks	standard	.60 (.12)	.66 (.10)	.80 (.56)	.73 (.38)	1.00 (1.00)	.86 (.65)	.94 (.85)			
	trimmed	.60 (.18)	.65 (.13)	.85 (.70)	.77 (.51)	1.00 (1.00)	.93 (.82)	.97 (.93)			
	trimmed++	.61 (.23)	.71 (.22)	.93 (.86)	.83 (.60)	1.00 (1.00)	.99 (.97)	1.00 (1.00)			
rk	standard	.62 (.15)	.72 (.22)	.77 (.48)	.67 (.23)	.86 (.65)	1.00 (1.00)	.85 (.61)			
	trimmed	.63 (.23)	.73 (.27)	.80 (.60)	.68 (.29)	.93 (.82)	1.00 (1.00)	.91 (.77)			
	trimmed++	.68 (.33)	.81 (.43)	.93 (.86)	.76 (.44)	.99 (.97)	1.00 (1.00)	.99 (.96)			
pdbs	standard	.62 (.15)	.71 (.17)	.79 (.55)	.72 (.35)	.94 (.85)	.85 (.61)	1.00 (1.00)			
	trimmed	.62 (.22)	.69 (.19)	.84 (.67)	.76 (.48)	.97 (.93)	.91 (.77)	1.00 (1.00)			
	trimmed++	.62 (.22)	.76 (.29)	.93 (.84)	.80 (.50)	1.00 (1.00)	.99 (.96)	1.00 (1.00)			

Secondary Structure Assignments for 2lzm:



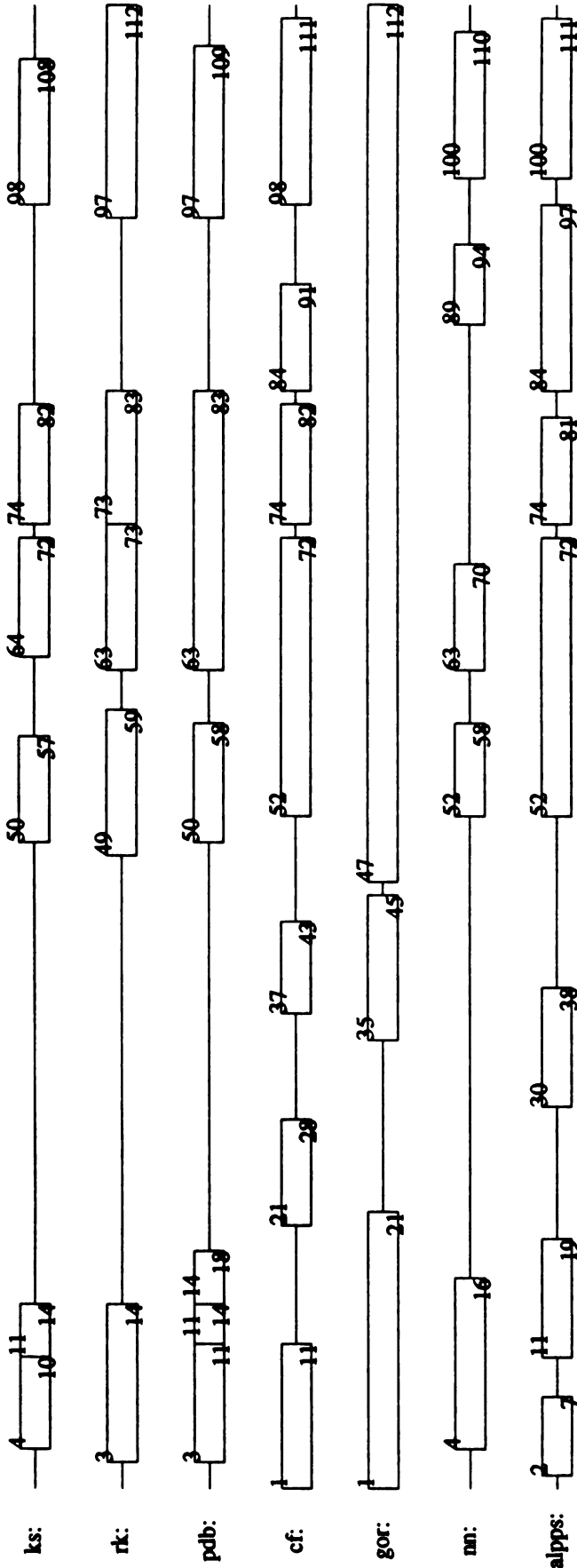
Observed	Residue Based Scores- Q (C)									
	Prediction Method	cf	gor	nn	alpps	ks	rk	pdbs		
ks	standard	.70 (.37)	.63 (.30)	.68 (.29)	.77 (.52)	1.00 (1.00)	.86 (.68)	.94 (.86)		
	trimmed	.71 (.41)	.64 (.35)	.69 (.33)	.78 (.57)	1.00 (1.00)	.95 (.89)	.95 (.90)		
	trimmed±	.76 (.54)	.64 (.42)	.74 (.42)	.83 (.67)	1.00 (1.00)	.99 (.97)	.95 (.90)		
rk	standard	.74 (.48)	.64 (.36)	.68 (.24)	.71 (.41)	.86 (.68)	1.00 (1.00)	.85 (.66)		
	trimmed	.76 (.55)	.65 (.41)	.69 (.30)	.73 (.49)	.95 (.89)	1.00 (1.00)	.93 (.84)		
	trimmed±	.81 (.63)	.65 (.44)	.76 (.40)	.81 (.65)	.99 (.97)	1.00 (1.00)	.93 (.85)		
pdbs	standard	.68 (.32)	.66 (.34)	.73 (.41)	.78 (.55)	.94 (.86)	.85 (.66)	1.00 (1.00)		
	trimmed	.67 (.33)	.67 (.40)	.74 (.45)	.80 (.61)	.95 (.90)	.93 (.84)	1.00 (1.00)		
	trimmed±	.72 (.44)	.68 (.47)	.78 (.53)	.88 (.77)	.95 (.90)	.93 (.85)	1.00 (1.00)		

Secondary Structure Assignments for 2tmv:



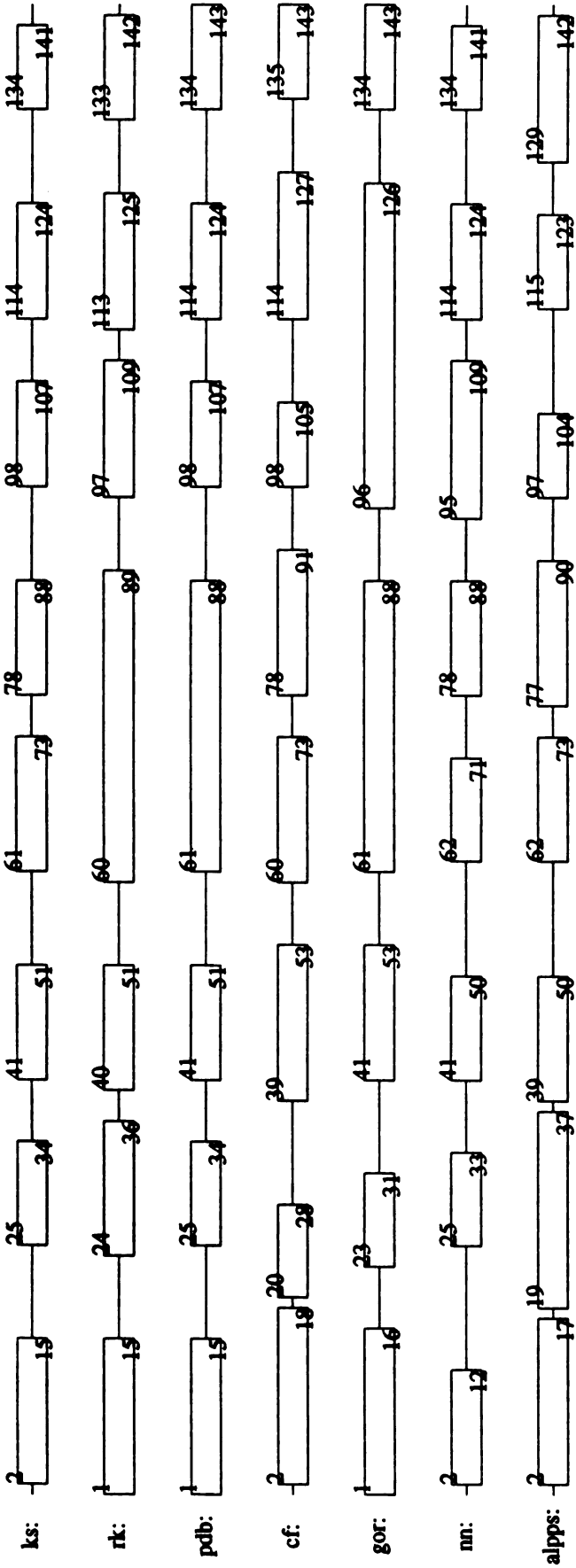
Observed	Residue Based Scores- Q (C)									
	Prediction Method	cf	gor	nn	alpps	ks	rk	pdb		
ks	standard	.62 (.24)	.58 (.00)	.75 (.56)	.75 (.49)	1.00 (1.00)	.83 (.67)	.79 (.61)		
	trimmed	.62 (.23)	.58 (.00)	.78 (.61)	.76 (.50)	1.00 (1.00)	.87 (.74)	.84 (.68)		
	trimmed†	.63 (.23)	.58 (.00)	.80 (.67)	.78 (.53)	1.00 (1.00)	.90 (.79)	.89 (.78)		
rk	standard	.54 (.08)	.50 (.00)	.74 (.49)	.77 (.55)	.83 (.67)	1.00 (1.00)	.91 (.82)		
	trimmed	.54 (.08)	.50 (.00)	.78 (.58)	.81 (.61)	.87 (.74)	1.00 (1.00)	.94 (.88)		
	trimmed†	.51 (.02)	.50 (.00)	.84 (.72)	.86 (.71)	.90 (.79)	1.00 (1.00)	.99 (.98)		
pdb	standard	.58 (.15)	.46 (.00)	.77 (.53)	.82 (.66)	.79 (.61)	.91 (.82)	1.00 (1.00)		
	trimmed	.59 (.17)	.46 (.00)	.81 (.63)	.86 (.74)	.84 (.68)	.94 (.88)	1.00 (1.00)		
	trimmed†	.58 (.16)	.46 (.00)	.86 (.74)	.92 (.85)	.89 (.78)	.99 (.98)	1.00 (1.00)		

Secondary Structure Assignments for 3c2c:



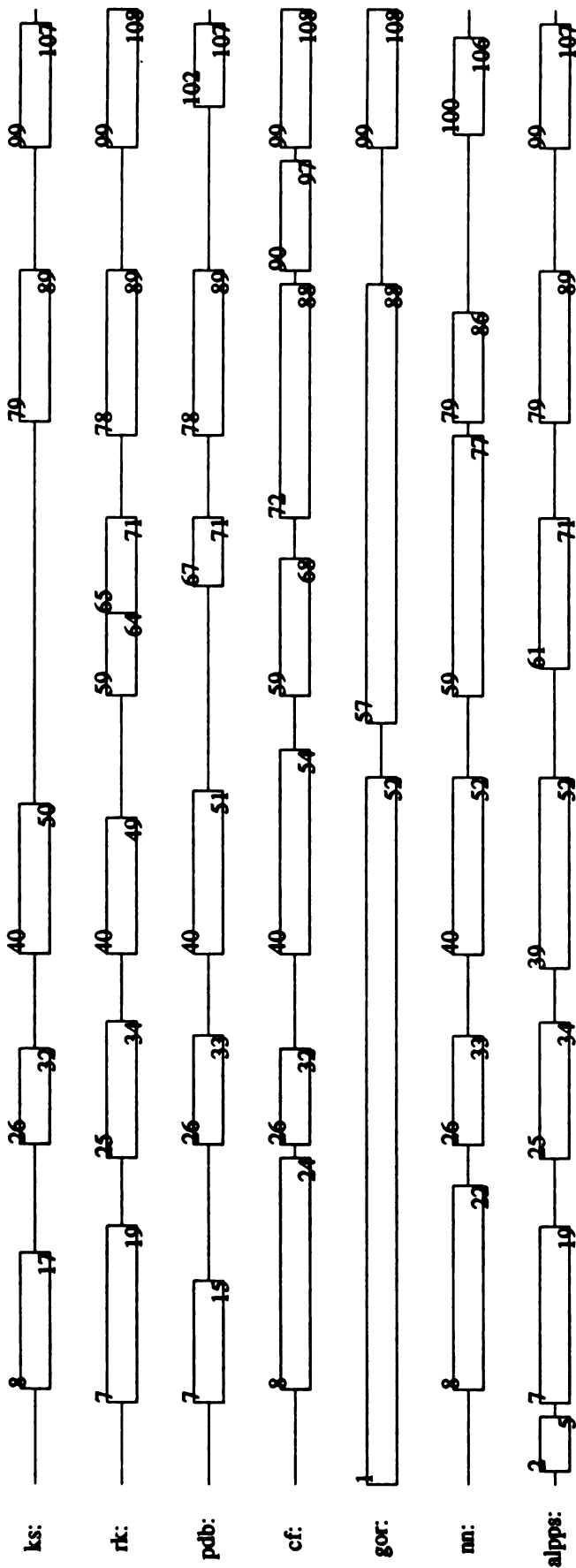
Observed	Prediction Method	Residue Based Scores- Q (C)									
		cf	gor	nn	alpps	ks	rk	pdbs			
ks	standard	.64 (.38)	.55 (.33)	.76 (.51)	.58 (.24)	1.00 (1.00)	.89 (.81)	.90 (.82)			
	trimmed	.63 (.39)	.52 (.32)	.79 (.54)	.57 (.29)	1.00 (1.00)	.96 (.91)	.95 (.90)			
	trimmed±	.63 (.46)	.51 (.33)	.79 (.49)	.56 (.39)	1.00 (1.00)	.97 (.94)	.96 (.91)			
rk	standard	.66 (.32)	.66 (.41)	.72 (.47)	.62 (.22)	.89 (.81)	1.00 (1.00)	.92 (.84)			
	trimmed	.67 (.41)	.65 (.41)	.75 (.51)	.61 (.26)	.96 (.91)	1.00 (1.00)	.94 (.88)			
	trimmed±	.68 (.48)	.67 (.43)	.79 (.56)	.63 (.39)	.97 (.94)	1.00 (1.00)	.95 (.90)			
pdb	standard	.62 (.23)	.65 (.40)	.75 (.52)	.64 (.29)	.90 (.82)	.92 (.84)	1.00 (1.00)			
	trimmed	.61 (.25)	.64 (.41)	.77 (.54)	.63 (.30)	.95 (.90)	.94 (.88)	1.00 (1.00)			
	trimmed±	.61 (.28)	.66 (.44)	.78 (.51)	.65 (.45)	.96 (.91)	.95 (.90)	1.00 (1.00)			

Secondary Structure Assignments for 3cla:



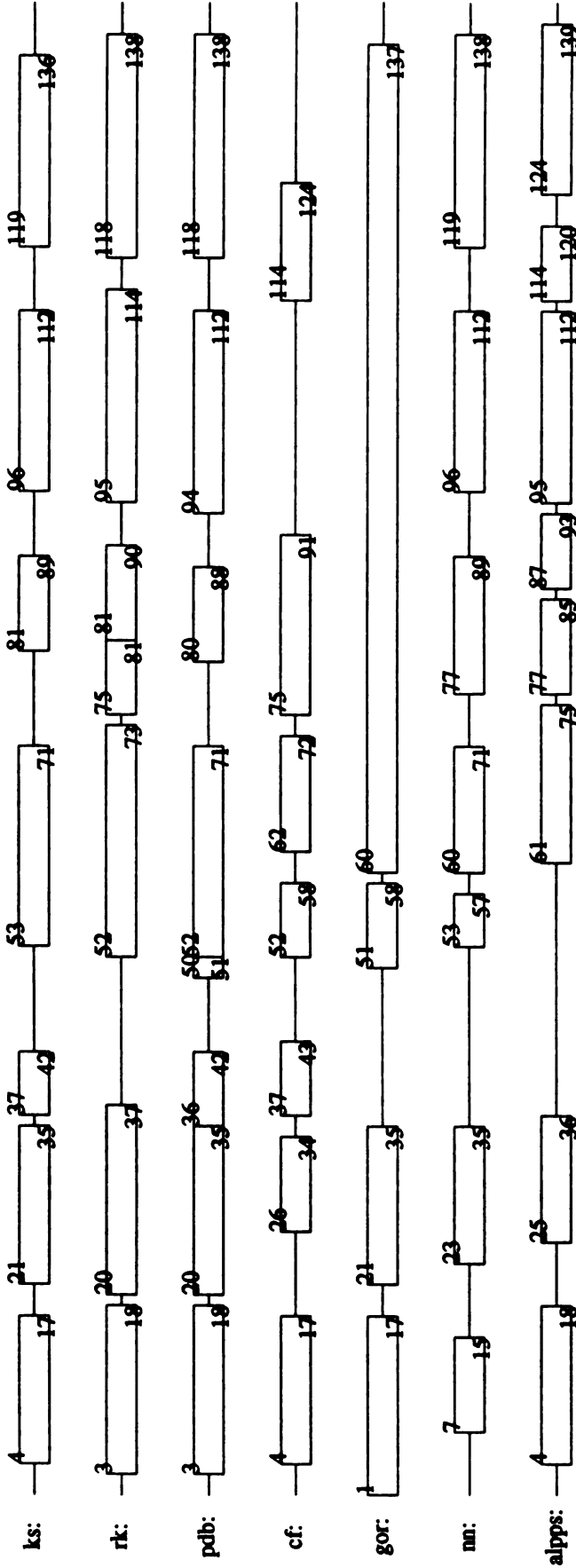
		Residue Based Scores- Q (C)									
Observed	Prediction Method	cf	gor	nn	alpps	ks	rk	pdbs			
ks	standard	.79 (.55)	.82 (.63)	.91 (.81)	.79 (.55)	1.00 (1.00)	.87 (.75)	.95 (.90)			
	trimmed	.84 (.66)	.86 (.71)	.95 (.90)	.86 (.71)	1.00 (1.00)	.95 (.89)	.96 (.92)			
	trimmed±	.92 (.81)	.92 (.82)	1.00 (1.00)	.93 (.85)	1.00 (1.00)	.98 (.95)	.97 (.94)			
rk	standard	.75 (.38)	.87 (.65)	.82 (.65)	.79 (.46)	.87 (.75)	1.00 (1.00)	.91 (.80)			
	trimmed	.80 (.52)	.93 (.82)	.90 (.79)	.84 (.61)	.95 (.89)	1.00 (1.00)	.98 (.96)			
	trimmed±	.86 (.62)	.96 (.87)	.95 (.89)	.92 (.76)	.98 (.95)	1.00 (1.00)	.99 (.97)			
pdb	standard	.77 (.47)	.87 (.71)	.86 (.71)	.77 (.46)	.95 (.90)	.91 (.80)	1.00 (1.00)			
	trimmed	.81 (.58)	.90 (.79)	.90 (.80)	.83 (.63)	.96 (.92)	.98 (.96)	1.00 (1.00)			
	trimmed±	.91 (.78)	.96 (.90)	.94 (.88)	.91 (.77)	.97 (.94)	.99 (.97)	1.00 (1.00)			

Secondary Structure Assignments for 3cpv:



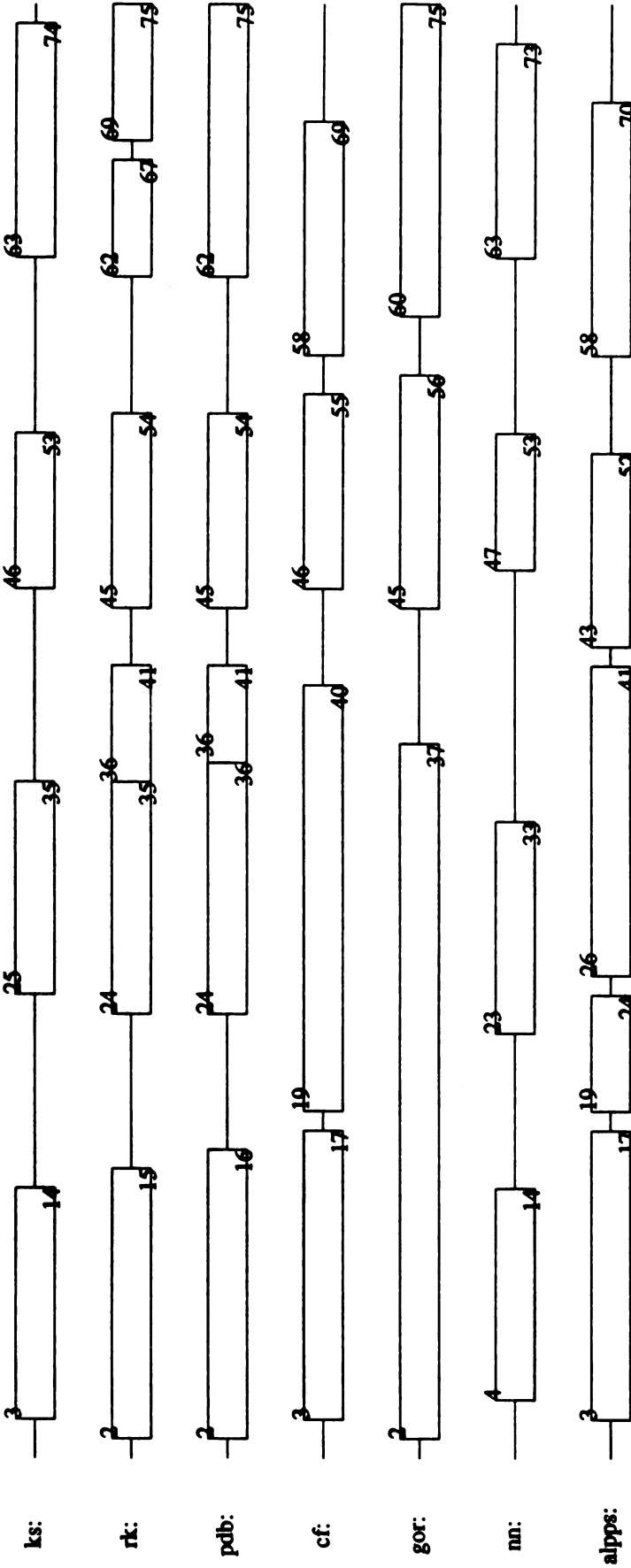
Observed	Residue Based Scores- Q (C)									
	Prediction Method	cf	gor	mn	alpps	ks	rk	pdb		
ks	standard	.65 (.43)	.56 (.29)	.70 (.46)	.78 (.63)	1.00 (1.00)	.80 (.65)	.87 (.74)		
	trimmed	.65 (.47)	.54 (.33)	.72 (.52)	.80 (.67)	1.00 (1.00)	.83 (.72)	.91 (.82)		
	trimmed±	.61 (.43)	.51 (.30)	.69 (.53)	.79 (.65)	1.00 (1.00)	.80 (.67)	.93 (.86)		
rk	standard	.70 (.33)	.74 (.45)	.80 (.56)	.89 (.76)	.80 (.65)	1.00 (1.00)	.81 (.66)		
	trimmed	.72 (.44)	.73 (.49)	.86 (.71)	.92 (.84)	.83 (.72)	1.00 (1.00)	.85 (.73)		
	trimmed±	.75 (.54)	.79 (.56)	.93 (.84)	.92 (.83)	.80 (.67)	1.00 (1.00)	.89 (.79)		
pdb	standard	.59 (.25)	.59 (.32)	.72 (.48)	.80 (.64)	.87 (.74)	.81 (.66)	1.00 (1.00)		
	trimmed	.60 (.36)	.58 (.36)	.76 (.58)	.84 (.72)	.91 (.82)	.85 (.73)	1.00 (1.00)		
	trimmed±	.61 (.42)	.57 (.35)	.79 (.65)	.86 (.76)	.93 (.86)	.89 (.79)	1.00 (1.00)		

Secondary Structure Assignments for 3hhb:



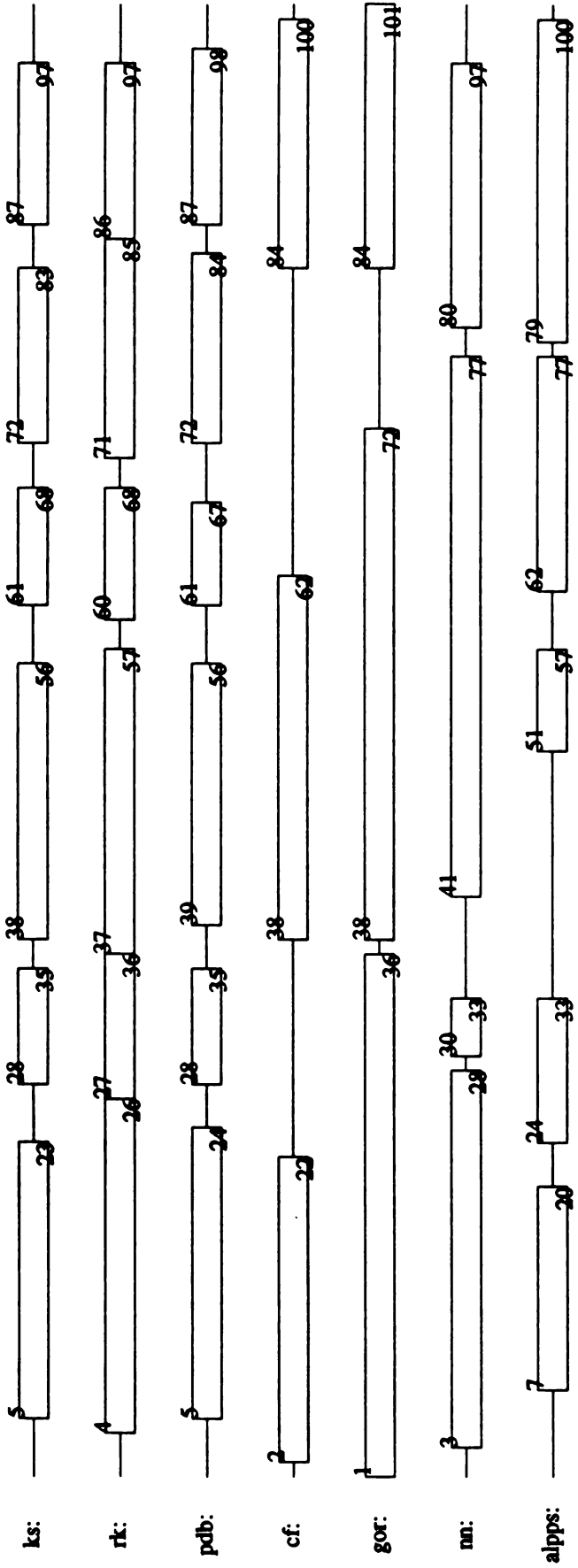
Observed	Residue Based Scores- Q (C)										
	Prediction Method	cf	gor	nn	alpps	ks	rk	pdb			
ks	standard	.62 (.22)	.76 (.38)	.85 (.68)	.68 (.24)	1.00 (1.00)	.82 (.56)	.90 (.76)			
	trimmed	.62 (.28)	.75 (.38)	.87 (.73)	.70 (.31)	1.00 (1.00)	.89 (.71)	.97 (.92)			
	trimmed±	.58 (.25)	.78 (.39)	.92 (.82)	.78 (.45)	1.00 (1.00)	.94 (.80)	.99 (.97)			
rk	standard	.60 (.18)	.88 (.60)	.83 (.65)	.79 (.45)	.82 (.56)	1.00 (1.00)	.86 (.59)			
	trimmed	.58 (.20)	.91 (.71)	.89 (.76)	.79 (.51)	.89 (.71)	1.00 (1.00)	.89 (.69)			
	trimmed±	.57 (.19)	.97 (.88)	.99 (.96)	.87 (.68)	.94 (.80)	1.00 (1.00)	.94 (.78)			
pdb	standard	.56 (.09)	.77 (.28)	.79 (.55)	.68 (.18)	.90 (.76)	.86 (.59)	1.00 (1.00)			
	trimmed	.56 (.16)	.78 (.33)	.82 (.64)	.69 (.25)	.97 (.92)	.89 (.69)	1.00 (1.00)			
	trimmed±	.53 (.17)	.81 (.37)	.90 (.78)	.78 (.38)	.99 (.97)	.94 (.78)	1.00 (1.00)			

Secondary Structure Assignments for 3kcb:



Observed	Residue Based Scores- Q (C)									
	Prediction Method	cf	gor	nn	alpps	ks	rk	pdh		
ks	standard	.65 (.28)	.72 (.48)	.91 (.81)	.61 (.17)	1.00 (1.00)	.81 (.64)	.81 (.65)		
	trimmed	.67 (.33)	.75 (.54)	.97 (.94)	.63 (.28)	1.00 (1.00)	.88 (.76)	.90 (.80)		
	trimmedt	.75 (.40)	.79 (.55)	1.00 (1.00)	.69 (.34)	1.00 (1.00)	.85 (.73)	.89 (.77)		
rk	standard	.71 (.20)	.77 (.35)	.75 (.52)	.69 (.15)	.81 (.64)	1.00 (1.00)	.97 (.93)		
	trimmed	.75 (.34)	.77 (.39)	.81 (.65)	.71 (.24)	.88 (.76)	1.00 (1.00)	.98 (.96)		
	trimmedt	.79 (.10)	.82 (.33)	.83 (.71)	.80 (.14)	.85 (.73)	1.00 (1.00)	1.00 (1.00)		
pdh	standard	.73 (.24)	.80 (.39)	.75 (.54)	.72 (.19)	.81 (.65)	.97 (.93)	1.00 (1.00)		
	trimmed	.79 (.41)	.81 (.45)	.81 (.67)	.76 (.31)	.90 (.80)	.98 (.96)	1.00 (1.00)		
	trimmedt	.81 (.12)	.87 (.38)	.87 (.75)	.83 (.15)	.89 (.77)	1.00 (1.00)	1.00 (1.00)		

Secondary Structure Assignments for 3wrp:



Observed	Prediction		Residue Based Scores- Q (C)									
	Method		cf	got	nn	alpps	ks	rk	pdb			
ks	standard	.60 (.10)	.66 (-.13)	.76 (.27)	.64 (.12)	1.00 (1.00)	.87 (.63)	.95 (.86)				
	trimmed	.60 (.13)	.66 (-.13)	.75 (.30)	.65 (.17)	1.00 (1.00)	.96 (.87)	1.00 (1.00)				
	trimmed†	.64 (.27)	.74 (-.15)	.88 (.59)	.71 (.29)	1.00 (1.00)	1.00 (1.00)	1.00 (1.00)				
rk	standard	.59 (-.01)	.77 (-.13)	.85 (.37)	.69 (.17)	.87 (.63)	1.00 (1.00)	.86 (.57)				
	trimmed	.59 (.03)	.76 (-.13)	.85 (.42)	.68 (.20)	.96 (.87)	1.00 (1.00)	.97 (.90)				
	trimmed†	.65 (.17)	.83 (-.07)	.98 (.89)	.75 (.38)	1.00 (1.00)	1.00 (1.00)	1.00 (1.00)				
pdb	standard	.61 (.11)	.67 (-.13)	.77 (.28)	.67 (.19)	.95 (.86)	.86 (.57)	1.00 (1.00)				
	trimmed	.62 (.16)	.66 (-.14)	.77 (.34)	.66 (.22)	1.00 (1.00)	.97 (.90)	1.00 (1.00)				
	trimmed†	.67 (.28)	.74 (-.15)	.90 (.63)	.72 (.34)	1.00 (1.00)	1.00 (1.00)	1.00 (1.00)				

		Aggregate Residue Based Scores- $Q(C)$									
Observed	Prediction	cf	gor	nn	alpps	ks	kr	pd	pd	pd	pd
	Method										
ks	standard	.66 (.28)	.69 (.35)	.78 (.55)	.71 (.40)	1.00 (1.00)	.86 (.72)	.89 (.77)			
	trimmed	.66 (.30)	.69 (.37)	.81 (.61)	.73 (.45)	1.00 (1.00)	.92 (.84)	.94 (.87)			
	trimmed±	.67 (.33)	.72 (.43)	.85 (.69)	.76 (.51)	1.00 (1.00)	.95 (.90)	.95 (.90)			
kr	standard	.65 (.23)	.71 (.35)	.74 (.47)	.71 (.36)	.86 (.72)	1.00 (1.00)	.91 (.79)			
	trimmed	.66 (.27)	.71 (.39)	.78 (.55)	.73 (.42)	.92 (.84)	1.00 (1.00)	.95 (.89)			
	trimmed±	.67 (.30)	.75 (.46)	.83 (.66)	.78 (.52)	.95 (.90)	1.00 (1.00)	.98 (.94)			
pdb	standard	.65 (.23)	.69 (.32)	.75 (.48)	.72 (.38)	.89 (.77)	.91 (.79)	1.00 (1.00)			
	trimmed	.66 (.27)	.69 (.35)	.78 (.55)	.74 (.44)	.94 (.87)	.95 (.89)	1.00 (1.00)			
	trimmed±	.67 (.30)	.72 (.42)	.83 (.65)	.79 (.54)	.95 (.90)	.98 (.94)	1.00 (1.00)			

Table IV-2: Aggregate Scores for All 20 Proteins

This table shows the residue based scores on the aggregation of the tallies for all 20 proteins.

Assignment Comparison Summary Statistics on Q Scores								
		Method	Mean	Median	SD	Worst	Best	Range
rk	ks	standard	.86	.86	.04	.77	.94	.17
		trimmed	.92	.91	.04	.83	.99	.16
		trimmed±	.94	.90	.06	.80	1.00	.20
pdb	rk	standard	.91	.89	.05	.81	.97	.16
		trimmed	.95	.93	.04	.85	1.00	.15
		trimmed±	.97	.94	.03	.89	1.00	.12
pdb	ks	standard	.89	.87	.06	.76	.98	.22
		trimmed	.94	.91	.05	.81	1.00	.19
		trimmed±	.95	.92	.05	.83	1.00	.17

Table IV-3: Assignment Comparison Summary Statistics on Q Scores

Sets of 20 (one for each protein) Q scores are taken from the residue based score tables of Figure 4-5. The combination of two secondary structure assignment methods and tallying method (*e.g.*, trimmed) specifies each set. This table lists summary statistics (mean, median, standard deviation, worst, best, and range) on each set of Q scores.

Secondary Structure Assignment Comparisons Ranking by Standard Q					
rk-ks		pdb-ks		pdb-rk	
.94	2ccy	.98	2ccy	.97	3icb
.89	3c2c	.96	1hmq	.97	1ccr
.89	1cc5	.95	3cln	.96	1ecd
.89	2lh1	.95	3wrp	.95	2cts
.89	2cts	.94	2lhb	.94	1cc5
.88	1hmq	.94	2lzm	.93	1mbd
.87	3cln	.93	1cc5	.92	2ccy
.87	1ccr	.92	2lh1	.92	3c2c
.87	156b	.90	3c2c	.92	1hmq
.87	3wrp	.90	3hhb	.92	2cyp
.86	1mbd	.88	1ccr	.92	1fdh
.86	2lzm	.88	1mbd	.91	3cln
.86	2lhb	.88	2cyp	.91	2tmv
.86	2cyp	.87	2cts	.90	2lh1
.84	1fdh	.87	3cpv	.86	3hhb
.83	2tmv	.86	156b	.86	3wrp
.82	3hhb	.81	1fdh	.85	2lhb
.81	3icb	.81	3icb	.85	2lzm
.80	3cpv	.79	2tmv	.83	156b
.77	1ecd	.76	1ecd	.81	3cpv

Table IV-4: Assignment Comparisons Ranking by Standard Q

Secondary structure assignments (Kabsch-Sander, Richards-Kundrot, PDB) on the 20 proteins are compared against each other in pairs. This table lists a ranking of the proteins based on standard Q scores for each of the three comparisons.

Secondary Structure Assignment Comparisons Ranking Trimmed Q					
rk-ks		pdb-ks		pdb-rk	
.99	2ccy	1.00	3wrp	1.00	1ccr
.96	3wrp	1.00	1cc5	.99	1hmq
.96	2cts	.98	2lh1	.98	3icb
.96	3c2c	.97	2ccy	.98	3cln
.96	1cc5	.97	2lhb	.98	1ecd
.95	2lzm	.97	3hhb	.97	3wrp
.95	3cln	.96	1hmq	.97	1cc5
.93	2lh1	.96	3cln	.97	2cts
.93	156b	.95	2lzm	.96	2ccy
.93	1hmq	.95	2cts	.96	1mbd
.93	1ccr	.95	3c2c	.95	1fdh
.93	2lhb	.94	1ccr	.95	2cyp
.92	1mbd	.93	1mbd	.94	2tmv
.92	1fdh	.93	2cyp	.94	3c2c
.91	2cyp	.92	156b	.93	2lzm
.89	3hhb	.91	3cpv	.92	2lh1
.88	3icb	.90	3icb	.91	2lhb
.87	2tmv	.88	1fdh	.89	3hhb
.83	3cpv	.84	2tmv	.88	156b
.83	1ecd	.81	1ecd	.85	3cpv

Table IV-5: Assignment Comparisons Ranking by Trimmed Q

Secondary structure assignments (Kabsch-Sander, Richards-Kundrot, PDB) on the 20 proteins are compared against each other in pairs. This table lists a ranking of the proteins based on trimmed Q scores for each of the three comparisons.

Secondary Structure Assignment Comparisons Ranking Trimmed± Q					
rk-ks		pdb-ks		pdb-rk	
1.00	3wrp	1.00	3wrp	1.00	3wrp
1.00	2cts	1.00	2lhb	1.00	3icb
1.00	2ccy	1.00	2lh1	1.00	1hmq
1.00	156b	1.00	1cc5	1.00	1ccr
.99	2lzm	1.00	2cts	1.00	1cc5
.99	2lhb	.99	3hhb	.99	2cts
.98	1cc5	.98	156b	.99	2tmv
.98	3cln	.98	1hmq	.99	1mbd
.97	3c2c	.97	2ccy	.99	3cln
.95	2lh1	.97	3cln	.99	2lhb
.94	1fdh	.96	3c2c	.99	1ecd
.94	3hhb	.95	2lzm	.98	1fdh
.94	1mbd	.94	1mbd	.97	2lh1
.94	1hmq	.94	1ccr	.97	2ccy
.94	2cyp	.93	3cpv	.97	2cyp
.93	1ccr	.92	2cyp	.96	156b
.90	2tmv	.91	1fdh	.95	3c2c
.85	1ecd	.89	2tmv	.94	3hhb
.85	3icb	.89	3icb	.93	2lzm
.80	3cpv	.83	1ecd	.89	3cpv

Table IV-6: Assignment Comparisons Ranking by Trimmed± Q

Secondary structure assignments (Kabsch-Sander, Richards-Kundrot, PDB) on the 20 proteins are compared against each other in pairs. This table lists a ranking of the proteins based on trimmed± Q scores for each of the three comparisons.

Summary Statistics on Residue Based Q Scores								
		Method	Mean	Median	SD	Worst	Best	Range
ks	cf	standard	.67	.67	.08	.53	.81	.28
		trimmed	.67	.68	.09	.52	.84	.32
		trimmed±	.69	.71	.12	.51	.92	.41
	gor	standard	.70	.70	.09	.55	.84	.28
		trimmed	.70	.69	.10	.52	.86	.34
		trimmed±	.74	.71	.12	.51	.92	.41
	nn	standard	.79	.77	.07	.62	.91	.28
		trimmed	.81	.79	.09	.62	.97	.35
		trimmed±	.85	.79	.11	.58	1.00	.42
alpps	standard	.71	.71	.08	.58	.85	.27	
	trimmed	.73	.72	.09	.56	.87	.31	
	trimmed±	.76	.74	.11	.54	.93	.39	
rk	cf	standard	.67	.67	.09	.52	.82	.30
		trimmed	.67	.68	.10	.48	.87	.39
		trimmed±	.70	.70	.12	.49	.90	.41
	gor	standard	.73	.69	.10	.50	.88	.38
		trimmed	.74	.72	.10	.50	.93	.43
		trimmed±	.78	.73	.12	.50	.97	.47
	nn	standard	.75	.70	.07	.56	.85	.29
		trimmed	.78	.73	.09	.56	.90	.33
		trimmed±	.84	.75	.12	.51	.99	.48
alpps	standard	.72	.75	.07	.60	.89	.29	
	trimmed	.74	.75	.09	.59	.92	.33	
	trimmed±	.79	.76	.10	.59	.93	.34	
pdb	cf	standard	.66	.67	.08	.55	.80	.25
		trimmed	.67	.69	.10	.55	.84	.29
		trimmed±	.70	.72	.13	.53	.91	.38
	gor	standard	.71	.67	.11	.46	.87	.41
		trimmed	.72	.68	.12	.46	.90	.44
		trimmed±	.76	.71	.14	.46	.96	.50
	nn	standard	.76	.72	.07	.57	.86	.29
		trimmed	.79	.74	.08	.58	.90	.32
		trimmed±	.84	.78	.10	.58	.98	.39
alpps	standard	.72	.74	.07	.60	.87	.27	
	trimmed	.74	.74	.09	.57	.92	.34	
	trimmed±	.79	.81	.10	.63	1.00	.37	

Table IV-7: Summary Statistics on Residue Based Q Scores

Sets of 20 (one for each protein) Q scores are taken from the residue based score tables of Figure 4-5. The combination of secondary structure assignment method (observation), prediction method, and tallying method (*e.g.*, trimmed) specifies each set. This table lists summary statistics (mean, median, standard deviation, worst, best, and range) on each set of Q scores.

Summary Statistics on Residue Based Q Scores								
		Method	Mean	Median	SD	Worst	Best	Range
rk	rg	standard	.74	.74	.08	.59	.88	.28
		trimmed	.75	.77	.09	.60	.93	.33
		trimmed±	.80	.79	.10	.62	.97	.35
ks	rg	standard	.71	.70	.09	.55	.84	.28
		trimmed	.71	.69	.10	.52	.86	.34
		trimmed±	.75	.71	.12	.51	.92	.41
pdb	rg	standard	.73	.72	.10	.56	.87	.31
		trimmed	.73	.73	.11	.56	.90	.34
		trimmed±	.77	.76	.12	.56	.96	.40

Table IV-8: GOR Summary Statistics without 2tmv

This table is similar to Table IV-7 in format. Only summaries of GOR predictions are shown, and only 19 of the 20 predictions are included in the data. The GOR prediction for Intact Tobacco Mosaic Virus [2tmv] contains no helices, and this Q score is not considered here.

Ranking of Individual Predictions by Standard Q											
cf			got			nn			alpps		
ks	rk	pdb	ks	rk	pdb	ks	rk	pdb	ks	rk	pdb
.81 1cc5	.82 1cc5	.80 2lh1	.84 1ccr	.88 3hbb	.87 3cln	.91 3cln	.85 3wrp	.86 3cln	.85 2ccy	.89 3cpv	.87 2ccy
.79 3cln	.78 2lh1	.78 1cc5	.82 3cln	.87 3cln	.87 2lh1	.91 3icb	.83 1cc5	.84 1cc5	.79 1mbd	.86 2ccy	.82 2mv
.78 2lh1	.77 1mbd	.77 3cln	.81 2lh1	.82 1mbd	.83 156b	.87 1fdh	.83 3hbb	.83 2lh1	.79 3cln	.79 3cln	.80 3cpv
.74 1mbd	.75 3cln	.76 2ccy	.80 1hmq	.81 1hmq	.82 1hmq	.86 1ccr	.82 3cln	.80 156b	.78 3cpv	.79 3hbb	.78 2lzm
.74 2ccy	.74 2lzm	.76 1mbd	.77 1cc5	.80 2ccy	.80 1mbd	.85 3hbb	.80 3cpv	.79 3hbb	.77 2lzm	.77 2mv	.77 3cln
.74 1ccr	.73 2ccy	.73 3icb	.77 1mbd	.79 2lh1	.80 3icb	.84 1cc5	.78 1fdh	.79 2lhb	.77 156b	.76 1mbd	.76 2lh1
.70 2lzm	.71 3icb	.68 2lzm	.76 3hbb	.78 1ccr	.79 1ccr	.82 2cts	.78 2lh1	.78 2ccy	.76 1fdh	.74 1fdh	.75 1mbd
.69 1fdh	.70 3cpv	.68 1ccr	.75 2ccy	.77 3icb	.77 3hbb	.80 2ccy	.78 1mbd	.78 1ccr	.75 1ccr	.74 156b	.72 2cts
.65 3icb	.69 1ccr	.64 156b	.72 3icb	.77 3wrp	.77 2ccy	.80 2lhb	.77 2cts	.77 3wrp	.75 2mv	.71 2lzm	.72 1ccr
.65 3cpv	.66 3c2c	.64 1hmq	.72 1fdh	.74 3cpv	.71 2lhb	.80 2lh1	.77 2lhb	.77 1mbd	.74 2lh1	.71 1ccr	.72 3icb
.64 3c2c	.65 1fdh	.62 1ccd	.72 156b	.72 2lhb	.70 1cc5	.77 1mbd	.77 1ccr	.77 2mv	.73 2lhb	.71 1cc5	.72 2lhb
.64 2cts	.65 1hmq	.62 2lhb	.66 3wrp	.71 156b	.67 3wrp	.76 3wrp	.76 2ccy	.75 2cts	.72 2cts	.71 2cts	.70 1cc5
.62 1ccd	.64 1ccd	.62 3c2c	.66 1ccd	.70 1ccd	.67 1ccd	.76 3c2c	.75 3icb	.75 3c2c	.68 3hbb	.69 3wrp	.69 156b
.62 3hbb	.62 2lhb	.61 3wrp	.66 2lhb	.69 1cc5	.66 2lzm	.76 156b	.74 2mv	.75 3icb	.66 2cyp	.69 3icb	.68 3hbb
.62 2mv	.61 2cts	.61 2cts	.66 2cyp	.67 1fdh	.65 3c2c	.75 2mv	.72 3c2c	.74 1fdh	.65 1cc5	.69 2lh1	.67 3wrp
.60 3wrp	.60 3hbb	.61 1fdh	.63 2lzm	.66 3c2c	.64 2cyp	.72 2cyp	.71 156b	.73 2lzm	.64 3wrp	.67 2lhb	.67 1fdh
.60 2lhb	.59 3wrp	.59 3cpv	.61 2cts	.64 2cyp	.62 1fdh	.72 1hmq	.68 2lzm	.72 3cpv	.63 1hmq	.65 1hmq	.66 2cyp
.59 1hmq	.55 2cyp	.58 2mv	.58 2mv	.64 2lzm	.59 3cpv	.70 3cpv	.65 2cyp	.67 1hmq	.61 3icb	.62 2cyp	.64 3c2c
.54 156b	.54 2mv	.56 3hbb	.56 3cpv	.59 2cts	.56 2cts	.68 2lzm	.63 1hmq	.66 2cyp	.58 1ccd	.62 3c2c	.63 1ccd
.53 2cyp	.52 156b	.55 2cyp	.55 3c2c	.50 2mv	.46 2mv	.62 1ccd	.56 1ccd	.57 1ccd	.58 3c2c	.60 1ccd	.60 1hmq

Table IV-9: Ranking of Predictions by Standard Q

This table lists a ranking of the proteins based on standard Q scores for each combination of prediction method and secondary structure assignment method.

Ranking of Individual Predictions by Trimmed Q											
cf			gor			nn			alpps		
ks	rk	pdb	ks	rk	pdb	ks	rk	pdb	ks	rk	pdb
.84 3cln	.87 1cc5	.84 1cc5	.86 3cln	.93 3cln	.90 3cln	.97 3icb	.90 3cln	.90 3cln	.87 2ccy	.92 3cpv	.92 2ccy
.83 1cc5	.80 3cln	.83 2lh1	.85 1ccr	.91 3hhb	.89 3hhb	.95 3cln	.89 3hhb	.90 1cc5	.86 3cln	.90 2ccy	.86 2tmv
.81 2lh1	.79 1mbd	.81 3cln	.81 2lh1	.83 1mbd	.85 156b	.90 1fdh	.87 1cc5	.86 2lh1	.80 3cpv	.84 3cln	.84 3cpv
.74 1mbd	.79 2lh1	.79 3icb	.81 1hmq	.82 1hmq	.84 1hmq	.90 1cc5	.86 3cpv	.84 2lhb	.80 156b	.81 2tmv	.83 3cln
.74 2ccy	.76 2lzm	.79 1mbd	.80 1cc5	.81 2ccy	.81 1ccr	.88 1ccr	.85 3wrp	.82 3hhb	.80 1mbd	.79 3hhb	.80 2lzm
.73 1ccr	.75 2ccy	.78 2ccy	.76 1mbd	.80 1ccr	.81 3icb	.87 3hhb	.82 2cts	.81 3icb	.79 1ccr	.78 1mbd	.77 1mbd
.71 2lzm	.75 3icb	.69 1ccr	.75 3icb	.80 2lh1	.81 1mbd	.86 2cts	.82 1fdh	.81 2ccy	.78 2lzm	.77 156b	.76 2lhb
.70 1fdh	.72 3cpv	.67 2lzm	.75 3hhb	.77 3icb	.78 3hhb	.85 2lhb	.81 3icb	.81 2tmv	.78 1fdh	.77 1fdh	.76 3icb
.67 3icb	.70 1ccr	.64 156b	.75 2ccy	.76 3wrp	.77 2ccy	.84 2ccy	.80 2lhb	.81 156b	.77 2lhb	.73 2lzm	.76 2lh1
.65 3cpv	.67 3c2c	.62 2lhb	.72 156b	.73 3cpv	.72 1cc5	.81 2lh1	.80 2lh1	.80 1ccr	.76 2lh1	.73 1ccr	.74 2cts
.64 2cts	.65 1fdh	.62 3wrp	.70 1fdh	.73 2lhb	.69 2lhb	.79 1mbd	.80 1mbd	.79 1mbd	.76 2tmv	.73 2cts	.73 1ccr
.63 3c2c	.63 2lhb	.61 2cts	.67 1ccr	.71 156b	.68 1ccr	.79 3c2c	.80 2ccy	.79 2cts	.74 2cts	.73 1cc5	.70 1cc5
.62 1ccr	.63 1ccr	.61 1ccr	.66 2cyp	.71 1ccr	.67 2lzm	.78 2tmv	.79 1ccr	.77 3c2c	.70 3hhb	.71 3icb	.69 3hhb
.62 3hhb	.62 2cts	.61 3c2c	.66 3wrp	.71 1cc5	.66 3wrp	.76 156b	.78 2tmv	.77 1fdh	.67 2cyp	.70 2lh1	.69 156b
.62 2tmv	.61 1hmq	.61 1hmq	.65 2lhb	.66 1fdh	.65 2cyp	.75 3wrp	.75 3c2c	.77 3wrp	.67 1cc5	.68 2lhb	.69 1fdh
.60 2lhb	.59 3wrp	.60 3cpv	.64 2lzm	.65 2cyp	.64 3c2c	.74 1hmq	.71 156b	.76 3cpv	.65 3wrp	.68 3wrp	.67 2cyp
.60 3wrp	.58 3hhb	.60 1fdh	.62 2cts	.65 3c2c	.59 1fdh	.73 2cyp	.69 2lzm	.74 2lzm	.63 3icb	.63 1hmq	.66 3wrp
.56 1hmq	.54 2tmv	.59 2tmv	.58 2tmv	.65 2lzm	.58 3cpv	.72 3cpv	.67 2cyp	.68 1hmq	.61 1hmq	.63 2cyp	.63 3c2c
.53 2cyp	.54 2cyp	.56 3hhb	.54 3cpv	.60 2cts	.56 2cts	.69 2lzm	.63 1hmq	.66 2cyp	.57 3c2c	.61 3c2c	.62 1ccr
.52 156b	.48 156b	.55 2cyp	.52 3c2c	.50 2tmv	.46 2tmv	.62 1ccr	.56 1ccr	.58 1ccr	.56 1ccr	.59 1ccr	.57 1hmq

Table IV-10: Ranking of Predictions by Trimmed Q

This table lists a ranking of the proteins based on trimmed Q scores for each combination of prediction method and secondary structure assignment method.

Ranking of Individual Predictions by Trimmedt Q											
cf			gor			nn			alpps		
ks	rk	pdb	ks	rk	pdb	ks	rk	pdb	ks	rk	pdb
.92 3cln	.90 1cc5	.91 3cln	.92 3cln	.97 3hbb	.96 2lh1	1.00 3icb	.99 3hbb	.98 1cc5	.93 3cln	.93 2ccy	1.00 2ccy
.88 1cc5	.90 1mbd	.90 1cc5	.89 1hmq	.96 3cln	.96 3cln	1.00 3cln	.98 3wrp	.94 2lh1	.93 2ccy	.92 3cln	.92 2tmv
.87 2lh1	.86 3cln	.89 2lh1	.89 1ccr	.91 1hmq	.91 1hmq	.98 1cc5	.98 1cc5	.94 3cln	.87 156b	.92 3cpv	.91 3cln
.80 1mbd	.81 2lh1	.88 1mbd	.88 2lh1	.90 1mbd	.91 156b	.96 1fdh	.95 3cln	.93 2lhb	.85 1fdh	.88 1fdh	.88 2lzm
.78 2ccy	.81 2lzm	.84 2ccy	.81 1mbd	.85 1ccr	.87 1ccr	.93 2lhb	.93 2lhb	.90 3hbb	.83 1mbd	.87 3hbb	.86 3cpv
.76 2lzm	.79 3icb	.81 3icb	.81 1cc5	.85 2lh1	.87 3icb	.92 1ccr	.93 3cpv	.90 3wrp	.83 2lzm	.86 2tmv	.84 1mbd
.75 3icb	.78 2ccy	.74 1ccr	.79 3icb	.85 2ccy	.86 1mbd	.92 3hbb	.92 1fdh	.89 1mbd	.83 2lhb	.84 1mbd	.83 3icb
.73 1ccr	.75 3cpv	.72 2lzm	.78 3hbb	.83 3wrp	.81 3hbb	.90 2lh1	.90 1mbd	.89 156b	.83 2lh1	.82 156b	.82 2lh1
.71 1fdh	.75 1ccr	.68 156b	.78 156b	.82 3icb	.81 2ccy	.89 2cts	.86 2cts	.87 3icb	.81 1ccr	.81 2lzm	.80 2lhb
.65 2cts	.68 3c2c	.67 3wrp	.77 2ccy	.81 2lhb	.76 2lhb	.88 3wrp	.85 1ccr	.87 1ccr	.79 3cpv	.80 3icb	.79 2cts
.64 3wrp	.68 2lhb	.64 1hmq	.77 1fdh	.79 3cpv	.75 1cc5	.88 2ccy	.85 2ccy	.86 2tmv	.78 3hbb	.78 1cc5	.78 3hbb
.63 3c2c	.67 1fdh	.62 2lhb	.74 3wrp	.78 156b	.74 3wrp	.86 1mbd	.85 2lh1	.85 2cts	.78 2tmv	.77 2cts	.78 1fdh
.63 2tmv	.65 3wrp	.62 2cts	.71 2lhb	.76 1cc5	.73 1cc5	.80 2tmv	.84 2tmv	.85 2ccy	.76 2cts	.76 2lhb	.77 156b
.61 3cpv	.64 1hmq	.61 3cpv	.68 1cc5	.72 1cc5	.68 2lzm	.84 1hmq	.83 3icb	.83 1fdh	.71 3wrp	.75 2lh1	.76 1ccr
.61 2lhb	.61 2cts	.61 3c2c	.67 2cyp	.72 1fdh	.66 3c2c	.79 156b	.79 3c2c	.79 3cpv	.69 3icb	.75 3wrp	.74 1cc5
.59 1hmq	.59 1cc5	.60 1fdh	.64 2cts	.67 3c2c	.66 2cyp	.79 3c2c	.76 156b	.78 2lzm	.69 1cc5	.75 1ccr	.72 3wrp
.59 1cc5	.57 3hbb	.58 2tmv	.64 2lzm	.65 2lzm	.62 1fdh	.74 2lzm	.76 2lzm	.78 3c2c	.65 2cyp	.71 1hmq	.69 2cyp
.58 3hbb	.54 2cyp	.57 1cc5	.58 2tmv	.65 2cyp	.57 3cpv	.73 2cyp	.74 1hmq	.77 1hmq	.64 1hmq	.65 2cyp	.65 3c2c
.54 156b	.51 2tmv	.55 2cyp	.51 3c2c	.62 2cts	.56 2cts	.69 3cpv	.67 2cyp	.66 2cyp	.56 3c2c	.63 3c2c	.64 1cc5
.51 2cyp	.49 156b	.53 3hbb	.51 3cpv	.50 2tmv	.46 2tmv	.58 1cc5	.51 1cc5	.58 1cc5	.54 1cc5	.59 1cc5	.63 1hmq

Table IV-11: Ranking of Predictions by Trimmedt Q

This table lists a ranking of the proteins based on trimmedt Q scores for each combination of prediction method and secondary structure assignment method.

4.4. Discussion

Before discussing scoring predictions it is important to look at the data collected on the helices assigned by different secondary structure assignment methods. Counting the helices in the feature diagrams of Figure 4-5, there is a range of 141 (Kabsch-Sander), 150 (Richards-Kundrot), and 152 (PDB) total observed helices on the 20 proteins. The next section will look at differences in secondary structure assignments and the effects of trimming. After looking at assignments some comments can be made on the scoring of predictions.

4.4.1. Secondary Structure Assignments

4.4.1.1. Differences in Secondary Structure Assignments

Some of the differences in the total number of helices assigned by different methods can be explained by the "run-on" helix problem. Using Richards-Kundrot assignments, one can find examples of two adjacent helices which have no residue separating the C-terminal of one helix and the N-terminal of the next. There are even cases where the C-cap residue is the N-cap residue of the next helix (*e.g.*, residue isoleucine-76 serves both roles in leghemoglobin [2lh1]). This comes about because the Richards-Kundrot assignment is based on fitting α -carbon locations with those of an ideal helix. The kink at a given residue may allow it to fit two adjacent helices. A feature of Richards-Kundrot is that it treats each helix as a unit. Kabsch-Sander assignments, which use hydrogen bonding patterns, define a helical unit as a set of contiguous residues which are assigned as helical. In the leghemoglobin example, Richards-Kundrot assigns one helix from residue 57 through residue 76 and a second from 76 through 81, while Kabsch-Sander

assigns one long helix from 58 through 81. (See Figure 4-6.) A visual examination of the structure on a graphics terminal leads to an assignment of one (kinked) helix. In another example (sperm whale myoglobin [1mbd]), Richards-Kundrot assign serine-58 to two adjacent helices, while Kabsch-Sander define serine-58 as a one residue turn. In this case a visual inspection gives the impression that there are two distinct helices. Had Kabsch-Sander treated serine-58 as a helical residue, the separation of these two distinct helices would have been lost.

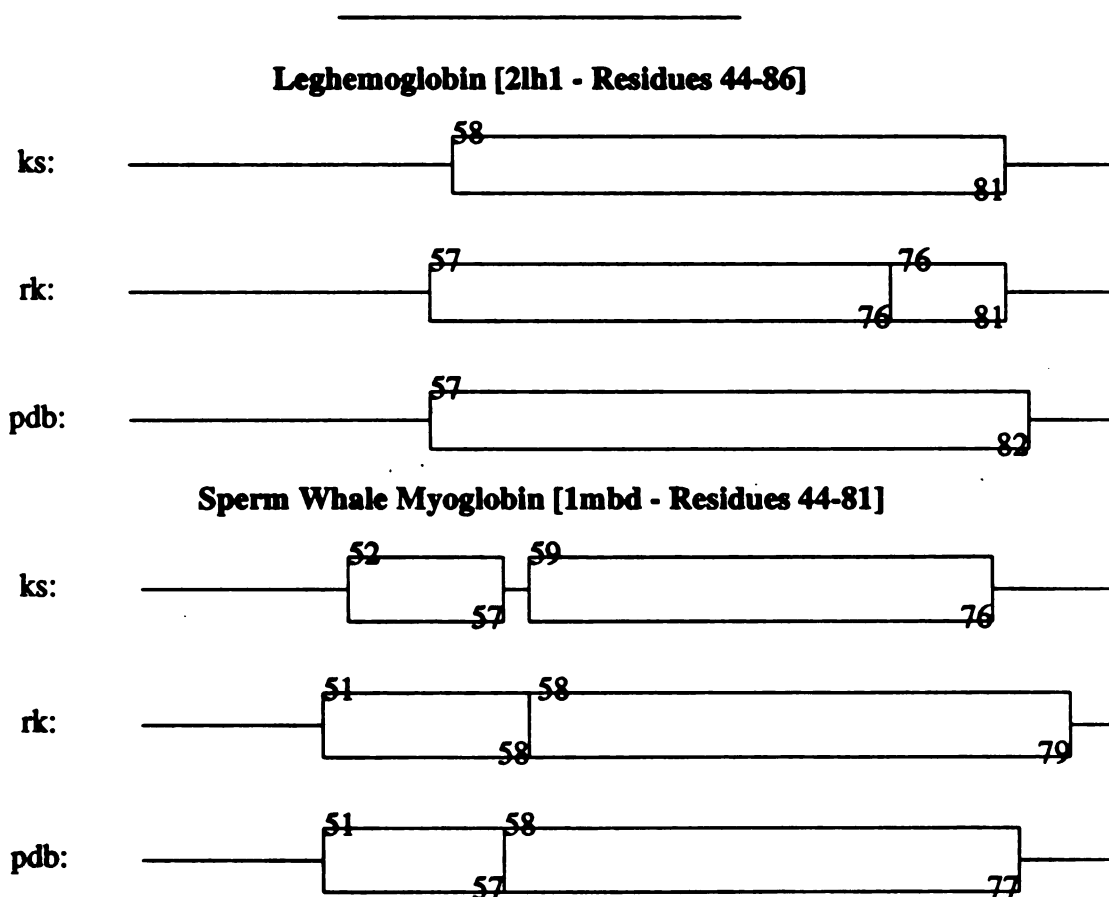


Figure 4-6: Run-on Helices

These two examples show places where the Richards and Kundrot assignment place one residue in two adjacent helices.

From a residue based perspective, the Kabsch-Sander and Kundrot-Richards assignments are not very different. After discounting capping residues through the trimming methods, all three assignments are very close in areas shown in Figure 4-6. The three are identical for 2lh1, and Kabsch-Sander and PDB are identical for 1mbd. Extended trimming brings all three into agreement on both subsequences.

Other examples of differences in helix counts arise from more divergent assignments. Many "stray" helices, ones which appear in only one of the three assignments, can be found. In Tobacco Mosaic Virus Coat Protein [2tmv], Kabsch-Sander finds a helix from residue 104 through residue 108. Neither of the other two assignments show a helix in the same area. On the other hand, Kabsch-Sander shows no helix before residue 20, while both Richards-Kundrot (8-13) and the PDB (9-14) assign a 5 residue helix closer to the beginning of the sequence. On balance, all three assignments find 6 helices, but there is a significant difference in where the helices are placed.

Table IV-2 shows the best aggregate Q scores between secondary structure assignments is only .91 (Richards-Kundrot and PDB). The .86 aggregate score (Richards-Kundrot and Kabsch-Sander) is low considering that 9 predictions have Q scores of .86 or better when compared against one (or more) secondary structure assignments.

Table IV-3 shows summary statistics for the cross assignment comparisons. When comparing Richards-Kundrot and Kabsch-Sander, only one sequence [2ccy, .94] had a Q score equal to or better than .90. The difference between these two assignments is emphasized by noting that there is no overlap between top six sequences in the PDB-KS and PDB-RK columns — even though the scores are at least .93. The hypothesis of proteins with secondary structure which is "easy" to assign can be dismissed. On the other

hand, at least one protein [3cpv] appears to fare poorly in all three comparisons.

4.4.1.2. Secondary Structure Assignments and Trimming

Although stray helices will not be tempered by trimming, run-on helices and moderate capping differences will. Table IV-2 shows what happens to the aggregate scores when trimming is applied. The biggest jump occurs for the Richards-Kundrot and Kabsch-Sander comparison (*e.g.*, .86 to .92 *Q*s). The range of *Q* scores decreases from .05 to .03. The aggregate scores continue to improve when extended trimming is applied. The aggregate *C* scores also improve with both forms of trimming. With extended trimming the correlation coefficient is .90 or better for all three comparisons.

The summary statistics in Table IV-3 show the same trends for both PDB cross comparisons. The scores in both PDB-RK and PDB-KS rise with trimming and the range falls. This continues with extended trimming. The RK-KS scores go up with trimming and the range drops slightly, but the trend does not carry over to extended trimming.

As seen in the discussion of Figure 4-4, trimming ameliorates helix capping differences between secondary structure assignments, but it does not change the scoring of stray helices. In the process of trimming capping residues from consideration in overlapping helices, some potential true positives are cut from the tallies. The false positives and false negatives of the stray helices are not changed by trimming. In the RK-KS comparisons, trimming improves the capping comparisons, but extended trimming exposes the stray helices. For two proteins [3cpv and 1ecd] the extended trimming scores are lower than the trimming scores.

4.4.2. Scoring Predictions

As a first look at the predictions, the helix counts show that CF (143), NN (140), and ALPPS (151) are similar to the assignment counts (141 to 152). The GOR count (90) is very low. On the other hand, the aggregate residue based scores of Table IV-2 place GOR above CF and close to ALPPS on all comparisons. This is one indication that residue based scoring does not necessarily reflect the fit between the topology of two secondary structures. This section will look more carefully at the residue based numbers before returning to the theme of scoring a prediction as an approximation of observed secondary structure.

4.4.2.1. Residue Scores

Although there is a ranking of methods (NN, ALPPS, GOR, CF) based on the Q scores, Table IV-7 reports all four methods having large (.25+) ranges which increase with trimming and extended trimming. The standard deviations and ranges for the GOR predictions appear to be slightly larger than the corresponding figures for each of the other three prediction methods. A review of the feature diagrams in Figure 4-5 shows that the GOR prediction for Tobacco Mosaic Virus Coat Protein [2tmv] contains no helices. The Q scores stay the same through trimming, and the C score is zero. This prediction could be considered as an outlier. When this prediction is removed from consideration the range and standard deviation statistics for GOR (see Table IV-8) look similar to those of the other three prediction methods.

Unlike the situation in assignments, it does appear that there are some proteins with secondary structures which are "easy" to predict. Looking at the rankings in Table IV-9, 3cln is consistently in the top five predictions. Other proteins lend themselves to good or

excellent prediction Q scores. For example, 2ccy is always in the top half of each list with Q scores ranging from .73 (CF-RK) to .87 (ALPPS-PDB). Similarly, some sequences [1ecd, 2cyp] were generally at the bottom of the Q rankings.

The rankings also show the influence of assignment methods on prediction scores. The best GOR Q score is for the prediction of 3hhb compared to the RK assignment (.88). Seven GOR-PDB scores are better than the one for 3hhb (.77), and six GOR-KS scores are better than 3hhb's .76. The .12 difference between the GOR-RK score and the GOR-KS score is significant. This difference is increased with trimming (.16) and extended trimming (.21).

4.4.2.2. Looking at the Feature Diagrams

A look at the feature diagrams for the GOR prediction and assignments on 3hhb gives more insight into interpreting the residue scores. As can be seen in the feature diagrams, GOR places one long helix from residues 60 to 137. The first two predicted helices correspond well with the first two helices of each of the three assigned secondary structures. The PDB and KS assignments show a third helix in the area of residues 37 to 42. Neither the RK assignment nor the GOR prediction have a helix in this area. Aside from a one residue turn (at 59), GOR shows two helices taking up the entire area between residues 51 and 137. RK places 5 helices (including a both parts of a run-on situation) in this same stretch, while both KS and PDB have 4 helices. Although the topology of the GOR prediction does not look very much like the layout of the RK assignment. The Q of the comparison is very good. Moreover, the Q score gets even better as turn residues from the RK assignment are discounted by extended trimming. Good Q scores do not tell the whole story.

One might argue that this secondary structure prediction is *prima facie* wrong because of the unreasonable length of the fourth helix. Q scores for reasonable predictions do not necessarily portray a complete evaluation a prediction as an approximation of an observed secondary structure. Consider the ALPPS and NN predictions for 1ccr. Assuming that any predicted helix of less than 20 residues is reasonable, only the ALPPS predicted A helix (residues 10-35) should be questioned. Since it is 25 residues, it may be too long. The NN-KS Q score (.86) is much better than the ALPPS-KS Q score (.75). Kabsch-Sander shows 4 helices — none of which are run-on. The ALPPS prediction does a good job of matching the B, C, and D helices. For purposes of using the prediction to generate a tertiary structure, the missing turn in the NN prediction may be more misleading than the extended helix in the ALPPS prediction.

4.5. Conclusions and Future Directions

This chapter looked at scoring by producing three sets of observed secondary structures and four sets of predictions over a set of 20 α/α proteins. The 7 sets of secondary structures were scored against the 3 observed secondary structures using standard residue based tallies and trimmed and extended trimming tallies. Scores were reported as both Q and C quality indices. Additionally, feature diagrams of the 140 secondary structures were used to examine the differences between secondary structure methods as well as to explore the concept of predictions as approximations of observed structure.

In comparing secondary structure assignment methods, there is a large range in the number of predicted helices and in the individual Q and C scores. Much of difference in Q and C scores was removed by trimming. Although the aggregate scores continue to improve after extended trimming, a couple of individual extended trimming scores are

lower than the trimmed versions. Trimming is not a method that simply adds points to residue scores. It allows scoring to discount residues which are subject to interpretation by different assignment methods.

Some proteins have observed secondary structures which can be well predicted by all four methods ("easy to predict"). Other proteins present observed secondary structures which are not well predicted by any of the four methods.

Residue based scoring does not give a complete measurement of how well a predicted secondary structure approximates an observed secondary structure. Residue tallies do not reflect any assessment of a prediction as a reasonable secondary structure. For example, GOR scores are generally close to ALPPS scores and better than CF scores, yet GOR shows a third less helices. Many GOR helices are more than 40 residues in length. The residue based penalty for missing turns relates solely to the length of the missed turn. Some complementary scoring system needs to be developed which gives additional information on the fit between the topology of the observed and predicted secondary structures.

Feature based scoring methods and a proposal for a method based on changes to a prediction which make it approximate an observed secondary structure are discussed in the next chapter.

Chapter 5

A Proposal for Feature-Based Scoring

5.1. Introduction

Chapter 4 addresses the problem of evaluating secondary structure predictions. The scoring techniques used in that chapter are variations of residue by residue comparisons of predictions and observations. One conclusion of that chapter is that residue based scoring fails to provide a complete evaluation of the relative "goodness of fit" between predicted and observed secondary structures. This chapter attempts to lay the groundwork for a feature based scoring system for secondary structure predictions on α/α proteins. This new evaluation technique would supplement residue based scoring systems.

Secondary structure prediction is not an end in itself. It is a valuable stepping stone on a path towards some other goal. Any secondary structure prediction should be evaluated in light of how well it furthers the underlying goal. For example, secondary structure prediction might be used to refine a model structure built by sequence homology to a known structure. In this case, the correct prediction of the ends of helices would be very important. For purposes of discussion in this chapter, it is assumed that the end goal is tertiary structure generation based on combinatorial methods (Cohen, Richmond, and Richards, 1979; Cohen and Kuntz, 1989).

At this time, residue based scoring is the accepted standard for evaluating secondary structure predictions. Residue based scoring is a valuable tool, but it does not adequately describe a comparison between predicted and observed secondary structure. The example shown in Figure 5-1 demonstrates a situation where two predictions have identical residue based tallies, and upon visual inspection, the two predictions are both reasonable

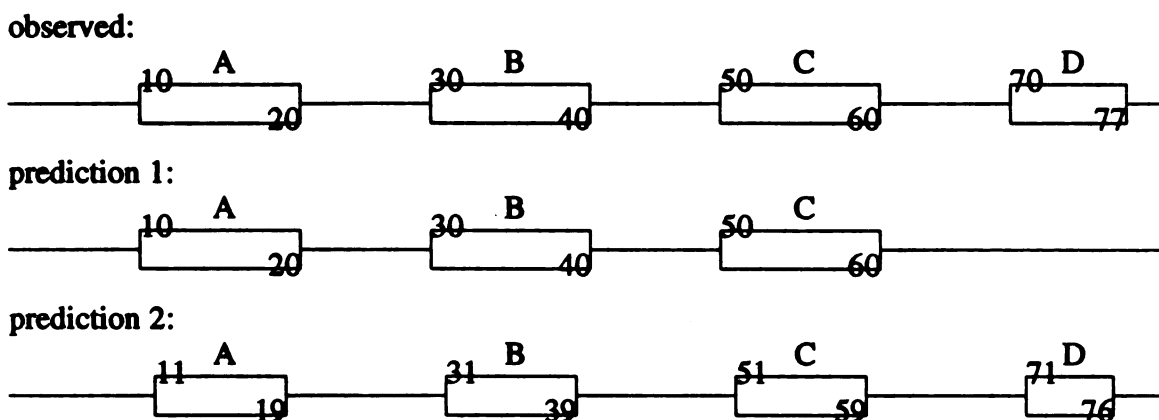


Figure 5-1: Residue Scoring Ambiguity

A secondary structure assignment and two predictions are shown in as feature diagrams. Both predictions are correct on all but 8 residues. Prediction 1, totally misses the D helix, while prediction 2 simply misses the single capping residues from each end of all four helices.

(e.g. no helix is less than 5 residues) yet different. In light of the goal of using secondary structure as a step towards a tertiary prediction, prediction 2 is a better approximation of the observed secondary structure.⁹ This assessment is more easily made when examining the feature diagrams, than when looking at the residue tallies. This chapter addresses the issue of devising a quantitative evaluation of a helical secondary structure prediction based on a comparison of the features in the observed and predicted secondary structures.

This idea is not new. As seen in the introduction to Chapter 4, early secondary structure predictions were evaluated by pairing observed and predicted helices. There was a sense of under- and over- predicted helices as well as realization of the difficulty in predicting the ends (especially the C-cap) of helices. These *ad hoc* evaluations were not

⁹ Using the *trimming* modifications described in chapter 4, the tallies do differ, and prediction 2 is the clear winner. Another example of identical tallies for two reasonable predictions can be assembled that are differentiated by trimming.

quantitative. As the number of known protein structures increased and more secondary structure prediction methods became available, quantitative residue based comparisons became the norm.

This chapter looks at two general types of feature based scoring methods. The first is based on the only known published attempt at quantitative feature based scoring. The remainder of the chapter introduces a new concept, *pseudo-string editing*, as a feature based measure. In order to develop this, an excursion through the established field of string edit distances is presented. The concept of pseudo-string editing is sketched with some examples, but additional work will be required to fully develop this new idea.

5.2. Registration and Analysis Methods

One general approach to evaluation using the feature diagrams consists of two major steps, *registration* and *analysis*. The predicted and assigned features need to be paired or *registered*.¹⁰ Some features will not have mates, while others have more than one. Figures 5-2 and 5-3 give three examples of different rules for registering predicted and observed features. Variations on these themes could also be considered. The registration then needs to be analyzed. Some comparisons can be made based on the registration alone. For example, does each assigned helix have one and only one mate in the predicted secondary structure? A second level of analysis looks in more detail at the pairings. For example,

¹⁰ Although the term *alignment* would describe this step, *alignment* has taken on a specific meaning in the context of protein sequence studies.

- How do the sizes of the assigned and predicted helices compare?
- How far apart are the mid-points of the respective assigned and predicted helices?

5.2.1. Taylor's Structure Percentage

In an appendix to an article on using templates to predict supersecondary structural motifs (Taylor and Thornton, 1984), Taylor (1984) developed a feature based score called a *structure percentage*. An example is shown in Figure 5-2. A registration of observed and predicted secondary structure units (strands and helices) is based on a minimum one residue overlap. The registration was used by Taylor to produce a *structure abstract* for the observed secondary structure and each prediction. Taylor's structure percentage was designed for comparing supersecondary structure motifs and not secondary structure predictions. Although the goals of the evaluation are different, Taylor's approach appears to be a good starting point — especially with respect to the issue of registration.

5.2.2. Alternative Registrations

Taylor's single residue overlap rule is dependent upon the secondary structure assignment method. If an observed helix and a predicted helix share only one residue, that residue is an N[C]-cap for one helix and a C[N]-cap for the other. As discussed in Chapter 4, the caps of helices are difficult to assign and predict. Why should an observed helix be registered with a predicted helix if they share but one residue in common? Figure 5-3 provides examples of two alternative registration rules which put more emphasis on correctly predicting the central region of the helix.

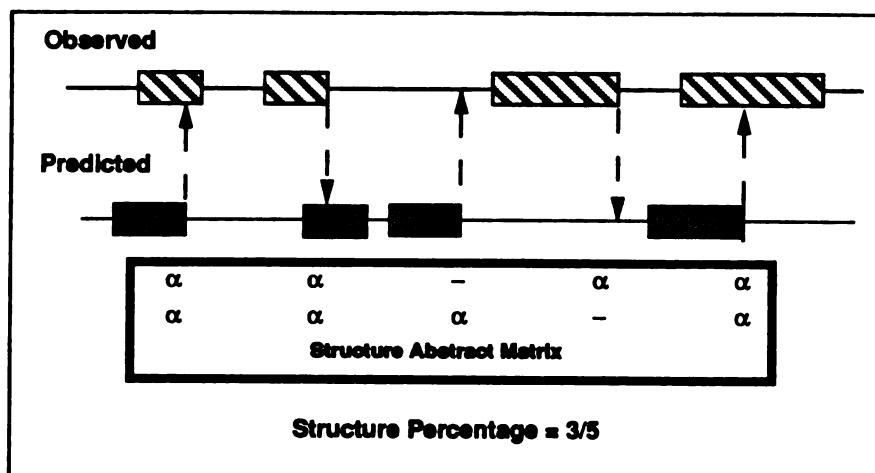


Figure 5-2: Taylor's Structure Percentage

First the predicted and observed features are registered. Starting at the N-terminal of the sequence, stop at the first C-terminal residue of a secondary structure unit (helix or strand). Go to the same residue in the parallel feature diagram and see if the same type of unit is found. If so, these two units are registered and a matching column can be entered in the structure abstract matrix. If not, the unit is not paired with another unit and a dash goes into the abstract. This is shown here for the third helices of both the assigned and predicted feature diagrams. Continue attempts at pairing using the N-most C-terminal of any unit which has whose pairing (or non-pairing) is not determined until all units have been considered.

The structure abstract matrix follows from the registration. A column is produced for each pairing (or non-pairing) arrow in the registration. One row is formed for the observed assignment and another for the prediction. The entries are α for helix and - for turn. The structure percentage is simply the number of columns with matching entries divided by the total number of columns.

Even with a more stringent set of registration rules, the calculation of the structure percentage is too simplistic for evaluating predictions. No regard is given to the type of error (*e.g.* a missing helix, extra turn, length of features) made by the prediction. All errors are given the same weight.

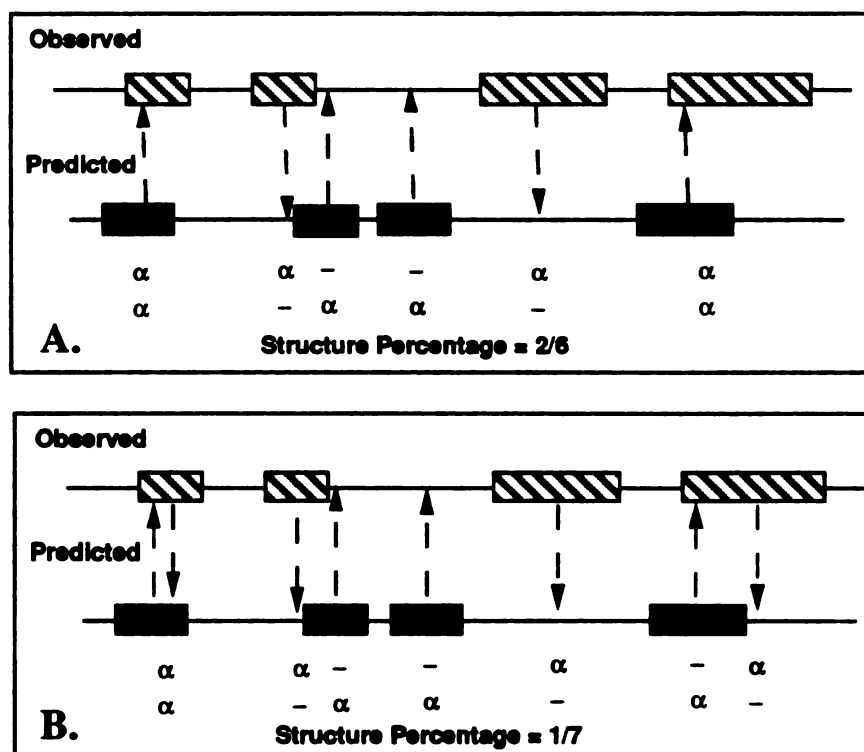


Figure 5-3: Alternative Registrations

Panel A- Single Central Residue Overlap - A registration that requires only a single residue overlap would be dependent upon the assignment algorithm. This example requires that a central residue of a secondary structure unit must overlap with a similarly assigned residue in the parallel feature diagram in order to define a pairing. In this definition the parallel residue can be in any position of the secondary structure unit. A broad definition of *central residue* (e.g., each secondary structure could have more than one central residue) lowers the dependence upon the assignment algorithm. It also focuses the scoring on accurate prediction of the cores secondary structure units.

Panel B- Double Central Residue Overlap - The single central residue overlap registration (defined in Panel A) does not guarantee that the cores of both of the paired units are covered by the extent of the opposite units. Although the core of the fourth predicted helix is covered by the fourth assigned helix, the core of the fourth assigned helix is not cover by the predicted helix. In this example, a pairing units occurs only when a central residue from each potential mate, finds a partner residue. A double central residue overlap rule demonstrates a high standard for pairing.

5.3. Pseudo-String Editing

When comparing a predicted secondary structure to an observed secondary structure for a given protein one might ask the question, what mistakes distinguish the prediction from the observation. Some mistakes are not as damaging to the long term goal (*e.g.*, tertiary structure generation) as others. Another way of asking this question is what alterations are necessary to take a predicted secondary structure and transform it into an acceptable approximation of the observed secondary structure. Consider the two predictions shown in Figure 5-4. Prediction 1 fails to show the turn between the B and C helices. An insertion of a short turn breaking helix BC into two helices would produce a good approximation by prediction 1. Prediction 2 has all four helices well registered with the observed secondary structure. The predicted A helix is more than twice as large as the observed A helix. Contracting the right (carboxy) half of the predicted A helix would produce an approximation by prediction 2. Some consideration might be given to expanding the left (amino) end of the predicted D helix, but unpredicted 6 residues may not be important to the approximation.

Each type of alteration could be given a cost in scoring points, and the prediction with the lowest score would be considered best. In the Figure 5-4 examples, the cost of inserting a turn might outweigh the cropping of a helix. A scoring method can be constructed based on this alteration scheme. Borrowing constructions from the well studied field of *string edit distances*, a new scoring method, *pseudo-string edit distances*, could prove to be a useful approach to feature based scoring.

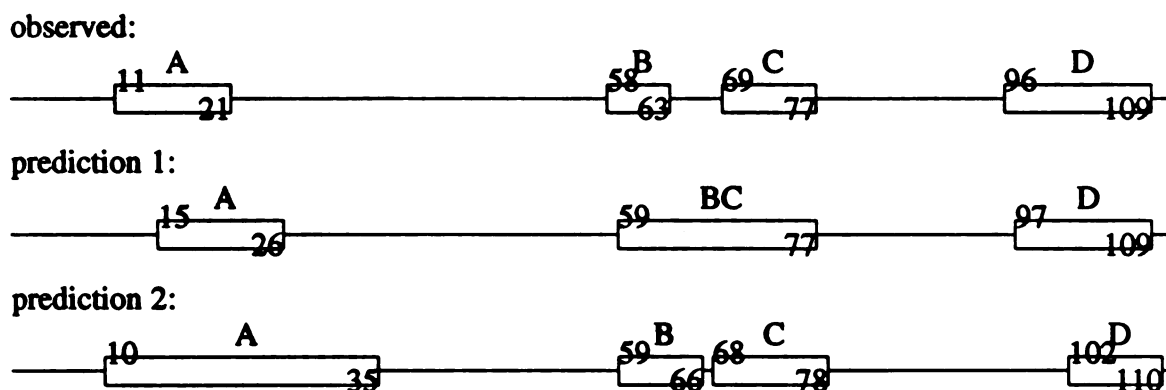


Figure 5-4: Feature Diagrams for Editing

These feature diagrams (based on 1ccr) are presented to consider what changes are necessary to transform each of the predictions into approximations of the observed feature diagram. This is discussed in the text.

5.3.1. String Edit Distances

This subsection gives an overview of string edit distances. It is presented to identify some of the issues that will need to be considered for an implementation of pseudo-string editing. A detailed examination of string editing can be found in Sankoff and Kruskal (1983).

INDUST-R-Y-	INDU--STRY
IN---TEREST	INTEREST--
delete D	substitute T for D
delete U	substitute E for U
delete S	insert R
insert E	insert E
insert E	delete R
substitute S for Y	delete Y
insert T	
IN--DUSTRY	INDUSTRY
INTEREST--	INTEREST
insert T	substitute T for D
insert E	substitute E for U
substitute R for D--	substitute R for S
substitute E for U	substitute E for T
delete R	substitute S for R
delete Y	substitute T for Y

Figure 5-5: String Edit Examples

Four examples of string edits are shown. In each case, a set of edits transforms **INDUSTRY** to **INTEREST**. **Bold** characters show places where the same characters can be aligned. Dashes show *insertions* and *deletions*. *Substitutions* are shown by different characters in the same position.

$$\begin{array}{ll}
 d(a,b) \geq 0 & \text{nonnegative} \\
 d(a,b) = 0 \text{ if and only if } a = b & \text{identity} \\
 d(a,b) = d(b,a) & \text{symmetry} \\
 d(a,b) + d(b,c) \geq d(a,c) & \text{triangle inequality}
 \end{array}$$

Figure 5-6: Distance Requirements

In this figure, d is a relation and a and b are objects (e.g., strings). These four requirements are necessary for a relation to be considered a mathematical distance metric.

Assume that you have two character strings, **INDUSTRY** and **INTEREST**.

Through the use of three string edit operations, *insertion*, *deletion*, and *substitution*, one string can be transformed into the other. Four examples are shown in Figure 5-5. By assigning a cost for each operation, totaling the costs of each set of operations, and determining the set of operations which minimizes the total cost, a metric can be placed on the distance between strings. The assignment of the individual operation costs needs to preserve notions of a metric which are summarized in Figure 5-6. These imply that the cost of an insertion must equal the cost of a deletion to preserve symmetry.

Table V-1 displays four cost schemes and their application to each of the four string edit examples. The first scheme is based on the assumption that all operations should cost the same. The second scheme also gives equal weight to operations after considering that a substitution is equivalent to an insertion plus a deletion. The third scheme encourages substitutions. There may be limitations placed on which characters are subject to substitution. A scheme could also give different costs to different sets substitutions as long as the metric constraints of Figure 5-6 are preserved. This is exemplified by the fourth scheme where **D-T** and **U-E** substitutions are preferred. Table V-1 demonstrates that the ranking of string edit sets is dependent upon the allocation of costs on

Table V-1-A			
Scoring Scheme	indel	substitute	comment
I	1	1	1 for each operation
II	1	2	<i>substitute = delete + insert</i>
III	2	3	reward substitutes
IV	2	2 for D-T or U-E 3 for others	reward substitutes to varying degrees

Table V-1-B				
Scoring Scheme:	I	II	III	IV
Edit Set				
INDUST-R-Y- IN---TEREST	7	8	15	15
IN--DUSTRY INTEREST--	6	8	14	13
INDU--STRY INTEREST--	6	8	14	12
INDUSTRY INTEREST	6	12	18	16

Table V-1: String Edit Scores

Four different scoring rules, I, II, III, and IV, (Table V-1-A) are applied to the four different string edits shown in Figure 5-5. The bold numbers in Table V-1-B show the lowest score for each set of scoring rules. The "shortest" string edit is dependent on the scoring rules.

each operation. Moreover, the allocation should be based on the underlying goals of calculating the string edit distance. For example, in studying homologies between protein primary sequences, some substitutions (*e.g.*, I [isoleucine] for L [leucine]) should be less costly than others (*e.g.*, G [glycine] for W [tryptophan]).

The actual distance between two strings is the *minimum* total cost of editing one string into the other. The problem of computing the optimal edit sequence is well studied (Masek and Patterson, 1983). Algorithms have execution times of order $O(jk)$ where the strings have lengths of j and k .

5.3.2. Pseudo-String Objects

In summary, string editing can be used to define a distance metric between strings. Characters are string edit *objects* which are manipulated by string edit *operations*. The metric is defined by an allocation of *costs* on operations. The distance is the minimum total cost of transforming one string into the other. Given this brief summary of string edit metrics, a parallel world of pseudo-string edits can be constructed. The objects, operations, and cost allocation schemes described in these three subsections are merely examples of a possible construction. Additional research will be necessary to produce a working pseudo-string editing metric.

Before describing pseudo-string editing, it is important to note that I am not proposing to look at secondary structure assignments and predictions as character strings of As and ts. Though one could take At strings (as shown in Figure 4-3) and produce string edit distances, these distances would be another residue based score. Pseudo-string objects need to be at the feature and not residue level of abstraction.

Feature diagrams can be viewed as pseudo-strings composed of *helix* and *turn* objects. Although feature based scoring is meant to complement residue based scoring by not considering individual residues, helix and turn units need some length attribute. This could be accomplished by having three sizes for each object type giving six different type pseudo-string objects (*PSOs*):

short helix
medium helix
long helix
short turn
medium turn
long turn.

The exact definitions of these PSO types need to be established.

In string editing, the string objects are discrete characters. In pseudo-string editing, the objects are fuzzy.¹¹ The number of residues represented two PSOs of a given PSO type may vary. For example, in Figure 5-1 the observed A helix is 10 residues long while the prediction 2 A helix has 8 residues. The same type of PSO (*e.g.*, a medium helix) could be used to represent each helix. Both the observed secondary structure and prediction 2 have matching pseudo-string representations.

5.3.3. Pseudo-String Operations

Continuing with a sample pseudo-string edit system, six edit operators (*PSOPs*) are shown in Figure 5-7. In the following descriptions, a turn is used as the sample object, but it is easy to see how the operators would act on any of the other five objects.

¹¹ This is not a reference to fuzzy set theory (Kandel, 1982). At this point in time, I am aware that the field of fuzzy set theory is worth exploring but I am leaving that to future work.

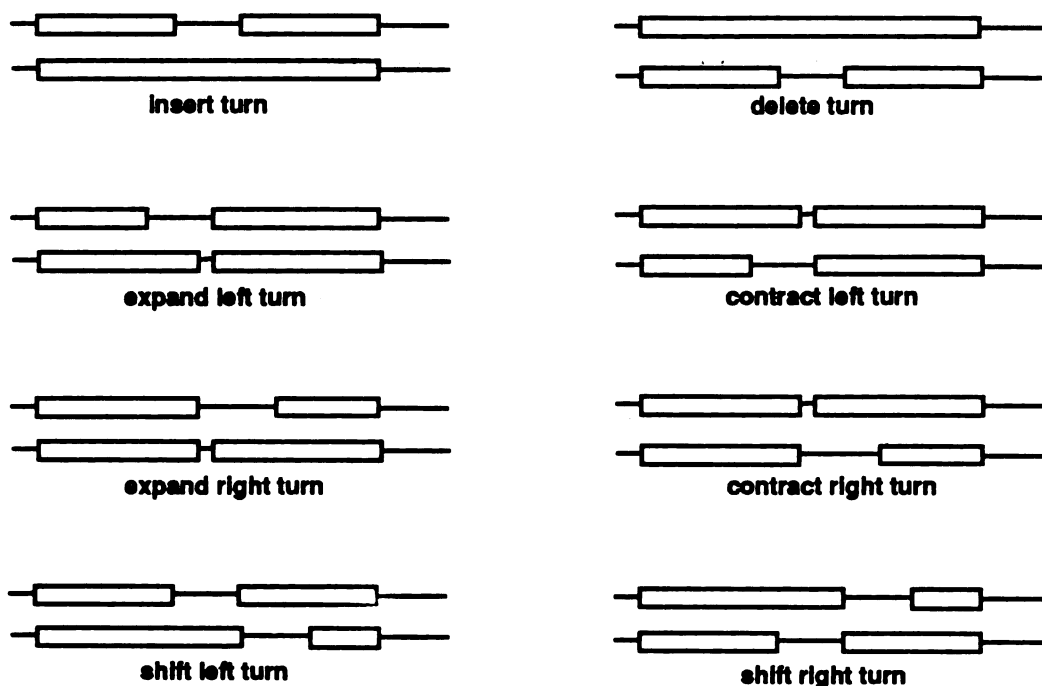


Figure 5-7: Pseudo-String Editing Operations

Three pairs of possible pseudo-string editing operations are shown. In each case, the operation takes the bottom feature diagram and produces the top feature diagram. In these examples involving turns, only the helix caps which actually touch the modified turn are changed. The caps at positions A and B are unchanged.

insert short turn- breaks the existing helix into two smaller helices and a short turn.

The N-cap of the first new helix and the C-cap of the second match the caps of the original single helix.

delete short turn - removes the turn and concatenates the two surrounding helices.

shift right turn - moves the turn to the right by lengthening the neighboring left helix and shrinking the neighboring right helix. The N-cap of the left helix and the C-cap of the right remain fixed.

shift left turn - moves the turn to the left by shrinking the neighboring left helix and lengthening the neighboring right helix. The N-cap of the left helix and the C-cap of the

right remain fixed.

expand left small turn - takes a small turn and leaves a medium turn. The C-cap of the adjacent left helix is moved to make room for the larger turn.

contract left medium turn - would reverse the previous operation.

expand right small turn - takes a small turn and leaves a medium turn. The N-cap of the adjacent right helix is moved to make room for the larger turn.

contract right medium turn - would reverse the previous operation.

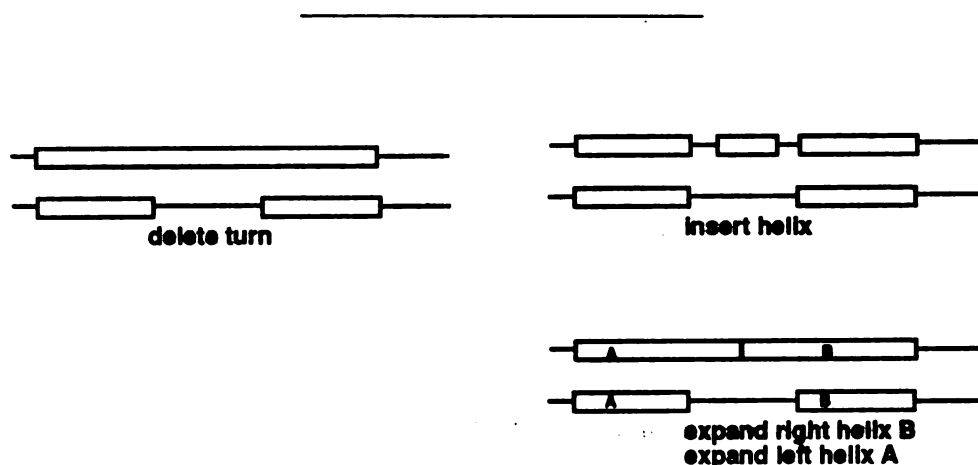


Figure 5-8: Nonreverse Pseudo-String Editing Operations

Two pseudo-string editing operations are shown. In each case, the operation takes the bottom feature diagram and produces the top feature diagram. This example shows that a *delete turn* is not the reverse of an *insert helix*. Similarly, applying *expand-left helix* and *expand-right helix* produces a slightly different result from the *delete turn*.

These operators can be viewed as pairs of reversible operations. Some of the reverse operators are necessary because of the requirement that either pseudo-string can be edited to match the other. As shown in Figure 5-8, a *delete turn* is not the same as an *insert helix*. The number of operators could be reduced if both pseudo-strings could be

edited simultaneously until the resulting pseudo-strings were the same. Other operations, *e.g.*, *shift*, are redundant but convenient.

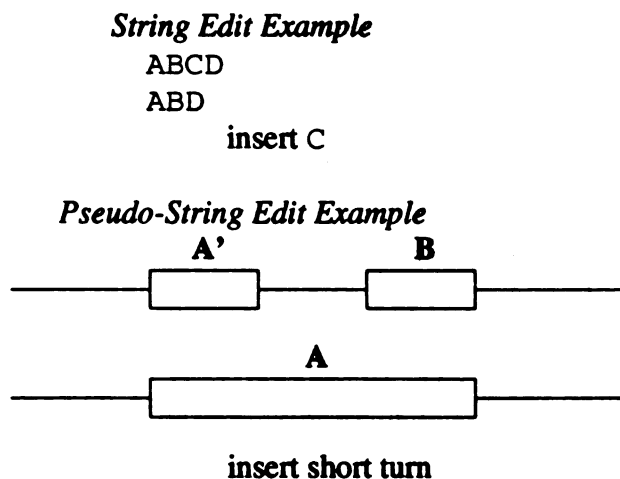


Figure 5-9: PSOPs and Adjacent PSOs

In both examples an insert is applied to the lower [pseudo-]string to produce the upper [pseudo-]string. Unlike string edit operations, pseudo-string edit operations can affect the object type of more than one object in a single operation.

Unlike string edit operators, a PSOP can affect more than one object during the course of a single operation. In Figure 5-9, the string insert pushes D to the right, but does not change the B or D characters. The pseudo-string insert splits long helix A into medium helix A' and medium helix B.

5.3.4. Pseudo-String Operational Costs

The final component is composed of the costs for each operation. Just as in the case of string edits, the metric constraints of Figure 5-6 must be observed. Although one could argue that the cost of inserting a helix should be different than the cost of deleting a helix, the symmetric relationship between these two operations demands that they be scored the same. This does not mean that inserting a helix must cost the same as insert-

ing a turn.

The underlying intended use for the secondary structure prediction is very important. For example, if the underlying use is tertiary structure generation based on a combinatorial approach (Cohen, Sternberg, and Taylor, 1980; Cohen and Kuntz, 1989), then the cost of changing the size of a helix should be less than the cost of inserting a helix. This is because the tertiary fold will not be changed as much by a shortened helix as it would by the presence of an additional helix. Although ordering the relative costs of indels (insertion or deletion), shifts, and size changes (expansion and contraction) flows from the underlying use, quantifying this ordering is much more difficult.

5.4. Conclusions

A feature based scoring method would complement residue based scoring for secondary structure predictions. Feature based scoring allows the purpose of undertaking a secondary structure prediction to be reflected in the scores.

Two general classes of feature based scoring are worth considering. A simple version of a "registration and analysis method" using any of three registration methods and Taylor's structure percentage analysis gives some additional information, but the analysis (structure percentage) is too simple. More work could be done on developing a more sophisticated and useful analysis scheme.

Pseudo-string edits are not string edits on a turn-helix string representation of residues. The pseudo-string edit distances show promise, but require much additional work before they can demonstrate that promise. This future work for pseudo-string edit distances includes:

- complete definitions for pseudo-string objects;
- complete definitions for pseudo-string operators; and
- costs for each operator-object combination.

Chapter 6

A Segment Based Approach to Protein Secondary Structure Prediction[†]

6.1. Introduction

Under suitable conditions, many proteins adopt a compact, globular fold which is dictated by the amino acid sequence of their polypeptide chain. However, the precise relationship of sequence to structure remains unresolved. Substantial experimental and theoretical efforts have been directed at understanding the protein folding problem. Experimentalists have uncovered evidence of native-like intermediates along the folding pathway (Hughson, Wright, and Baldwin, 1990; Goto and Fink, 1990; Ptitsyn *et al.*, 1990). Theoretical methods are also being used to explore possible folding pathways, and to gain an understanding of the forces which stabilize folded proteins. To date, three approaches have been employed: energy minimization and/or molecular dynamics (Levitt and Warshel, 1975; Nemethy and Scheraga, 1977; Weiner *et al.*, 1984; McCammon, Gelin, and Karplus, 1977; Karplus and McCammon, 1981; Beveridge and Jorgenson, 1986), Lattice models (Skolnick and Kolinski, 1989), and semi-empirical sub-structure condensation (Cohen, Richmond, and Richards, 1979; Cohen, Sternberg, and Taylor, 1980).

Energy minimization techniques and molecular dynamics approaches offer the promise of a rigorous treatment of the inter- and intra-molecular forces in protein structures. However, several practical details of these methods remain unresolved. The

[†] This chapter is similar to the manuscript of a paper prepared for publication: Scott R. Presnell, Bruce I. Cohen, and Fred E. Cohen; **A Segment Based Approach to Protein Secondary Structure Prediction.**

current potential functions do not provide an accurate representation of energy states, bulk solvent is not handled adequately, and optimization algorithms cannot sample the entirety of conformation space. Moreover, computing power enables us to visualize no more than 2 or 3 nanoseconds of dynamic variation while protein folding requires milliseconds to seconds (Udgaonkar and Baldwin, 1988; Roder, Eloeve, and Englander, 1988).

Monte Carlo simulations utilizing simplified lattice frameworks have been performed in an attempt to elucidate general rules for globular protein folding. These simulations have incorporated several different lattice types (cubic, diamond, and "210" or "knights walk") to simulate the folding pathways of four helix bundles such as apoferritin and somatotropin (Sikorski and Skolnick, 1989), and β -barrel proteins such as plastocyanin (Skolnick and Kolinski, 1990). These methods provide an interesting vignette of the possible folding pathways. However, the simulations currently require that position specific conformational preferences be built into the backbone atom representation of each simulated sequence.

Semi-empirical methods also suffer some limitations; but they have found utility in the development of structure models. The present semi-empirical condensation methods are based on a hierarchical definition of globular proteins. The classical hierarchy structure presents the following categories: primary, secondary, and tertiary structure. Typically, condensation schemes fold primary structure into secondary structure, then secondary structure is assembled into tertiary structure. This strict ordering is not intended to be an accurate reflection of a protein folding pathway: some aspects of secondary structure formation may be influenced by specific tertiary interactions. However, current

work on the molten globule state indicates that the intermediate stages in protein folding show a large fraction of the secondary structure apparent in the native state (Hughson, Wright, and Baldwin, 1990; Goto and Fink, 1990; Ptitsyn *et al.*, 1990). The body of work that focuses on the transition of secondary structure to tertiary structure will not be considered here (Cohen, Richmond, and Richards, 1979; Cohen, Sternberg, and Taylor, 1980; Cohen, Sternberg, and Taylor, 1982). This work concentrates on advanced techniques for relating primary structure to secondary structure.

Several groups have reported methods for the prediction of secondary structure in globular proteins (for a review see: Schulz, 1988). These methods group loosely into two classes. The collection of known structures provides a database of information about the propensity of the individual amino acid to reside in specific types of secondary structure. The first class of methods is based on statistical analyses of this data. The progenitor of these methods was developed by Chou and Fasman (Chou and Fasman, 1974; Chou and Fasman, 1978). Specialized treatments of context in the amino acid sequence (e. g. Markov dependence) or the information content within the sequence were developed into prediction methods by Garnier, *et al.* (referred to here as the GOR method, Garnier, Osguthorpe, and Robson, 1978; Gilbrat, Garnier, and Robson, 1987). The most advanced of these methods now combines many predictive schemes, or combinations of predictions from sequences homologous to the sequence of interest (Levin and Garnier, 1988; Nishikawa and Ooi, 1986). Recently, computational neural networks have been used to investigate the mapping of protein sequence to secondary structure (Qian and Sejnowski, 1988; Holley and Karplus, 1989; Kneller, Cohen, and Langridge, 1990).

The second class of methods rely on biophysical principles as a basis for the predic-

tion of interactions among the amino acids. This approach was first outlined by Nagano (Nagano, 1973; Schulz *et al.*, 1974), and Lim (Lim, 1974a; Lim, 1974b). For globular proteins, these principles include compactness of form, the presence of a hydrophobic core, or cores, and a polar outer shell. The geometries inherent in the two archetypes of secondary structure, α -helix and β -sheet, afford restrictions on the types of amino acid side- and main- chain interactions. Our methods for the generation and analysis of patterns that recognize sequence-structure correlates have followed from this second class of techniques.

Long range interactions are believed to play a critical role in the formation of complete tertiary structure. Kabsch and Sander (1984) were among the first to note that identical or similar sequences of up to five residues can adopt decidedly different three dimensional structures. Hence, five residues of context is not enough to define unique three dimensional structure. Classical secondary structure prediction methods typically achieve results of no greater than 65% accuracy. This limitation has often been attributed to the absence of long range interactions in the prediction algorithms. At first glance, the specification of long range interactions seems a daunting problem. But, general knowledge of the nature, or types of long range interactions expected can be included into predictive schemes. In previous work, Cohen *et al.* (1986) described the specification of regular expression patterns that could incorporate long range interactions from the estimated turn distribution in proteins. The patterns developed under the PLANS system were able to accurately locate turns in the three classes of globular proteins: (α/α , α/β , β/β) with success rates approaching 90%.

We report here the extension of the PLANS work to predict regular secondary struc-

ture in α/α proteins. This advancement involved two explicit developments. First, the general problem of regular secondary structure prediction was divided into subtasks. Regular expression patterns were developed to recognize the individual component parts of helices, the amino terminus, the core, and the carboxy terminus. Those patterns were designed to function in an autonomous fashion. Second, *A Language for the Prediction of Protein Substructures* (ALPPS) was formulated to coordinate the development and analysis of *meta*-patterns: patterns of patterns. In the case of helical structure prediction, meta-patterns coordinate the recognition of the helix components.

The patterns developed in this work recognize three helical features to differing extents. Scoring the success of feature prediction, we are able to detect 95% of the helix core structures, with a 10% over-prediction rate (for every 10 helix core features predicted correctly, one over-prediction will occur). N- and C-terminal helix caps are much more difficult to recognize. One half of these features are detected, but a 25% over-prediction rate is observed. The recognition of individual features at these rates produces prediction accuracies that exceed the statistically based prediction algorithms. The residue based scores from the pattern based work presented here do not surpass the scores obtained by the latest neural network algorithms. However, pattern based methods allow complete inspection, and consequent structural interpretation, of the tools used to predict structure. Interpretation of the internal weights of neural network connections is difficult for all but the simplest architectures.

In summary, we can analyze protein structures for specific features of helical secondary structure, and develop patterns to recognize those features. We can also orchestrate the recognition of these features in specific orderings via a language that describes meta-

patterns. The developments described in this work mark the emergence of a prediction scheme that uses the hierarchical organization in protein structure to facilitate the the inclusion of sequential, long range interactions.

6.2. Methods

Definitions. Our previous work on the prediction of turns considered protein sequences from three structural classes of proteins α/α , α/β , β/β (Levitt and Chothia, 1976). In this work, we have focused on the features that stabilize individual α -helices within protein structures constructed primarily of helices (α/α protein structures). This is not intended to be a representative sampling of all globular proteins. Instead, it provides a limitation on the variety of structural interactions to identify, characterize, and predict. For α/α proteins the polypeptide chain has only two conformational states: helix and turn. All forms of helical structure (3_{10} , α , and π helix) are treated identically. Similarly, any region that interconnects regular secondary structure is considered a turn.

One drawback to class specific structure prediction algorithms is the problem of determining the structure class of the protein under scrutiny. Protein class determination in the absence of a crystal structure remains a difficult problem in biochemistry (Sheridan *et al.*, 1985; Klein and Delisi, 1986; Deleage and Roux, 1987). However, work in sequence analysis (Bowie *et al.*, 1990), and machine learning techniques are beginning to provide new algorithms for this task (Kim, 1991). Advanced experimental methods (Lee *et al.*, 1990) are also providing new methods beyond the classical circular dichroism techniques for determining structure class from experimental data (Johnson Jr., 1990).

Data Sets. Twenty polypeptide chains from the collection of α/α proteins in the Brookhaven protein data bank (PDB, Bernstein *et al.*, 1977) were pooled and split into

two sets of ten chains each. One set was used for the development and analysis of patterns. The sequences of the other set were sequestered from examination for the purposes of unbiased evaluation after pattern development was complete (Table VI-1).

Except for the structure of Tobacco Mosaic Virus Coat protein, the data sets were selected from crystal structures that have an atomic resolution greater than 2.5 Å. Identity scores were evaluated after alignment with the multiple sequence alignment program PIMA (Smith and Smith, 1990). The greatest amount of identity between any two polypeptide chains was 42% (between fetal human hemoglobin γ chain and human adult human hemoglobin β chain). The next highest identity was 27%.

Table VI-1: Proteins Used for Pattern Development and Analysis

Development Set		
PDB	Protein	
1ccr	Cytochrome C	(Ochi <i>et al.</i> , 1983)
1fdh	Human Fetal Hemoglobin (gamma chain)	(Frier and Perutz, 1977)
2ccy	Cytochrome C prime	(Finzel <i>et al.</i> , 1985)
2cts	Citrate Synthase	(Remington, Wiegand, and Huber, 1982)
2lh1	Leghemoglobin	(Arutyunyan <i>et al.</i> , 1980)
2lhb	Hemoglobin V	(Honzatko, Hendrickson, and Love, 1985)
2lzm	T4 Lysozyme	(Matthews, 1975)
3c2c	Cytochrome C2	(Bhatia, 1981)
3cln	Calmodulin	(Babu, Bugg, and Cook, 1988)
3cpv	Parvalbumin B	(Moews and Kretsinger, 1975)
Test Set		
156b	Cytochrome B562	(Lederer <i>et al.</i> , 1981)
1cc5	Cytochrome C5	(Carter <i>et al.</i> , 1985)
1ecd	Erythrocyruorin	(Steigemann and Weber, 1979)
1hmq	Hemerythrin	(Stenkamp, Sieker, and Jensen, 1983)
1mbd	Myoglobin	(Phillips and Schoenborn, 1981)
2cyp	Cytochrome c Peroxidase	(Finzel, Poulos, and Kraut, 1984)
2tmv	Tobacco Mosaic Virus Coat Protein	(Namba, Pattanayek, and Stubbs, 1989)
3hhb	Human Hemoglobin (alpha chain)	(Fermi <i>et al.</i> , 1984)
3icb	Vitamin D-dependent Calcium-binding Protein	(Szebenyi and Moffat, 1986)
3wrp	Trp Aporepressor	(Lawson <i>et al.</i> , 1988)

Secondary structure assignment. Secondary structure assignment was performed with the aid of the program DEFINE (Richards and Kundrot, 1988). Briefly, the algorithm within DEFINE evaluates the fit of actual secondary structure backbone C_{α} atoms to an ideal secondary structure through a difference distance matrix technique. Specific structure "masks" are used to make secondary structure assignments to the three dimensional structure. The cumulative root mean squared difference between the ideal and actual structures is used to evaluate the assignment of the desired structure type. In this study, 0.75 \AA was used as the maximum RMS difference between the ideal and actual helical structures. This yielded a secondary structure assignment that was consistent with most authors assignments. While other programs are available for structure assignment (Kabsch and Sander, 1983a; Sklenar, Etchebest, and Lavery, 1989), DEFINE was employed because it more closely matched the crystallographers' helical structure assignments.

Evaluation. The accuracy of the algorithms presented here were evaluated using several measures. Feature based scoring was used to evaluate the predictive capabilities of the PLANS patterns. If a pattern appears within four residues of the targeted feature of structure (e.g. the core of an α -helix), that event is considered a true positive (*TP*), otherwise the event is registered as a false positive (*FP*). The absence of a prediction for a targeted feature is registered as a false negative (*FN*). Since PLANS patterns do not explicitly predict the absence of a structure element, true negatives (*TN*) cannot be recorded. We represent the success of a feature based prediction as the quotient of the correctly predicted features and the total number of features. This index, often referred to as Q_2 , is defined as follows (Schulz and Schirmer, 1979): This index all but ignores the possibility of over-prediction, so we also report the quotient of over-predicted

$$Q_2 = \frac{TP}{TP+FN} \quad (3)$$

features (*FP*) and the total number of features (referred to simply as *O*):

$$O = \frac{FP}{TP+FN} \quad (4)$$

The primary goal of the patterns and algorithms developed in this work was the prediction of helical features. However, structural feature analysis is not a commonly accepted standard for algorithm comparison. Residue based scoring, in which each residue is given equal weight, was used to evaluate the success of helical predictions for the purposes of comparison to other algorithms. Residue based scoring and *trimming* are discussed in Chapter 4.

Pattern Development. In our previous work, PLANS was developed to facilitate expression of sequence-structure correlates as regular expression patterns (Cohen *et al.*, 1986; see also Chapter 2). We continue to use that regular expression syntax and algorithm as the basis for turn and helix component pattern development in this work.¹²

Turn Patterns. In the ALPPS assisted prediction of regular secondary structure, the first step is to predict the likely locations of turns. This is done using the composite turn prediction pattern, TU, developed previously (Cohen *et al.*, 1986). TU is the synthesis of several patterns that predict turn location. A cluster of hydrophilic residues provides the

¹² The PLANS patterns described here, which show primarily sequential relationships, are presented in a simplified syntax. The general PLANS pattern syntax is *pattern_name* : *pattern*.

Φ represents one of a set of hydrophobic residues; A, V, I, L, M, C, K, F, W, or Y.

Ψ represents one of a set of hydrophilic residues; A, D, E, H, K, N, Q, R, S, or T.

- represents one of the acidic amino acids; D or E.

+ represents one of the basic amino acids; K or R.

. represents any amino acid.

[X
Y] represents residues X or Y.

Ψ represents the complement of the set Ψ.

strongest signal. A pattern that explicitly recognizes the special role of proline in initiating and terminating helices is also included. Weaker turns are recognized as an interrupted collection of hydrophilic residues spaced appropriately from the strongest turn indications. The relationships between the constituent aspects of TU and the composite pattern is shown in Figure 6-1.

Helix Component Patterns. The characteristics of helical residues vary as a function of their position within the helical structure. There are often differences in the spatial distribution of residues (e.g. the amphiphilic character of a helix), as well as differences in residue types between the center and the termini. Several authors have noted specific sequential characteristics of helices from globular proteins (Richardson and Richardson, 1988), and others from membrane associated proteins (Rees, DeAntonio, and Eisenberg, 1989; Sternberg and Gullick, 1990). In order to efficiently describe patterns that recognize diverse α -helical characteristics in α/α proteins, we have chosen to subdivide helices into three specific components for study: the central section or core region of the helix, the amino terminal area of the helix (referred to hereafter as the N-cap) and carboxy terminal area (C-cap). While the principle of structure subdivision is general, the PLANS patterns subsequently described are specific to soluble, globular proteins. The patterns would have to be reformulated for integral membrane proteins.

Helix Core. The criteria for the prediction of the helix core regions incorporate several different biophysical properties (Figure 6-2). Sequential placement of hydrophobic residues in a pattern suggestive of a *hydrophobic patch* on one face of a helix would facilitate the creation of a hydrophobic interaction with another part of the protein (Schiffer and Edmundson, 1968).

PATCH: $\Phi.. \Phi \Phi.. \Phi$

Acidic and basic residue can be placed in such a way as to generate a *charged pair* on one surface of the helix.

PAIR: $-...+$

Empirical rules can be used to identify a putative helix-helix interaction site (Richmond and Richards, 1978).

INTER: $\Phi \Psi \Psi \left[\begin{array}{c} A \\ G \end{array} \right] \Psi \Psi \Phi$

Long stretches of hydrophilic residues indicate a turn or loop regions on the surface of the molecule. By process of elimination the remaining areas are likely to contain helices.

NO-PHIL: $\overline{\Psi \Psi \Psi \Psi \Psi \Psi \Psi \Psi}$

We have found that these criteria are effective when given equal weights (i.e. not hierarchical). These individual PLANS patterns, along with others, are collected into an aggregate helix core pattern called HCORE.

Helix N-cap. Three sub-types of patterns have been developed to describe the amino terminal capping sites of the helices (Figure 6-3). Those patterns that give the most reliable indication of a helix N-cap stem from the combination of a residue commonly found at the exact helix N-cap site (in this case N, S, T, or D) and a proline residue (Richardson and Richardson, 1988).

NSIMPLE: $\left[\begin{array}{c} N \\ S \\ D \\ T \end{array} \right] P$

Patterns containing a residue from the set N, D or S, one or more acidic residues one to three residues from the N-cap site, and one or more large hydrophobic residue five to six residues from the N-cap site are also highly reliable.

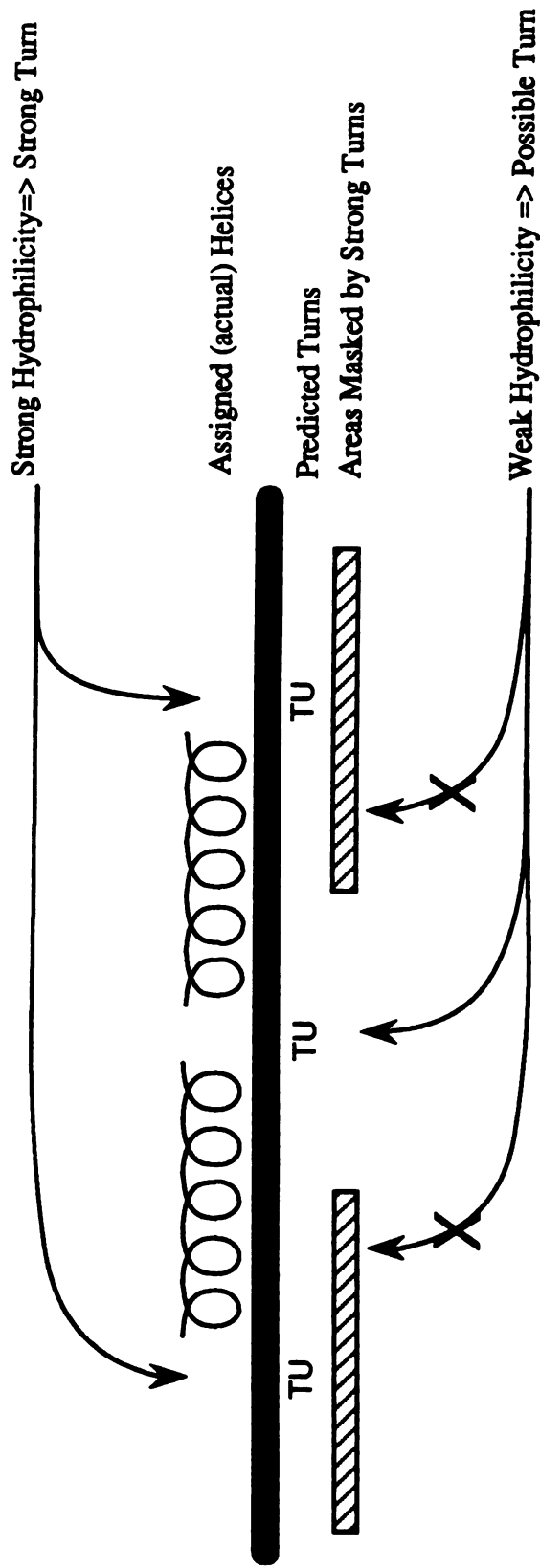


Figure 6-1: Turn Prediction Reviewed

A high concentration of hydrophilic residues often signals the presence of a turn in globular protein sequences. A block of sequence around these strong indicators are "masked" from further consideration. Only those regions of the sequence free from the mask are considered when examining the sequence for less likely turn indications (medium concentrations of hydrophilic residues).

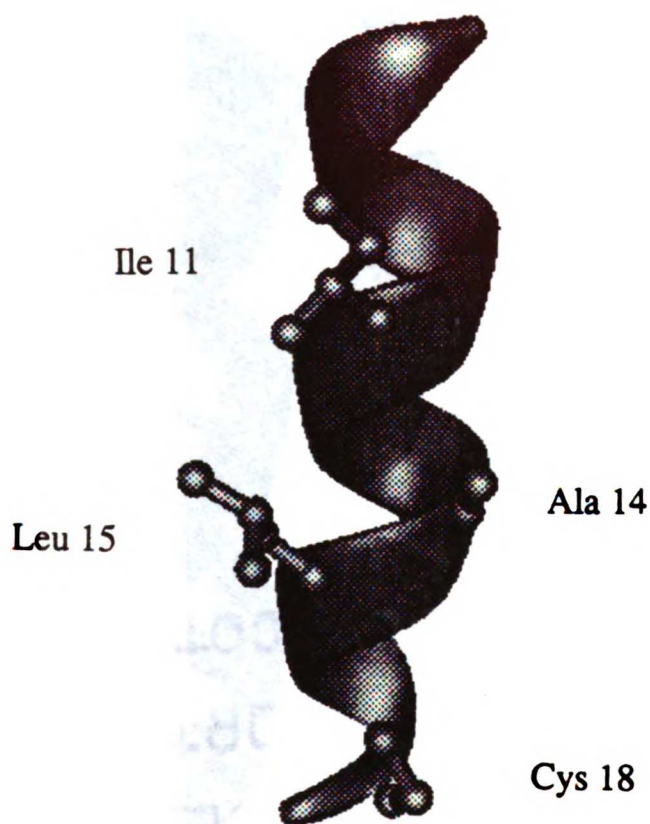


Figure 6-2-(a): Helical Core — Hydrophobic Patch

Patterns that are indicative of a helical sequences include those that would (a) form a hydrophobic patch on one side of the helix (3cpv residues 7 to 19); (b) form a charged pair of residues (3cpv residues 39 to 51); and (c) form a putative helix-helix interaction site (1fdh residues 57 to 77). A low density of hydrophilic residues might also indicate a helical region.

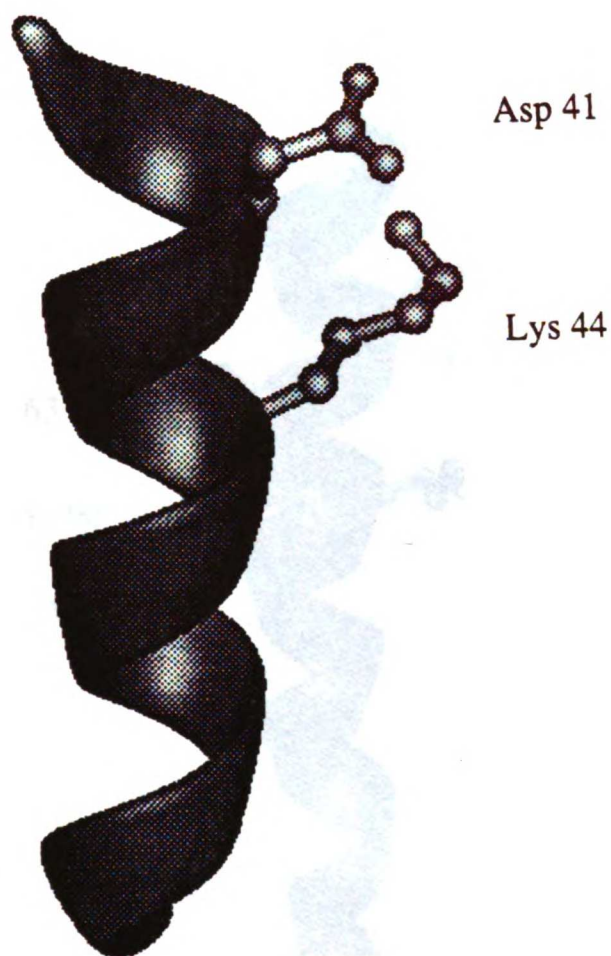


Figure 6-2-(b): Helical Core — Charged Pair

Patterns that are indicative of a helical sequences include those that would (a) form a hydrophobic patch on one side of the helix (3cpv residues 7 to 19); (b) form a charged pair of residues (3cpv residues 39 to 51); and (c) form a putative helix-helix interaction site (1fdh residues 57 to 77). A low density of hydrophilic residues might also indicate a helical region.

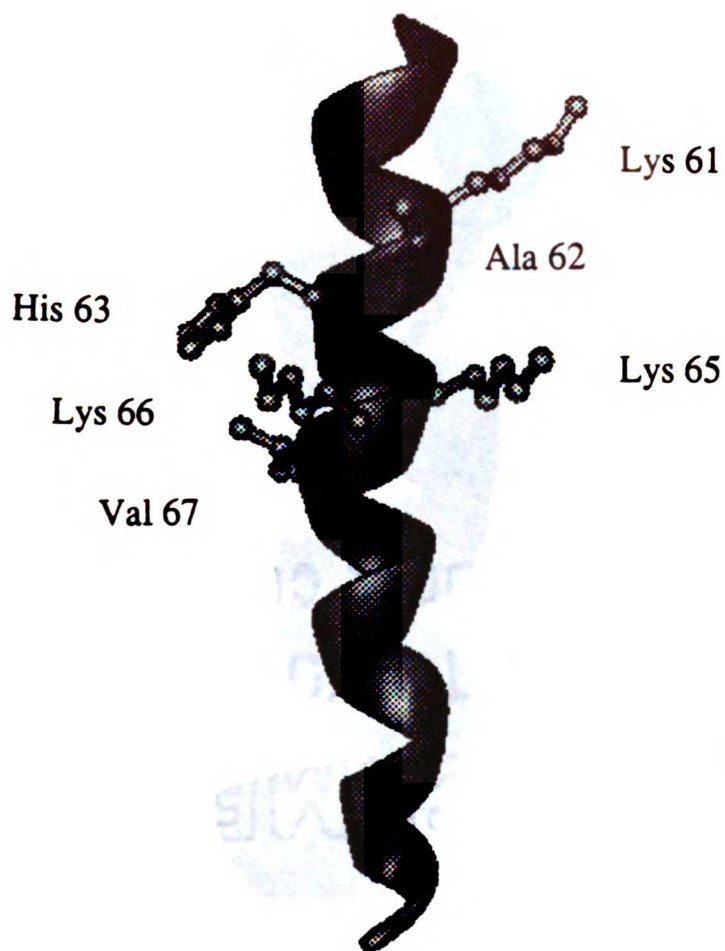


Figure 6-2-(c): Helical Core — Putative Helix-Helix Interaction Site

Patterns that are indicative of a helical sequences include those that would (a) form a hydrophobic patch on one side of the helix (3cpv residues 7 to 19); (b) form a charged pair of residues (3cpv residues 39 to 51); and (c) form a putative helix-helix interaction site (1fdh residues 57 to 77). A low density of hydrophilic residues might also indicate a helical region.



Figure 6-3-(a): N-Cap Examples — Specific Residues

Sequence patterns indicative of N-cap sites include (a) simple juxtapositions of specific residues (1fdh residues 57-77); (b) acidic and large hydrophobic residues placed just after a residue with a strong N-cap preference (3cpv residues 59-65); and (c) a Strong N-cap residue "terminating" a putative hydrophobic patch for a helix (3cpv residues 39 to 49).

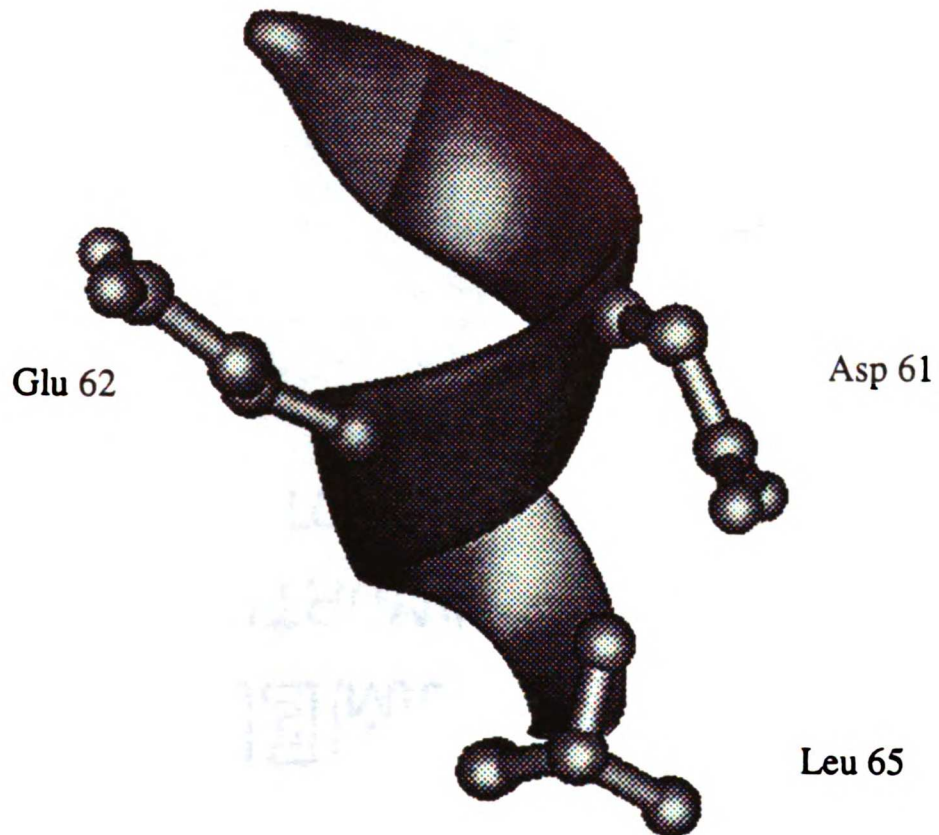


Figure 6-3-(b): N-Cap Examples — Acidic and Large Hydrophobic Residues

Sequence patterns indicative of N-cap sites include (a) simple juxtapositions of specific residues (1fdh residues 57-77); (b) acidic and large hydrophobic residues placed just after a residue with a strong N-cap preference (3cpv residues 59-65); and (c) a Strong N-cap residue "terminating" a putative hydrophobic patch for a helix (3cpv residues 39 to 49).

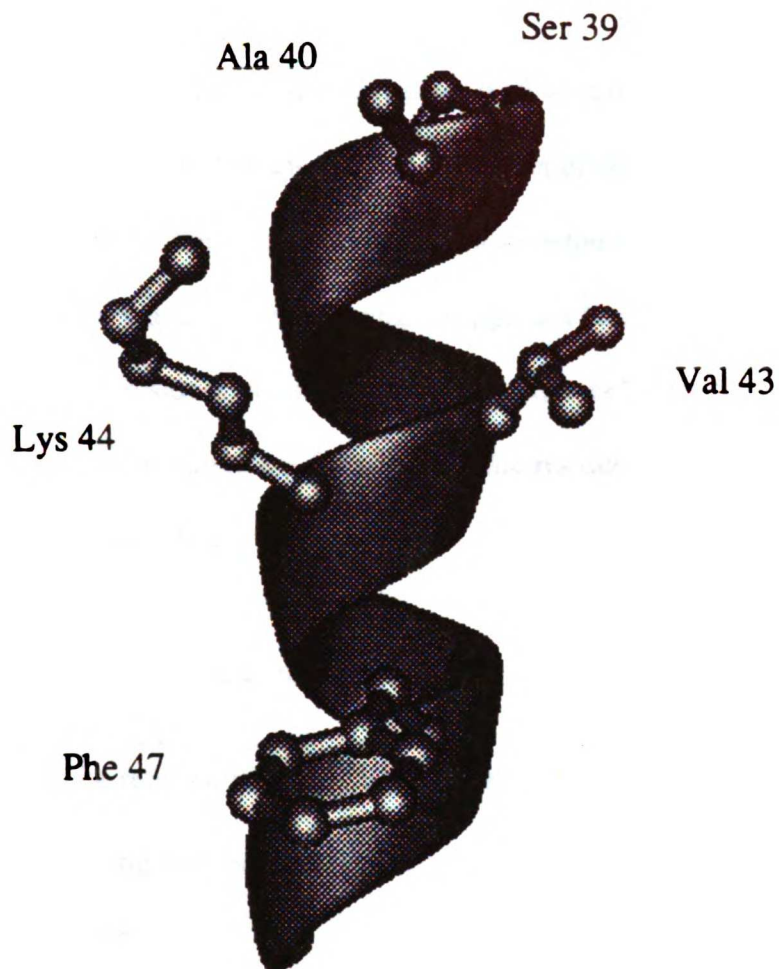
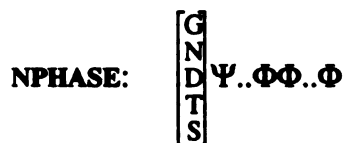


Figure 6-3-(c): N-Cap Examples — "Terminating" a Putative Hydrophobic Patch

Sequence patterns indicative of N-cap sites include (a) simple juxtapositions of specific residues (1fdh residues 57-77); (b) acidic and large hydrophobic residues placed just after a residue with a strong N-cap preference (3cpv residues 59-65); and (c) a Strong N-cap residue "terminating" a putative hydrophobic patch for a helix (3cpv residues 39 to 49).



The placement of the acidic residue correlates well with the hypothesis that interaction with the helix dipole is important to the stabilization of helical structure. These two patterns are considered at the same level in the N-cap pattern hierarchy. The next most reliable patterns require a residue that strongly suggests a N-cap site "in phase" with a cluster of hydrophobic residues on one face of the putative helix. In the case of N-caps, it seems that there is a requirement for a hydrophilic residue, just after the N-cap position, to terminate the hydrophobic patch.



This effect will be referred to as the "hydrophobic phasing" of the cap site.

These patterns, along with others are collected together into a composite N-cap pattern referred to as NCAP.

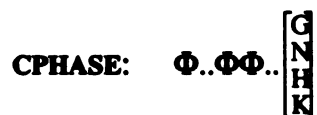
Helix C-cap. The C-cap prediction scheme follows an analogous hierarchical construction of patterns, but the critical residues differ. The pattern that provides the most reliable, independent, indication of a carboxy terminal site for a helix is the juxtaposition of a G at the C-cap site and a P one or two residues after the helix.

CSIMPLE: GP

Patterns containing, one of G, H or K (residues indicative of the C-cap site) and one or more basic residues one to three positions from the C-cap site, or a large hydrophobic residue three to four positions upstream of the C-cap site have also proven predictive.



By analogy with the N-cap patterns, a residue that strongly suggests a C-cap specific site "in phase" with a hydrophobic patch on one face of a putative helix constitutes the final class of patterns used in C-cap prediction.



These patterns, along with others are collected together into a composite C-cap pattern referred to as CCAP.

ALPPS Pattern Language. ALPPS, as described in Chapter 3, was used to produce patterns for this work. By utilizing the information in the required sequential ordering of the secondary structure component patterns, we take the next step in introducing long range interactions into secondary structure prediction.

The location of the turn predictions generated by the TU pattern is used to segment the sequence into blocks. After initial segmentation, all the sequence blocks are hidden from consideration. Then, those blocks that contain strong evidence for helical structure are exposed for evaluation. This is done using the PLANS pattern for helix core recognition: HCORE. Each visible block is then examined for orderings of PLANS patterns that match the region definitions supplied in the form of an ALPPS pattern. The subsequent PLANS patterns are then evaluated within the context of a block. Helical regions are specified under four possible conditions. Under the best of conditions, the the amino terminus, the core, and the carboxy terminus of a helix are recognized by the PLANS patterns NCAP, HCORE, CCAP respectively. The sequence beginning with the location of the NCAP pattern and ending with the CCAP pattern is marked as a helical region. If one or

more of the capping patterns is not recognized within the block, helical regions are constructed from the available information. The ordering of region definitions is significant, and there is at most one helical region per block. Figure 6-4 presents a symbolic representation of the different stages of prediction for the the protein carp parvalbumin B.

A complete listing of the ALPPS and PLANS patterns used to produce the results discussed in this chapter can be found in Appendix C.

Figure 6-4: A Working Example of an ALPPS Prediction

The first row of the figure on the next page is a description of the actual secondary structure for carp parvalbumin b (3cpv). The next three rows show where the sequence is broken into blocks using the turn prediction algorithm. The next three rows indicate the low-level PLANS patterns that recognize the different helical foundations (N-cap, helix core, and C-cap). ALPPS then uses the location of these PLANS patterns to define secondary structure. In this case, ALPPS first looks for the correct juxtaposition of N-cap core and C-cap then calls the region between the the N-cap and C-cap a helix. If one of the helix caps cannot be found, the ALPPS constructs a helix covering the distance between the remaining cap pattern and the helix core patterns, or possibly to the end of the block. If there are no caps at all, the helical core pattern might be used to define the helical region.

6.3. Results and Discussion

Table VI-2 shows the predictive capabilities of individual PLANS patterns on the specific structure sub-types: turns, helical cores, N-caps and C-caps. Because of a lack of false negatives (*FN*), strict Q_3 or C values as described earlier cannot be calculated: the PLANS patterns are scored on a basis of feature recognition. Multiple identifications of the same feature had no effect on the accuracy scores.

Table VI-2: Feature Scores

Development sequence set					
Class	Q_2	O	TP	FN	FP
Turns	84%	12%	63	12	9
Cores	95%	10%	81	4	9
N-Cap	61%	20%	51	33	17
C-Cap	58%	27%	49	35	23
Test sequence set					
Turns	82%	11%	47	10	6
Cores	95%	10%	63	4	7
N-Cap	51%	30%	34	33	20
C-Cap	35%	32%	23	44	21
Both sequence sets					
Turns	83%	11%	110	22	15
Cores	95%	11%	144	8	16
N-Cap	56%	25%	85	66	37
C-Cap	48%	29%	72	79	44

The patterns used to predict the locations of turns show a high level of success. The 2% decrease in prediction accuracy from the development set to the test set suggests the possibility that some patterns recognize specific features of the development set rather than generalized biophysical principles of turn stabilization. Within an error range of four residues, the strong turn patterns rarely generate false indications. However, the

weaker turn prediction patterns are not as reliable an indicator of the placement of actual turns. These patterns are more dependent on additional signals, such as the expected periodic distance between turns (Cohen *et al.*, 1986). As a consequence, most of the over or under prediction (false positives and false negatives) are a result of the noise in these secondary signals.

In a specific example of an over prediction (false positive), the first helix of Cytochrome c prime is broken by the presence of a weak turn indicator. Weak turn indications are not utilized if they are within eleven residues of strong turn indications. In this particular instance, the weak turn indications occur 15 residues from the nearest strong turn indicator. Hence, the pattern is accepted as an authentic turn (Figure 6-5).

Lysozyme contains an example of an under prediction (false negative). The last block should be split into two blocks, but it is not. There are some weakly hydrophilic residues in the area we would like to call a turn. While the area is sufficiently distant from the previous turn, the signal is below the threshold for accepting a weak turn indicator (Figure 6-6).

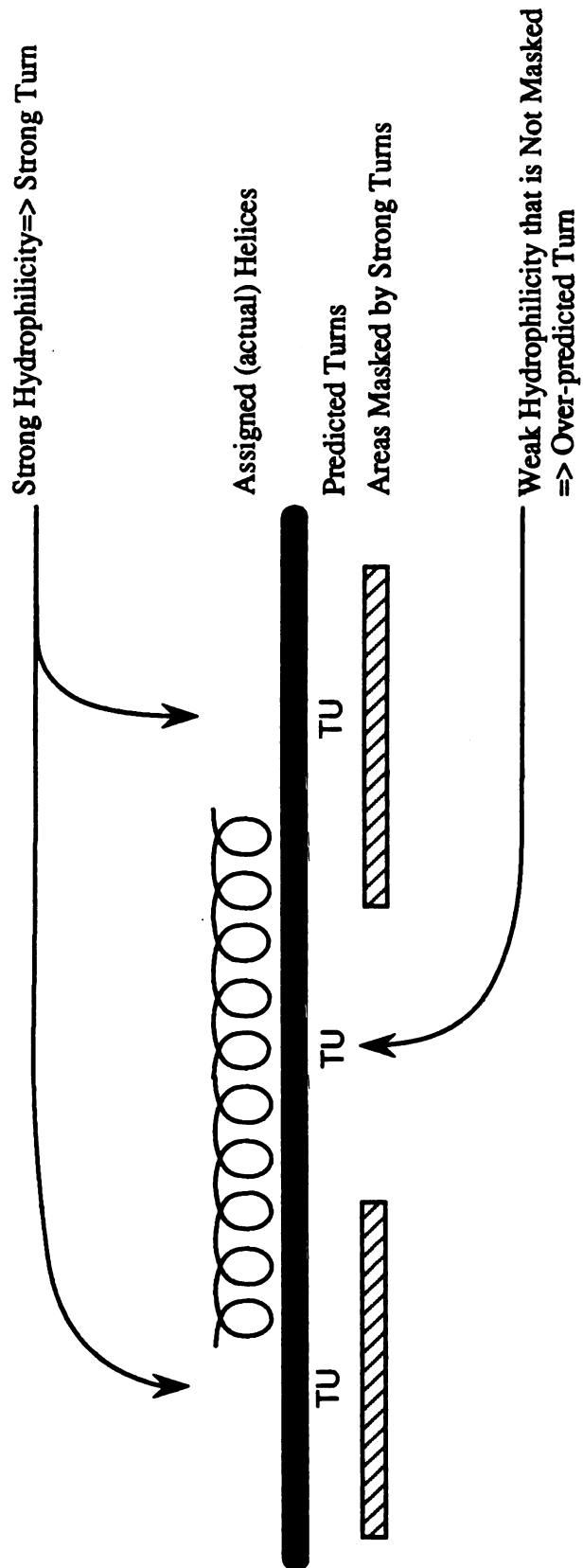


Figure 6-5: Over-predicted Turns

A segment of weakly hydrophilic residues is found sufficiently far away from adjacent strong turns that it is considered valid. However, the prediction of this turn produces two blocks where there should be only one.

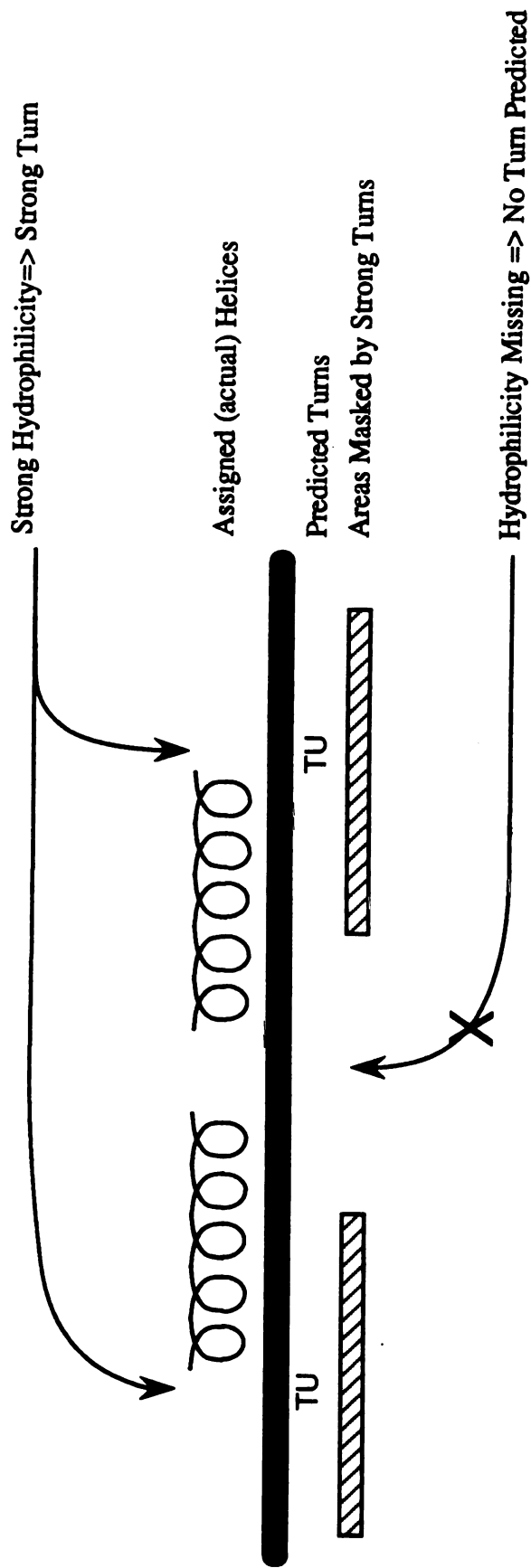


Figure 6-6: Under-predicted Turns

A segment of sequence, while sufficiently far away from the adjacent strong turns, does not contain enough sequential evidence to predict the location of a turn. This sequence leads to the production of only one block where there should be two.

The patterns used to predict helix core regions also appear successful. The lack of a significant decrease in prediction accuracy is likely to indicate the recognition of general principles in helix core formation. The few false positives for helix core feature stem from a displacement or extension of the predicted helical core, either beginning too soon, and/or ending too late in the sequence. Currently, it is difficult to associate this character with a specific structure or sequence phenomenon. Short helices (of length 5 to 10) are under-reported (false negatives), as are those helices with a strong hydrophilic character. Prediction of the core region of these types of helices is generally dependent on the recognition of complementary charged residue pairs, or a putative tertiary helix-helix interaction site (Richmond and Richards, 1978).

The predictive capability of the N- and C- cap patterns is significantly lower than that of the helix core patterns in both the development and test sets of protein sequences. Similarly, the decrease in predictive accuracy from the development to the test set for the cap patterns is currently greater (approx. 10% to 20%) than the decrease for the turn or helix core prediction. This suggests a tendency for the capping patterns to recognize specific features of the proteins in the development set, instead of general principles of the amino acid sequences that initiate and terminate helices. There is no specific structural feature that identifies over-prediction. However, the lack of crucial residues in the amino acid sequence near the site of the N- or C- cap typically characterizes under-prediction. Often the capping patterns will have as a constituent one of a class of residues commonly found at the terminus of a helix. The N-cap positions in helices are often one of the residues G, N, S, T, or D. The C-cap positions are usually G, K, H, or N. Further, proline is often a constituent of C-cap areas of sequence, appearing one or two residues after the cap position. If a helix does not begin or end in one of these residues,

the likelihood of a correctly predicted helix cap is low.

These helix capping patterns suggest that one of a specific set of residues is required (but not sufficient) to initiate or terminate a helix. A systematic approach to the mutagenesis of N-caps in barnase by Serrano and Fersht (1989) has provided some additional data on this aspect of helix structure. Threonines found at two different helix N-caps were mutated to several alternative residue types to examine the energetic contributions of different residues to helical structure stabilization. On the whole, the energetic stability provided by the alternative residues commonly found at the N-caps corroborate the statistical data presented by Richardson and Richardson (1988). However, the choice of a "best" residue to terminate a specific helix appears to be dependent on the tertiary interactions at the particular site. In this sense, the statistical data reported by Richardson and Richardson is not sufficient to completely specify the N- and C- caps.

Multiple regions can be defined by the user specified ALPPS prediction patterns. In the ALPPS pattern for predicting helical structure, four different region types have been specified as described in the methods section and in Figure 6-4. These regions reflect the amount of information available to specify the extent of the predicted helix. The number of assigned regions (helices) can be evaluated against the number of predicted regions for the protein sequences examined. Table VI-3 presents the number of each type of region identified in each protein. There was no particular relationship between the type of region predicted and the quality of the prediction for that region. Nor was there a more general relationship between the distribution of region types and the overall quality of a sequence prediction. However, some sequences appear to be more difficult to predict than others given any prediction method. For example, both cytochrome C peroxidase

and erythrocrucorin are predicted equally poorly by the methods of Chou and Fasman (1974), GOR (Garnier, Osguthorpe, and Robson, 1978), neural networks (Kneller, Cohen, and Langridge, 1990), and ALPPS. The rank orderings of the proteins by success rate using the different prediction methods were similar (data not shown).

Table VI-3: ALPPS Region Scores

Development set							
Protein	Assigned Helices	Predicted Helices	Both ends	No Nt	No Ct	No ends	Repechage
1ccr	5	4	1	1	0	2	0
1fdh	9	9	3	1	2	3	0
2ccy	5	6	2	1	2	1	0
2cts	21	24	5	7	5	6	1
2lh1	7	8	1	5	1	1	0
2lhb	8	8	0	2	3	3	0
2lzm	11	9	4	1	1	3	0
3c2c	5	7	4	0	3	0	0
3cln	7	8	5	0	3	0	0
3cpv	7	7	4	0	1	1	1
Totals	85	90	29	18	21	20	2
Test set							
156b	4	5	2	1	1	1	0
1ecd	8	7	2	3	1	1	0
1mbd	8	6	2	2	1	1	0
3icb	6	5	3	0	2	0	0
1cc5	4	3	1	0	1	1	0
2cyp	13	11	3	1	6	0	1
3wrp	6	5	0	1	2	2	0
1hmq	5	5	1	1	0	3	0
2tmv	6	6	1	0	3	2	0
3hhb	7	8	2	0	3	3	0
Totals	67	61	17	9	20	14	1

Table VI-4 presents the results of evaluating the predicted helical regions on a residue by residue basis in comparison to helical residues defined by the automatic assignment algorithms. In this two state prediction scheme, those regions not predicted as heli-

cal were scored as turns. The Chou and Fasman algorithm (Chou and Fasman, 1974) produced a predictive accuracy (Q_3) of 65% when applied to the collection of 20 all-alpha protein sequences used in this study; the GOR algorithm (Garnier, Osguthorpe, and Robson, 1978) provided an accuracy of 71%; the neural net scheme of Kneller et al. (Kneller, Cohen, and Langridge, 1990) was 78% accurate; and the ALPPS algorithm provided an accuracy of 71%. Trimming the N- and C-cap locations in both the assigned sequence and the predicted sequence improved the Chou and Fasman, GOR, the neural net, and ALPPS based predictions %2, %4, %6, and 6% respectively. Most of the individual prediction scores increase as the endpoint locations are eliminated from consideration, but some scores stay constant or decrease. This suggests that the overall error rate in prediction stems mainly from the difficulty in assigning the N- and C-caps. It is our experience that those predictions that do not benefit from neglecting the endpoints were poor predictions from the start. These data give a relative indication of the ability each algorithm possesses to predict the core features of secondary structure.

The primary source of error in the ALPPS prediction of helices results from failures in the underlying PLANS patterns. These errors can be subdivided into two levels: the segmentation of the sequence into structural units, or block definition, and the specification of helices within those units, or region definition. The primary source of block definition comes from correct identification of turns; here with the PLANS pattern TU. Failure at this level results in either the scission of a segment of regular secondary structure or the concatenation of two segments of regular secondary structure. Based on our previous work in turn prediction, we had anticipated and planned for these failures. We were able to describe a simple length based heuristic for splitting exceptionally long blocks. This is analogous to the PLANS work where weak turn predictions were *masked*

away from the location of strong turn indicators based on the expected distance between turns for a given protein class. We were not able to develop a consistently accurate heuristic for recognizing appropriate situations for adjacent block concatenation.

The errors in region specification are of two types: mis-assignment of the helical core, or mis-assignment of the helical endpoints. When one or both of the helix termini cannot be determined, helix is defined over the extent of the helical core pattern. This usually leads to under-prediction, but can lead to over-prediction when false positive helical core signals are generated. Predicted helical termini can also be located in a manner which erroneously shortens the helical region.

ALPPS also provides information to drive a display of the secondary structure prediction. Using the graphical display program MIDAS (Ferrin *et al.*, 1988), one can evaluate a secondary structure prediction in light of the the actual three dimensional structure of the protein. Graphical display of the structures facilitates an inspection of the postulated sequence-structure correlates for structural relevance. Examination of algorithm errors is also greatly simplified by coloring the residues with respect to the types of errors (under vs. over) in the secondary structure prediction. A MIDAS ribbon rendering which highlights the differences in the assigned and the predicted secondary structure for Cytochrome c, is presented in Figure 6-7.

6.4. Conclusions

We have developed PLANS patterns that recognize individual components of secondary structure. In the work presented here, we present concepts used to recognize the distinct structural components of α -helices: N-cap, core region, and C-cap. Currently

Table VI-4: ALPPS Residue by Residue Scores

Development set						
Protein	Q_3	C	TP	TN	FP	FN
1ccr	0.71	0.42	38	41	16	16
1fdh	0.74	0.35	91	17	7	31
2ccy	0.85	0.58	91	18	11	7
2cts	0.70	0.36	220	89	48	80
2lh1	0.69	0.23	87	19	22	25
2lhb	0.66	0.23	79	21	18	32
2lzm	0.71	0.41	87	30	8	39
3c2c	0.61	0.22	48	21	31	12
3cln	0.79	0.46	90	23	14	16
3cpv	0.88	0.75	64	32	8	4
Totals	0.73	0.39	895	311	183	262
Trimmed	0.80	0.56	543	225	86	101
Test set						
156b	0.73	0.49	48	28	5	22
1cc5	0.71	0.41	33	26	13	11
1ecd	0.60	0.01	75	7	11	43
1hmq	0.64	0.12	62	11	15	25
1mbd	0.75	0.26	104	12	11	26
2cyp	0.61	0.23	103	78	62	50
2tmv	0.77	0.54	57	62	15	20
3hhb	0.78	0.45	91	20	8	22
3icb	0.69	0.15	46	6	14	9
3wrp	0.69	0.17	64	6	5	26
Totals	0.69	0.33	683	256	159	254
Trimmed	0.75	0.47	408	192	72	126

we can identify almost all of the helical regions via their core structure features. However, we can identify only some of the N-cap, and C-cap positions with certainty.

With the success of the pattern based turn prediction, it was our expectation that the predicted location of turns would be a useful basis for sub-dividing a protein sequence into regions for independent evaluation. This expectation has been fulfilled. A language to facilitate a segment based approach to the prediction of regular secondary structure, ALPPS, has been designed and implemented. Initial pattern development was performed

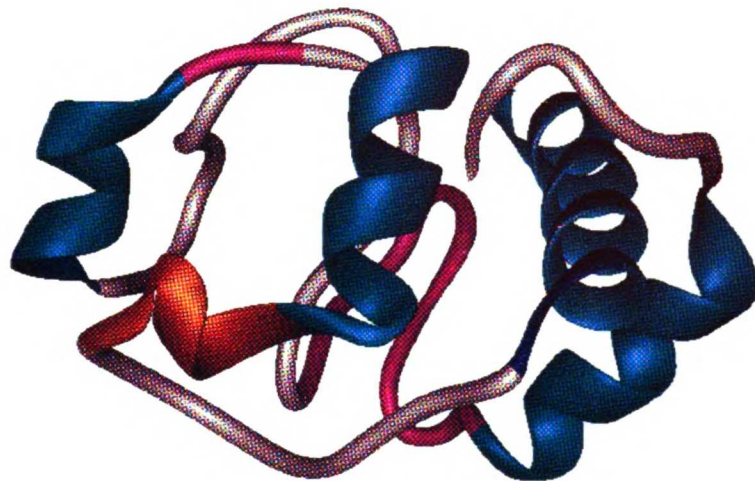


Figure 6-7: Sample ALPPS Result

Blue indicates where the secondary structure assignment and prediction match. Red indicates those segments of the sequence falsely predicted as helical. Violet outlines those segments where sequence was assigned helical but not predicted as such. The orange segment indicates an entire helix that was not predicted.

on a database of ten α/α proteins. These patterns were applied to an independent, non-homologous data set. The results presented here compare favorably with the current secondary structure prediction algorithms (Table VI-5). The ALPPS method is similar to the GOR method when all residues are considered. However when the scoring is focused on the core of the helical segments, the ALPPS algorithm fairs better. On the basis of individual residue scores, the neural network algorithm and weights developed by Kneller et. al. performs 3% better than the ALPPS system.

Effective, practical use of secondary structure prediction methods is facilitated by

Table VI-5: Comparison of Prediction Methods for All Helical Proteins

Method	Raw $Q_3 (C)$	Trimmed $Q_3 (C)$	Reference
Chou & Fasman	0.65 (0.23)	0.68 (0.30)	(Chou and Fasman, 1978)
Garnier, et al.	0.71 (0.36)	0.75 (0.46)	(Garnier, Osguthorpe, and Robson, 1978)
Qian, et al.	0.67 (NC)	NC (NC)	(Qian and Sejnowski, 1988)
Kneller, et al.	0.79 (0.55)	0.85 (0.69)	(Kneller, Cohen, and Langridge, 1990)
This Work	0.71 (0.36)	0.80 (0.56)	

NC is used to show *Not Calculated*. The values determined for Chou and Fasman, Garnier, et al. and the current work were taken by applying the respective algorithms to the 20 protein dataset described in methods, and comparing that to the assignments as generated by the algorithm of Richards and Kundrot. The weights generated in the neural net simulation by Kneller et al., were also used in a run against the 20 protein dataset, but compared against the helical assignment as determined by the algorithm of Kabsch and Sander, as this assignment was used in the training of the neural network. The value for Qian and Sejnowski were taken unmodified from Kneller et al. (Kneller, Cohen, and Langridge, 1990).

ability to determine the basis for individual predictions. With methods based on statistical analyses, this determination is infeasible because the data are primarily numerical distributions. The problems are similar with computational neural network learning algorithms. However, the database of rules contained in a pattern matching system such as ALPPS is interpretable by the experimentalist. This allows the experimentalist to perform an evaluation of confidence based on the biochemical knowledge incorporated into the pattern. Secondary structure prediction then overcomes the concept of a "black box" procedure.

Several secondary structure prediction algorithms may in fact be useful when used in concert. Predictions with the ALPPS system and the neural network software often complement each other. When the neural network software incorrectly predicts a partic-

ular region, often the ALPPS evaluation will be closer to the targeted assignment, and vice versa. In future work, we plan to integrate the two methods to obtain the best information from each package.

Work on extensions to the ALPPS method is also underway. PLANS patterns that recognize the components of β -structure, and the ALPPS patterns that would coordinate the prediction of β -structure regions are currently under development. The eventual incorporation of combined α and β component structure patterns for the set of α/β proteins is also anticipated. We have only explored the lower half of the structural hierarchy. We are now incorporating concepts into the syntax of ALPPS that will facilitate the description of higher level concepts in protein structure. These will include super-secondary structure and "motifs" such as four helix bundles (Presnell and Cohen, 1989) or nucleotide binding folds (Rao and Rossman, 1973).

One of the innovative features incorporated into the ALPPS language was the ability to utilize the sequential ordering of sub-structure features as recognized by PLANS patterns. We have exploited that aspect of the ALPPS meta-pattern language to construct the first example of a predictive algorithm for regular secondary structure that explicitly incorporates non-local sequence-structure correlations. The results presented here strongly suggest that processing the sequence into structurally reasonable segments can provide an advantage over non-hierarchical methods of prediction.

The developments in predictive schemes have also brought to our attention the complexities in describing the target. Scoring algorithms that focus on the prediction of individual residues have been used as a standard of comparison for several years. One artifact of these algorithms is that each residue is considered equally important. How-

ever, researchers and algorithms and cannot agree amongst themselves on the precise location of the terminal residues of regular secondary structure. Moreover, the semi-empirical condensation methods of model construction, which use secondary structure as input, are insensitive to the exact end or beginning of a particular stretch of secondary structure (Cohen and Kuntz, 1989). Of much greater importance is the number and location of the individual secondary structure features. The trimming technique presented here affords one method to examine this issue. We are continuing to develop new scoring algorithms that focus on the segmented, feature oriented, nature of regular secondary structure. This should provide an evaluation technique that indicates the utility of secondary structure predictions for subsequent steps in the modeling process.

Chapter 7

Conclusions

“[T]he prediction of helix locations has almost become a science.”

— Richard E. Dickerson (Dickerson *et al.*, 1971)

Twenty years later, the prediction of helix locations is still an open problem.

Secondary structure can provide a low-resolution view of the structure of a protein — a view that can be used for many purposes, *e.g.*, the epitope search described in Chapter 1 or tertiary structure prediction described in Chapter 6. Most of the standard secondary structure prediction techniques have failed to adequately focus on the prediction of helix [strand] *locations*. Instead, the emphasis has been on maximizing some quality index (usually *Q* or *C*) which is based solely on residue counts. As is shown in Chapter 4, residue based scores do not necessarily reflect how well a prediction approximates observed secondary structure. Segment based secondary structure prediction, as described in this thesis, is attempt to produce predictions which do approximate observed secondary structure.

As described in Chapter 2, local sequence patterns can provide some sequence-structure correlates — particularly for turns when combined with the concept of link length (appropriate spacing between turns). The non-local information provided by the appropriate spacing is an important caveat. Sequence-structure correlates for helix components can also be derived from local sequence information, but these patterns need to be placed into a larger framework in order to predict helix locations. ALPPS provides the framework through its various segment types and metapattern syntax.

The results of using ALPPS and a segment based approach, detailed in Chapter 6, are promising. Because the predictions begin with a partitioning of the sequence into segments based on turn predictions, no predicted helix is unreasonably long. Similarly, minimum sizes are applied to predicted helices. The bootstrap kernel of segment based prediction allows the use of local sequence information to go into refining the prediction with more long range information.

One of the areas of future improvement for ALPPS is an implementation of the frame concept sketched at the end of Chapter 3. Prediction may also be improved by moving to a multiple pass technique. Preliminary predictions of helix locations may be subject to change on a second pass when supersecondary structure (*e.g.*, $\alpha - \beta - \alpha$) or motif (*e.g.*, four helix bundle) signals are recognized.

Although residue based scores do not completely evaluate secondary structure predictions, they do reveal some information. This is particularly true after capping differences are removed by trimming. Based on Q or C scores, our predictions are superior to Chou-Fasman (Chou and Fasman, 1978) and GOR (Garnier, Osguthorpe, and Robson, 1978), but not as good as a recent neural net technique (Kneller, Cohen, and Langridge, 1990). Neural nets and machine learning in general offers the capacity of exhaustive searching that might exhaust (bore) a human. Work should go into combining aspects of a neural net (or other machine learning) approach into our segment based approach.

Pseudo-string edit distances described in the chapter on feature based scoring, could improved machine learning by providing an evaluation tool which reflects the goal of making predictions which approximate secondary structure assignments. Though all of the components of pseudo-string editing (objects, operations, and costs) are outlined in Chapter 5, the remaining work will be difficult, but also rewarding.

References

- R. M. Abarbanel, "Protein Structural Knowledge Engineering," *PhD Thesis*, University of California, San Francisco, 1984.
- S. S. Abdel-Meguid, H.-S. Shieh, W. W. Smith, H. E. Dayringer, B. N. Violand, and L. A. Bente, "Three-dimensional Structure of a Genetically Engineered Variant of Porcine Growth Hormone," *Proc Natl Acad Sci USA*, 84:6434-6437, 1987.
- J. Allen, *Natural Language Understanding*, Benjamin/Cummings, Menlo Park, 1987.
- C. B. Anfinsen, E. Haber, M. Sela, and F. H. White, "The kinetics of the formation of native ribonuclease during oxidation of the reduced polypeptide chain," *Proc Natl Acad Sci USA*, 47:1309-1314, 1961.
- P. Argos and J. Palau, "Amino Acid Distribution in Protein Secondary Structures," *Int J Peptide Res*, 19:380-393, 1982.
- P. Argos, J. Schwarz, and J. Schwarz, "An Assessment of Protein Secondary Structure Prediction Methods Based on Amino Acid Sequence," *Biochimica et Biophysica Acta*, 439:261-273, 1976.
- E. G. Arutyunyan, I. P. Kuranova, B. K. Vainshtein, and W. Steigemann, "X-ray Structural Investigation of Leghemoglobin. VI. Structure of Acetate-ferrileghemoglobin at a Resolution of 2.0 Angstroms," *Kristallografiya*, 25:80, 1980.
- Y. S. Babu, C. E. Bugg, and W. J. Cook, "Structure Of Calmodulin Refined At 2.2 Angstroms Resolution," *J Mol Biol*, 204:191-204, 1988.
- D. J. Barlow and J. M. Thornton, "Helix geometry in proteins," *J Mol Biol*, 201(3):601-19, 1988.
- F. C. Bernstein, T. F. Koetzle, G. J. B. Williams, E. F. Meyer, M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasumi, "The Protein Data Bank: a Computer Archive," *J Mol Biol*, 112:535-542, 1977.
- C. S. O. C. A. B. Systems, *Ann NY Acad Sci*, p. 482, 1986.
- G. E. Bhatia, *Refinement Of The Crystal Structure Of Oxidized Rhodospirillum Rubrum Cytochrome C2*, PhD Thesis, University Of California, San Diego, 1981.
- J. L. Bittle, R. A. Houghten, H. Alexander, T. M. Shinnick, R. A. Lerner, D. J. Rowlands, and F. Brown, "Protection against foot-and-mouth disease by immunization with a chemically synthesized peptide predicted from the viral nucleotide sequence," *Nature (London)*, 298:30-33, 1982.
- T. L. Blundell and L. N. Johnson, *Protein Crystallography*, Academic Press, New York, 1976.
- J. U. Bowie, N. D. Clarke, C. O. Pabo, and R. T. Sauer, "Identification of Protein Folds: Matching Hydrophobicity Patterns of Sequence Sets With Solvent Accessibility Patterns of Known Structures," *Proteins: Structure, Function, and Genetics*, 7:257-264, 1990.
- E. K. Bradley, J. F. Thomason, F. E. Cohen, P. A. Kosen, and I. D. Kuntz, "Studies of Synthetic Helical Peptides Using Circular Dichroism and Nuclear Magnetic

- Resonance," *J Mol Biol*, 215:607-622, 1990.
- B. J. Brandhuber, T. Boone, W. Kenney, and D. B. McKay, "Three-Dimensional Structure of Interleukin-2," *Science*, 238:1707-1709, 1987.
- E. A. T. W. Budge, *The Rosetta Stone*, Trustees of the British Museum, London, 1950.
- D. C. Carter, K. A. Melis, S. E. O'Donnell, B. K. Burgess, W. F. Furey Jr., B. C. Wang, and C. D. Stout, "Crystal Structure Of Azotobacter Cytochrome c5 At 2.5 Angstroms Resolution," *J Mol Biol*, 184:279, 1985.
- C. Chothia, M. Levitt, and D. Richardson, "Structure of proteins: packing of alpha-helices and pleated sheets," *Proc Natl Acad Sci USA*, 74:4130-4134, 1977.
- P. Y. Chou and G. D. Fasman, " β -turns in proteins," *J Mol Biol*, 115:135-175, 1977.
- P. Y. Chou and G. D. Fasman, "Conformational parameters for amino acids in helical, β -sheet, and random coil regions calculated from proteins," *Biochemistry*, 13:211-221, 1974.
- P. Y. Chou and G. D. Fasman, "Empirical predictions of protein conformation," *Ann Rev Biochem*, 47:251-276, 1978.
- B. I. Cohen, S. R. Presnell, and F. E. Cohen, "Pattern Based Approaches to Protein Structure Prediction," *Methods in Enzymology*, 202:xxx, 1991 (*in press*).
- B. I. Cohen, S. R. Presnell, M. Morris, R. Langridge, and F. E. Cohen, "Pattern Recognition and Protein Structure Prediction," *Proceedings of the 24th Hawaii International Conference on System Sciences*, p. 574-578, IEEE, 1991.
- F. E. Cohen, R. M. Abarbanel, I. D. Kuntz, and R. J. Fletterick, "Turn Prediction in Proteins Using a Pattern-Matching Approach," *Biochemistry*, 25:266-275, 1986.
- F. E. Cohen, P. A. Kosen, I. D. Kuntz, L. B. Epstein, T. L. Ciardelli, and K. A. Smith, "Structure-Activity Studies of Interleukin-2," *Science*, 234:349-352, 1986a.
- F. E. Cohen and I. D. Kuntz, "Tertiary Structure Prediction," in: *Prediction of Protein Structure and the Principles of Protein Conformation*, G. D. Fasman, ed., p. 647-705, Plenum Press, New York, 1989.
- F. E. Cohen, T. J. Richmond, and F. M. Richards, "Protein Folding: Evaluation of Some Simple Rules for the Assembly of Helices into Tertiary Structures with Myoglobin as an Example," *J Mol Biol*, 132:275-288, 1979.
- F. E. Cohen and I. D. Kuntz, "Prediction of the Three-Dimensional Structure of Human Growth Hormone," *Proteins: Structure, Function, and Genetics*, 2:162-166, 1987.
- F. E. Cohen, M. J. E. Sternberg, and W. R. Taylor, "Analysis and Prediction of Protein β -Sheet Structures by a Combinatorial Approach," *Nature (London)*, 285:378-382, 1980.
- F. E. Cohen, M. J. E. Sternberg, and W. R. Taylor, "Analysis and Prediction of the Packing of α -Helices against a β -Sheet in the Tertiary Structure of Globular Proteins," *J Mol Biol*, 156:821-862, 1982.
- I. Crawford, T. Niermann, and K. Kirschner, "Prediction of secondary structure by evolutionary comparison: application to the alpha subunit of tryptophan synthase," *Proteins: Structure, Function, and Genetics*, 2:118-129, 1987.

- G. Deleage and B. Roux, "An Algorithm for Protein Secondary Structure Prediction based on Class Prediction," *Protein Engineering*, 1(4):289-294, 1987.
- R. E. Dickerson, T. Takano, D. Eisenberg, O. B. Kallai, L. Samson, and A. Cooper, "Ferricytochrome C: General Features of the Horse and Bonito Proteins at 2.8 Å Resolution," *J Biol Chem*, 246:1511-1535, 1971.
- G. Fermi, M. F. Perutz, B. Shaanan, and R. Fourme, "The Crystal Structure Of Human Deoxyhaemoglobin At 1.74 Angstroms Resolution," *J Mol Biol*, 175:159-174, 1984.
- T. Ferrin, C. Huang, L. Jarvis, and R. Langridge, "The MIDAS Display System," *J Mol Graphics*, 6:13-37, 1988.
- B. C. Finzel, T. L. Poulos, and J. Kraut, "Crystal Structure Of Yeast Cytochrome c Peroxidase Refined At 1.7-Angstroms Resolution," *J Biol Chem*, 259:13027-13036, 1984.
- B. C. Finzel, P. C. Weber, K. D. Hardman, and F. R. Salemme, "Structure Of Ferricytochrome c(Prime) From Rhodospirillum molischianum At 1.67 Angstroms Resolution," *J Mol Biol*, 186:627-643, 1985.
- J. A. Frier and M. F. Perutz, "Structure Of Human Foetal Deoxyhaemoglobin," *J Mol Biol*, 112:97, 1977.
- J. R. Garnier, D. J. Osguthorpe, and B. Robson, "Analysis of the accuracy and implications of simple methods for predicting the secondary structure of globular proteins." *J Mol Biol*, 120:97-120, 1978.
- J.-F. Gibrat, J. Garnier, and B. Robson, "Further developments of protein secondary structure prediction using information theory. New parameters and consideration of residue pairs," *J Mol Biol*, 198:425-443, 1987.
- Y. Goto and A. L. Fink, "Phase Diagram For Acidic Conformational States Of Apomyoglobin," *J Mol Biol*, 214:803-5, 1990.
- M. Gribskov, A. D. McLachlan, and D. Eisenberg, "Profile analysis: detection of distantly related proteins," *Proc Natl Acad Sci USA*, 84(13):4355-8, 1987.
- A. V. Guzzo, "The Influence of Amino Acid Sequence on Protein Structure," *Biophys J*, 5:809-822, 1965.
- L. W. Hardy, J. S. Finer-Moore, W. R. Montfort, M. O. Jones, D. V. Santi, and R. M. Stroud, "Atomic structure of thymidylate synthase: target for rational drug design," *Science*, 235:448-55, 1987.
- W. G. J. Hol, P. T. van Duijnen, and H. J. C. Berendsen, *Nature (London)*, 273:443-446, 1979.
- L. H. Holley and M. Karplus, "Protein Secondary Structure Prediction with a Neural Network," *Proc Natl Acad Sci (USA)*, 86:152-156, 1989.
- R. B. Honzatko, W. A. Hendrickson, and W. E. Love, "Refinement Of A Molecular Model For Lamprey Hemoglobin From *Petromyzon marinus*," *J Mol Biol*, 184:147, 1985.
- F. M. Hughson, P. E. Wright, and R. L. Baldwin, "Structural Characterization of a Partly Folded Apomyoglobin Intermediate," *Science*, 249:1544-1548, 1990.

- W. C. Johnson Jr., "Protein Secondary Structure And Circular Dichroism: A Practical Guide," *Proteins: Structure, Function, and Genetics*, 7:205-214, 1990.
- W. Kabsch and C. Sander, "How good are predictions of protein secondary structure?" *FEBS Lett*, 155(2):179-182, May 1983.
- W. Kabsch and C. Sander, "On the use of sequence homologies to predict protein structure: Identical pentapeptides can have completely different conformations," *Proc Natl Acad Sci USA*, 81:1075-1078, 1984.
- W. Kabsch and C. Sander, "Dictionary of Protein Secondary Structure: Pattern Recognition and Geometrical Features," *Biopolymers*, 22:2577-2637, 1983.
- A. Kandel, *Fuzzy Techniques in Pattern Recognition*, Wiley, New York, 1982.
- S. Karlin, B. E. Blaisdell, E. S. Mocarski, and V. Brendel, "A Method to Identify Distinctive Charge Configurations in Protein Sequences, with Applications to Human Herpesvirus Polypeptides," *J Mol Biol*, 205:165-177, 1989.
- M. Karplus and J. A. McCammon, "The Internal Dynamics of Globular Proteins," *CRC Crit Rev*, 9:293-349, 1981.
- P. S. Kim and R. L. Baldwin, *Nature (London)*, 307:329-334, 1984.
- S. H. Kim, *personal communication*, 1991.
- P. Klein and C. Delisi, "Prediction of Protein Structural Class from the Amino Acid Sequence," *Biopolymers*, 25:1659-1672, 1986.
- D. Kneller, F. E. Cohen, and R. Langridge, "Improvements in Protein Secondary Structure Prediction by an Enhanced Neural Network," *J Mol Biol*, 214:171-182, 1990. eptide chain.
- I. D. Kuntz, "Protein folding," *J Amer Chem Soc*, 94:4009-4012, 1972.
- C. L. Lawson, R. Zhang, R. W. Schevitz, Z. Otwinowski, A. Joachimiak, and P. B. Sigler, "Flexibility of the DNA-binding Domains of Trp Repressor," *Proteins: Structure, Function, and Genetics*, 3:18-31, 1988.
- F. Lederer, A. Glatigny, P. H. Bethge, H. D. Bellamy, and F. S. Mathews, "Improvement Of The 2.5 Angstroms Resolution Model Of Cytochrome b562 By Redetermining The Primary Structure And Using Molecular Graphics," *J Mol Biol*, 148:427-448, 1981.
- D. C. Lee, P. I. Haris, D. Chapman, and R. C. Mitchell, "Determination of Protein Secondary Structure Using Factor Analysis of Infrared Spectra," *Biochemistry*, 29:9185-9193, 1990.
- J. M. Levin and J. Garnier, "Improvements in a secondary structure prediction method based on a search for local sequence homologies and its use as a model building tool," *Biochimica et Biophysica Acta*, 955:283-95, 1988.
- M. Levitt and C. Chothia, "Structural patterns in globular proteins," *Nature (London)*, 261:552-557, 1976.
- M. Levitt and J. Greer, "Automatic identification of secondary structure in globular proteins," *J Mol Biol*, 114:181-293, 1977.
- M. Levitt and A. Warshel, "Computer simulation of protein folding," *Nature*, 253:694-698, 1975.

- H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, Englewood Cliff, 1981.
- P. N. Lewis, F. A. Momany, and H. A. Scheraga, "Folding of polypeptide chains in proteins: A proposed mechanism for folding," *Proc Natl Acad Sci USA*, 68:2293-2297, 1971.
- P. N. Lewis and H. A. Scheraga, "Predictions of Structural Homologies in Cytochrome c Proteins," *Arch Biochem Biophys*, 144:576-583, 1971.
- V. I. Lim, "Structural principles of the globular organization of protein chains. A stereochemical theory of globular protein secondary structure," *J Mol Biol*, 88:857-872, 1974.
- V. I. Lim, "Algorithms for prediction of α -helical and β -structural regions in globular proteins," *J Mol Biol*, 88:873-894, 1974.
- P. Manavalan and W. C. Johnson, "Circular Dichroism is Sensitive to Classes of Protein Tertiary Structure," *Nature (London)*, 305:831-832, 1983.
- W. J. Masek and M. S. Patterson, "How To Compute String-Edit Distances Quickly," in: *Time Warps, String Edits, and Macromolecules : The Theory and Practice of Sequence Comparison*, D. Sankoff and J. B. Kruskal, ed., p. 337-349, Addison-Wesley, Reading, Mass, 1983.
- B. W. Matthews, "Comparison of the Predicted and Observed Secondary Structure Of Bacteriophage T4 Lysozyme," *Biochimica et Biophysica Acta*, 405:442-451, 1975.
- J. McCammon, B. Gelin, and M. Karplus, "Dynamics of Folded Proteins," *Nature (London)*, 267:585-590, 1977.
- A. D. McLachlan and M. Stewart, "Tropomyosin Coiled-coil Interactions: Evidence for an Unstaggered Structure," *J Mol Biol*, 98:293-308, 1975.
- P. C. Moews and R. H. Kretsinger, "Refinement Of The Structure Of Carp Muscle Calcium- Binding Parvalbumin By Model Building And Difference Fourier Analysis," *J Mol Biol*, 91:201-225, 1975.
- D. Morris, *personal communication*, 1990.
- K. Nagano, "Logical analysis of the mechanism of protein folding. I. Predictions of helices, loops, and beta-structures from primary structure," *J Mol Biol*, 75:401-420, 1973.
- K. Namba, R. Pattanayek, and G. Stubbs, "Visualization Of Protein-Nucleic Acid Interactions In A Virus. Refined Structure Of Intact Tobacco Mosaic Virus At 2.9 Angstroms Resolution By X-ray Fiber Diffraction." *J Mol Biol*, 208:307, 1989.
- G. Nemethy and H. A. Scheraga, "Protein folding," *Quarterly Reviews of Bioph*, 10:239-352, 1977.
- K. Nishikawa and T. Ooi, "Amino acid sequence homology applied to the prediction of protein secondary structures, and joint prediction with existing methods," *Biochimica et Biophysica Acta*, 871:45-54, 1986.
- H. Ochi, Y. Hata, N. Tanaka, M. Kakudo, T. Sakurai, S. Aihara, and Y. Morita, "Structure Of Rice Ferricytochrome c At 2.0 Angstroms Resolution," *J Mol Biol*, 166:407, 1983.

- M. Sundaralingam and Y. C. Sekharudu, "Water-Inserted α -Helical Segments Implicate Reverse Turns as Folding Intermediates," *Science*, 244:1333-1337, 1989.
- L. H. Pearl and W. R. Taylor, "A Structural Model for the Retroviral Proteases," *Nature (London)*, 329:351-354, 1987.
- E. Pfaff, M. Mussgay, H. O. Bohm, G. E. Schultz, and H. Schaller, "Antibodies against a preselected peptide recognize and neutralize foot and mouth disease virus," *EMBO*, 1(7):869-874, 1982.
- S. E. V. Phillips and B. P. Schoenborn, "Neutron Diffraction Reveals Oxygen-histidine Hydrogen Bond In Oxymyoglobin," *Nature (London)*, 292:81, 1981.
- S. R. Presnell and F. E. Cohen, "Topological Distribution of Four- α -Helix Bundles," *Proc Natl Acad Sci*, p. 6592-6596, 1989.
- L. G. Presta and G. D. Rose, "Helix Signals in Proteins," *Science*, 240:1632-1641, 1988.
- O. B. Ptitsyn, R. H. Pain, G. V. Semisotnov, E. Zerovnik, and O. I. Razgulyaev, "Evidence For A Molten Globule State As A General Intermediate In Protein Folding," *FEBS Letters*, 262:20-24, 1990.
- N. Qian and T. J. Sejnowski, "Predicting the Secondary Structure of Globular Proteins Using Neural Network Models," *J Mol Biol*, 202:865-884, 1988.
- S. T. Rao and M. G. Rossman, "Comparison of super-secondary structure in proteins," *J Mol Biol*, 76:241-256, 1973.
- D. C. Rees, L. DeAntonio, and D. Eisenberg, "Hydrophobic Organization of Membrane Proteins," *Science*, 245:510-513, 1989.
- S. Remington, G. Wiegand, and R. Huber, "Crystallographic Refinement And Atomic Models Of Two Different Forms Of Citrate Synthase At 2.7 And 1.7 Angstroms Resolution," *J Mol Biol*, 158:111, 1982.
- F. M. Richards and C. E. Kundrot, "Identification of Structural Motifs From Protein Coordinate Data: Secondary Structure and First-Level Supersecondary Structure," *Proteins: Structure, Function, and Genetics*, 3:71-84, 1988.
- J. S. Richardson and D. C. Richardson, "Amino Acid Preferences for Specific Locations at the Ends of alpha Helices," *Science*, 240:1648-1652, 1988.
- T. J. Richmond and F. M. Richards, "Packing of α -helices: Geometrical constraints and contact areas," *J Mol Biol*, 119:537-555, 1978.
- D. M. Ritchie and K. Thompson, "The UNIX Time-sharing System," *Communications of the ACM*, 17(7):365-375, 1974.
- H. Roder, G. A. Eloeve, and S. W. Englander, "Structural Characterization of Folding Intermediates in Cytochrome C by H-exchange Labelling and Proton NMR," *Nature (London)*, 335:700-704, 1988.
- G. D. Rose, "Prediction of chain turns in globular proteins on a hydrophobic basis," *Nature (London)*, 272:586-590, 1978.
- D. Sankoff and J. B. Kruskal, ed., *Time Warps, String Edits, and Macromolecules : The Theory and Practice of Sequence Comparison*, Addison-Wesley, Reading, Mass, 1983.

- C. Schellman, "The α_L Conformation at the Ends of Helices," in: *Protein folding : proceedings of the 28th Conference of the German Biochemical Society, held at the University of Regensburg, Regensburg, West Germany, September 10-12, 1979*, R. Jaenicke, ed., p. 53-61, Elsevier/North-Holland Biomedical Press, New York, 1980.
- M. Schiffer and A. B. Edmundson, *Biophys J*, 8:29-39, 1968.
- M. Schiffer and A. B. Edmundson, "Use of helical wheels to represent the structures of proteins and to identify segments with helical potential," *Biophys J*, 7:121-135, 1967.
- G. E. Schultz, "A Critical Evaluation of Methods for Prediction of Protein Secondary Structures," *Ann Rev Biophys Biophys Chem*, 17:1-21, 1988.
- G. E. Schulz, "A Critical Evaluation of Methods for Prediction of Protein Secondary Structures," *Ann Rev Biophys Chem*, 17:1-21, 1988.
- G. E. Schulz, C. D. Barry, J. Friedman, P. Y. Chou, G. D. Fasman, A. V. Finkelstein, V. I. Lim, O. B. Pittsyn, E. A. Kabat, T. T. Wu, M. Levitt, B. Robson, and K. Nagano, "Comparison of Predicted and Experimentally Determined Secondary Structure of Adenyl Kinase," *Nature (London)*, 250:140-142, 1974.
- G. E. Schulz and R. H. Schirmer, *Principles of Protein Structure*, Springer-Verlag, New York, NY, 1979.
- D. B. Searls, *The Computational Linguistics of Biological Sequences*, Unisys Center for Advanced Information Technology, Paoli, PA, 1990.
- L. Serrano and A. R. Fersht, "Capping and alpha-helix stability," *Nature (London)*, 342:296-299, 1989.
- R. P. Sheridan, J. S. Dixon, R. Venkataghavan, I. D. Kuntz, and K. P. Scott, "Amino Acid Composition and Hydrophobicity Patterns of Protein Domains Correlate with Their Structures," *Biopolymers*, 24:1995-2023, 1985.
- A. Sikorski and J. Skolnick, "Monte Carlo Simulation of Equilibrium Globular Protein Folding. α -helical Bundles with long loops," *Proc Natl Acad Sci (USA)*, 86:2668-2672, 1989.
- H. Sklenar, C. Etchebest, and R. Lavery, "Describing Protein Conformation: A General Algorithm Yielding Complete Helicoidal Parameters and a Unique Overall Axis," *Proteins: Structure, Function, and Genetics*, 6:46-60, 1989.
- J. Skolnick and A. Kolinski, "Computer Simulations of Globular Protein Folding and Tertiary Structure," *Annu Rev Phys Chem*, 40:207-235, 1989.
- J. Skolnick and A. Kolinski, "Simulations of the Folding of a Globular Protein," *Science*, 250:1121-1126, 1990.
- T. F. Smith and R. F. Smith, "Automatic Generation of Primary Sequence Patterns from Sets of Related Protein Sequences," *Proc Natl Acad Sci USA*, 87:118-122, 1990.
- R. Stallman, *GNU Emacs Manual*, Free Software Foundation, Cambridge, Massachusetts, 1986.
- G. L. Steele, *Common Lisp the Language*, Digital Press, Burlington, Massachusetts, 1984.

- W. Steigemann and E. Weber, "Structure Of Erythrocrucorin In Different Ligand States Refined At 1.4 Angstroms Resolution," *J Mol Biol*, 127:309-338, 1979.
- R. E. Stenkamp, L. C. Sieker, and L. H. Jensen, "Adjustment Of Restraints In The Refinement Of Methemerythrin And Azidomethemerythrin At 2.0 Angstroms Resolution," *Acta Crystallogr., Sect. B*, 39:697-703, 1983.
- M. J. Sternberg and W. J. Gullick, "A Sequence Motif in the Transmembrane Region of Growth Factor Receptors with Tyrosine Kinase Activity Mediates Dimerization," *Protein Engineering*, 3:245-248, 1990.
- D. M. E. Szebenyi and K. Moffat, "The Refined Structure Of Vitamin D-dependent Calcium-binding Protein From Bovine Intestine. Molecular Details, Ion Binding, And Implications For The Structure Of Other Calcium-binding Proteins," *J. Biol. Chem*, 261:8761-8777, 1986.
- W. R. Taylor, "An Algorithm to Compare Secondary Structure Predictions," *J Mol Biol*, 173:512-514, 1984.
- W. R. Taylor and J. M. Thornton, "Recognition of Super-secondary Structure in Proteins," *J Mol Biol*, 173:487-512, 1984.
- J. B. Udgaonkar and R. L. Baldwin, "NMR Evidence for an Early Framework Intermediate on the Folding Pathway of Ribonuclease A," *Nature (London)*, 335:694-699, 1988.
- S. J. Weiner, P. A. Kollman, D. A. Case, U. C. Singh, C. Ghio, G. Alagona, S. Profeta, and P. Weiner, "A New Force Field for Molecular Mechanical Simulation of Nucleic Acids and Proteins," *J Am Chem Soc*, 106:765-784, 1984.
- K. Wuthrich, *NMR of Proteins and Nucleic Acids*, Wiley, New York, 1986.

Appendix A

Common Lisp Source Code for *def-alpps* Macro

```
.....*****  
;;;  
;;;* Match-Set System  
;;;* file: alpps.lisp  
....*  
;;;  
.....*****  
;;;  
;;; These functions are the basis of the segmentation capabilities of  
;;; ALPPS.  
;;;  
  
(in-package 'alpps)  
  
(use-package '(user lisp loop plans))  
  
(import '(loop:mit-loop ))  
(export '(segment-sequence substring hide-blocks expose-blocks split-blocks  
          cat-blocks alpps-eval-all))  
  
;;; def-alpps macro  
(defvar *alpps-pattern-names* nil)  
(setf *alpps-pattern-list* nil)  
  
;;; defaults for automatic turn regions  
(setf *turn-region-name* "auto-turn")  
(setf *turn-region-target* "Cturn")  
(setf *turn-region-symbol* "+")  
(setf *turn-region-color* "Yellow")  
  
(defmacro def-alpps (name lamda-list &rest body)  
  (let ((it (gensym)))  
    `(let ((,it  
           (cons 'progn  
                 (cons (cons 'segment-known-sequence ',lamda-list  
                             ',body))))  
      (setf *alpps-pattern-list*  
            (cons ',name *alpps-pattern-list*))  
      (setf *alpps-pattern-names*  
            (cons (format nil "~s" ',name) *alpps-pattern-names*))  
      (defun ,name ()  
        ,it))))
```

```

(defvar *top-alpps-blocks* nil "place to put seqs from alpps")
(defvar *bottom-alpps-blocks* nil "place to put seqs from alpps")
(defvar *top-visible-alpps-blocks* nil "place to put seqs from alpps")
(defvar *bottom-visible-alpps-blocks* nil "place to put seqs from alpps")
(defvar *top-visible-alpps-regions* nil "place to put seqs from alpps")
(defvar *bottom-visible-alpps-regions* nil "place to put seqs from alpps")
(defvar *default-seq* nil "working sequence name for alpps")
(defvar *default-resseq* nil "working sequence for alpps")
(defvar *default-spat* nil "working alpps pattern")
(defvar *summary-seq* "SUMMARY")

```

```

(defun alpps-eval ()
  "perform alpps patterns see s-patterns-table"
  (mit-loop for spat in *alpps-pattern-list*
    do
      (setf *region-color-list* nil)
      (setf *default-spat* spat)
      (eval (eval (list spat )))
      (setf *default-sb* (get-default-block-pointer))
      (setf *alpps-sb-list*
        (append *alpps-sb-list* (list *default-sb*)))
      (send-to-midas)
      (tally-regions-and-types)))

```

```

(defun alpps-eval-all ()
  "this function should not be in alpps package"
  (setf *alpps-tallys-list* ())
  (setf *alpps-region-type-tallys-list* ())
  (setf *alpps-sb-list* ())
  (start-total-residue-scoring)
  (mit-loop for seq in match-set::*sequence-deck*
    do
      (setf *default-seq* seq)
      (setf *default-resseq* (match-set::resseq-obj-seq
        (gethash seq match-set::resseq-hash)))
      (alpps-eval )))

```



```
;;;
```

```
;;; STRUCTURES FOR BLOCKS
```

```
(defstruct (seq-blocks :named)
  "holds blocks for a given sequence"
  (name) ;( :type string)
  (alpps-pattern) ;( :type string)
  (tolerance) ;( :type integer)
  (length) ;( :type integer)
  (block-list) ;( :type list)
  (predicted-ss-seq) ;( :type string)
  (predicted-helices) ; ( :type list)
  (predicted-strands) ; ( :type list)
  (helix-results) ; ( :type list)
  (strand-results) ; ( :type list)
  (turn-results) ; ( :type list)
  (big-Q)
  (big-C-alpha)
  (big-C-beta)
  (visible-list)) ;( :type list)
```

```
(defstruct (block)
  "main object for blocks"
  (seq) ;( :type string)
  (left-tol) ;( :type string)
  (right-tol) ;( :type string)
  (ss-seq) ;( :type string)
  (length) ;( :type integer)
  (tolerance) ;( :type integer)
  (start) ;( :type integer)
  (end) ;( :type integer)
  (visible) ;( :type bool)
  region
  (markings)) ;( :type list)
```

```
(setf seq-blocks-hash (make-hash-table :size 100 :test #'equal))
;( "hash table for holding seq-blocks")
```

```
(defun segment-sequence (&key (seq *default-seq*) (tol 0) pat
                        (turn-regions nil)
                        (spat *default-spat*))
  (let ((resseq (match-set::resseq-obj-seq
                (gethash seq match-set::resseq-hash))))
    (setq *default-resseq* resseq)
    (segment-known-sequence :seq seq :resseq resseq :turn-regions turn-regions
                           :tol tol :pat pat :spat spat)))
```

```
(defvar tol-pad-char #\? "character for tol area before and after seq")
```

```
(defun segment-known-sequence (&key (seq *default-seq*)
                                    (resseq *default-resseq*)
                                    (turn-regions nil)
                                    (tol 0) pat
                                    (spat *default-spat*))
  (let* ((tol-pad (make-string tol :initial-element tol-pad-char))
        (block-list (make-block-list :seq seq :resseq resseq
                                     :tol tol :pat pat
                                     :turn-regions turn-regions
                                     :left-tol tol-pad
                                     :right-tol tol-pad)))
    (setf (block-left-tol (car block-list)) "")
    ; (setf block-list (reverse block-list))
    (setf (block-right-tol (car (reverse block-list))) "") ; remove padded tol
```

```
(setf (gethash (format nil "~s+~s" seq spat) seq-blocks-hash)
      (make-seq-blocks
       :name seq
       :length (length resseq)
       :alpps-pattern spat
       :tolerance tol
       :block-list block-list
       :visible-list block-list))))
```

```
(defun get-default-block-pointer ()
  (setf *default-sb*
        (gethash (format nil "~s+~s" *default-seq* *default-spat*)
                  seq-blocks-hash)))
```

(defun remove-mid-numbers (l)

"takes a list of numbers and removes middle adjacent numbers.

e.g. (remove-mid-numbers '(

(1 3 4 5 6 7 8 12 13 14 15 18 19 27 28 29))
 ^{^ ^ ^ ^ ^ ^ ^}

gives (1 3 8 12 15 18 19 27 29)"

(mit-loop for i in l with result = () and j = nil and look-back = nil

do

(if (null j)

(setf result (cons i result))

(if (equal i (+ 1 j))

(setf look-back t)

(progn

(if look-back

(setf result (cons j result)))

(setf look-back nil)

(setf result (cons i result))))))

(setf j i)

finally

(if look-back

(setf result (cons j result)))

(return (reverse result)))

(defun get-run-beginnings-only (l &key (run-length 3))

**"takes a list of numbers and returns the beginnings of runs
of length 3 or more.**

e.g. (get-run-beginnings-only '(

'(1 3 4 5 6 7 8 12 13 14 15 18 19 27 28 29))

^ ^ ^ ^ ^ ^ ^

gives (3 12 27)"

(mit-loop for i in l and j in (cdr l) with result = () and hold = (car l) and

count = 1

do

(if (equal (+ i 1) j)

(if (equal count 1)

(progn

(setf hold i)

(setf count 2))

(incf count))

(if (>= count run-length)

(progn

(setf result (cons hold result))

(setf hold i)

(setf count 1))

(progn

(setf hold i)

(setf count 1))))

finally

(if (>= count run-length)

(setf result (cons hold result)))

(return (reverse result)))

```

(defun make-block-list (&key (seq *default-seq*) resseq (tol 0) pat (offset 0)
                      (left-tol "") (right-tol "")
                      (turn-regions nil))
  " block out a sequence "
  (let* ((l (length resseq))
         (match-results (plans-match resseq pat))
         (turn-starts ()) this-block
         block-list obj (start 1)
         (full-seq (format nil "~a~a~a" left-tol resseq right-tol)))
    (progn
      (if turn-regions
          (setf turn-starts (get-run-beginnings-only match-results)))
        (setf match-results (delete 1 (remove-mid-numbers match-results))))
      (mit-loop for i in match-results
                do
                  (setf this-block
                        (make-block
                          :seq (substring resseq :start start :end i)
                          :ss-seq nil
                          :left-tol (substring full-seq
                                                :start start
                                                :end (+ start tol -1))
                          :right-tol (substring full-seq
                                                  :start (+ i tol 1)
                                                  :end (+ i tol tol))
                          :length (- (+ i 1) start)
                          :start (+ start offset)
                          :end (+ i offset) :region nil
                          :tolerance tol :visible t
                          :markings nil))
                  (setf block-list
                        (cons this-block block-list))
                  (if (and turn-regions (member start turn-starts))
                      (setf (block-region this-block)
                            (make-region
                              :name *turn-region-name*
                              :seq (block-seq this-block)
                              :length (block-length this-block)
                              :start (block-start this-block)
                              :target *turn-region-target*
                              :symbol *turn-region-symbol*
                              :color *turn-region-color*
                              :end (block-end this-block)
                              :visible t)))
                      (setq start i))

```

```

(if (not (equal start 1))
  (progn
    (setf this-block
      (make-block
        :seq (substring resseq :start start :end 1)
        :ss-seq nil
        :left-tol (substring full-seq
          :start (- start tol)
          :end (- start 1))
        :right-tol (substring full-seq
          :start (+ 1 1)
          :end (+ 1 tol))
        :length (- (+ 1 1) start)
        :start (+ start offset)
        :end (+ 1 offset)
        :tolerance tol :visible t
        :markings nil))
      (setf block-list
        (cons this-block block-list))
      (if (and turn-regions (member start turn-starts))
        (setf (block-region this-block)
          (make-region
            :name *turn-region-name*
            :seq (block-seq this-block)
            :length (block-length this-block)
            :start (block-start this-block)
            :target *turn-region-target*
            :symbol *turn-region-symbol*
            :color *turn-region-color*
            :end (block-end this-block)
            :visible t))))))
  (reverse block-list))))

```

```

(defun substring (string &key (start 1) (end (length string)))
  " returns a substring of string . Does some checking for reasonable bounds."
  (let (result)
    (if (or (string-equal string "") (null string))
        (setf result string)
        (progn
          (if (< start 1) (setf start 1))
          (if (> end (length string)) (setf end (length string)))
          (setf result
                (make-string (- (+ 1 end) start) :initial-element #\space))
          (mit-loop for i from (- start 1) to (- end 1)
                    with j = 0
                    do
                    (setf (elt result j) (elt string i))
                    (incf j))))))
  result))

```

```

(defun show-all-blocks (&key (seq *default-seq*) (spat *default-spat*))
  (preshow-all-blocks :seq seq :spat spat)
  (format t "[a]" *top-alpps-blocks*)
  (format t "[a]" *bottom-alpps-blocks*))

```

```

(defun preshow-all-blocks (&key (seq *default-seq*) (spat *default-spat*))
  "retrun list of blocks for a given sequence in up-down fashion"
  (let* ((seq-ptr (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
         (tol (seq-blocks-tolerance seq-ptr))
         (top-list nil)
         (l (seq-blocks-length seq-ptr)))
    (setf *top-alpps-blocks*
          (make-string l :initial-element #\space))
    (setf *bottom-alpps-blocks*
          (make-string l :initial-element #\space))
    (mit-loop
     for block in (seq-blocks-block-list seq-ptr)
     with tol = (seq-blocks-tolerance seq-ptr)
     do
      (setf top-list (not top-list))
      (mit-loop for i from (max 1 (- (block-start block) tol))
                to (min 1 (+ (block-end block) tol))
                and j in (coerce
                          (format nil "~a~a~a" (block-left-tol block)
                                                (block-seq block)
                                                (block-right-tol block)) 'list)
                do
                (if top-list
                    (setf (elt *top-alpps-blocks* (- i 1)) j)
                    (setf (elt *bottom-alpps-blocks* (- i 1)) j)))
      ))))

```



```

(defun show-visible-blocks (&key (seq *default-seq*) (spat *default-spat*))
  "retrun list of visible blocks for a given sequence"
  (setf *default-block-list*
        (seq-blocks-visible-list
         (gethash (format nil "~s+~s" seq spat) seq-blocks-hash)))

  (mit-loop
   for block in (seq-blocks-visible-list
                (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
   with result = nil and region-list = nil
   do

     (setf *default-block* block)
     (setf result
            (cons (list (block-start block) (block-end block)) result))
     (if (block-region block)
         (setf region-list
                (cons (list (region-start (block-region block))
                            (region-end (block-region block)))
                      region-list)))

   finally
     (print (reverse region-list))
     (return (reverse result))))

```

```
(defun hide-blocks (&key (seq *default-seq*) (spat *default-spat*) pat
                    (spat-min-count 1)
                    (no-limit nil))
```

"looks for segments which have pattern and are then hidden"

```
(mit-loop
  for block in (seq-blocks-visible-list
               (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
  with result = nil and plans-results = nil
  do
  (if (null no-limit)
      (setf plans-results (plans-match (block-seq block) pat))
      (setf plans-results (interval
                          (- (block-start block) (block-tolerance block))
                          (+ (block-end block) (block-tolerance block))
                          (plans-match *default-resseq* pat))))
  (if (< (length plans-results) spat-min-count)
      (setf result (cons block result))
      (setf (block-visible block) nil))
  finally
  (setf (seq-blocks-visible-list
        (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
        (reverse result))))
```

```
(defun expose-blocks (&key (seq *default-seq*) (spat *default-spat*) pat
                      (spat-min-count 1) (no-limit nil))
```

"remove blocks from hiding based on pat"

```
(mit-loop
  for block in (seq-blocks-block-list
               (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
  with result = nil and plans-results = nil
  do
  (if (not (block-visible block)) ; only build if necessary
      (if (null no-limit)
          (setf plans-results (plans-match (block-seq block) pat))
          (setf plans-results (interval
                              (- (block-start block) (block-tolerance block))
                              (+ (block-end block) (block-tolerance block))
                              (plans-match *default-resseq* pat))))
      (if (or (block-visible block)
              (not (< (length plans-results) spat-min-count)))
          (progn
            (setf (block-visible block) t)
            (setf result (cons block result))))
      finally
  (setf (seq-blocks-visible-list
        (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
        (reverse result))))
```

```

(defun expose-all-blocks (&key (seq *default-seq*) (spat *default-spat*))
  (let ((sb (gethash (format nil "~s+~s" seq spat) seq-blocks-hash)))
    (mit-loop
     for block in (seq-blocks-block-list sb)
     do
      (setf (block-visible block) t)
      (setf (seq-blocks-visible-list sb)
            (seq-blocks-block-list sb))))))

```

```

(defun split-long-blocks
  (&key (seq *default-seq*) (max-length 100) (spat *default-spat*))
  "looks for segments which are longer than max-length and divide them
in half"
  (mit-loop
   for block in (seq-blocks-block-list
                 (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
   with block-results = nil and visible-results = nil
   do
    (if (not (block-visible block))
        (setf block-results (cons block block-results))
        (if (not (> (block-length block) max-length))
            (progn
             (setf block-results (cons block block-results))
             (setf visible-results (cons block visible-results)))
            (progn
             (mit-loop for i in (cut-block :block block
                                           :cut-point (round (/ max-length 2)))
             do
              (setf block-results (cons i block-results))
              (setf visible-results (cons i visible-results))))))
    finally
    (setf (seq-blocks-block-list
          (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
          (reverse block-results))
      (setf (seq-blocks-visible-list
            (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
            (reverse visible-results))))))

```

```

(defun cut-block (&key block cut-point)
  "given a block, return a list of 2 cut at cut-point"
  (let ((left-block nil) (right-block nil)
        (i 0) (offset 0))
    (format t "using cut-block~%")
    (setf tol (block-tolerance block))
    (setf offset (- (block-start block) 1))
    (setf left-block
      (make-block
       :seq (substring (block-seq block) :end cut-point)
       :ss-seq nil
       :left-tol (block-left-tol block)
       :right-tol (substring (block-seq block)
                             :start (+ cut-point 1)
                             :end (+ cut-point tol))
       :length cut-point
       :start (block-start block)
       :end (+ cut-point offset)
       :region nil
       :tolerance tol
       :visible t
       :markings nil))
      (setf right-block
        (make-block
         :seq (substring (block-seq block) :start cut-point)
         :ss-seq nil
         :left-tol (substring (block-seq block)
                              :start (- cut-point tol)
                              :end (- cut-point 1))
         :right-tol (block-right-tol block)
         :length (- (block-length block) (- cut-point 1))
         :start (block-end left-block)
         :end (block-end block)
         :region nil
         :tolerance tol
         :visible t
         :markings nil))
        (list right-block left-block)))

```

```

(defun split-blocks (&key (seq *default-seq*) pat (spat *default-spat*))
  "looks for segments which have pattern and are then split"
  (mit-loop
   for block in (seq-blocks-block-list
                 (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
   with block-results = nil and visible-results = nil and new-blocks = nil
   do
     (if (not (block-visible block))
         (setf block-results (cons block block-results))
         (if (not (plans-match (block-seq block) pat))
             (progn
              (setf block-results (cons block block-results))
              (setf visible-results (cons block visible-results)))
             (progn
              (setf new-blocks
                    (reverse (make-block-list :resseq (block-seq block) :pat pat
                                             :left-tol (block-left-tol block)
                                             :right-tol (block-right-tol block)
                                             :tol (block-tolerance block)
                                             :offset (- (block-start block) 1))))
              (setf block-results (append new-blocks block-results))
              (setf visible-results (append new-blocks visible-results))))))
   finally
   (setf (seq-blocks-block-list
         (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
         (reverse block-results))
   (setf (seq-blocks-visible-list
         (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
         (reverse visible-results))))

```

```

(defun cat-blocks (&key (seq *default-seq*) pat1 pat2 (spat *default-spat*))
  "looks for adjacent segments which have pattern and are then merged"
  (let* ((block-list (seq-blocks-block-list
                     (gethash (format nil "~s+~s" seq spat) seq-blocks-hash)))
         (first-block (car block-list)))
    (setf block-list (delete first-block block-list))
    (mit-loop
     for second-block in block-list
     with block-results = nil and visible-results = nil
     do
      (if (not (block-visible first-block))
          (progn
           (setf block-results (cons first-block block-results))
           (setf first-block second-block))
          (if (not (and
                   (block-visible second-block)
                   (plans-match (block-seq first-block) pat1)
                   (plans-match (block-seq second-block) pat2)))
              (progn
               (setf block-results (cons first-block block-results))
               (setf visible-results (cons first-block visible-results))
               (setf first-block second-block))
              (merge-to-one-block first-block second-block)))
          finally
          (setf block-results (cons first-block block-results))
          (if (block-visible first-block)
              (setf visible-results (cons first-block visible-results)))
          (setf (seq-blocks-block-list
                (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
                (reverse block-results))
          (setf (seq-blocks-visible-list
                (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
                (reverse visible-results))))))

```

```

(defun merge-to-one-block (first-block second-block)
  "combines two block structs into the first"
  (setf (block-seq first-block)
        (format nil "~a~a" (block-seq first-block)
                  (substring (block-seq second-block) :start 2)))
  (setf (block-ss-seq first-block)
        (format nil "~a~a" (block-ss-seq first-block)
                  (substring (block-ss-seq second-block) :start 2)))
  (setf (block-length first-block) (+ (block-length first-block)
                                       (block-length second-block)))
  (setf (block-end first-block) (block-end second-block))
  (setf (block-markings first-block)
        (merge-block-markings (block-markings first-block)
                              (block-markings second-block))))

```

```

(defun merge-block-markings (first-list second-list)
  "combines two block marking lists"
  )

```

```

(defun equal-pair (a b)
  "test for equality of all member of list"
  (if (and (null a) (null b))
      T
      (if (and (equal (car a) (car b))
                (equal-pair (cdr a) (cdr b)))
          T
          nil)))

```



```

.....*****
;;;
;;;* Match-Set System
;;;* file: globals.lisp
.....*
.....*****

```

```

(in-package :match-set)
(use-package '(lisp user loop plans))

```

```

;;;; LISTS for patterns
(defvar *plans-results-list* nil)
(defvar *alpps-pattern-list* nil)
(defvar *alpps-pattern-names-list* nil)
(defvar *plans-pattern-list* nil)
(defvar *ssblock-types* '("H" "S" "L" "T" ) "sec struct types")

```

```

(defvar *ms-output-directory* "." "place for ms output")
(defvar *default-plans-output-file* "plans-summary.tbl")
(defvar *default-alpps-tally-output-file* "alpps-summary.tbl")
(defvar *default-plans-seq-output-file* "plans-dump.roff")
(defvar *send-helix-tally* t)

```

```

;;; flags for controlling running of ms
(defvar *do-alpps-runs* t)
(defvar *control-file-lock-out* nil "if T, don't reload control-file")
(defvar *just-peeking* nil "if T, don't show primary sequence")
(defvar *dump-target* nil "possible target for general run")

```

```

;;; bags of sec struct symbols for look-down
(defvar *Helix* (coerce "a34jk" 'list))
(defvar *Strand* (coerce "be" 'list))
(defvar *Cturn* (coerce "tljk" 'list))

```

```

(setf *hit-char-list* '(! @ # $ % ^ & * ( #) #- #_
                      #= #+ #\ #\~ #' #< #> #{ #}
                      #[ #] #: #\; #' #' #? #V #\, #\))

```

```

(deftype bool ()
  "a logical type"
  '(or (satisfies null) (satisfies symbolp)))

```

```
;;; residue sequence object
(defstruct (resseq-obj :named)
  "main object for dealing with matches"
  (name)
  (seq)
  (ss-seq)
  (ss-3-seq) ; place to put abt type ss-assignments
  (loops) ;; (:type list "start stops for each class")
  (turns)
  (strands)
  (helices)
  (match-list) ;; (:type list "((1 2 3) (3 9 10) ...)")
  (results-list) ;; (:type list "(T L F)")
  (alpps-runs) ;; (:type list "pointers to alpps results")

(defvar resseq-hash (make-hash-table :size 100 :test #'equal)
  "hash table for holding resseq-objs")
```

```

...*****
;;;
;;;* Match-Set System
;;;* file: helix-output.lisp
...*
;;;
...*
;;;
...*
;;;
...*****
;;;

```

```
(in-package :alpps)
```

```
(use-package '(lisp user loop plans match-set))
```

```
(setf tab-char #\tab)
```

```
(setf *default-alpps-helix-output-file* "helix-results.tbl")
```

```
(defun send-helix-tallys (&key (file ""))
```

```
  "build an output file with alpps helix results"
```

```
  (let (file-name (assigned-total 0) (predicted-total 0) (type-totals ()))
```

```
    (type-count (length *region-color-list*)) )
```

```
  (progn
```

```
    (mit-loop for k from 1 to type-count
```

```
      do
```

```
        (setf type-totals (cons 0 type-totals))))
```

```
  (if (equal file "")
```

```
      (setf file *default-alpps-helix-output-file*)))
```

```
  (setq file-name (format nil "~a/~a"
```

```
                      match-set::*ms-output-directory* file))
```

```
  (format t "type-count ~d~%" type-count)
```

```
  (with-open-file
```

```
    (output-stream file-name
```

```
      :direction :output
```

```
      :if-exists :new-version
```

```
      :if-does-not-exist :create)
```

```
    (format output-stream ".sp 2~%")
```

```
    (format output-stream "~a~%~a~%~a~%~a~%~a~%~a~%"
```

```
      ".TS"
```

```
      "box;"
```

```
      "c s s s s s"
```

```
      "l l c c c s"
```

```
      "l l c c c c"
```

```
      "l l || n | n | n | n.")
```

```
    (format output-stream "Match-Set ALPPS Helix Tallys~%")
```

```
    (format output-stream
```

```
      "Sequence~aPattern~aAssigned~aPredicted~aRegions~%"
```

```
      tab-char tab-char tab-char tab-char)
```

```

(format output-stream
  "~a~a~a~aType~aCount~%=~%"
  tab-char tab-char tab-char tab-char tab-char)
(mit-loop for i in (reverse *alpps-tallys-list*) and
  for j in (reverse *alpps-region-type-tallys-list*) and
  assigned-count = 0
  do
  (setf assigned-count
    (length
      (match-set::resseq-obj-helices
        (gethash (car i) match-set::resseq-hash))))
  (incf assigned-total assigned-count)
  (incf predicted-total (fifth i))
  (format output-stream
    "~a~a~a~a~a~a~a~%"
    (car i) tab-char
    (cadr i) tab-char
    assigned-count tab-char
    (fifth i))
  (mit-loop for k from 0 to (- type-count 1)
  do
  (incf (nth k type-totals) (nth (+ k 2) j))
  (format output-stream
    "~a~a~a~a~a~a~a~d~%"
    tab-char tab-char tab-char tab-char
    (car (nth k *region-color-list*)) tab-char
    (nth (+ k 2) j)))
  finally
  (format output-stream
    "=~%"")
  (format output-stream
    "Totals~a~a~d~a~d~%"
    tab-char tab-char assigned-total tab-char
    predicted-total)
  (mit-loop for k from 0 to (- type-count 1)
  do
  (format output-stream
    "~a~a~a~a~a~a~a~d~%"
    tab-char tab-char tab-char tab-char
    (car (nth k *region-color-list*)) tab-char
    (nth k type-totals)))
  (format output-stream ".TE~%")))))))

```

```

.....*****
;;;
;;;* Match-Set System
;;;* file: look-down.lisp
....*
;;;
.....*****

```

```

(in-package :match-set)
(use-package '(lisp user loop plans ))

```

```

(defvar *true-pos* 0)
(defvar *tol-pos* 1)
(defvar *false-pos* 2)

```

```

(defun look-down (&key ss-seq pos tol goal)
  "look down a ss-seq from position pos and try to find goal. returns
  *true-pos* *tol-pos* false-pos*"
  (let ((true-pos nil) (found nil))
    (setf pos (- pos 1))
    (if (member (nth pos ss-seq) goal)
        (setf true-pos t)
        (mit-loop for i from (max (- pos tol) 0) to
                  (min (+ pos tol) (- (length ss-seq) 1))
                  do
                    (if (member (nth i ss-seq) goal)
                        (progn
                          (setf found t) (loop::loop-finish))))))
    (if true-pos
        *true-pos*
        (if found
            *tol-pos*
            *false-pos*))))

```

```

(defun make-pat-results-list (&key ss-seq tol goal match-list)
  (mit-loop for i in match-list with positive = 0
            and true-pos = 0 and tol-pos = 0 and false-pos = 0
            do
              (setf positive (look-down :ss-seq ss-seq
                                       :pos i :tol tol :goal goal))
              (case positive
                ((0) (incf true-pos))
                ((1) (incf tol-pos))
                ((2) (incf false-pos)))
              finally
                (return (list true-pos tol-pos false-pos))))

```

```

.....*****
;;;
;;;* Match-Set System
;;;* file: ms.lisp
.....*
.....*****
;;;

```

```

(in-package :match-set)
(use-package '(lisp user loop plans alpps))

```

```

(defun m-s ()
  (ms-main))

```

```

(defun ms-main ()
  "main-line for ms"
  (setup-for-ms)
  (mit-loop for protein in *sequence-deck*
    do
      (evaluate-protein (gethash protein resseq-hash )))
  (send-plans-results)
  (if *do-alpps-runs*
    (progn
      (alpps-eval-all)
      (send-alpps-tallys)))
  (send-plans-sequences)
  (if *send-helix-tallys*
    (alpps::send-helix-tallys))
  (say-good-bye))

```

```

(defun evaluate-protein (obj)
  (do-plans-matching obj))

```

```

(defun print-proteins ()
  (mit-loop for protein in *sequence-deck*
    do
      (print (gethash protein resseq-hash ))))

```

```

(defun do-plans-matching (obj)
  "perform match on all patterns on one resseq"
  (let ((seq (resseq-obj-seq obj))
        (ss-seq (coerce (resseq-obj-ss-seq obj) 'list))
        (match-list ())
        (matches ())
        (results ())
        (new-plans-results-list ())
        (results-list ()))
    (mit-loop
     for pattern in (reverse *plans-pattern-list*)
     and plans-results in (reverse *plans-results-list*)
     do
      (setq matches (plans-match seq (car pattern)))
      (setq match-list (cons matches match-list))
      (setq results (make-pat-results-list :ss-seq ss-seq
                                           :tol *plans-tol*
                                           :goal (eval (cadr pattern))
                                           :match-list matches))
      (setq results-list (cons results results-list))
      (setq new-plans-results-list (cons (list-add results plans-results)
                                         new-plans-results-list)))
      (setq *plans-results-list* new-plans-results-list)
      (setf (resseq-obj-results-list obj) results-list)
      (setf (resseq-obj-match-list obj) match-list)))

  (setf protein-data-loaded nil)

```

```
(defun setup-for-ms ()
  (if (not *control-file-lock-out*)
    (load
     (format nil "~a/~a" *match-set-working-dir* "control.lisp")))
    (if *do-alpps-runs*
      (progn
        (setf alpps::*alpps-pattern-list* nil) ; reset to prevent doubles
        (load
         (format nil "~a/~a" *match-set-working-dir* "alpps-control.lisp"))))
      (if (not protein-data-loaded)
        (progn
          (load
           (format nil "~a/~a" *match-set-working-dir* *protein-data*))
          (setf protein-data-loaded t)))
        (setf *plans-results-list* ())
        (mit-loop for i in *plans-pattern-list* do
          (setf *plans-results-list* (cons '(0 0 0)
                                           *plans-results-list*)))
        (init-plans-pat-file *plans-pattern-file*))

  (defun say-good-bye ()
    (format t "good-bye~%"))
```



```

...*****
;;;
;;;* Match-Set System
;;;* file: plans-output.lisp
...*
;;;
...*****

```

```

(in-package :match-set)
(use-package '(lisp user loop plans alpps))

```

```

(setf tab-char #\tab)
(setf break-length 150)

```

```

(defun send-plans-sequences (&key (file ""))
  "build an output file with plans sequence results"
  (let (file-name obj hits len)
    (progn
      (if (equal file "")
          (setf file *default-plans-seq-output-file*))
          (setq file-name (format nil "~a/~a"
                                  *ms-output-directory* file)))
      (with-open-file
        (output-stream file-name
                       :direction :output
                       :if-exists :new-version
                       :if-does-not-exist :create)
          (format output-stream ".ps 8~%"")
          (format output-stream ".vs 9~%"")
          (format output-stream ".nf~%"")
          (format output-stream ".ll 10i~%"")
          (format output-stream ".po .5i~%"")
          (format output-stream ".pl 8i~%"")
          (format output-stream ".de NP~%"")
          (format output-stream
                  ".tl '\n(mo\n(dy\n(yr'Match-Set PLANS Sequence Results'~%"")
                  (format output-stream ".sp~%"")
                  (format output-stream "..~%"")
                  (format output-stream ".wh 0 NP~%"")
                  (format output-stream "\fC~%"")
                  (format output-stream ".NP~%"")
                  (format output-stream "Pattern Symbols:~%"")
                  (mit-loop for i from 0 to (- (length *plans-pattern-list*) 1)
                            with hit-char = nil and pat = nil do
                              (setq hit-char (nth i *hit-char-list*))
                              (setq pat (car (nth i *plans-pattern-list*)))
                              (format output-stream "~a- ~a~%" hit-char pat))
                  (format output-stream "~%~%"")

```

```

(mit-loop for protein in *sequence-deck*
  do
    (setq obj (gethash protein resseq-hash))
    (setf len (length (resseq-obj-seq obj)))
    (if (<= len break-length)
      (progn
        (format output-stream ".ne ~d~%"
          (+ (length *plans-pattern-list*) 5))
        (format output-stream "Sequence Name:~a Length: ~d~%"
          (resseq-obj-name obj) len)
        (if (null *just-peeking*)
          (format output-stream "~a~%" (resseq-obj-seq obj)))
        (if (null *dump-target*)
          (format output-stream "~a~%"
            (resseq-obj-ss-seq obj))
          (format output-stream "~a~%"
            (reduce-string (resseq-obj-ss-seq obj)
              :show *dump-target*)))
        (if *do-alpps-runs*
          (dump-alpps-predictions :start 0
            :end nil
            :protein protein
            :output output-stream))
        (mit-loop for i from 0 to
          (- (length *plans-pattern-list*) 1)
          with hit-char = nil do
            (setq hits
              (nth i (resseq-obj-match-list obj)))
            (setq hit-char (nth i *hit-char-list*))
            (format output-stream
              "~a~%"
              (make-match-string
                len hits
                :hit-char hit-char)))
            (if (null *just-peeking*)
              (format output-stream "~a~%"
                (resseq-obj-seq obj))))))

```

```

(mit-loop for j from 0 to (truncate (/ len break-length))
  with start = 0 and end = len
  do
    (setf start (* j break-length))
    (setf end (min len (+ start break-length)))
    (format output-stream ".ne ~d~%"
      (+ (length *plans-pattern-list*) 5))
    (format output-stream
      "Sequence Name:~a From: ~d To: ~d~%"
      (resseq-obj-name obj) (+ start 1) end)
    (if (null *just-peeking*)
      (format output-stream "~a~%"
        (subseq
          (resseq-obj-seq obj) start end)))
    (if (null *dump-target*)
      (format output-stream "~a~%"
        (subseq (resseq-obj-ss-seq obj)
          start end))
      (format output-stream "~a~%"
        (reduce-string
          (subseq (resseq-obj-ss-seq obj)
            start end)
          :show *dump-target*)))
    (if *do-alpps-runs*
      (dump-alpps-predictions
        :start start
        :end end
        :protein protein
        :output output-stream))
    (mit-loop for i from 0 to
      (- (length *plans-pattern-list*) 1)
      with hit-char = nil do
        (setq hits
          (nth i
            (resseq-obj-match-list obj)))
        (setq hit-char (nth i *hit-char-list*))
        (format output-stream
          "~a~%" (subseq
            (make-match-string
              len hits
              :hit-char hit-char)
            start end)))
    (if (null *just-peeking*)
      (format output-stream "~a~%~%"
        (subseq (resseq-obj-seq obj)
          start end))))))

```

```

(defun seq-portion (seq size part)
  "returns the partth sized portion of seq"
  (let* ((len (length seq))
         (start (* part size))
         (end (+ (min len (+ start size)) 0)))
    (subseq seq start end)))

(defun send-plans-results (&key (file ""))
  "build an output file with plans results in format for class analysis"
  (let (file-name obj results t-p tol-p f-p d-p (total 0) (res-count 0)
        (hit-count 0) (hit-percent 0))
    (progn
      (if (equal file "")
          (setf file *default-plans-output-file*))
      (setq file-name (format nil "~a/~a"
                             *ms-output-directory* file))
      (with-open-file
        (output-stream file-name
                       :direction :output
                       :if-exists :new-version
                       :if-does-not-exist :create)
        (format output-stream ".sp 2~%")
        (format output-stream "~a~%~a~%~a~%~a~%~a~%~a~%"
                ".TS"
                "box;"
                "c s s s s s"
                "l c c c c c"
                "l c c c c c"
                "l || n | n | n | n | n | n.")
        (format output-stream "Match-Set PLANS Results~%")
        (format output-stream "Pattern~aTrue~aTolerated~aFalse~aDecent~a"
                tab-char tab-char tab-char tab-char tab-char)
        (format output-stream " Hit of~a Hit of~%"
                tab-char)
        (format output-stream "~aPositives~aPositives~aPositives~aPercentage~a"
                tab-char tab-char tab-char tab-char tab-char)
        (format output-stream "All Residues~aUnmasked~%"
                tab-char)

```

```

(mit-loop for protein in *sequence-deck*
  do
    (setq obj (gethash protein resseq-hash ))
    (setf res-count (length (resseq-obj-seq obj)))
    (format output-stream "~%a: ~d res~%"
      (resseq-obj-name obj) res-count)
    (incf total (+ res-count))
    (format output-stream ".\
      tab-char res-count)
    (mit-loop for i from 0 to (- (length *plans-pattern-list*) 1)
      do
        (setq results (nth i (resseq-obj-results-list obj)))
        (setf t-p (nth 0 results))
        (setf tol-p (nth 1 results))
        (setf f-p (nth 2 results))
        (setf hit-count (+ t-p tol-p f-p))
        (if (not (equal hit-count 0))
          (setf d-p
            (/ (* (+ t-p tol-p) 100.0)
              hit-count))
          (setf d-p 0))
        (format output-stream
          " ~a~d~a~d~a~d~a~,1f"
          (car (nth i *plans-pattern-list*)) tab-char
          t-p tab-char tol-p
          tab-char f-p
          tab-char d-p)
        (if (not (equal 0 res-count))
          (format output-stream
            "~a~,1f~%"
            tab-char
            (/ (* 100.0 hit-count) res-count))
          (format output-stream "~%"))))

```

```

(format output-stream "=~%~a: ~d res~%" "Totals" total)
(format output-stream ".\
  tab-char total)
(mit-loop for i from 0 to (- (length *plans-pattern-list*) 1)
  do
    (setq results (nth i *plans-results-list*))
    (setf t-p (nth 0 results))
    (setf tol-p (nth 1 results))
    (setf f-p (nth 2 results))
    (if (not (equal (+ t-p tol-p f-p) 0))
      (setf d-p (/ (* (+ t-p tol-p) 100.0) (+ t-p tol-p f-p)))
      (setf d-p 0))
    (format output-stream " ~a~d~a~d~a~d~a~,1f~"
      (car (nth i *plans-pattern-list*)) tab-char
      t-p tab-char tol-p
      tab-char f-p
      tab-char d-p)
    (if (not (equal 0 total))
      (format output-stream
        "~a~,1f~%"
        tab-char
        (/ (* 100.0 (+ t-p tol-p)) total))
      (format output-stream "~%"))
    (format output-stream ".TE~%" ))))

```

```

(defun send-alpps-tallys (&key (file ""))
  "build an output file with alpps results"
  (let (file-name (nils 0) (alphas 0) (betas 0) (turns 0) (row-tot 0))
    (progn
      (if (equal file "")
          (setf file *default-alpps-tally-output-file*))
      (setq file-name (format nil "~a/a"
                             *ms-output-directory* file))
      (with-open-file
        (output-stream file-name
                       :direction :output
                       :if-exists :new-version
                       :if-does-not-exist :create)
        (format output-stream ".sp 2~%")
        (format output-stream "~a~%~a~%~a~%~a~%~a~%"
                 ".TS"
                 "box;"
                 "c s s s s s s"
                 "l l c c c c c"
                 "l l c c c c c"
                 "l l | | n | n | n | n | n | n.")
        (format output-stream "Match-Set ALPPS Tallys~%")
        (format output-stream
                 "Sequence~aPattern~aNILS~aTurns~aHelices~aStrands"
                 tab-char tab-char tab-char tab-char tab-char tab-char tab-char)
        (format output-stream
                 "~aHelix~aStrand~%"
                 tab-char tab-char)
        (format output-stream
                 "~a~a~a~a~aPercent~aPercent~%_~%"
                 tab-char tab-char tab-char tab-char tab-char tab-char)

```

```

(mit-loop for i in (reverse alpps::*alpps-tallys-list*)
  do
    (incf nils (third i))
    (incf alphas (fifth i))
    (incf betas (sixth i))
    (incf turns (fourth i))
    (format output-stream
      "~a~a~a~a~a~a~a~a~a~a"
      (car i) tab-char
      (cadr i) tab-char
      (caddr i) tab-char
      (fourth i) tab-char (fifth i) tab-char (sixth i))
    (setf row-tot (+ (third i) (fourth i) (fifth i) (sixth i)))
    (format output-stream
      "~a~,1f~a~,1f~%"
      tab-char (/ (* (fifth i) 100.0) row-tot)
      tab-char (/ (* (sixth i) 100.0) row-tot)))
  (format output-stream
    "=~%~a~a~a~a~a~a~a~a~a~a"
    "Totals" tab-char tab-char nils tab-char turns
    tab-char alphas tab-char betas)
  (setf row-tot (+ nils alphas betas turns))
  (format output-stream
    "~a~,1f~a~,1f~%"
    tab-char (/ (* alphas 100.0) row-tot)
    tab-char (/ (* betas 100.0) row-tot))
  (format output-stream ".TE~%" )))

(defun dump-alpps-predictions (&key start end protein output)
  "puts out predicted sequence"
  (mit-loop for spat in (reverse alpps::*alpps-pattern-list*)
    with sb = nil and result = nil and better-result
    do
      (setf sb (gethash (format nil "~s+~s" protein spat)
        alpps::seq-blocks-hash))
      (setf result (subseq (alpps::seq-blocks-predicted-ss-seq sb)
        start end))
      (setf better-result (string-char-replace
        :string result :old #\a :new #\h))
      (setf result (string-char-replace
        :string result :old #\t :new #\c))

      (format output "~a~%" result)))

```



```

.....*****
;;;
;;;* Match-Point System
;;;* file: regions.lisp
.....*
;;;*
;;;* CHANGES:
;;;* 4/4/90 mid pattern for region
;;;* 10/4/90 changes for tallying region types
;;;* 2/19/91 fix minimum residue count for regions after chopping TU
.....*
;;;*
.....*
.....*****
;;;

```

```
(in-package 'alpps)
```

```
(use-package '(user lisp plans loop))
```

```
(import '(loop:mit-loop))
```

```
(export '(make-regions hide-regions expose-regions check-region
          report-all-regions tally-regions))
```

```
(defvar blue "blue")
```

```
(defvar *default-color* Blue)
```

```
(defvar *region-color-list* nil)
```

```
(defstruct (region)
```

```
  "main object for regions"
```

```
  (name)
```

```
  (seq)
```

```
  (ss-seq)
```

```
  (target)
```

```
  (symbol)
```

```
  (length)
```

```
  (start) ; with respect to entire sequence
```

```
  (end)
```

```
  (color)
```

```
  (visible))
```

```
;;; change for minimum AFTER dropping TU
```

```
;;; after change to eliminate TU from regions
```

```

(defun make-a-region (&key block start-pat end-pat name color target symbol
                    (mid-pat nil)
                    (start-n t)
                    (end-c t)
                    (min-size 1) (no-limit nil))
  " make a region in a given block"
  (let* ((full-seq (format nil "~a~a~a" (block-left-tol block)
                                (block-seq block)
                                (block-right-tol block)))
        (offset (- (block-start block) (length (block-left-tol block)) 1))
        start-list stop-list mid-list
        (start nil)
        (stop nil))
    (block nil
      (if (null no-limit)
          (progn
            (setf start-list (plans-match full-seq start-pat))
            (setf stop-list (plans-match full-seq end-pat)))
          (progn
            (setf start-list
                  (interval
                   (- (block-start block) (block-tolerance block))
                   (+ (block-end block) (block-tolerance block))
                   (plans-match *default-resseq* start-pat)))
            (setf stop-list
                  (interval
                   (- (block-start block) (block-tolerance block))
                   (+ (block-end block) (block-tolerance block))
                   (plans-match *default-resseq* end-pat)))
            (setf offset 0)))
      (if start-n
          (setf start (car start-list))
          (setf start (car (last start-list))))
      (if end-c
          (setf stop (car (last stop-list)))
          (setf stop (car stop-list)))
      (if (or (null start) (null stop) (< (- stop start) (- min-size 1)))
          (return nil))
    )
  )

```

```

(if (not (null mid-pat))
  (progn
    (setf mid-list
      (interval start stop (plans-match *default-resseq* mid-pat)))
    (if (null mid-list)
        (return nil))))
(setf start (max (+ start offset) (+ (block-start block) 1)))
(setf stop (min (+ stop offset) (- (block-end block) 1)))
;; recheck for minimum after TU has been eliminated
(if (< (- stop start) (- min-size 1))
    (return nil))
(setf (block-region block)
  (make-region
    :name name
    :length (+ 1 (- stop start))
    :start start
    :target target
    :symbol symbol
    :color color
    :end stop
    :visible t))
(return t)))

(setf *nil-region*
  (make-region
    :name "*nil-region*"
    :visible nil))

```

```

(defun make-regions (&key (seq *default-seq*) (spat *default-spat*)
                    (color *default-color*)
                    (start-n t)
                    (end-c t)
                    (mid-pat nil)
                    (min-size 0)
                    (no-limit nil)
                    start-pat end-pat name target (symbol " "))
  "make regions in visible blocks"
  (setf *region-color-list* (cons (list name color) *region-color-list*))
  (mit-loop
   for block in (seq-blocks-visible-list
                 (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
   with count = 0
   do
   (if (and (block-visible block)
            (null (block-region block))
            (not (null (make-a-region :block block :start-pat start-pat
                                     :symbol symbol
                                     :color color
                                     :no-limit no-limit
                                     :start-n start-n
                                     :end-c end-c
                                     :mid-pat mid-pat
                                     :min-size min-size
                                     :end-pat end-pat :name name
                                     :target target))))
       (incf count))
   finally
   (return count)))

(defun hide-regions (&key (seq *default-seq*) (spat *default-spat*)
                    name)
  "hide all blocks with regions named name"
  )

(defun expose-regions (&key (seq *default-seq*) (spat *default-spat*)
                      name)
  "expose all blocks with regions named name"
  )

```

```
(defun check-regions (&key (seq *default-seq*) (spat *default-spat*)
                    pat)
  "do something for blocks whose regions contain pat"
  )
```

```
(defun report-all-regions (&key (seq *default-seq*) (spat *default-spat*))
  "builds a list which includes the seq-name and the list of region
  targets. A "NIL" is given if there is no region in a given block."
  ; e.g. ("2kai-b" "Cturn" "beta" "Cturn" "beta" "Cturn" "beta" "Cturn" "beta")
  (let ((sb (gethash (format nil "~s+~s" seq spat) seq-blocks-hash))
        reg (goals ()) (names ()) (result ()))
    (setf result (cons (format nil "~a" (seq-blocks-name sb)) result))
    (setf result (cons (format nil "~a"
                            (seq-blocks-alpps-pattern sb)) result))

    (mit-loop
     for block in (seq-blocks-block-list sb)
     do
     (setf reg (block-region block))
     (if (null reg)
        (progn
         (setf goals (cons
                    (format nil "~a" "NIL") goals)))

        (progn
         (setf goals (cons
                    (format nil "~a" (region-target reg)) goals))
         (setf names (cons
                    (format nil "~a" (region-name reg)) names))))

    finally
    (setf result (cons goals result))
    (return (reverse (cons names result))))))
```

```
(setf *default-tally-goals* '("NIL" "Cturn" "helix" "beta"))
```

```

(defun tally-regions-and-types (&key (seq *default-seq*) (spat *default-spat*)
                               (goal-list *default-tally-goals*))
  "adds tally lists to both region goal and type lists"
  (let ((report (report-all-regions :seq seq :spat spat))
        (goal-result ()) (type-result ()))
    (mit-loop for i from 0 to (- (length goal-list) 1)
              do
                (setf goal-result
                      (cons (count (nth i goal-list) (third report)
                                   :test #'equal) goal-result))
              finally
                (setf goal-result
                      (cons (cadr report) (reverse goal-result)))
                (setf *alpps-tallys-list* (cons (cons (car report) goal-result)
                                                *alpps-tallys-list*)))
    (mit-loop for j from 0 to (- (length *region-color-list*) 1)
              do
                (setf type-result
                      (cons (count (car (nth j *region-color-list*))
                                   (fourth report)
                                   :test #'equal) type-result))
              finally
                (setf type-result
                      (cons (cadr report) (reverse type-result)))
                (setf *alpps-region-type-tallys-list*
                      (cons (cons (car report) type-result)
                            *alpps-region-type-tallys-list*))))))

(defun tally-regions (&key (seq *default-seq*) (spat *default-spat*)
                     (goal-list *default-tally-goals*))
  "given a goal-list (e.g. ((helix reg-name) (Cturn reg)))
  the function produces
  a count list: (3fxn 1 2 3)"
  (let ((report (report-all-regions :seq seq :spat spat))
        (result ()))
    (mit-loop for i from 0 to (- (length goal-list) 1)
              do
                (setf result
                      (cons (count (nth i goal-list) (third report)
                                   :test #'equal) result))
              finally
                (setf result
                      (cons (cadr report) (reverse result)))
                (return (cons (car report) result))))))

```

```

(defun tally-region-types (&key (seq *default-seq*) (spat *default-spat*)
                          goal-list)
  "returns a list of region-type and counts"
  (let (;; (report (report-all-regions :seq seq :spat spat))
        (report (test-report-all-regions))
        (result ()))
    (mit-loop for i from 0 to (- (length *region-color-list*) 1)
              do
                (setf result
                      (cons (count (car (nth i *region-color-list*)))
                            (fourth report)
                            :test #'equal) result))
              finally
                (setf result
                      (cons (cadr report) (reverse result)))
                (return (cons (car report) result))))))

(defun test-report-all-regions ()
  "builds a list which includes the seq-name and the list of region
  targets. A "NIL" is given if there is no region in a given block."
  ; e.g. ("2kai-b" "Cturn" "beta" "Cturn" "beta" "Cturn" "beta" "Cturn" "beta")
  (let ((sb *default-sb*)
        (reg (goals ()) (names ())) (result ()))
    (setf result (cons (format nil "~a" (seq-blocks-name sb)) result))
    (setf result (cons (format nil "~a"
                              (seq-blocks-alpps-pattern sb)) result))

    (mit-loop
     for block in (seq-blocks-block-list sb)
     do
       (setf reg (block-region block))
       (if (null reg)
           (progn
            (setf goals (cons
                      (format nil "~a" "NIL") goals)))
           (progn
            (setf goals (cons
                      (format nil "~a" (region-target reg)) goals))
            (setf names (cons
                      (format nil "~a" (region-name reg)) names))))))

    finally
      (setf result (cons goals result))
      (return (reverse (cons names result))))))

```

```

.....*****
; ; ; ;
; ; ; ; * Match-Set System
; ; ; ; * file: register.lisp
; ; ; ; *
; ; ; ; *
.....*****
; ; ; ;

```

```

(in-package :match-set)
(use-package '(lisp user loop plans))

```

```

(defun determine-next-standard (a p)
  "determine whether a or p is next c terminus standard"
  (let ((at-a nil))
    (if (null a)
        (setf at-a nil)
        (if (null p)
            (setf at-a t)
            (if (< (cadr a) (cadr p))
                (setf at-a t)
                (setf at-a nil))))))
  at-a))

```



```

(defun raw-registration-scoring (registration)
  "given a registration e.g. '((3 6) (3 7)) ((12 16) (9 15)) ((20 25) ( ))'
  function will return a list of (length-a len-p n-overlap c-over)"
  (mit-loop for i in registration
    with n-over = nil and c-over = nil and result = nil
    and len-a = nil and len-p = nil
    do
      (if (car i)
        (setf len-a (- (cadr (car i)) (car (car i))))
        (setf len-a nil))
      (if (cadr i)
        (setf len-p (- (cadr (cadr i)) (car (cadr i))))
        (setf len-p nil))
      (if (and len-a len-p)
        (progn
          (setf n-over (- (car (car i)) (car (cadr i))))
          (setf c-over (- (cadr (cadr i)) (cadr (car i))))
          (setf result
            (cons (list len-a len-p n-over c-over) result)))
        (if len-a
          (setf result (cons (list len-a nil nil nil) result))
          (setf result (cons (list nil len-p nil nil) result))))
      finally
      (return (reverse result))))

```

```

(defun mid-point-compares (registration)
  "given a registration e.g. '(((3 6) (3 7)) ((12 16) (9 15)) ((20 25) ()))
  function will return a list of midpoint differences '(-1 2 nil)"
  (mit-loop for i in registration
    with mid-a = nil and mid-p = nil and result = nil
    and len-a = nil and len-p = nil
    do
      (if (car i)
          (progn
            (setf len-a (- (cadr (car i)) (car (car i))))
            (setf mid-a
                  (truncate (/ (+ (car (car i)) (cadr (car i))) 2))))
          (setf mid-a nil))
        (if (cadr i)
            (progn
              (setf len-p (- (cadr (cadr i)) (car (cadr i))))
              (setf mid-p
                    (truncate (/ (+ (car (cadr i)) (cadr (cadr i))) 2))))
            (setf mid-p nil))
          (if (and mid-a mid-p)
              (setf result
                    (cons (list len-a len-p (- mid-a mid-p)) result))
              (setf result (cons nil result)))
            finally
              (return (reverse result))))))

```

```

(defun register-prediction (prediction assignment)
  "return a pairing of blocks"
  (let ((a-left (cdr assignment)) (p-left (cdr prediction))
        (a (car assignment)) (p (car prediction))
        at-a other-used
        (result nil))
    (mit-loop while (or a p)
      do
        (setf at-a (determine-next-standard a p))
        (if at-a
          (progn
            (setf other-used nil)
            (if (and p (>= (cadr a) (car p)))
              (progn
                (setf other-used t)
                (setf result (cons (list a p) result)))
              (setf result (cons (list a ()) result)))
            (setf a (car a-left))
            (setf a-left (cdr a-left))
            (if (and other-used
                    (and (or (null a)
                          (and (car p-left)
                               (> (car a) (cadr p))))))
              (progn
                (setf p (car p-left))
                (setf p-left (cdr p-left))))
            (progn
              (setf other-used nil)
              (if (and a (>= (cadr p) (car a)))
                (progn
                  (setf other-used t)
                  (setf result (cons (list a p) result)))
                  (setf result (cons (list () p) result)))
                (setf p (car p-left))
                (setf p-left (cdr p-left))
                (if (and other-used
                        (or (null p)
                            (and (car a-left) (> (car p) (cadr a))))
                  (progn
                    (setf a (car a-left))
                    (setf a-left (cdr a-left))))))
              finally
                (return (reverse result))))))

```

```
(defun pull-nil-pairs (list)
  "remove NIL pairs from list eg (1 NIL)"
  (let ((result nil))
    (mit-loop for i in (reverse list)
              do
                (if (or (null (car i)) (null (cadr i)))
                    ()
                    (setf result (cons i result)))
              finally
                (return result))))
```

```

.....*****
;;;
;;;* Match-Set System
.....*
;;;* file: utils.lisp
.....*****
;;;

```

```

(in-package :user)
(use-package '(loop))

```

```

(export '(string-char-replace list-add make-match-string interval
        prune-left prune-right))

```

```

;;; make-match-string
;;;      given the length of the protein sequence and a list of
;;;      the hit points, the function returns a string of hits.
;;;
;;;
;;; (make-match-string 5 '(1 3 4))
;;; " * * * "
(defun make-match-string (length hits &key (show-misses nil) (misses nil)
                        (tols nil) (hit-char #\*) (miss-char #\-)
                        (tol-char #\#))
  "build a string for display on m-p window"
  (let ((result (make-string length :initial-element #\space)))
    (progn
      (mit-loop for hit in hits
                do
                  (setf (elt result (- hit 1)) hit-char))
      (if show-misses
          (progn
            (mit-loop for miss in misses
                      do
                        (setf (elt result (- miss 1)) miss-char))
            (mit-loop for tol in tols
                      do
                        (setf (elt result (- tol 1)) tol-char))))
          result)))

```

```

(defun interval-old (start stop list)
  " make a list of all elts of list between start and stop"
  (let ((picks ()))
    (mit-loop for i from start to stop do
      (setf picks (cons i picks)))
    (intersection picks list)))

(defun prune-left (list cutoff)
  "cuts sorted list from the left based on >= cutoff"
  (member cutoff list :test #'<=))

(defun prune-right (list cutoff)
  "cuts sorted list from the right based on >= cutoff"
  (reverse (member cutoff (reverse list) :test #'>=)))

(defun interval (start stop list)
  " make a list of all elts of list between start and stop"
  (let ((left (prune-right list stop))
        (right (prune-left list start)))
    (sort (intersection right left) #'<)))

(defun list-add (a b)
  "returns list which is sum of two lists"
  (let (result)
    (mit-loop for i in a and j in b
      do
        (setf result (cons (+ i j) result)))
    (reverse result)))

(defun string-char-replace (&key (string "") (old #\space) (new #\space))
  "take string and change from old char to new char whenever old is found"
  (let ((result string) pos)
    (progn
      (if (member old (coerce string 'list)) ; check for existence of old
        (mit-loop while (setf pos (position old (coerce result 'list)))
          do
            (setf (elt result pos) new)))
      result)))

```

```
(defun reduce-string (string &key (show nil) (blank-char #\space))
  "take string and only show chars in show. replace others with blank-char."
  (let ((result (make-string (length string) :initial-element blank-char)))
    (progn
      (mit-loop for i from 0 to (- (length string) 1)
        do
          (if (member (elt string i) show)
              (setf (elt result i) (elt string i))))
      result)))
```

```
(defun break-point-string (string &key (size 50) (break-size 3)
  (break-char #\space))
  "take a string and divide it into sections of length size. sections
  are marked with break-size break-chars."
  (let ((result nil) (sects (truncate (/ (length string) size)))
        (break (make-string break-size :initial-element break-char)))
    (progn
      (mit-loop for i from 0 to (- sects 1)
        do
          (if (null result)
              (setf result
                    (format nil "~a"
                          (subseq string
                                (* i size) (* (+ i 1) size))))
              (setf result
                    (format nil "~a~a~a" result break
                          (subseq string (* i size) (* (+ i 1) size))))))
          finally
            (if (null result)
                (setf result (format nil "~a" string))
                (if (not (equal (subseq string (* i size) ""))
                    (setf result (format nil "~a~a~a" result break
                                      (subseq string (* i size) ""))))))
            (return result))))))
```

Appendix B

Common Lisp Source Code for *Trimmed Scoring*

```
.....*****
****
**** * score package
**** * file: res-scoring.lisp
**** *
**** *
(in-package :score)
(use-package '(lisp user loop))
(setf *do-cuts-with-pairing* t)

;;; tolerated residue scoring 9/19/90
;;; cut out residues around helix caps and then score
;;; pred-ss, assgn are strings of a,t
(defun res-score-seq-tol-pattern (&key (goal #\a)
                                (tol 1) (assgn-ends t)
                                (do-pairing *do-cuts-with-pairing*)
                                (assgn-cuts ())
                                (assgn-ss "")
                                (pred-cuts ())
                                (pred-ss "")
                                (pred-ends nil))
  "returns a list (TP TN FP FN) for the default seq pattern combo
  accounting for the tolerance"
  (let (cut-assgn-ss cut-pred-ss (helix-cuts ()))
    (if assgn-ends
        (setf helix-cuts (cons assgn-cuts helix-cuts)))
    (if pred-ends
        (setf helix-cuts (cons pred-cuts helix-cuts))))

    (if do-pairing
        ; only paired helices are cut
        (setf helix-cuts
              (sort (remove-duplicates
                    (remove-paren (pull-nil-pairs
                                   (register-prediction
                                    assgn-cuts pred-cuts))))
                    #'<)))

    (setf cut-pred-ss (make-tol-cuts :cuts helix-cuts
                                    :tol tol :seq pred-ss))
    (setf cut-assgn-ss (make-tol-cuts :cuts helix-cuts
                                     :tol tol :seq assgn-ss))
    (res-score-seq-pattern :goal goal
                          :assgn-ss cut-assgn-ss
                          :pred-ss cut-pred-ss)))
```



```

(defun make-tol-cuts (&key (seq "") (cuts ()) (tol 1))
  "takes a sequence and makes cuts based on a tolerance around points"
  (let (result (len (length seq)) (off-char #\0))
    (setf result (format nil seq))
    (mit-loop for i in (remove-paren cuts)
      do
        (if (eql tol 0)
          (setf (elt result (- i 1)) off-char)
          (mit-loop for j from (- i tol) to (+ i tol)
            do
              (if (and (> j 0) (< j len))
                (setf (elt result (- j 1)) off-char))))
          finally
            (return (remove off-char result))))))

(defun remove-paren (L)
  "depth first remove paren"
  (cond ((null L) nil)
        ((atom L) (list L))
        (T (append (remove-paren (car L)) (remove-paren (cdr L))))))

(defun compute-big-Q (&key p-a p-b p-t N)
  "returns Q for the default seq pattern combo"
  (/ (+ p-a p-b p-t) N))

(defun compute-big-C (results-list)
  "returns correlation coefficient given results list (TP TN FP FN)"
  (let* ((TP (car results-list))
         (TN (cadr results-list))
         (FP (caddr results-list))
         (FN (caddr results-list)))
    (denom
     (sqrt (* (+ TN FN) (+ TN FP) (+ TP FN) (+ TP FP))))))
  (if (equalp denom 0)
    0
    (/ (- (* TP TN) (* FN FP)) denom))))

```

```

.....*****
;;;
;;;* score package
;;;* file: register.lisp
.....*
;;;
.....*****
;;;

```

```

(in-package :score)
(use-package '(lisp user loop))

```

```

(defun determine-next-standard (a p)
  "determine whether a or p is next c terminus standard"
  (let ((at-a nil))
    (if (null a)
        (setf at-a nil)
        (if (null p)
            (setf at-a t)
            (if (< (cadr a) (cadr p))
                (setf at-a t)
                (setf at-a nil))))))
  at-a))

```

```

(defun raw-registration-scoring (registration)
  "given a registration e.g. '(((3 6) (3 7)) ((12 16) (9 15)) ((20 25) ()))
function will return a list of (length-a len-p n-overlap c-over)"
  (mit-loop for i in registration
    with n-over = nil and c-over = nil and result = nil
    and len-a = nil and len-p = nil
    do
      (if (car i)
        (setf len-a (- (cadr (car i)) (car (car i))))
        (setf len-a nil))
      (if (cadr i)
        (setf len-p (- (cadr (cadr i)) (car (cadr i))))
        (setf len-p nil))
      (if (and len-a len-p)
        (progn
          (setf n-over (- (car (car i)) (car (cadr i))))
          (setf c-over (- (cadr (cadr i)) (cadr (car i))))
          (setf result
            (cons (list len-a len-p n-over c-over) result)))
        (if len-a
          (setf result (cons (list len-a nil nil nil) result))
          (setf result (cons (list nil len-p nil nil) result))))
      finally
      (return (reverse result))))

```

```

(defun mid-point-compares (registration)
  "given a registration e.g. '((3 6) (3 7)) ((12 16) (9 15)) ((20 25) ( ))
  function will return a list of midpoint differences '(-1 2 nil)"
  (mit-loop for i in registration
    with mid-a = nil and mid-p = nil and result = nil
    and len-a = nil and len-p = nil
    do
      (if (car i)
        (progn
          (setf len-a (- (cadr (car i)) (car (car i))))
          (setf mid-a
            (truncate (/ (+ (car (car i)) (cadr (car i))) 2))))
        (setf mid-a nil))
      (if (cadr i)
        (progn
          (setf len-p (- (cadr (cadr i)) (car (cadr i))))
          (setf mid-p
            (truncate (/ (+ (car (cadr i)) (cadr (cadr i))) 2))))
        (setf mid-p nil))
      (if (and mid-a mid-p)
        (setf result
          (cons (list len-a len-p (- mid-a mid-p)) result))
        (setf result (cons nil result)))
      finally
        (return (reverse result))))

```

```

(defun register-prediction (prediction assignment)
  "return a pairing of blocks"
  (let ((a-left (cdr assignment)) (p-left (cdr prediction))
        (a (car assignment)) (p (car prediction))
        at-a other-used
        (result nil))
    (mit-loop while (or a p)
      do
      (setf at-a (determine-next-standard a p))
      (if at-a
        (progn
          (setf other-used nil)
          (if (and p (>= (cadr a) (car p)))
            (progn
              (setf other-used t)
              (setf result (cons (list a p) result)))
            (setf result (cons (list a ()) result)))
          (setf a (car a-left))
          (setf a-left (cdr a-left))
          (if (and other-used
                  (and (or (null a)
                          (and (car p-left)
                               (> (car a) (cadr p))))))
            (progn
              (setf p (car p-left))
              (setf p-left (cdr p-left))))))
        (progn
          (setf other-used nil)
          (if (and a (>= (cadr p) (car a)))
            (progn
              (setf other-used t)
              (setf result (cons (list a p) result)))
            (setf result (cons (list () p) result)))
          (setf p (car p-left))
          (setf p-left (cdr p-left))
          (if (and other-used
                  (or (null p)
                      (and (car a-left) (> (car p) (cadr a))))))
            (progn
              (setf a (car a-left))
              (setf a-left (cdr a-left))))))
        finally
        (return (reverse result))))))

```



```

....*****
;;;
;;;* socre package
;;;* file: utils.lisp
....*
;;;
....*****

```

```

(in-package :user)
(use-package '(loop))

```

```

(export '(string-char-replace list-add make-match-string interval
        prune-left prune-right make-length-ss-seq make-ss-seq ))

```

```

;;; make-match-string
;;;      given the length of the protein sequence and a list of
;;;      the hit points, the function returns a string of hits.
;;;
;;;
;;; (make-match-string 5 '(1 3 4))
...  " * * * "
;;;
(defun make-match-string (length hits &key (show-misses nil) (misses nil)
                        (tols nil) (hit-char #\*) (miss-char #\.)
                        (tol-char #\#))
  "build a string for display on m-p window"
  (let ((result (make-string length :initial-element #\space)))
    (progn
      (mit-loop for hit in hits
                do
                  (setf (elt result (- hit 1)) hit-char))
      (if show-misses
          (progn
            (mit-loop for miss in misses
                      do
                        (setf (elt result (- miss 1)) miss-char))
            (mit-loop for tol in tols
                      do
                        (setf (elt result (- tol 1)) tol-char))))
          result)))

```

```

(defun prune-left (list cutoff)
  "cuts sorted list from the left based on >= cutoff"
  (member cutoff list :test #'<=))

(defun prune-right (list cutoff)
  "cuts sorted list from the right based on >= cutoff"
  (reverse (member cutoff (reverse list) :test #'>=)))

(defun interval (start stop list)
  " make a list of all elts of list between start and stop"
  (let ((left (prune-right list stop))
        (right (prune-left list start)))
    (sort (intersection right left) #'<)))

(defun list-add (a b)
  "returns list which is sum of two lists"
  (let (result)
    (mit-loop for i in a and j in b
              do
                (setf result (cons (+ i j) result)))
    (reverse result)))

(defun string-char-replace (&key (string "") (old #\space) (new #\space))
  "take string and change from old char to new char whenever old is found"
  (let ((result string) pos)
    (progn
      (if (member old (coerce string 'list)) ; check for existence of old
          (mit-loop while (setf pos (position old (coerce result 'list)))
                    do
                      (setf (elt result pos) new)))
      result)))

```



```
(defun make-length-ss-seq (&key (length 0) (hits nil) (symbol #\a))
  "builds a string the length of the seq which represents the predicted
  secondary structure based on regions and their targets. The default assignment
  is t."
```

```
(let ((result (make-string length :initial-element #\t)))
  (mit-loop for block in hits
    do
      (mit-loop for i from (- (car block) 1) to (- (cadr block) 1)
        do
          (setf (elt result i) symbol))
      finally
        (return result))))
```

```
(defun make-ss-seq (&key (seq "") (hits nil) (symbol #\a))
  "builds a string the length of the seq which represents the predicted
  secondary structure based on regions and their targets. The default assignment
  is t."
```

```
(let ((result (make-string (length seq) :initial-element #\t)))
  (mit-loop for block in hits
    do
      (mit-loop for i from (- (car block) 1) to (- (cadr block) 1)
        do
          (setf (elt result i) symbol))
      finally
        (return result))))
```

```
.....
```

```
.....; 3 examples of "same function"
```

```
.....; The mit-loop is very expensive in time. Version 2 checks for
.....; existence of old before entering the loop. Version 3 lacks
.....; this feature. Version one does check, but goes thru the loop
.....; many more times than is necessary. There was a version 0,
.....; version 1 without the member check, which was very slow.
.....; 2/2/90 bic
.....;
```

```
.....
```

```
(defun string-char-replace-3 (&key (string "") (old #\space) (new #\space))
  "take string and change from old char to new char whenever old is found"
```

```
(let ((result string))
  (progn
    (mit-loop while (member old (coerce string 'list))
      do
        (setf (elt result (position old (coerce result 'list)))
              new))
    result)))
```

```
(defun string-char-replace-2 (&key (string "") (old #\space) (new #\space))
  "take string and change from old char to new char whenever old is found"
  (let ((result string) pos)
    (progn
      (if (member old (coerce string 'list)) ; check for existence of old
          (mit-loop while (setf pos (position old (coerce result 'list)))
                    do
                      (setf (elt result pos) new)))
          result)))
```

```
(defun string-char-replace-1 (&key (string "") (old #\space) (new #\space))
  "take string and change from old char to new char whenever old is found"
  (let ((result string))
    (progn
      (if (member old (coerce string 'list)) ; check for existence of old
          (mit-loop for i from 0 to (- (length string) 1)
                    do
                      (if (equal (elt string i) old)
                          (setf (elt result i) new))))
          result)))
```

```
(defun reduce-string (string &key (show nil) (blank-char #\space))
  "take string and only show chars in show. replace others with blank-char."
  (let ((result (make-string (length string) :initial-element blank-char)))
    (progn
      (mit-loop for i from 0 to (- (length string) 1)
                do
                  (if (member (elt string i) show)
                      (setf (elt result i) (elt string i))))
      result)))
```

```

(defun break-point-string (string &key (size 50) (break-size 3)
                          (break-char #\space))
  "take a string and divide it into sections of length size. sections
are marked with break-size break-chars."
  (let ((result nil) (sects (truncate (/ (length string) size)))
        (break (make-string break-size :initial-element break-char)))
    (progn
      (mit-loop for i from 0 to (- sects 1)
                do
                  (if (null result)
                      (setf result
                            (format nil "~a"
                                    (subseq string
                                             (* i size) (* (+ i 1) size))))
                      (setf result
                            (format nil "~a~a~a" result break
                                    (subseq string (* i size) (* (+ i 1) size))))))
                finally
                  (if (null result)
                      (setf result (format nil "~a" string))
                      (if (not (equal (subseq string (* i size)) ""))
                          (setf result (format nil "~a~a~a" result break
                                                  (subseq string (* i size))))))
                    (return result))))))

```

Appendix C

ALPPS and PLANS Patterns for α/α Proteins

ALPPS Patterns

;;;

;;; N.B. Comments come *after* the line or statement.

;;;

(in-package :alpps)

(use-package '(lisp user loop plans))

(defvar *default-min-region* 4)

;;; Minimum helix length

(def-alpps all-alpha-min-1 (:pat "TU" :tol 1) ; :no-limit t

;;; Split the sequence based on the TU turn pattern.

(split-long-blocks :max-length 40)

;;; Split any resulting blocks such that no block is longer than 40 residues

(hide-blocks :pat "anything")

(expose-blocks :pat "HA" :no-limit t :spat-min-count 2)

(make-regions :start-pat "Nt" :end-pat "Ct"

 :target "helix"

 :symbol "b"

 :color "Blue"

 :min-size *default-min-region*

 :no-limit t

 :name "both-ends")

;;; Nt-HA-Ct helix.

(make-regions :start-pat "Nt" :end-pat "HA"

 :target "helix"

 :symbol "n"

 :color "Green"

 :min-size *default-min-region*

 :no-limit t

 :name "no-ct")

;;; Nt-HA helix.

(make-regions :start-pat "HA" :end-pat "Ct"

 :target "helix"

 :color "Red"

 :symbol "c"

 :min-size *default-min-region*

 :no-limit t

 :name "no-nt")

```

;;; HA-Ct helix.
(make-regions :start-pat "HA" :end-pat "HA"
  :target "helix"
  :symbol "e"
  :color "Yellow"
  :min-size *default-min-region*
  :no-limit t
  :name "no-ends")
;;; Parse again for the existence of only one HA.
(expose-blocks :pat "HA" :no-limit t :spat-min-count 1)
(make-regions :start-pat "Nt" :end-pat "Ct"
  :target "helix"
  :symbol "b"
  :color "Blue"
  :min-size *default-min-region*
  :no-limit t
  :name "repechage")
;;; Nt-HA-Ct helix.
)

```

PLANS Patterns

;;;

;;; N.B. Comments about the patterns come *after* the pattern itself.

;;; Syntax: <name> : <pattern>

;;; == HELIX CORE PATTERNS ==

chpr1: "[DE]...[HKR]"{2,2}

;;; charged pair neg to pos N to C

chpr2: "[HKR]...[DE]"{2,2}

;;; charged pair pos to neg N to C

chpr1a: "[DE].[AVLIMFWY].[HKR]"{2,2}

;;; charged pair neg to pos with phobic interior face

chpr2a: "[HKR].[AVLIMFWY].[DE]"{2,2}

;;; charged pair pos to neg with phobic interior

chpr1c1: "[^PGN][DE][^PGN][AVILMFYW][^PGN][HKR][^PGN]"{3,3}

;;; Neg to Pos charge pair with hydrophobic backside NO PDG or N

chpr1c2: "[^PGN][HKR][^PGN][AVILMFYW][^PGN][DE][^PGN]"{3,3}

;;; Pos to Neg charge pair with hydrophobic backside NO PDG or N

chpr1c3: "[^PGN][DE][^PGN][^PGN][HKR][^PGN]"{2,2}

;;; Neg to Pos charge pair with hydrophobic backside NO PDG or N

chpr1s1: ("[DE].[AVILMFYW].[HKR]"{2,2}) and

(not (density(>=,5,7,"[DEGHKNPQRSTY]"{3,3})))

;;; helix charge pair pattern - -> +

chpr1s2: ("[HKR].[AVILMFYW].[DE]"{2,2}) and

(not(density(>=,5,7,"[DEGHKNPQRSTY]"{3,3})))

;;; helix charge pair + -> -

chpr1s3: ("[DE]..[HKR]"{1,1}) and

(not (density(>=,4,6,"[DEGHKNPQRSTY]"{2,2})))

;;; helix charge pair pattern - -> +

CP: (chpr1c1 or chpr1c3 or chpr1s1 or chpr1s3)

;;; Current working set of charge pairs. Only those aligned with the helix

;;; dipole moment are used.

h1: "[AVILMCKFWY]..[AVILMCKFWY][AVILMCKFWY]..[AVILMCKFWY]"
 h2: "[AVILMCKFWY]...[AVILMCKFWY][AVILMCKFWY]..[AVILMCKFWY]"
 h3: "[AVILMCKFWY][AVILMCKFWY]...[AVILMCKFWY]..[AVILMCKFWY]"
 h4: "[AVILMCKFWY][AVILMCKFWY]..[AVILMCKFWY]...[AVILMCKFWY]"
 h5: "[AVILMCKFWY][AVILMCKFWY]..[AVILMCKFWY][AVILMCKFWY]"
 h6: "[AVILMCKFWY]..[AVILMCKFWY][AVILMCKFWY]...[AVILMCKFWY]"
 h7: "[AVILMCKFWY]..[AVILMCKFWY]...[AVILMCKFWY][AVILMCKFWY]"
 h8: "[AVILMCKFWY]...[AVILMCKFWY]..[AVILMCKFWY][AVILMCKFWY]"
 ::: Patterns of 4 phobics encompassing 1-2 turns of the helix.

ha: (h1{-3,-3} and h1){0,3}
 hb: (h1{-4,-4} and h1){0,4}
 hc: (h2{-4,-4} and h2){0,4}
 hd: (h2{-5,-5} and h2){0,5}
 he: (h3{-1,-1} and h3){0,1}
 hf: (h3{-4,-4} and h3){0,4}
 hg: (h4{-1,-1} and h4){0,1}
 hh: (h4{-4,-4} and h4){0,4}
 hi: (h5{-1,-1} and h5){0,1}
 hj: (h5{-4,-4} and h5){0,4}
 hk: (h5{-5,-5} and h5){0,5}
 ::: Longer groups of phobic diamond patterns.

helix: (ha or hb or hc or hd or he or hf or hg or hh or hi or hj or hk)
 ::: Bring it all together.

p1: "[PGQNDERKSTH]...[PGQNDERKSTH]...[PGQNDERKSTH]"
 p2: "[PGQNDERKSTH]..[PGQNDERKSTH]...[PGQNDERKSTH]"
 p3: "[PGQNDERKSTH]...[PGQNDERKSTH]..[PGQNDERKSTH]"
 pstripe: (p1 or p2 or p3)
 ::: Philic area's the absence of which seems to be predictive of helix.
 ::: originally this stemmed from the idea that a philic stripe was
 ::: predictive of helix.

H1: (helix)
 H2: (aend{2,2} and abegin{2,4} and (not a2{-4,4})) ; old a6
 H3: (CP)
 H4: (gs)
 H5: (density(=,0,8, pstripe)){6,6}
 H6: (h1{0,6} or h2{0,6} or h6{0,6})
 ::: The high level patterns that are predictive of helix.

HA: (H1 or H2 or H3 or H4 or H5 or H6)
 ::: Core helix structure: Final Pattern.

;;; == C-CAP PATTERNS ==

c1: "[ACFIKLMVWY]..[ACFIKLMVWY][ACFIKLMVWY]..[GKNH]" {7,7}
 c2: "[ACFIKLMVWY]..[ACFIKLMVWY][ACFIKLMVWY]...[GKNH]" {8,8}
 c3: "[ACFIKLMVWY]..[ACFIKLMVWY]...[ACFIKLMVWY][GKNH]" {8,8}
 c4: "[ACFIKLMVWY]...[ACFIKLMVWY]..[ACFIKLMVWY][GKNH]" {8,8}
 c5: "[ACFIKLMVWY][ACFIKLMVWY]..[ACFIKLMVWY][GKNH]" {5,5}
 c6: "[ACFIKLMVWY]..[ACFIKLMVWY]...[ACFIKLMVWY][GK]" {8,8}

;;; Hydrophobic phasing of the C-cap

Cz: "[ACFIKLMVWY]..[ACFIKLMVWY][ACFIKLMVWY]..[DEGKPNQRS]
 [GKHN]" {8,8}
 Cy: "[ACFIKLMVWY]...[ACFIKLMVWY][ACFIKLMVWY]..[DEGKPNQRS]
 [GKHN]" {9,9}
 Cv: "[ACFIKLMVWY]..[ACFIKLMVWY][ACFIKLMVWY].[DEGKPNQRS].P" {8,8}
 Cw: "[ACFIKLMVWY]..[ACFIKLMVWY][ACFIKLMVWY]..
 [ACFIKLMVWY]..[DEGKPNQRS][GKHN]" {11,11}
 Cx: "[ACFIKLMVWY]..[ACFIKLMVWY][ACFIKLMVWY]..
 [ACFIKLMVWY]...[DEGKPNQRS][GKHN]" {12,12}

;;; Hydrophobic patch + terminating hydrophilic residue followed by a C-cap

;;; Not currently used.

Cu: ("[GKHN][L].[CFILMVWY]..[CFILMVWY]" or
 "[GKHN][L]..[CFILMVWY].[CFILMVWY]")

HK: (density(>=,2,3,"[HK]") and (not density(>=,2,3,"[K]")))

;;; High density of basic residues R is not used here.

CA: (("GK" {0,0} or "P" {-2,-1}) and
 ("K" {1,3} or "R" {2,2} or "[QH]" {1,1} or "H" {5,5}))
 CB: (("G" {0,0} or "P" {-2,-1}) and ("FMW" {4,4} or "L" {3,3}))
 CC: (("GK" {0,0} and ("P" {-2,-1} or "[TW]" {-2,-2}))
 CD: (("K" {1,3} or "R" {2,2} or "[QH]" {1,1} or "H" {5,5}) and "[TW]" {-2,-2})
 CE: (("K" {1,3} or "R" {2,2} or "[QH]" {1,1} or "H" {5,5}) and
 ("[FMW]" {4,4} or "L" {3,3}))
 CF: (("FMW" {4,4} or "L" {3,3}) and "[TW]" {-2,-2})
 CG: ("G" {0,0} and "P" {-2,-1})

CH: (("K" {1,3} or "R" {2,2} or "[QH]" {1,1} or "H" {5,5}) and
 ("[FMW]" {4,4} or "L" {3,3}) and "[TW]" {-2,-2})

CS: (("GK" {0,0} or "P" {-1,-1}) and
 ("K" {1,3} or "R" {2,2} or "[QH]" {1,1} or "H" {5,5}) and
 ("[FMW]" {4,4} or "L" {3,3}))

;;; Combinations of residue patterns that mimic the statistical information
 ;;; in the Richardson and Richardson Science paper 1988.

Ca: (CG)

Cb: (CH)

Cc: (CS and "[^ACFILMV]" {-2,-2})

Cd: (c6)

Ce: (("[GK]" {0,0} and "W" {-2,-2}) or ("P" {-1,-1} and "[WT]" {-2,-2})
 or ("N.P" or "NP"))

Cf: (HK and (not Cc{-10,0}) and (not Cb{-10,0}) and (not Ca{-10,0}) and
 HA{0,13} and (not density(>,2,5,HA){5,5}))

;;; The high level patterns that are predictive of a C-cap.

Ct: (Ca or Cb or Cc or Cd or Ce)

;;; C-cap: Final Pattern

;;; == N-CAP PATTERNS ==

n1: "[NSTDE]..[ACFIKLMVWY][ACFIKLMVWY]..[ACFIKLMVWY]"
 n2: "[NSTDE]...[ACFIKLMVWY][ACFIKLMVWY]..[ACFIKLMVWY]"
 n3: "[NSTDE][ACFIKLMVWY]...[ACFIKLMVWY]..[ACFIKLMVWY]"
 n4: "[NSTDE][ACFIKLMVWY]..[ACFIKLMVWY]...[ACFIKLMVWY]"
 n5: "[NSTDE][ACFIKLMVWY]..[ACFIKLMVWY][ACFIKLMVWY]"

;;; Hydrophobic phasing of the N-cap

Nz: "[GDNST][DEKPNQRS]..[ACFIKLMVWY][ACFIKLMVWY]..[ACFIKLMVWY]"
 Ny: "[GDNST][DEKPNQRS]..[ACFIKLMVWY][ACFIKLMVWY]...[ACFIKLMVWY]"
 Nw: "[GDNST][ADEKPNQRS]..[ACFIKLMVWY]..[ACFIKLMVWY]
 [ACFIKLMVWY]..[ACFIKLMVWY]"
 Nx: "[DNST]..[CFIKLMVWY]...[ACFIKLMVWY]..[ACFIKLMVWY]
 [ACFIKLMVWY]..[ACFIKLMVWY]"

;;; Hydrophobic patch + terminating hydrophilic residue followed by a N-cap

;;; Currently, these are more predictive than without the hydrophilic residue.

DE: (density(>=,2,3,"[DE]"))

NA: ("[NS]"{0,0} and ("D"{-3,-2} or "E"{-3,-1}))
 NB: ("[NS]"{0,0} and ("LFW"{-4,-4} or "M"{-5,-4}))
 NC: ("[ND]"{0,0} and "P"{-2,-2})
 ND: (("D"{-3,-2} or "E"{-3,-1}) and "P"{-1,-1})
 NE: (("LFW"{-4,-4} or "M"{-5,-4}) and ("D"{-3,-2} or "E"{-3,-1}))
 NF: (("LFW"{-4,-4} or "M"{-5,-4}) and "P"{-1,-1})
 NG: ("[GNSDT]"{0,0} and "P"{-1,-1})

NH: (("D"{-3,-2} or "E"{-3,-1}) and ("LFW"{-4,-4} or "M"{-5,-4})
 and "P"{-1,-1})

NS: ("[DNS]"{0,0} and ("D"{-3,-2} or "E"{-3,-1})
 and ("LFW"{-4,-4} or "M"{-5,-4}))

;;; Combinations of residue patterns that mimic the statistical information

;;; in the Richardson and Richardson Science paper 1988.

Na: (NG)

Nb: (NS)

Nc: (NA and (Nw or Ny or Nz) and (not Na{0,10}) and (not Nb{0,10}))

Nd: (NE and (Nw or Ny or Nz) and (not Na{0,10}) and (not Nb{0,10}))

Ne: (NB and (n1 or n4 or n5) and (not Na{0,10}) and (not Nb{0,10}))

**Nf: (DE and (not Nc{0,10}) and (not Nb{0,10}) and (not Na{0,10}) and
HA{-13,0} and (not density(>,2,5,HA){5,5}))**

Ng: (group(2,NC) and (not NG{-1,-1}))

::: The high level patterns that are predictive of a N-cap.

Nt: (Na or Nb or Nc or Nd or Ne or Ng or group(5,Nf))

::: N-cap: Final Pattern

;;; == TURN PATTERNS: refer to the 1986 Biochemistry paper ==

a1: (gs or CP)

;;; Primary helical mask for turns.

a2: (density(=,0,5,alpha_strong_phobic)
and (not "[DE]...[KHR]"{-2,4})
and (not "[DE]..[KHR]"{-2,4})
and (not gs{-2,2})){2,2}

;;; AA_Turn1_no-phobics

a3: (density(=,4,4,alpha_philic)){1,1}

;;; Strong Turn of four Hydrophilic residues in sequence.

a4: (a3)

;;; AA_Turn2_4-philics

HP: ("[VLIAWYKFCT][VLAIWKYFCT]P[VLAIWYFCT]"{2,2} and
(not density(>=,1,5, "[NQRS]")){1,1})

;;; Potential situation of a proline in helical region

AP: ("P"{-1,-1} and (density(>=,1,5, "[DEKNQRST]")){1,1})

a5: ("P"{-1,-1}
and (not a2{-11,0})
and (not a4{-11,0})
and (not HP{-1,-1})))

;;; AA_Turn3_proline + Must be a philic residue in the surrounding area.

a6: (aend and abegin{-2,0}
and (not (a2{-11,11} or a4{-11,11} or a5))))

;;; AA_Turn4_helix-ends

a7: ((not ap{-1,1}) and (not a1{-2,2}) and (not a9{-11,11}))

;;; AA_Turn5_weak

a8: (group(7,a7))

;;; AA_turn5_group

a9: (group(7,(a2 or a4 or a5 or a6)))

;;; AA_T_possible

tu: (a8 or a9)

TU: (a8 or a9)

;;; *The* turn pattern

abegin: "[DEGHKNQRS][ACFIKLMPTVWY][DEGHKNQRS]
[ACFIKLMPTVWY]"{-3,-3}

;;; Pattern frequently found at the beginning of helices.

aend: "[ACFIKLMPTVWY][DEGHKNQRS][ACFIKLMPTVWY][DEGHKNQRS]"

;;; Pattern frequently found at the end of helices.

alpha_philic: "[DEGKPNQRS]"

;;; Alpha/alpha hydrophilic residues

alpha_phobic: "[ACFIKLMPTVWY]"

;;; Alpha/alpha hydrophobic residues

alpha_strong_phobic: "[ACFILMVW]"

;;; Alpha/alpha strong hydrophobic residues

ap: (not density(>=,2,3,alpha_philic)){-1,-1}

;;; alpha_turn

g1: "[ACFIKLMTVWY][ADEHKNQRST]%2,2[AG]
[ADEHKNQRST]%2,2[ACFIKLMTVWY]"{4,4}

;;; Cluster of 'phobic bounded by 'philics (helix-helix interactions)

g2: "[ACFIKLMTVWY]%2,2[ADEGHKNQRST][ADEHKNQRST][AG]
[ADEHKNQRST]%2,2.[ACFIKLMTVWY]"{5,5}

;;; Cluster of 'phobic bounded by 'philics (helix-helix interactions)

g3: "[ACFIKLMTVWY].[ADEHKNQRST]%2,2[AG]
[ADEHKNQRST]%2,2[ACFIKLMTVWY]%2,2"{5,5}

;;; Cluster of 'phobic bounded by 'philics (helix-helix interactions)

gs: ((g1 or g2 or g3) and (not ma) and (not ga))

;;; A Gly-Ala type helical site without too many Alas.

ma: (density(>=,3,9,"A")){5,5}

;;; Too many alanines in one area are a bad sign for helix-helix packing

ga: (density(>=,2,9,"G")){5,5}

;;; Too many glycines in one area are a bad sign for helix-helix packing

anything: "."

END

