Exploiting whole–PDB Analysis in Novel Bioinformatics Applications

Varun Ramraj

Lincoln College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary Term 2014

To my parents, for constantly encouraging and enabling me to seek fertile grounds upon which to cultivate both academic and life experiences.

Acknowledgements

The story of my studies at Oxford begins with my supervisor, Robert Esnouf, answering my cold–call from Canada in March of 2009. I visited Oxford, and Strubi, in August of that year for a short three–day stint; Erika Mancini kindly organised for me to be put up in a very comfortable guest room at Jesus College. Robert and I drafted the three–pronged research programme presented in its refined form this thesis, and I thank him for enthusiastically taking me under his wing for a "slightly different," highly–computational structural biology project! Jenny Hayward and Susan Daenke have patiently handled my incessant questions and requests for various documents, signatures, and other administrivia during (and after!) the application process. I applied in November of 2009, and came up to Oxford in October 2010. For the past three–and–a–half years, Robert has been a supportive and caring supervisor, who has always been a phone call (or impromptu meeting) away and has (almost) never objected to me barging into his office. I daresay his rigorous application of the Scientific Method (and "sanity–checking" of results) has rubbed off on me, and I have become a far more competent scientist by his instruction!

Special thanks go to Dave Stuart, Gwyndaf Evans, and Robert, who were instrumental in securing me a joint studentship with the Nuffield Department of Medicine and the Diamond Light Source; this provided me with a large, active forum to distribute my software tools for the structural biology community, which satisfied a personal research goal. It also provided me with some much–needed funding to carry my studies through to completion! Gwyndaf has been a supportive mentor at Diamond and I have a much better appreciation for large–scale science after working with him and the Diamond MX computing/IT team (Bill Pulford, Alun Ashton and Graeme Winter). Thanks also to Robert and Gwyndaf for devoting many eye–watering hours poring over the thesis and offering poignant suggestions, to Jon Diprose for providing sound programming advice and help with debugging code, and to Prof. Chris Holmes and Dr Zamin Iqbal for providing access to high–end computing workstations. Stijn van Dongen (*MCL*), Robbie Joosten (*PDB_REDO*), and Sameer Velankar and Jose Dana (PDBe–related tools and web services) have been invaluable in their assistance and in providing custom web services for my use. Louise Durning and Carmella Elan–Gaston at Lincoln College have always taken an interest in my work and progress, and it has been a pleasure to be a part of that College community. Finally. thanks to Dave Stuart and Tassos Perrakis (Netherlands Cancer Institute) for enthusiastically agreeing to examine me and for providing insightful feedback to improve my thesis.

One often amasses a wealth of supportive friends and colleagues during the course of a doctorate, and indeed, I am very grateful to many wonderful people who were part of the journey. In no particular order, I would like to thank

my colleagues at Strubi, Wojtek Rzępała, Karen Lee, James Benoit, Adrian Ballard, Francisco Grajales, Annika Bruger, Alexander Ide, Kirstin Bilham, Yang Li, Caroline Bristow, Pádraig Belton, Melanie Beer, Marc Nowell, Tiffany Harte, Bryant Adibe, Christopher Noon, and Richard Cave–Bigley for their support, friendship and neverending willingness to invite me to an afternoon potable, dinner, get–together, trip or even a stimulating 'phone call whenever it seemed like I needed a breather. I would also like to thank Dr Kendall Ho, Dr Charles Krasic and Dr Arvind Gupta for helping me create a strong foundation for research by supervising me during my Master's; without their guidance and support, I would not have made it this far! My grandparents also deserve special mention here as they have always maintained a silent confidence that I would be "okay" and succeed at achieving my goals, going so far as to contribute financially in my academic endeavours. Lastly, I would also like to acknowledge Keenan Federico, Karim Hooda, Vladimir Avram, and Kip Warner for transcontinental encouragement to constantly strive for my best.

Finally, thank–you to my parents, without whom this entire journey would simply never have taken place. Sharing every single triumphant and nerve–wracking moment, they have patiently helped me stay my course whenever required, and I am relieved and excited to offer them this thesis as proof that their time and efforts were well–spent!

Exploiting whole–PDB Analysis in Novel Bioinformatics Applications

Varun Ramraj

Lincoln College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Hilary Term 2014

Abstract

The Protein Data Bank (PDB) is the definitive electronic repository for experimentally–derived protein structures, composed mainly of those determined by X–ray crystallography. Approximately 200 new structures are added weekly to the PDB, and at the time of writing, it contains approximately 97,000 structures. This represents an expanding wealth of high–quality information but there seem to be few bioinformatics tools that consider and analyse these data as an ensemble. This thesis explores the development of three efficient, fast algorithms and software implementations to study protein structure using the entire PDB.

The first project is a crystal–form matching tool that takes a unit cell and quickly ($<$ 1 second) retrieves the most related matches from the PDB. The unit cell matches are combined with sequence alignments using a novel Family Clustering Algorithm to display the results in a user–friendly way. The software tool, *Nearest–cell*, has been incorporated into the X–ray data collection pipeline at the Diamond Light Source, and is also available as a public web service.

The bulk of the thesis is devoted to the study and prediction of protein disorder. Initially, trying to update and extend an existing predictor, RONN, the limitations of the method were exposed and a novel predictor (called *MoreRONN*) was developed that incorporates a novel sequence–based clustering approach to disorder data inferred from the PDB and DisProt. *MoreRONN* is now clearly the best–in–class disorder predictor and will soon be offered as a public web service.

The third project explores the development of a clustering algorithm for protein structural fragments that can work on the scale of the whole PDB. While protein structures have long been clustered into loose families, there has to date been no comprehensive analytical clustering of short (~6 residue) fragments. A novel fragment clustering tool was built that is now leading to a public database of fragment families and representative structural fragments that should prove extremely helpful for both basic understanding and experimentation.

Together, these three projects exemplify how cutting–edge computational approaches applied to extensive protein structure libraries can provide user–friendly tools that address critical everyday issues for structural biologists.

Contents

List of Figures

Chapter 1

Introduction: Analysis of Protein Structure and Function

1.1 Overview

Proteins are molecular machines, responsible for essential functions of life. They are both synthesised from, and help regulate, gene function through a complex series of interactions and molecular pathways. The structure–function paradigm states that three–dimensional structure is directly responsible for a protein's function, as it promotes specific interactions that are wholly reliant on 3D geometry. Therefore, the field of structural biology, which aims to study protein structure, is vital for the understanding of protein function and regulation. Since the late 1960s, structural biology has grown exponentially due to the refinement of techniques used to determine protein structure. The major experimental techniques applied today are X–ray crystallography and nuclear magnetic resonance (NMR); however, electron microscopy (EM), small angle X–ray scattering (SAXS), circular dichroidism (CD) and a range of biophysical techniques also contribute at vary-

ing levels of structural studies. Since the early 1970s, results of these studies, *i.e.* resolved protein structures, along with the experimental data used to obtain the structure, have been deposited into the Protein Data Bank (PDB). With the advent of the Internet, the PDB has evolved into a large world–wide project (the world–wide PDB, or wwPDB) and it is growing quicker than ever before as newly resolved structures are deposited each week. The establishment of the PDB pre–dates modern bioinformatics; however, today's *in silico* structural biology, which complements experimentation, is almost entirely dependent on the PDB as the *de facto* repository of high–quality structures. Reciprocally, advances in structural bioinformatics have helped to curate and improve the quality of PDB depositions. In the last 20 years, other databases have been established as repositories for sequences, families and other macromolecular properties and are generating huge amounts of data. However, the PDB continues to be the best resource for structural information and an excellent complement to these other data sources.

Despite the existence of so much data, structural biology is still primarily concerned with the solution of individual protein structures. While this experimental output is critical and is the most fundamental way of increasing our knowledge, structural biology could look further into the existing data *en masse*. The common theme for this thesis is to mine the PDB to look at **protein space** as opposed to individual structures. As evolution operates on ecological scales, so does it operate at the molecular level; for example, studying the breadth of known protein space can provide very interesting and important insights about evolutionary pressure on protein families to look (and therefore function) in a certain way.

In this thesis, we follow three threads of computational studies on protein structure:

I. Crystal matching in the PDB to quickly find closely related crystal forms (and by extension, sometimes protein structures)

II. Disorder prediction (the majority of this thesis)

III. Clustering of short structural fragments in the PDB

The remainder of this chapter overviews protein structure, associated databases and basic computational concepts that are required for protein structure bioinformatics. Since this thesis is not experimental, the level of detail is restricted to the basics required to understand the software tools that are developed in subsequent chapters.

1.2 Proteins, Protein Structure and Disorder

1.2.1 Protein Synthesis and Primary Structure

The nucleus of each cell in an organism contains a complete copy of the genome for the organism. It is encoded in deoxyribonucleic acid (DNA) and undergoes the process of **transcription** into ribonucleic acid (RNA), which is then **translated** into an amino–acid sequence — a protein. Amino acid structures, names and abbreviations are shown in figure 1.1, which reflects the fact that different side chain compositions confer different electrochemical and stereochemical properties on amino acids. Note that all naturally encoded amino acids are enantiomorphic, with only the L–isomer used (with the exception of threonine which is stereomeric, owing to having two chiral centres). The resulting polypeptide consists of a main backbone of N and H^N (amino terminal), a central Cα atom (chiral carbon), and C and O (carboxyl terminus) atoms; side chains of individual amino acids protrude from the chiral Cα. Note that glycine is the exception, where the "side chain" is a single hydrogen atom; therefore, glycine is also the only achiral amino acid. The side chains of all other amino acids are anchored to the Cα atom by a Cβ atom.

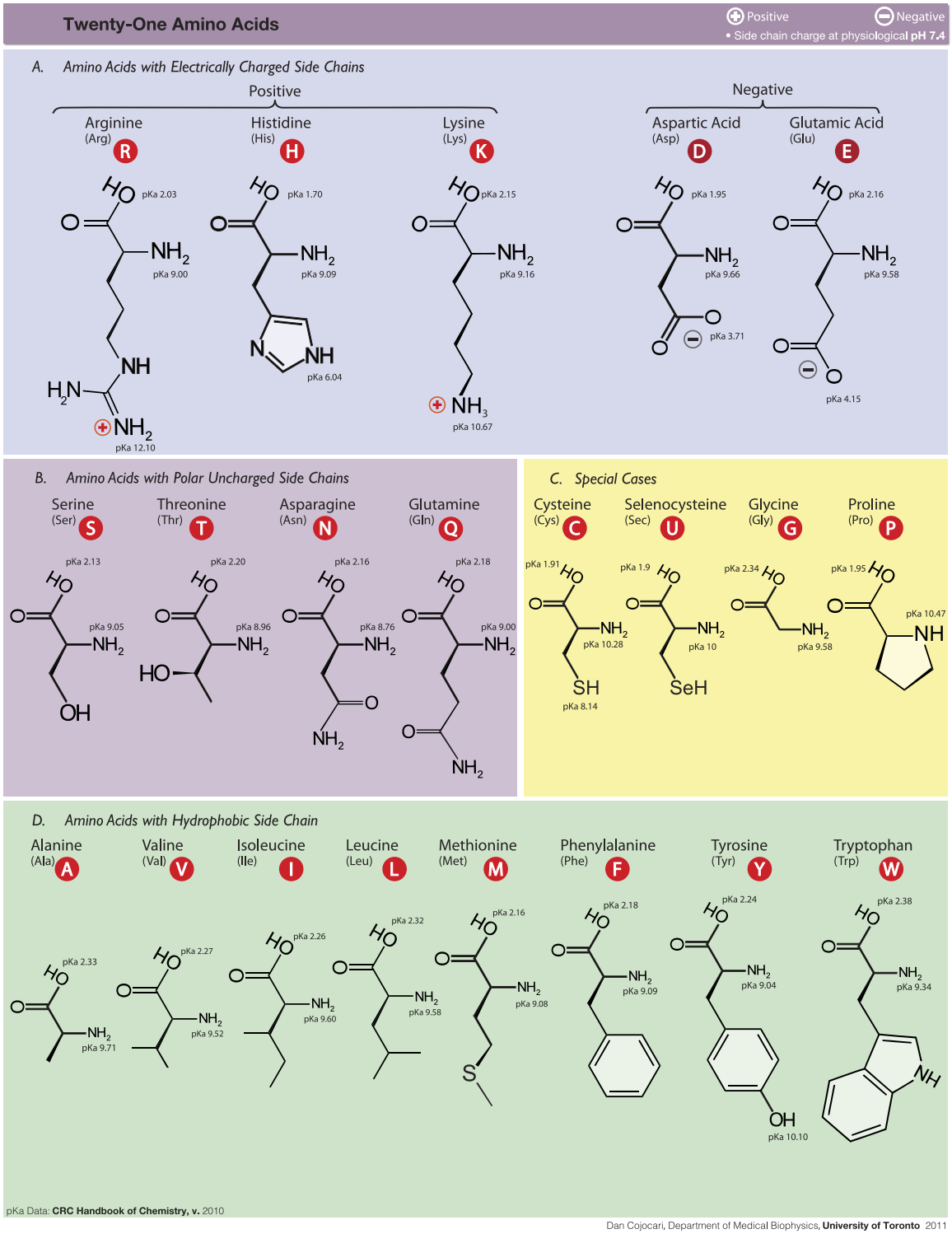


Figure 1.1: The structures of the 20 standard amino acids and selenocysteine (which only occurs in eukaryotes), along with their three–letter and one–letter abbreviations. The amino acids are organised into families based on their electric charge. Figure adapted from Cajocari et al. (2014).

Transcription occurs in specific regions in the DNA known as open reading frames (ORFs) with the help of proteins such as polymerases and ribonucleic acids. The transcribed messenger RNA (mRNA) strand contains binding regions for ribosomes that are involved in the translation process. Ribosomes read mRNA in triplets known as codons (see table 1.1), with each codon corresponding to a particular amino acid that is bonded through a condensation reaction to the growing peptide chain product. While the genetic code only encodes 20 normal amino acids, a modified cysteine, selenocysteine, is occasionally incorporated into peptides during translation (as opposed to a post–translational modification) and is therefore considered to be a separate, 21st, amino acid (Böck et al., 1991). Multiple codons map to a single amino acid and generally vary only in the third base (called the "wobble" position). The codon "ATG", which codes for methionine, is also the start codon in eukaryotes. There are three stop codons, "TAA", "TAG" and "TGA", the latter of which encodes for selenocysteine under certain conditions (Böck et al., 1991; Donovan et al., 2010).

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	**TGA**	**Stp***
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
ATG	**Met***	ACG	Thr	AAG	Lys	AGG	Arg
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 1.1: The standard genetic code, showing the mapping of DNA codons to amino acids. ATG, which codes for methionine, is also the start codon in eukaryotes. TGA is normally a stop codon but can also code for selenocysteine in certain conditions. This table was generated using the R statistics program and the "`seqinr`" package (Charif et al., 2007).

The chirality is the same for all amino acids (except glycine); it is shown in figure 1.2 (adapted from Esnouf, 1992) along with the three backbone torsion angles, ω, ϕ and ψ. Due to delocalisation of electrons across the peptide linkage, the central Cα—C—N—Cα linkage is approximately coplanar, and the bonds are usually in *trans* conformation ($\omega \simeq 180°$). A plot of the ϕ and ψ angles for each residue in a protein is called a Ramachandran plot (Ramachandran et al., 1963), which visually depicts possible ϕ and ψ values for each residue and can reveal some insight into secondary structure. Sample plots are shown in figure 1.3.

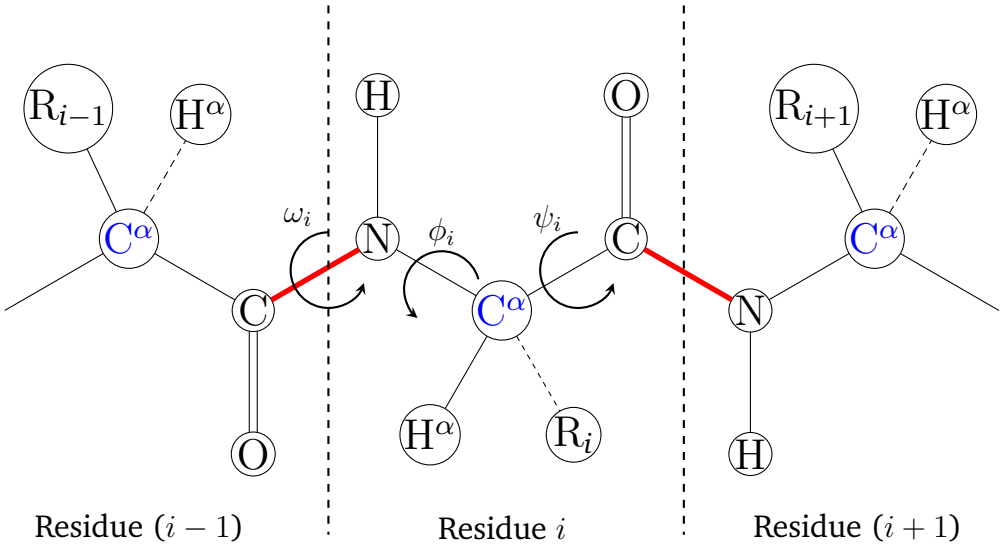


Figure 1.2: Main chain torsion angles. The peptide bonds are highlighted in red, while the Cα atoms are shown in blue. The R's are side chains whose chemical nature depends on the specific amino acid. Figure adapted from Esnouf (1992).

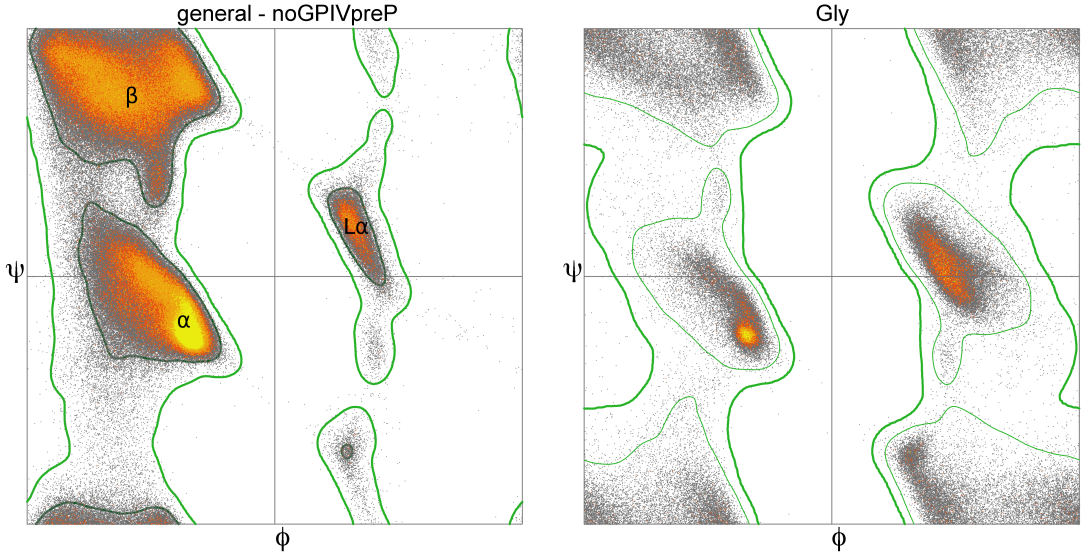


Figure 1.3: Sample Ramachandran plots. The data points in the plots are from a survey of approximately 825,000 amino acid residues taken from high–quality PDB structures by Read et al. (2011). The plot on the left shows all residues not including glycine, proline, valine and isoleucine. The contours demarcate the regions associated with α–helices, β–sheets and left–handed (Lα) helices. The distribution of glycine is shown separately at right since its small hydrogen side chain permits it to adopt a wide variety of ϕ–ψ conformations and would therefore add noise to the plot on the left. The inner contours enclose 98% of the data, whilst the outer contours enclose 99.95%. Figure adapted from Read et al. (2011) with permission from Jane Richardson.

1.2.2 Secondary Structure

Secondary protein structure is indicated by regular patterns of hydrogen bonding giving rise to two classes of structure: helices and sheets. Proteins may contain exclusively one class or a mixture of both. A Ramachandran plot (figure 1.3) can help visualise how residues in a protein are involved in secondary structure.

There are three types of helix; each is formed by a series of hydrogen bonds between the carbonyl oxygen of residue i and the amide hydrogen of either residue $i+3$ (called a 3_{10}–helix), residue $i+5$ (the rare π–helix) or residue $i+4$, which is the commonly occurring α–helix. Steric hindrance prevents the 3_{10}–helix or π–helix from occurring in long stretches, whereas α–helices can be ≥ 50 residues, as in the HA_2 chain of hæmagglutinin (Wilson et al., 1981). Due to the chirality of the Cα atoms, helices tend to be right–handed, although one notable exception is the left–handed helix (such as the one found in collagen) which contains many proline and glycine residues.

The second major conformation is composed of extended β–strands. β–sheets are formed by hydrogen bonding between multiple strands that are remote in the primary sequence. If adjacent pairs of strands run in the same direction as each other, they are parallel, otherwise, they are anti–parallel. Generally, sheets are either wholly parallel or anti–parallel with only around 20% sheets having mixed orientations (Brändén et al., 1991). Sheets can have local 'β–bulges', where additional residues appear in the middle of the strand but are not involved in hydrogen bonding. β–sheets can lie on top of each other (a β–sandwich) or even form complete loops of β–structure called β–barrels. For parallel β–barrels, α–helices are often found between the parallel sheets in a β–α–β pattern, usually with eight α–helices and eight β–sheets. This is known as a TIM barrel, named after triosephosphate isomerase. Figure 1.4 (Wu et al., 2007)[1] shows a ribbon diagram and a partial

[1]Reprinted from the Journal of Molecular Biology, Volume 366, Issue 5, Wu *et al.*, 2007, "A

space–filling model of the TIM barrel found in the α subunit of tryptophan synthase (αTS). More complex β–structures, such as β–propellers, are also possible.

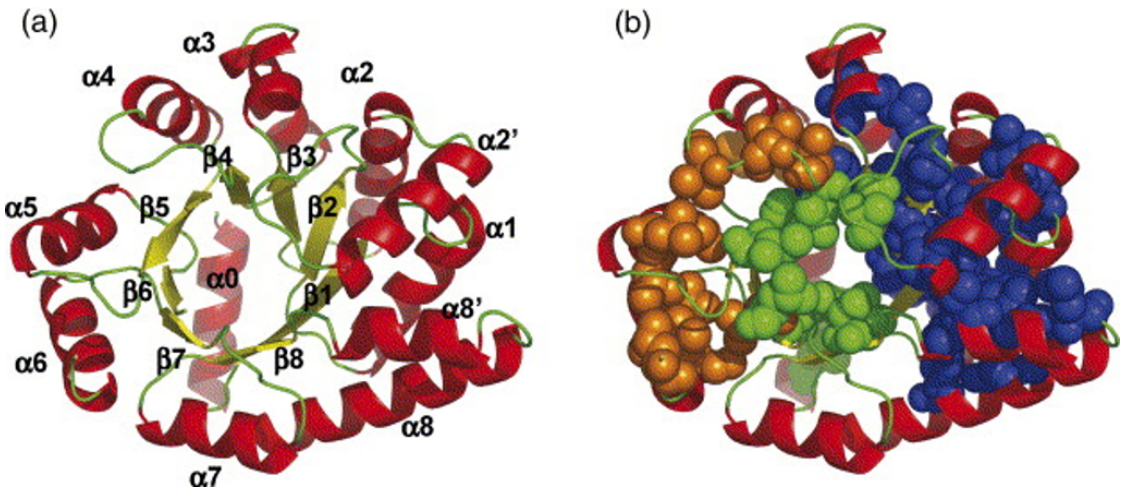


Figure 1.4: Figure 1 from Wu et al. (2007), showing (a) a ribbon diagram, and (b) a partial space–filling model of the TIM barrel in αTS. The springs represent α–helices and the arrows depict the direction of β–sheets. The space–filling model highlights hydrophobic regions as orange, green and blue spheres. These regions are rich in isoleucine, leucine and valine.

Loop regions occur in between α or β structures (in the latter case, loops between anti–parallel β–strands are called β–hairpins), and are usually polar or charged (able to interact with the solvent) (Brändén et al., 1991). Loops frequently serve as active sites in proteins and are therefore functionally and evolutionarily conserved.

1.2.3 Tertiary and Quaternary Structure, and Specificity

Tertiary structure is the resulting 3D shape taken by a protein domain. It is formed through hydrophobic interactions and other van der Waals forces[2]; hydrophobic regions are buried into the protein while hydrophilic regions face the solvent. It is in this native 3D form that most proteins become functionally ac-

Tightly Packed Hydrophobic Cluster Directs the Formation of an Off-pathway Sub-millisecond Folding Intermediate in the α Subunit of Tryptophan Synthase, a TIM Barrel Protein", Pages 1624-1638, Copyright 2007, with permission from Elsevier.

[2]Note that two nearby cysteines can form disulphide bonds under certain conditions during this folding process. These are covalent bonds rather than hydrogen bonds.

tive; binding sites and other regulatory regions usually need to be in their native 3D state in order to accept their binding partners. Thus, the amino acid sequence gives rise to an unique 3D structure, and subsequently, the function of the protein; this is known as the **structure–function paradigm**. It is important to note that, while primary structure between two proteins may be different, the tertiary folded state may look very similar and therefore lend similar functionality (Brändén et al., 1991). Finally, quaternary structure describes **complexes**, or non–covalent interactions between multiple structural domains or monomeric protein species (Price et al., 1999). Proteins often need to form or be part of a complex in order to function properly (*e.g.* hæmoglobin is a complex of four monomers in an $\alpha_2\beta_2$ arrangement).

1.2.4 The Structure–Function Paradigm and Protein Disorder

Given that the primary structure could adopt numerous conformations, it is remarkable that primary sequence leads to a unique tertiary structure imparting the correct functionality. The initial hypotheses about the role of structure in specificity were investigated by Fischer, Mirsky and Pauling and other pioneering structural biologists and biochemists, using enzymes as a convenient target of study (Dunker et al., 2001a). Enzyme–substrate specificity is defined as the specific recognition and chemical interaction between an enzyme and its substrate; indeed, enzymes are generally highly specific and will not interact with any substrate other than their intended targets. Fischer's "lock–and–key" hypothesis implies a rigid active site structure for the enzyme (the "lock") which can be activated by the unique substrate (the "key"), and thus further bolsters the structure–function paradigm; the enzyme's structure is directly responsible for its unique function and specificity (Lemieux et al., 1994). The lock–and–key theory worked well to explain known protein-protein or enzyme-substrate interactions until Karush described the heterogeneity of bovine serum albumin. This protein is able to bind to small, hy-

drophobic molecules of vastly different shapes. Two independent theories were formulated: either different configurations of the binding site exist in equilibrium ("configurational adaptability") (Karush, 1950), or the binding site is able to actively change shape to accommodate differently shaped substrates ("induced fit") (Koshland, 1958). Both configurational adaptability and induced fit rely on the observation that the "lock" does not have a single shape and can accommodate different "keys".

Thus, while the structure–function paradigm can be applied to many protein interactions such as enzymes, which were the first proteins studied to have a well–defined structure, it is simplistic and does not account for these non–conforming[3] regions. This unstructured state is known as "**disorder**" and its identification is the central topic in this thesis.

Theoretically, a disordered region in a protein can be identified by its lack of sustained rigid conformation; experimentally speaking, a region of disorder in a polypeptide displays no electron density in a protein crystal structure. However, these regions can be essential for protein function (Dunker et al., 2001a). Disordered regions may, however, be induced into a stable, experimentally discernible shape through interaction with substrates or cofactors that alter the disordered region's local environment and cause it to become spatially fixed. While natively disordered regions do not display any conformational preference, they can transiently adopt a preferred structure upon interaction with a substrate. By having this ability to develop a transient structure, disordered proteins still satisfy the structure-function paradigm (Wright et al., 1999).

For the experimentalist, disorder is often viewed to simply have "nuisance value." It is often associated with bad expression and aggregation of constructs as well as a barrier to growth of high–quality crystals. Much effort is expended trying

[3]Literally *and* figuratively!

to design well–behaved crystals, and disorder prediction is an important practical tool in this context.

1.3 The Protein Data Bank

The Protein Data Bank (PDB) (Bernstein et al., 1977) is a comprehensive repository of protein structure and experimental data used to determine the structure. It was established in 1971[4] at the Brookhaven National Laboratory to be the *de facto* international repository for 3D macromolecular structural and experimental data (Berman et al., 2000). The worldwide PDB (wwPDB) is the latest iteration of the collaborative project, and it is hosted and mirrored in three countries under three different umbrella projects (Berman et al., 2003):

1. United States — managed by the RCSB (Rutgers University, San Diego Supercomputing Centre, and the Centre for Advanced Research in Biotechnology within the National Institute of Standards and Technology).

2. Europe — managed by the MSD-EBI at the European Bioinformatics Institute, Hinxton.

3. Japan — managed by *PDBj* at the Institute of Protein Research in Osaka.

Of these three projects, the RCSB is the main archive maintainer, with sole write access to the database. However, depositions can be made through any of the three portals, and each maintains its own web–based interface for interacting with PDB data (Berman et al., 2003). For the projects in this thesis, the MSD-EBI mirror was used as it is geographically the closest to Oxford and therefore offers the fastest communication and data transfer.

[4]Much before the advent of bioinformatics!

Each PDB entry is stored in two standardised formats, the legacy PDB format (Bernstein et al., 1977; Berman et al., 2000), and the new XML format (Westbrook et al., 2005). The PDB data formats are able to store metadata for a wide variety of organisms and experimental techniques. The PDB contains just over 97,000 entries as of January 27, 2014, and two breakdowns of the data are shown in tables 1.2 and 1.3.

Method	**Proteins**	**Nucleic Acids**	**Prot./NA Complexes**	**Other**	**Total**
X–ray	**80250**	1498	4185	4	85937
NMR	9011	1070	197	7	10285
EM	498	51	170	0	719
Hybrid	55	3	2	1	61
Other	155	4	6	13	178
Total	89969	2626	4560	25	**97180**

Table 1.2: A survey of holdings in the PDB as of January 27, 2014, laid out as presented by the RCSB mirror and organised by experimental method. Out of 97,180 entries, a vast majority (80,250) are proteins only, and have been resolved by X–ray crystallography.

Organism	**Number of deposited structures**
Eukaryota	49766 (24792 human)
Bacteria	35840
Virus	6431
Archaea	3629
Unassigned	2852
Other	761

Table 1.3: A survey of structures in the PDB as of January 27, 2014, laid out as presented by the RCSB mirror and arranged by taxonomy. Over half of the deposited structures are eukaryotic (wherein half are human). Unassigned and "other" structures tend to be engineered or incompletely annotated. Note that a PDB entry can contain more than one type of structure.

In this thesis, we concentrate on protein structures (*i.e.* no nucleic acid) that have been resolved by X–ray crystallography. Conveniently, this is where the PDB's strength lies, with 82.5% of the entries in this category. The PDB is updated weekly and it is currently growing at a rate of around 200 structures per week (10,000 per

year)[5].

NMR records contain ensembles of coordinates, as discussed before. EM records, until now, tended to be somewhat artificial, as they incorporated crystal or NMR–derived models to provide structural building blocks (see section 1.4). In this thesis however, we restrict ourselves to protein X–ray data, and therefore, PDB data derived from NMR, EM and other experimental methods are not described further, save for the use of NMR records in disorder prediction, discussed in chapter 3.

For protein crystal PDB entries, there are three sections of importance:

1. Headers – These contain metadata such as organism name, expression system, experimenters and contact information, and reference publications. They also contain information on caveats in the experiment (for instance, cleavage of a section of protein in order for it to crystallise properly). Finally, of great importance is the "CRYST1" record, which contains information on crystallographic parameters, and is discussed in more detail in section (1.5).

2. SEQRES records – Describe the amino acid sequences for each protein species, usually derived from UniProt DNA sequences.

3. ATOM records – These contains 3D coordinates and other data for each atom in the crystal.

The hierarchy of protein structure has led to a corresponding hierarchy in bioinformatics methods for analysing and grouping structure. Ultimately however, all structure–based databases and classification schemes, *e.g.* Pfam (Punta et al., 2012), SCOP (Lo Conte et al., 2000) or CATH (Pearl, 2003), rely on experimental data that have been deposited into the PDB. The PDB, therefore, is the defini-

[5]On May 14, 2014, the PDB passed the 100,000 structure mark. Serendipitously, 2014 was declared the International Year of Crystallography to celebrate the centennial of X–ray diffraction and the 400th anniversary of Johannes Kepler's observations of the symmetry of ice crystals. The author is pleased to have submitted his thesis in the same year!

tive resource for protein structure, and forms the core of this thesis. The research philosophy and techniques developed in this thesis revolve primarily around X–ray crystallography data in the PDB and therefore, X–ray techniques are discussed more expansively later in this chapter. Preceding that however, other common structural biology methods are summarised in section 1.4.

1.4 NMR and EM

Protein NMR spectroscopy was pioneered in the late 1960s by Richard E. Ernst and Kurt Wüthrich, among others (Wüthrich, 2001). It exploits quantum mechanical properties of atomic nuclei which in turn are affected by the local environment around each atom, including the geometric alignment of nearby atoms. Protein samples are prepared as an aqueous solution and are then placed inside a powerful, but uniform, magnetic field. Radio waves are then aimed at the sample, which then interact with the atomic nuclei in the protein. Absorption is measured in the frequency spectrum and complex calculations are performed to arrive at the final 3D structure of the sample. NMR's major advantage is that it is able to discern structure of proteins in solution, albeit very concentrated and often with unnatural solvents. Therefore, NMR is useful for probing the structure of flexible proteins, or proteins that contain disordered regions; furthermore, NMR is much better at resolving small proteins (less than 70 residues). The final NMR data set is an ensemble of protein structure models, the first model usually being the best refinement.

Electron microscopy is most useful for determining structures of large complexes (*i.e.* quaternary structure); as its name implies, an electron beam is aimed at the sample and the resulting resulting images are used to build up a 3D structure, often at limited resolution[6]. Cryo–electron microscopy examines the sample at cryogenic

[6]Usually between 3Å to 10Å or lower, although the method is fast improving.

temperatures, which reduce the radiation damage. The resolution of cryo–EM is improving and has become much more popular since the 1980s (Adrian et al., 1984) in the study of large molecular assemblies and complexes. EM is not typically capable of near–atomic resolution; in such cases, X–ray or NMR data may be used to better describe the 3D structure, resulting in a full–atom model.

NMR and EM (along with other biophysical methods such as circular dichroidism) have contributed to a small portion of the structural data found in the PDB, but X–ray crystallography remains the *de facto* structural biology technique for solving protein structures. Section (2.2) provides a brief overview of crystallography and organisation of X–ray data in PDB entries, as the computational methods discussed in this thesis are almost entirely informed by crystallographic data in the PDB.

1.5 Quality of Structural data in the PDB

As most of the work in this thesis revolves around selecting a "high–quality" subset of the PDB, we must first define what this means. Quality metrics fall into two categories: global quality, which is measured on the entire structure, and local quality, which is a set of criteria to evaluate individual amino acids in a protein.

1.5.1 Global quality

In a PDB file, the CRYST1 record contains information on the unit cell parameters and the space group. Other header records contain global quality metrics, called R–values, that reflect how well the model agrees with the experimental

diffraction data. The standard R–factor (known as R_{work}) is defined as:

$$R_{work} = \frac{\sum |F_{obs} - F_{calc}|}{\sum |F_{obs}|} \tag{1.1}$$

where F_{obs} and F_{calc} are the amplitudes of the observed and calculated structure factors, respectively. A similar measure, called R_{free} (Brünger, 1992), is calculated using the same formula, but applied to the small subset of data that are not used in refining the model. At each stage of the iterative refinement, therefore, the R_{free} provides a way of validating the progress of the refinement, in particular, the refinement protocol.

The 3D coordinates of each atom from the model refinement are stored in the ATOM records of the PDB entry. Coordinates are defined relative to a point of highest crystallographic symmetry for the relevant space group. In context of an X–ray structure, each ATOM record contains:

1. 3D coordinates in (x, y, z) format

2. Occupancy — a measure of whether the atom is found in all unit cells in the same place. If it is, the occupancy is 1.0; otherwise, this indicates that the atom is found in multiple conformations, multiple species are present, or the atom lies on a symmetry axis or point. The occupancy is a fraction, where the sum of all conformations for an atom should always add up to 1.0.

3. The B–factor or temperature factor — a measure of how rigid the atom is in the crystal; higher numbers indicate that the atom is moving a lot (temporal disorder), or its position is somewhat ill–defined with respect to the unit cell (spatial disorder). In both cases the electron density is smeared out or blurred.

Indicators of global quality can be found in other headers in the PDB. Two such notable indicators are:

1. RMS deviations for average bond angle and bond length — a measure of the leftover strain in the model; ideally, the model should minimise energy as much as possible.

2. Experimental resolution — High–resolution data should yield high–quality structures. At 1.5Å or higher, individual atoms can be resolved.

Once a model is refined, the average B–factor, R–value and average RMS bond length and angle deviations are very useful global quality metrics with which to evaluate a deposited PDB structure, and provide first–order quality control. However, there can be stretches of low–quality structure or individual low–quality atom groups even in the best PDB structures; in such cases, the global metrics are not useful (Tickle, 2012). For instance, the clustering project discussed in chapter 7 relies entirely on selecting high–quality short fragments in the PDB, as opposed to complete structures. For this project, therefore, a set of criteria for local quality was developed as a second–order filter.

1.5.2 Local quality metrics and PDB_REDO

The four measures of per–residue quality used in this thesis are RMS bond length deviation, RMS bond angle deviation, real–space residual (RSR) and real–space correlation coefficient (RSCC). The RMS bond length and angle values are deviations in length or angle (respectively) from the accepted refinement (target) values.

RSR is a measure of the similarity between the electron–density map calculated from the structural model and the electron–density map obtained using experimental structure factor amplitudes (Tickle, 2012). One method of calculating RSR is shown in 1.2 below and was taken from Jones et al. (1991).

$$\text{RSR} = \frac{\sum |\rho_{obs} - \rho_{calc}|}{\sum |\rho_{obs} + \rho_{calc}|} \tag{1.2}$$

The sums are over all points on the electron–density map that cover the residue currently being examined, while ρ is the electron density at each point. *obs* and *calc* subscripts denote observed and calculated electron density, respectively. RSR values increase from 0 ("perfect fit").

RSCC is similar to RSR in that it is a "best–fit" metric between the structural model and experimental data. Unlike RSR however, RSCC is scale–independent. Atoms modelled correctly in regions of weak electron density will still be highly correlated with the experimental data, which makes RSCC a more robust measure than RSR (Brändén et al., 1990; Jones et al., 1991; Tickle, 2012). RSCC values range from –1 ("anti–correlated") to 1 ("perfectly correlated"), with 0 indicating that there is no correlation.

Vriend (1990) developed a molecular modelling package called *WHAT_IF* which contains a variety of tools for working with protein structures. One component, called *WHAT_CHECK*, was developed to validate PDB structures. *WHAT_CHECK* automatically checks for a variety of potential problems in refined structures, including unusual bond angles or lengths, unexpected atoms, unusual B–factors or other anomalies[7]. For our purposes, it represents an easy "first pass" to remove residues that are clearly unsuitable for further consideration. It is an integral part of the *PDB_REDO* (Joosten et al., 2009a) pipeline.

It is important to note that currently, no satisfying all–purpose metric is fully able to capture the notion of quality for a given residue. RSR and RSCC are not, for instance, weighted by the number of atoms or the size of the modelled atoms in the electron–density map, which could change the quality score significantly. The combination of RSR and RSCC are used in the applications described in this thesis since they form the basis of *PDB_REDO* quality measurements. However, Joosten et al. (in press) have recently devised a z–score that is a closer single–

[7] *WHAT_CHECK* calculates RSR using the Jones et al. (1991) equation.

metric approximation for a single residue which may, in the future, supersede the use of RSR and RSCC for applications similar to those described in this thesis.

The *PDB_REDO* software pipeline is designed to optimise and re–refine X–ray structures. It takes the deposited atomic model and structure factors associated with a PDB entry and re–refines (and partially rebuilds) the structure. It can highlight flaws in the original refinement, especially if the structure is old and was refined using legacy methods (Joosten et al., 2009a,b). The *PDB_REDO* server maintains *PDB_REDO*–refined versions of PDB structures with added metadata about structure quality, including *WHAT_CHECK* reports after the tool is run on re–refined structures. Since it both improves existing PDB structures and evaluates them with *WHAT_CHECK*, it is the ideal "one–stop" resource for isolating high–quality residues and eliminating problematic ones. Furthermore, it refines structures in a common way which standardises systematic errors.

As part of *PDB_REDO*'s refinement of a structure, it generates all four per–residue metrics discussed above. A more detailed description of the *PDB_REDO* suite (including *WHAT_IF* and *WHAT_CHECK*) can be found in chapter 7.

1.5.3 A note on inconsistencies in data recording

Due to the age of the PDB and bespoke annotation conventions between different labs and experimenters, deposited data can be extremely variant, which causes problems for computational data analysis and retrieval. This is more apparent in older PDB entries, but can crop up at any time. For the remainder of this thesis, unless otherwise specified, it should be assumed that any PDB entries that deviate from the standard format or contain esoteric or incomplete annotations have been discarded.

1.6 Biological similarity

Jacob (1977) described evolution as a "tinkerer", not an inventor. Indeed, biology is much better at adapting DNA or protein molecules than creating new ones (Durbin et al., 1998). Genes or proteins with similar functions and common ancestry are **homologous** to each other. The related term for measuring sequence or structure is **similarity**.

1.6.1 Sequence similarity

Similarity in DNA or amino acid primary sequences can be identified by treating the sequences as text strings and performing **alignment** on them. This is the crux of modern bioinformatics, although computers have been used since the early 1970s for crystallographic applications. As sequences evolve, biology often maintains invariant stretches interspersed between insertions, deletions and mutations. For proteins, biology often adds post–translational modifications to amino acids, which we ignore here. Therefore, central to bioinformatics is the development of scoring models for assessing the similarity between primary sequences. Scoring models can be simple, however, it is desirable for a model to take into account evolutionary history and functional motifs, as these directly impact the course of evolution of the primary structure (Durbin et al., 1998).

1.6.1.1 Pairwise alignments

Scoring models for **pairwise alignment** of protein primary sequences are intimately tied to various probabilistic **substitution matrices**. The first such matrices were the Point Accepted Mutation (PAM) matrices developed by Margaret Dayhoff and her colleagues (Dayhoff et al., 1972, 1978), which relied on a Markov model of protein evolution to score individual (point) amino acid replacements over differ-

ent lengths of time. However, PAM matrices assume that substitution rates at any given position in the sequence are constant, whereas in reality, some regions are conserved over millions of years and others evolve rapidly (Gonnet et al., 1992). The BLOSUM family of matrices (Henikoff et al., 1992) attempts to solve this problem by using alignments of distantly related proteins from the BLOCKS database (Henikoff et al., 1991; Pietrokovski et al., 1996; Henikoff et al., 2000) to derive the mutation frequencies of amino acids[8]. The two most commonly used matrices are BLOSUM50 and BLOSUM62; the former is preferable for gapped alignments (Pearson et al., 1992), whilst the latter is more suitable for ungapped alignments[9] (Durbin et al., 1998) and is the matrix used for the bulk of the software in this thesis. For reference, the BLOSUM62 matrix is shown in figure 1.5.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**4**	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0
R	-1	**5**	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3
N	-2	0	**6**	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3
D	-2	-2	1	**6**	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3
C	0	-3	-3	-3	**9**	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
Q	-1	1	0	0	-3	**5**	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2
E	-1	0	0	2	-4	2	**5**	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2
G	0	-2	0	-1	-3	-2	-2	**6**	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3
H	-2	0	1	-1	-3	0	0	-2	**8**	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	**4**	2	-3	1	0	-3	-2	-1	-3	-1	3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	**4**	-2	2	0	-3	-2	-1	-2	-1	1
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	**5**	-1	-3	-1	0	-1	-3	-2	-2
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	**5**	0	-2	-1	-1	-1	-1	1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	**6**	-4	-2	-2	1	3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	**7**	-1	-1	-4	-3	-2
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	**4**	1	-3	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	**5**	-2	-2	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	**11**	2	-3
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	**7**	-1
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	**4**

Figure 1.5: The BLOSUM62 substitution matrix as generated by the BLAST algorithm from the newest BLOCKS database in January 2014. Note that the diagonal values, shown in red, are all positive. This means that non–varying amino acids are scored higher in an alignment. The matrix is symmetric.

[8]The BLOCKS database actually contains multiple alignments of *conserved* sequences categorised into protein families, but distances between families, *i.e.* between more distantly related proteins, are used to derive the BLOSUM matrices.

[9]The BLOSUM62 matrix is therefore less deviant than the BLOSUM50.

A BLOSUM matrix is calculated by counting the observed relative frequencies of amino acids in highly conserved regions of protein families in the BLOCKS database. The "62" in BLOSUM62, for instance, refers to a 62% identity between the proteins in the BLOCKS database. The log–odds score is then calculated, as shown in equation 1.3, for each of the 210 possible substitutions for the 20 standard amino acids (Eddy, 2004):

$$S_{i,j} = \log \frac{p_i \cdot M_{i,j}}{p_i \cdot p_j} = \log \frac{M_{i,j}}{p_j} = \log \frac{\text{observed frequency}}{\text{expected frequency}} \tag{1.3}$$

where $M_{i,j}$ is the probability of amino acid i mutating to j, and p_i and p_j are the frequencies of the two amino acids in the BLOCKS database.

The concept of **dynamic programming** forms the basis of most early pairwise alignment methods (Pearson et al., 1992). Dynamic programming algorithms keep all potential alignments in memory as the sequences are read forwards to calculate the running scores. The best alignment is then constructed backwards once the maximum score is known. The most fundamental dynamic programming alignment algorithms are Needleman–Wunsch (Needleman et al., 1970), which is used to globally align two sequences, and Smith–Waterman (Smith et al., 1981), which is a variant that produces optimal local alignments. Both algorithms split the problem of pairwise alignment into smaller sub–problems and permit gaps to be inserted into the alignment. When dealing with long primary sequences, it is generally more useful to align locally since the sequences can vary in length. Local alignment is better suited in these cases to reveal functional sites or other conserved motifs. Therefore, a brief explanation of Smith–Waterman follows, although it should be noted that Needleman–Wunsch is almost identical in its operation.

Smith–Waterman requires a scoring model that defines a score for a match, mismatch and gap. Generally, gaps and mismatches are negative scores. The PAM

and BLOSUM matrices are often used as they provide a convenient scoring method that is based on evolution. Assuming we are attempting to align sequence A of length i and sequence B of length j, Smith–Waterman has three basic steps:

1. Initialise a matrix of size: $(i + 1) \times (j + 1)$. Set the top left corner coordinate as a blank with a score of 0.

2. Fill the matrix with the appropriate running scores.

3. Trace back from highest score until 0 is encountered.

Filling the matrix (step 2) is the key. The score for each coordinate is calculated by referring to the previously calculated coordinates to the left, above and to the diagonal–left of the current coordinate as shown in equation 1.4 below:

$$M_{i,j} = \max\left((M_{i-1,j-1} + S_{i,j}), (M_{i,j-1} + W), (M_{i-1,j} + W), 0\right) \tag{1.4}$$

where, usually, S is the substitution score (from BLOSUM62, for example) and W is the gap penalty. If the score derived from the neighbours is negative, then a score of 0 is assigned. As part of this step, a pointer to the the coordinate that provided the maximum score is maintained. Finally, in step 3, the trace back begins from the highest score, following the path backwards and visiting each of the coordinate pointers that have been stored, until a 0 score is encountered. This represents the optimal alignment.

Whilst Smith–Waterman does return the optimal alignment every time, its run time is directly proportional to the lengths of *both* input sequences, that is, $O(i * j)$, which is slow when long sequences are being aligned. The Basic Local Alignment Search Tool (BLAST) (Altschul et al., 1990) is the *de facto* algorithm for fast, local pairwise alignments and implements a heuristic alignment algorithm that is inspired

by the Smith–Waterman approach[10]. We focus here on the "`blastp`" variant, which is used to align protein sequences.

The Smith–Waterman algorithm can be modified to find all high scoring local alignments, as opposed to the optimal one. These high–scoring pairs (HSPs) can be predicted statistically through the "E–value" as shown below in equation 1.5, which calculates the number of HSPs with score of *at least* S. As long as the two sequences are sufficiently long (of lengths m and n), the parameters K and λ characterise this result (Karlin et al., 1990).

$$E = Kmn\left(e^{-\lambda S}\right) \tag{1.5}$$

The constant K is a scale for the size of the search space (in this case, the size of the BLAST database) and λ is a scaling factor for the scoring system (such as the BLOSUM matrix). They are known as Karlin–Altschul parameters and the details of their derivation are given in Karlin et al. (1990). In place of length n when comparing against the entirety of the BLAST sequence database, BLAST assumes that the database is a single long sequence of length N; the E value in this case is called the "database E–value", and this is the value that is reported by the algorithm. Lower E–values represent better matches.

The BLAST database is any large collection of curated sequences, `blastp`, for instance, can be run against SEQRES sequences in the PDB! Overlapping sub–sequences known as "words" are extracted; each word is of length w. This word size varies depending on what is being compared, but for proteins, the default word size is 3. Each word in the query sequence is compared with the database of words, and upon a match, a local alignment is begun at that point in the query. Word pairs that score greater than some threshold T are kept. After all words in the query have

[10]Whilst BLAST is heuristic and therefore cannot give optimal alignments in all cases, its speed and relative accuracy more than make up for this.

been compared to the database and extended through local alignment as much as possible, BLAST assembles the best alignment for each query–database pair. The resulting pairs, along with the E–values, are reported to the user (Altschul et al., 1997).

1.6.1.2 Multiple sequence alignments

The next logical step after a pairwise alignment is a **multiple sequence alignment** (MSA), where the best overall alignment is obtained for more than two input sequences. The most popular MSA algorithm is CLUSTAL/CLUSTALW (Higgins et al., 1988; Thompson et al., 1994) which was re–implemented in C++ along with a brand new web service by Larkin et al. (2007). The basic CLUSTAL approach relies on multiple pairwise alignments to generate a "guide tree" of alignments based on evolutionary distance. The sequences are then progressively aligned in pairwise fashion following the order indicated by the guide tree (Higgins et al., 1988). CLUSTALW improves on this by altering weights on the guide tree based on pairwise sequence similarity, by allowing different substitution matrices to be used at each stage of the alignment based on the divergence of the sequences being compared, and by incorporating different gap penalties based on amino acid composition (Thompson et al., 1994; Larkin et al., 2007). Sievers et al. (2011) recently developed a modified CLUSTALW algorithm called Clustal Omega, which outperforms its predecessors. It is similar to CLUSTALW but builds the guide tree in a different way by relying on HMMs generated from input and reference sequences. It can also utilise HMMs generated from databases such as Pfam (Punta et al., 2012) to improve the pairwise alignments themselves. MUSCLE (Edgar, 2004) is another common MSA package that uses a metric called the log–expectation score to measure distances. It also introduces unique methods of building and partitioning pairwise–comparison distance trees and has been measured to outperform other

MSA methods, both in speed and accuracy.

1.6.1.3 Clustering and sampling

Given that sequence alignment is fundamental to protein bioinformatics, it is often helpful to go a step further and group similar sequences together into protein "families". This process is known as **clustering**, and the representative clustering algorithm discussed here is CD–HIT (Li et al., 2001; Li et al., 2002), as it features throughout this thesis.

CD–HIT implements the greedy approach developed by Holm et al. (1998) and is an elegant solution for fast clustering. The algorithm begins by sorting all input sequences in decreasing order of length and picking the longest sequence as the first cluster **representative**. The remaining sequences are then compared to it based on identity (Smith–Waterman), and if a sequence A is above the identity threshold, it is included in the cluster. Otherwise, a new cluster is started with sequence A as the representative. In order to reduce the number of pairwise comparisons that need to be made (keeping in mind that Smith–Waterman is slow for long sequences), an index table of short peptide words (from 2 to 5 residues) is made in advance. The occurrences of these words are measured when a new sequence is presented for clustering, and only when the number of word occurrences is greater than the required threshold is an alignment performed to confirm the sequence identity (Li et al., 2001).

The main output from CD–HIT is a list of cluster representatives. If the clustering process was effective, the number of representatives is far lower than the number of input sequences, and each representative efficiently encapsulates a larger sequence space; in other words, they **sample** the available sequence space. Clustering and sampling are illustrated in figure 1.6.

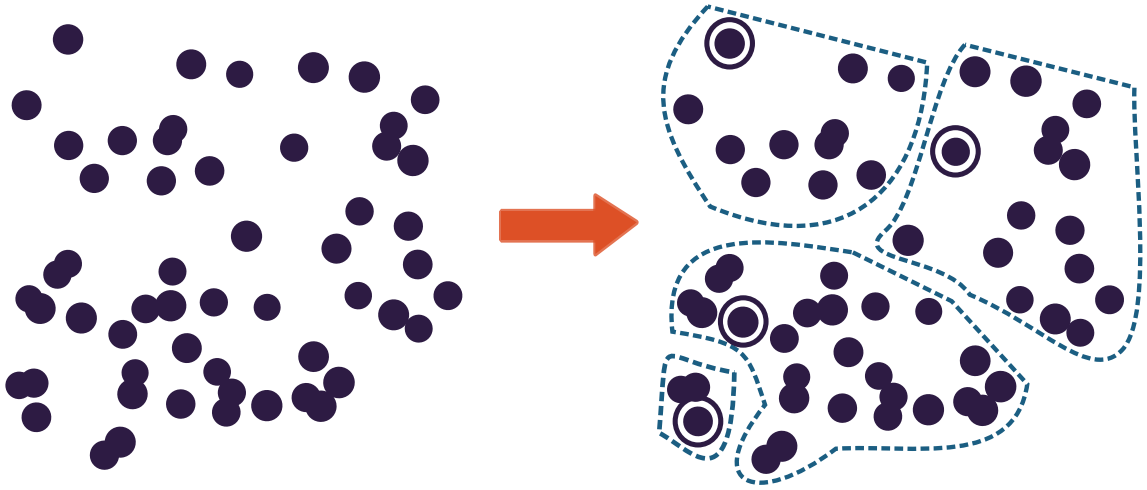


Figure 1.6: Clustering and sampling. Each circle on the left can be thought of as a protein sequence. An algorithm such as CD–HIT might cluster them together as shown on the right. Representative sequences for each cluster (i.e. samples) are highlighted as bull's–eyes.

The key metric for CD–HIT comparisons is **identity**, that is, two residues must be the same to score in the alignment. However, given that protein folding is determined to a large extent by the electrochemical properties of amino acids, related residues such as lysine and arginine can often be interchanged without greatly affecting structure (see figure 1.1). The BLOSUM62 matrix captures this sort of substitution well, as it tends to heavily penalise amino acid substitutions where the properties of the new amino acid are different from the old, whilst being more lenient with substitutions of similar amino acids. Therefore, sequence **similarity** is arguably a more biologically relevant way to cluster them, especially in the context of disorder prediction, where amino acids with similar properties will tend to be similarly disordered. However, since CD–HIT cannot cluster in this fashion, a more general clustering algorithm called *MCL* (Enright et al., 2002) was used and tuned to cluster by similarity. Details about *MCL* and its application to disorder prediction are discussed in chapter 5.

1.6.2 Structural similarity

Protein structure can display convergent evolution in motifs and binding regions despite arising from completely unrelated primary sequences (Higgins et al., 2000). Therefore, sequence alignment methods need to be complemented by methods that compare proteins at the structural level. The SCOP (Lo Conte et al., 2000) and CATH (Pearl, 2003) databases are often used in automatic protein structure classification and prediction; classification of domains in a newly–solved crystal structure, for example, can reveal functional information almost immediately. Tertiary structure prediction, or protein folding algorithms, rely on SCOP and CATH databases for training (Csaba et al., 2009). However, the SCOP and CATH databases have not been updated for many years and quantitative, manual curation of the myriad different tertiary motifs can quickly become an overwhelming task.

Therefore, we restrict ourselves in this thesis to a brief look at secondary structural analysis instead. The programme called Define Secondary Structure of Proteins (DSSP; Kabsch et al., 1983) is a secondary structure assignment tool[11] that analyses ATOM records from a PDB file and, based primarily on hydrogen bonding patterns, gives a clear, unambiguous description of the secondary structure. DSSP is discussed further in chapter 7.

Most quantitative pairwise comparisons of tertiary structure rely on some form of **superposition** of the atomic coordinates of conserved structural elements, as derived from crystallographic data (ATOM records). Most methods, such as *PDBeFold*[12] (Krissinel et al., 2004) rely on a pre–definition of the corresponding residues in a pair of structures, such as the one provided by DSSP. The program SHP (Stuart et al., 1979), however, is a general approach that does not rely on secondary structural elements at all. Instead, it relies on matching Cα coordinates between two

[11] DSSP is not a secondary structure predictor!
[12] Formerly known as "SSM."

structures using a rotational search procedure coupled with a dynamic programming algorithm to define and refine the list of correspondences before calling the program MATFIT (Kabsch, 1976; McLachlan, 1982) to fit the two structures together as well as possible, that is, to perform the actual superposition.

As MATFIT is used heavily in this thesis, a short introduction follows. The purpose of MATFIT is to align (superpose) and compare two matrices (Kabsch, 1976; McLachlan, 1982; Ramsay, 1990). In the case of SHP and the work in this thesis, the matrices are sets of (x, y, z) Cα coordinates. The alignment, or superposition, consists of applying rotations and translations to one matrix until it is as similar to the second matrix as possible. The difference between the two matrices at the end of the superposition is the root–mean–square difference (RMSD). Alternatively, the RMSD can be considered an error, RMSE, where one matrix is *expected* to superpose onto the other. The terms RMSD and RMSE are used in their respective contexts in this thesis.

Whilst the inner workings of MATFIT are beyond the scope of this thesis, it is important to note that MATFIT can provide either a fast or a slow alignment; the latter is more exact (thereby generating more exact RMS differences or errors) but requires the computation of a rotation matrix and translation vector, which is more time–consuming. The fast method negates this computation, but the resulting RMS values are more error–prone. Thus, for comparing very similar structures ($\leq$ 0.3Å RMS), the fast computation is not useful (see chapter 7).

1.7 Chapters in this thesis

Comprehensive usage of the data in the PDB to inform *in silico* structural biology techniques is complex due to the size of the data set and differences in quality of data and annotation methods. The work presented in this thesis focuses on efficient

bioinformatics methods for using the PDB in three distinct areas of structural biology, namely crystallographic unit cell matching, disorder prediction and structural fragment clustering.

All of the software projects developed during the course of the research described here rely on a bespoke method of farming key information out of the PDB. This pipeline, called PRAXIS, is discussed across all chapters as it was modified and enhanced throughout these projects.

Chapter 2 explores crystal matching and identification of families in the PDB based on unit cell and sequence information. The work discussed in this chapter has led to a software tool called *Nearest–cell* that is publicly available and can be easily incorporated into bioinformatics pipelines.

Chapter 3 discusses disorder prediction techniques and highlights one predictor, RONN, which was improved to use the latest PDB data. Section 3.7 in particular discusses one method of partitioning the PDB to try and improve prediction scores.

Chapters 4 and 5 build on section 3.7 using two other methods of partitioning, clustering and sampling the PDB for disorder prediction. A new and better predictor, called *MoreRONN*, has been developed as a result of this work.

Chapter 6 chronicles attempts at predicting disorder using *MoreRONN* and *ordered* regions in the PDB, including stumbling blocks that were encountered due to hardware limitations.

Chapter 7 records the development of a new method of clustering short structural fragments in the PDB and some initial analysis and application for this work. The clustering algorithm that was developed for this project has applications for disorder (and order) prediction as well.

Chapter 8 contains an overarching discussion with some concluding words on the usability and expansibility of the work that was conducted.

Chapter 2

Locating Crystal Matches in the PDB with *Nearest–cell*

2.1 Motivation

X–ray crystallography is the primary method of determining protein structure at (near) atomic resolution (see chapter 1). Synchrotron facilities, such as the Diamond Light Source in the UK, have dedicated a large amount of resources and funding to developing cutting edge X–ray beamline facilities for the elucidation of protein structure. At such large facilities where effective use of beamtime for macromolecular crystallography (MX) directly impacts throughput, the efficiency of the MX experimental process is subject to scrutiny and constant improvement. Automated tools such as X–ray data analysis pipelines, which form a core component of MX experimentation, are continuously being developed and evolved (Bahar et al., 2006; Keegan et al., 2007; Panjikar et al., 2009; Winter et al., 2011).

Despite the success of MX, however, there are still challenges upstream, that is, in protein production and crystallisation. This is especially true for challeng-

ing targets such as protein complexes, membrane proteins, or proteins that require chaperones. One risk is that incomplete or incidental protein species may be crystallised. In such cases, it would be useful for the beamline scientist to compare the new crystal data as it is collected against the PDB to determine whether an incorrect target has been crystallised.

This chapter describes the development of a software tool, *Nearest–cell*, which is able to compare a preliminary indexing solution against known crystal parameters in the PDB and retrieve the nearest matches. *Nearest–cell* has been embedded into the *fast_dp* MX analysis pipeline at Diamond. It runs as a separate service and has been described in Ramraj et al. (2012). Initially, this project was designed as an introductory programming exercise to become familiar with the PDB, but quickly proved its usefulness for structural biologists and beamline scientists, and was therefore implemented for general use at Diamond and as a public web service.

Nearest–cell retrieves closest matches to the crystal currently under examination from a local database copy of the PDB which is pre–curated by a custom data pipeline called PRAXIS (see section 2.3). Given that the PDB contains just over 100,000 total records (http://www.pdbe.org; Bernstein et al., 1977) at the time of writing, of which ~85,000 are crystal depositions, this represents a significant amount of data that needs to be sifted through in a short amount of time, given that speed is as important as quality of results in a bioinformatics pipeline. In order to display the results cleanly and reduce clutter in cases where there are hundreds of crystal matches, *Nearest–cell* utilises a custom clustering algorithm called "Family Clustering". This algorithm uses both unit cell and sequence data to group the results, and is discussed in section 2.4.2.

2.2 Crystal morphology and the reduced *P1* cell

Crystals are tightly–packed 3D structures made up of repeating copies of the protein or complex of interest, typically in an equal volume of solvent. The smallest portion of a crystal that can be repeated by translation alone to form the full crystal is called the **unit cell**. A unit cell is defined by three axis lengths (a, b, c) and three angles between the axes (α, β, γ) and can contain more than one copy of the protein (see figure 2.1). In contrast, the **asymmetric unit** is the smallest portion of the unit cell that can be repeated in 3D space through translations, rotations and inversions[1] to reconstitute the whole crystal. The asymmetric unit is a simple fraction of the unit cell, the ratio of volumes depending on the number of crystallographic symmetry operators in the given crystal form.

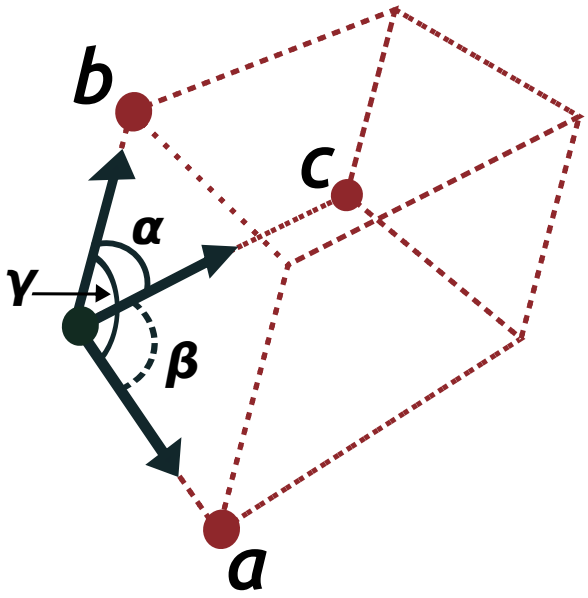


Figure 2.1: A unit cell, showing axis lengths and angles.

Crystallographic symmetry can be derived from the **space group**. In three dimensions, there are 230 distinct types of symmetry; however, since proteins are stereo–active, mirror image and inverted proteins are not found[2]. This reduces the

[1] For non–enantiomorphic species.

[2] Note, however, that at the time of writing, the PDB contains 5 synthetic left–handed structures

number of available space groups for normal protein crystals to 65. The simplest and most general space group contains no crystallographic symmetry and is known as *P1*; all other space groups describe a higher level of symmetry. The definition of the origin for a particular space group is arbitrary, but is conventionally defined by reference to the symmetry operators and is usually one of the points of highest symmetry. Since both the origin and frame of reference change as the space group changes, the comparison of unit cells is made much more complex. Software packages such as PHENIX (Adams et al., 2010) can transform unit cells in higher symmetry space groups to equivalent *P1* cells.

In addition to crystallographic symmetry, which is global, local non–crystallographic symmetry can also occur when there is more than one copy of the protein within the asymmetric unit of a crystal; this concept is not explored in this thesis except in the Family Clustering Algorithm (see chapter 2).

Growing crystals is a trial–and–error process which begins with the purification of a protein sample from a given expression system (human, yeast or bacteria, usually). Different crystallisation conditions are attempted, and successfully grown crystals are then exposed to a (usually) monochromatic X–ray beam. X–ray sources tend to be either small "in–house" systems (*e.g.* rotating anode sources), which are usually of lower resolution, or they are of the synchrotron variety. The latter is a large electron storage ring that can produce very high–intensity X–ray beams from which tunable X–ray wavelengths can be selected, which greatly reduces data collection time and helps in resolving crystal structure at high resolution. Note that many high resolution structures have come from in–house sources, but synchrotrons allow studies on weaker diffracting systems and smaller crystal samples, and allow much faster data collection. The Diamond Light Source is the UK's national synchrotron; some of the work in this thesis has been implemented there as

in the '*P -1*' space group. These structures are not considered further.

part of their X–ray data analysis pipeline.

As the X–rays are scattered upon interacting with atoms in the crystal, the diffraction pattern is recorded on a detector as a series of spots. The crystal sample is rotated, and therefore, the diffraction pattern changes as different atom planes come into the path of the X–ray beam. Intense spots on the diffraction pattern are a result of constructive interference; the angle at which constructive interference occurs was first described by William Lawrence Bragg (Bragg, 1913) as follows:

$$n\lambda = 2d\sin\theta \tag{2.1}$$

where n is an integer, λ is the wavelength, d is the space between the planes in the crystal lattice and θ is the angle between the incident ray and the scattering planes. These parameters are described in figure 2.2. Note that the wave hitting the lower atomic plane travels an extra distance of $2d\sin\theta$, as per the equation. As the crystal sample is rotated, many atomic planes interfere with each other, resulting in very sharp peaks surrounded by mostly destructive interference, where little diffraction is recorded.

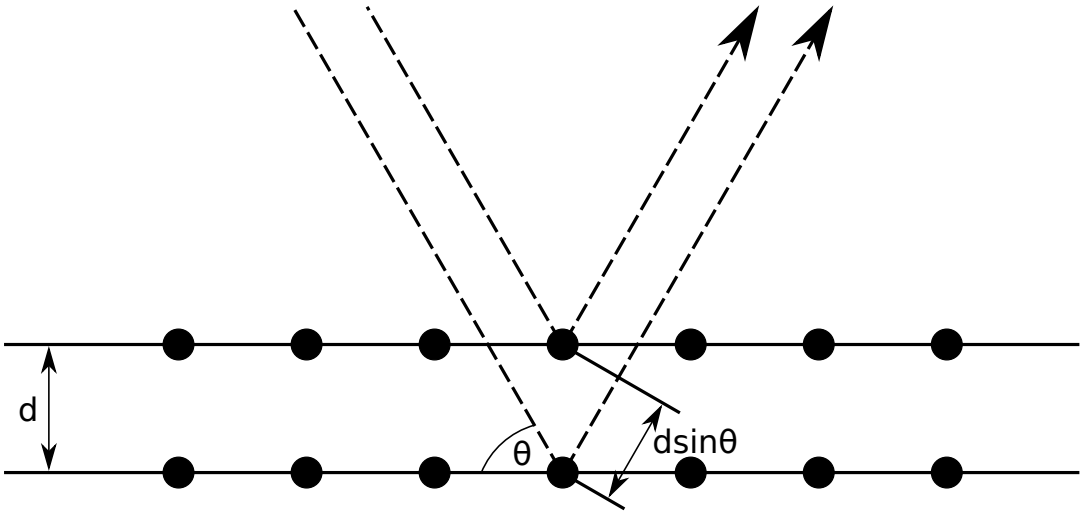


Figure 2.2: X–ray diffraction in a portion of a crystal lattice, describing the parameters of the Bragg equation. Figure taken from Wikipedia (2014).

Structure factors are mathematical functions that describe the amplitude and phase of a wave diffracted from the crystal (summed over each atom in a unit cell) for a given direction characterised by a unique triplet of integers (h, k, l), the Miller index. They can also be calculated by applying a Fourier transformation to electron density derived from an atomic model. Whilst the detector is able to detect amplitude[3], the phase of the diffracted wave must be inferred through either molecular replacement in cases where a similar structure has been solved, or through heavy atom replacement or anomalous diffraction in cases where no reference structure is available. This is known as the **phase problem**.

Once the electron density map is obtained, it can be interpreted as atomic coordinates to build the 3D model. The model is generally built by assigning the amino acid backbone first, followed by an estimation of the side chains. This preliminary model is then re–refined iteratively against the experimental data until no further improvements are obtained. At this stage, it is possible to **validate** the quality of the model, and indeed, many quality metrics are deposited into the PDB along with the structure itself.

2.2.1 Superposition and RMS value

Given that *P1* cells are all "standardised" insomuch as there is no symmetry, the unit cells can be superposed to determine how similar or different they are without worrying about different symmetry conventions. This distance is reported as an RMS value; the question of similarity versus difference is a semantic one – if one is comparing two unit cells, the RMS value is a difference (RMSD), if one is measuring how the two cells deviate, the RMS value is an error (RMSE). Given that tertiary structure is responsible for how a crystal forms, very similar proteins often

[3]Brighter spots appear darker on the diffraction image. The intensity of a spot is proportional to the square of the structure factor amplitude.

have similar unit cells, and therefore, small RMS differences. Figure 2.3 graphically demonstrates how a superposition works.

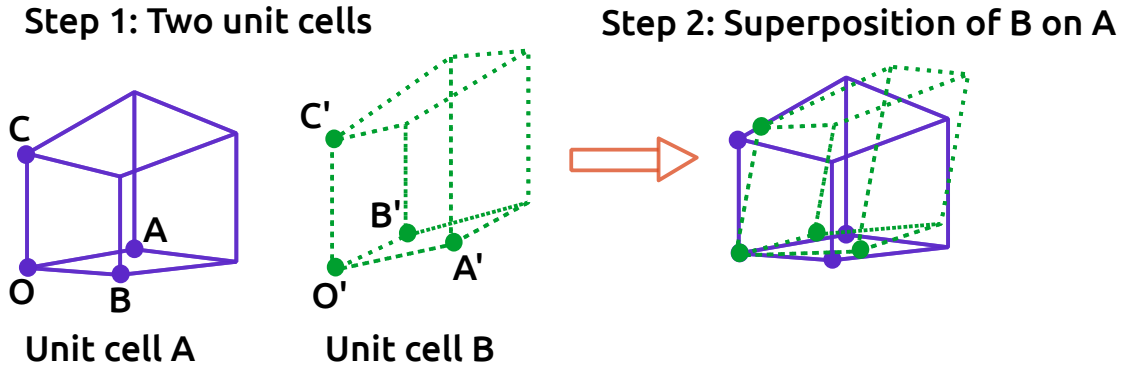


Figure 2.3: Superposition of unit cell B on unit cell A. Unit cell B is drawn with dotted lines for clarity. Note that the origins (O and O') are superposed perfectly.

One software package that quickly performs superposition and RMS calculations is MATFIT, which is a FORTRAN implementation of the iterative superposition method described in McLachlan (1972) and refined in Kabsch (1976, 1978). Basically, two sets of atomic position vectors are superposed as well as possible; then, the rotation matrix and translation vector to transform one position set to the other are calculated, and the residual RMS difference returned to the user. While MATFIT is agnostic in its calculations regardless of what the input vector sets are, the ordering of the sets is important since this defines an equivalence relationship. In this application, the *P1* unit cell vectors are used to generate three non–redundant vertices of the unit cell that, along with the origin, are provided as input to MATFIT (Ramraj et al., 2012).

The FORTRAN MATFIT code was isolated from the CALPHA source code (Esnouf, 1997) and compiled as a standalone shared library for integration into *Nearest–cell*. MATFIT is applied to a very different superposition problem in chapter 7.

2.3 PRAXIS 1.0

The first version of the feature–set implemented in PRAXIS includes: *1*) a PDB update script, which updates local PDB copies from the PDB servers each week; and *2*) an XML parser that mines each updated PDB entry and stores relevant CRYST1 data into a PostgreSQL database in order to easily run queries and filter the available data.

PRAXIS exists on disk as a combination of BASH scripts to synchronise PDB data and C++ code to handle XML parsing and database updating. The whole process is run each week as a UNIX *cron* job on Wednesday mornings, to coincide with the weekly PDB updates. The workflow is shown in figure 2.4 and each step is discussed below in the text.

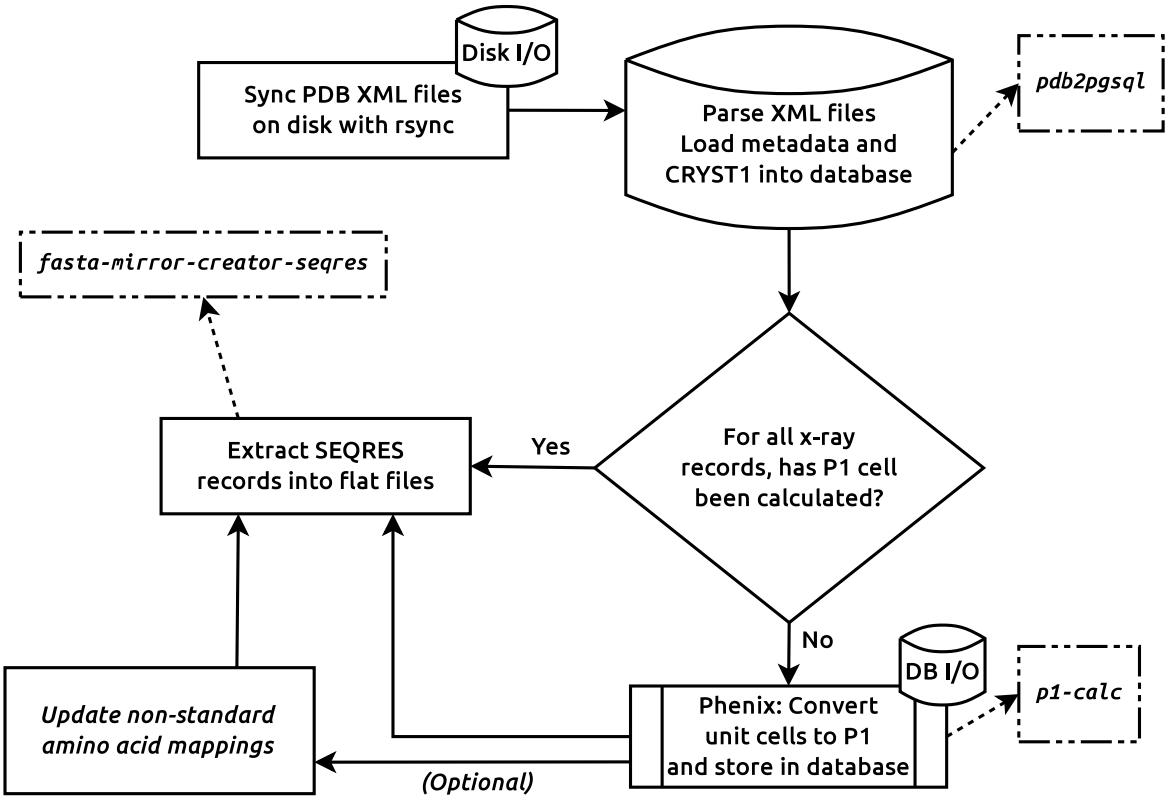


Figure 2.4: PRAXIS workflow. The dotted lines connect the task to be performed with the name of the actual application that performs it. PHENIX is invoked externally to convert cells to *P1*. Optionally, non–standard amino acids (such as selenomethionine) can be reduced to their standard counterparts as the SEQRES sequences are being constructed.

2.3.1 Synchronising local PDB mirror

The local PDB mirror (in XML format) is updated using an *rsync* BASH script provided by the EBI (personal communication with Sameer Velankar and Jose Dana at EBI). The *rsync* logs, which contain information on entries that were added/changed or deleted, are used subsequently in order to perform downstream tasks only on the files that were changed. This prevents unnecessary work on files that do not need updating and therefore speeds up the pipeline.

2.3.2 XML parsing with `pdb2pgsql`

Due to the hierarchical nature of XML files, XML parsing is much slower than reading data from a fixed–column flat file such as the legacy PDB format. Since certain data (such as CRYST1 records) would be required repeatedly, it was decided that this data should be parsed during PRAXIS updates and stored to a database, where it can be retrieved much faster. The XML parsing application (called `pdb2pgsql`) extracts key information from each updated file and stores it to a PostgreSQL database. The *rsync* logs from the synchronisation step are used to inform which entries need to be inserted or deleted from the database. The powerful XPath XML parsing language (Clark et al., 1999) was used to extract the desired data. XPath querying is integrated into `libxml-2.0`, which is the XML parsing library of choice for PRAXIS.

Table 2.1 shows relevant examples of XML structure and the corresponding XPath tag. It illustrates differences in XPath queries depending on where the data is located in the XML document, and how it is represented. XPath queries follow a convention similar to the UNIX Filesystem Hierarchy Standard (FHS), beginning at the root, or outermost tag, and nesting inwards. Tag attributes which are inside a tag begin with “@", as shown in the first row of table 2.1. Comments are nested between “(:" and “:)" tags and are not processed by XML parsers.

XML Structure	XPath Query

`<datablock datablockName="1VRT">`
`<...>`
`</datablock>`
	`/datablock/@datablockName`
	
	`(: Attributes begin with "@" :)`
	`(: Returns "1VRT" :)`

`<datablock datablockName="1VRT">`
`  <cellCategory>`
`  <cell>`
`  <length_a>`
`  141.8`
`  </length_a>`
`  </cell>`
`  </cellCategory>`
`</datablock>`
	`/datablock/cellCategory/`
	`  cell/length_a/`
	
	`(: Returns the value 141.8 :)`

Table 2.1: Example XML snippets and corresponding XPath queries to retrieve the values highlighted in bold face. The XPath comments in between "(:" and ":)" tags describe what is returned as a result of running the query.

The full list of XPath queries is shown in Listing 1.

```
(: PDB ID (this is an attribute of the <datablock> tag :)
/datablock/@datablockName

(: Description :)
/datablock/structCategory/struct/pdbx_descriptor/

(: Title (sometimes a description is stored here) :)
/datablock/structCategory/struct/title/

(: Organism from which sample was obtained :)
/datablock/entity_src_natCategory/entity_src_nat/
                                  pdbx_organism_scientific/
/datablock/entity_src_genCategory/entity_src_gen/
                                  pdbx_gene_src_scientific_name/

(: Method (X-RAY, NMR et c.) - an attribute of the <exptl> tag :)
/datablock/exptlCategory/exptl/@method

(: Unit cell parameters :)
/datablock/cellCategory/cell/length_a/
/datablock/cellCategory/cell/length_b/
/datablock/cellCategory/cell/length_c/
/datablock/cellCategory/cell/angle_alpha/
/datablock/cellCategory/cell/angle_beta/
/datablock/cellCategory/cell/angle_gamma/

(: Space group :)
/datablock/symmetryCategory/symmetry/space_group_name_H-M/

(: Experiment Resolution (high and low) :)
/datablock/reflnsCategory/reflns/d_resolution_high/
/datablock/reflnsCategory/reflns/d_resolution_low/
```

Listing 1: XPath queries to extract data from each PDB XML entry for storage in the PRAXIS database. Comments are denoted in between "(:" and ":)" as per XPath standard, and are included here for clarity.

2.3.3 *P1* cell and volume pre–calculation with `P1-calc`

Once the database is updated, an SQL query farms out all entries without corresponding pre-calculated *P1* cells, and PHENIX is invoked on the unit cell and space group for each entry. The calculated *P1* cell is then stored in a separate table within the database. These three tasks are encapsulated into the programme `P1-calc`.

P1 cell volume is also calculated, as this volume can be used as a rough cut-

off when comparing *P1* cells, in order to filter down the number of calculations performed by MATFIT. The cell volume is equivalent to the scalar triple product (STP) of the three vectors, described below.

$$\text{STP} = \vec{a} \cdot (\vec{b} \times \vec{c}) \tag{2.2}$$

This is equivalent to the absolute value of the determinant of a 3 x 3 square matrix of the components of vectors $\vec{a}, \vec{b}$ and $\vec{c}$, as shown in equation 2.3.

$$\text{STP} = \begin{vmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{vmatrix} = a_x \begin{vmatrix} b_y & b_z \\ c_y & c_z \end{vmatrix} - a_y \begin{vmatrix} b_x & b_z \\ c_x & c_z \end{vmatrix} + a_z \begin{vmatrix} b_x & b_y \\ c_x & c_y \end{vmatrix} \tag{2.3}$$

This formula is implemented in the PRAXIS C++ code, having been converted from its Fortran counterpart, which is part of the MATFIT suite.

2.3.4 SEQRES extraction with `fasta-mirror-creator-seqres`

As the Family Clustering algorithm was being developed, it became necessary to have quick access to the SEQRES sequences of each PDB file. As hardware limitations on the database server made it impossible to store them in PostgreSQL format, a C++ program was written to parse out the SEQRES sequences from each XML file and store them in FASTA-delimited format as a separate mirror directory on disk.

The program `fasta-mirror-creator-seqres` parses the XML file to retrieve the amino acid sequence three–letter codes, which are then translated to one–letter codes with reference to the internal table of amino acids (see section 2.3.5) to build the SEQRES sequence. The sequence is stored in FASTA–delimited format on disk.

While it may seem redundant to create another file–based mirror on disk, once again, it is noteworthy that XML parsing is much slower than simply reading textual

data from disk, and so this was a necessary feature which aided greatly, not only in *Nearest–cell*, but also in the development of *MoreRONN* (see chapter 5). Furthermore, storing sequence data directly in the PRAXIS database was impossible due to limited disk space on the database server.

This component acts only on added or changed PDB entries.

2.3.5 Non–standard amino acids

Non–standard amino acids occur usually as a result of post–translation modifications. They pose a problem for us as they are universally reduced to 'X' as the one–letter code in SEQRES sequences. As this represents a very large loss of data[4], especially in disorder studies, an amino acid mapping script was implemented to retrieve standard mappings for all non–standard amino acids, which are then stored in the PRAXIS database as a separate table. This script relies on a web service developed and put in place at the EBI for our specific use case by Jose Dana and Sameer Velankar, to whom we are very grateful for this service. As mappings do not change, it was only necessary to retrieve this data once, with an update every few weeks or so to fix or add any new mappings that may have arisen.

This step is marked as "optional" in figure 2.4 as it is run manually every few weeks.

2.3.6 A note on semantics: "deleted" versus "obsoleted" entries

When PRAXIS encounters an obsoleted entry, all references to it are removed in the database and all generated flat–files and XML mirror files are purged from disk. Obsoleted entries are identified by the word "Deleted" in the *rsync* log because a superseded entry in the PDB corresponds to two *rsync* operations — a new file

[4]Rather than "marking the spot", 'X' obfuscates useful information.

is created and the old one is deleted. In terms of PDB *modus operandi* however, deleted entries are still preserved, even if obsoleted.

2.4 Design and Development of *Nearest–cell*

Nearest–cell is written entirely in C++, although it integrates the MATFIT Fortran library for performing the superposition and it also depends on PHENIX (which is written in C but provides access and bindings in Python). While it is command-line driven, a Python CGI web service front end was developed in order to enable public use. The algorithm operates as shown in figure 2.5 (adapted from Ramraj et al., 2012).

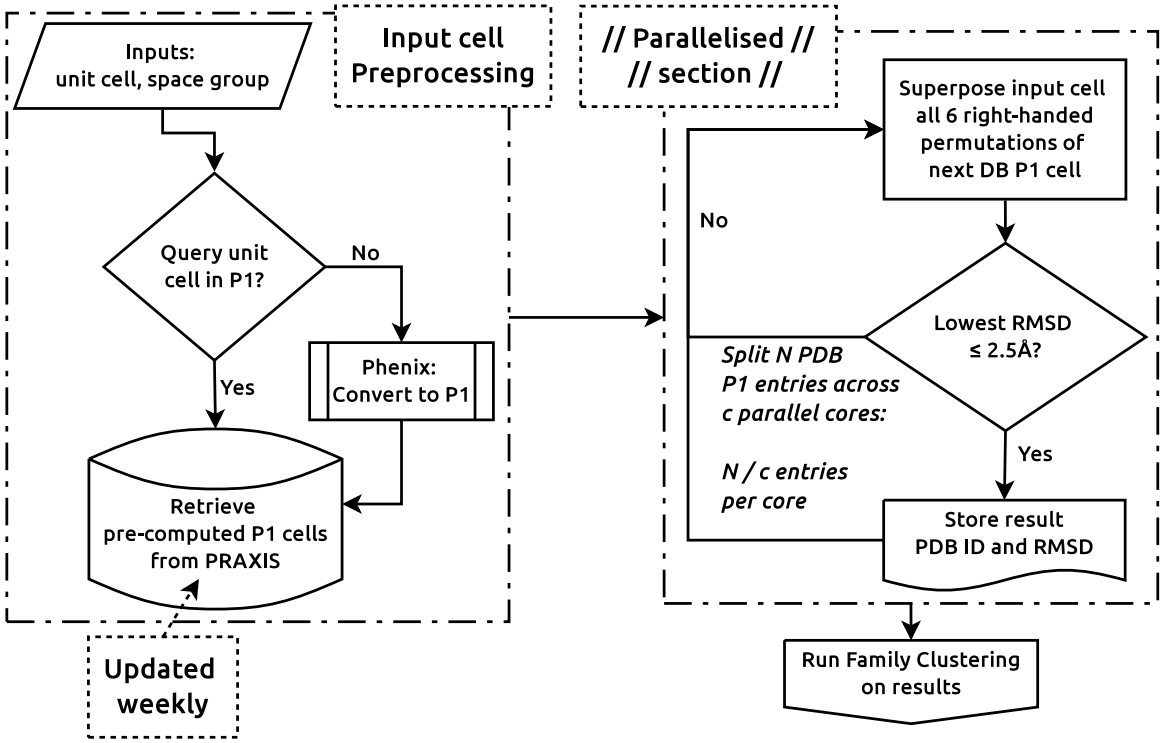


Figure 2.5: Workflow for *Nearest–cell*. If the space group is not *P1*, the input cell is reduced using PHENIX. The pre–calculated *P1* cells are retrieved from PRAXIS. Superposition is performed with MATFIT as discussed in the next section.

2.4.1 Superposition with MATFIT

In order to superpose two unit cells, they must first be decomposed into the component cell–edge vectors. *Nearest–cell* includes a C++ implementation of Fortran code from CALPHA (Esnouf, 1997) to perform this parameterisation. Thus, the unit–cell parameters $(a, b, c, \alpha, \beta, \gamma)$ are transformed into $(a_x, a_y, a_z), (b_x, b_y, b_z)$ and (c_x, c_y, c_z), describing the three component vectors $\vec{a}$, $\vec{b}$ and $\vec{c}$. These component vectors serve as part of the input to MATFIT.

In *Nearest–cell*, four coordinates are superposed with MATFIT, namely $\vec{o}, \vec{a}, \vec{b}$ and $\vec{c}$, where $\vec{a}, \vec{b}$ and $\vec{c}$ are broken into their components as specified above, and $\vec{o}$, which represents the origin $(0, 0, 0)$. MATFIT then superposes one cell onto the other and reports the RMS difference of the closest match while keeping the origin fixed (Ramsay, 1990). The database *P1* cell is then rotated through all right-handed permutations, running MATFIT each time (see figure 2.6). If the minimum of all six RMS differences is less than 2.5 Å, the database *P1* cell is included in the results.

$$\left(\vec{o}, \vec{a}, \vec{b}, \vec{c}\right) \leftrightarrow \left(\vec{o}, \vec{c}, \vec{a}, \vec{b}\right) \leftrightarrow \left(\vec{o}, \vec{b}, \vec{c}, \vec{a}\right) \leftrightarrow \left(\vec{o}, -\vec{a}, \vec{c}, \vec{b}\right) \leftrightarrow \left(\vec{o}, \vec{b}, \vec{a}, -\vec{c}\right) \leftrightarrow \left(\vec{o}, \vec{c}, -\vec{b}, \vec{a}\right)$$

Figure 2.6: The six right–handed permutations for unit cell rotation. Note that the origin is always fixed; the other three axes are rotated to form the permutations.

2.4.2 Family Clustering Algorithm

In situations where the input cell does not have many hits against the PDB, the results are of a manageable size. However, if the input cell is common, such as for myoglobin or lysozyme, there are numerous depositions in the PDB of similar crystal structures. In order to visualise the results effectively, a method of grouping large numbers of results was required. The Family Clustering algorithm was developed to address this need, and it works by grouping database hits by their **sequence** similarity. The algorithm is described in figure 2.7.

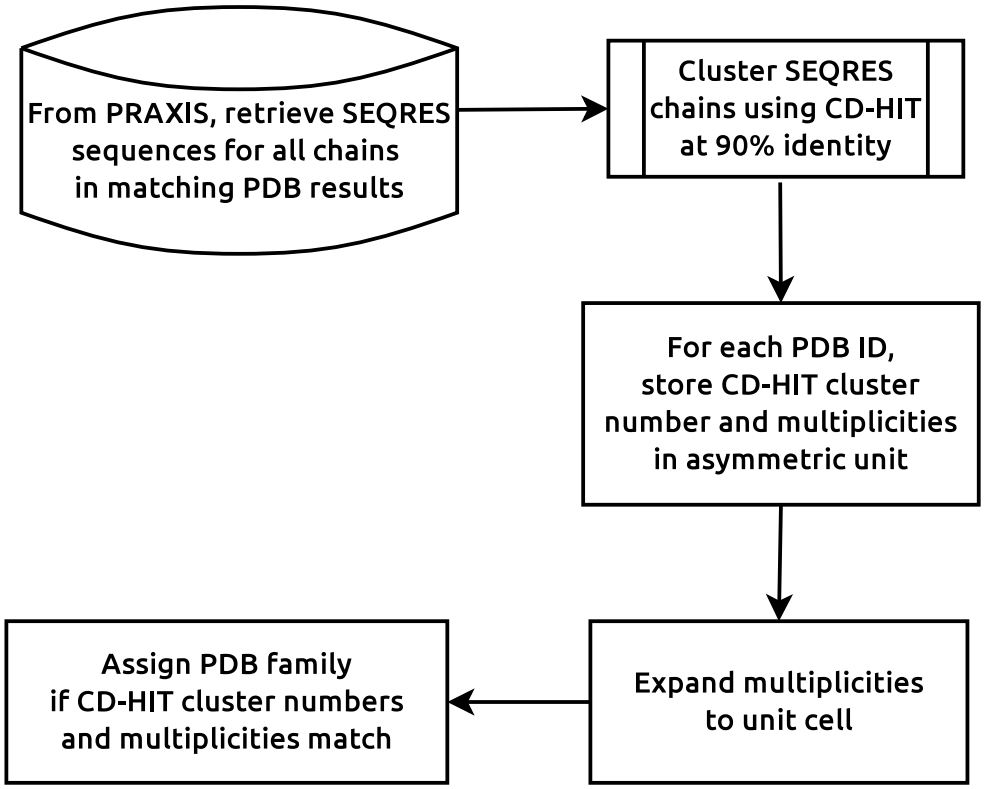


Figure 2.7: The Family Clustering algorithm. Family representatives are sorted by increasing RMSD (best hits appear first). Multiplicities are stored in PRAXIS for quick access, but they are available within PHENIX. The CD–HIT version used in this algorithm was modified as discussed in text.

Family Clustering reconciles sequential and structural similarity and through its chain-grouping mechanism, ensures that integrity is preserved at the PDB record level, rather than at the sequence chain level. It uses chain multiplicity to determine how many copies of the asymmetric unit are present in the unit cell. Multiplicity is an invariant that is based on the space group; PHENIX maintains a list of multiplicities for each space group. These multiplicities were manually added to PRAXIS for quick and easy retrieval without needing to depend on PHENIX each time. When the CD–HIT step is complete, the chains are multiplied by the corresponding multiplicity value for the original space group. Two PDB entries are placed into the same family only when the CD–HIT cluster number *and* the chain copy numbers match.

2.4.2.1 Debugging and modifying CD–HIT

An introduction to CD–HIT is provided in section 1.6.1.3.

In order to use CD–HIT for Family Clustering, it was necessary to alter its cluster output slightly to include full FASTA headers instead of just the first few characters as in the original CD–HIT output. This allowed parsing of the CD–HIT output file using previously written FASTA parsers.

During this modification process, a small bug was fixed in the CD–HIT source code, in order for the programme to correctly output the amount of time taken for an analysis. A patch was submitted and was integrated into the CD–HIT source code[5]. The new binary executable is called `cdhit-modified` to differentiate it from the stock binary (`cdhit`) and to allow the two versions to co-exist on a single machine. All custom software built as part of this thesis that require CD–HIT use `cdhit-modified`.

2.4.3 Web Service

The web service is written in Python CGI and invokes the *nearest–cell* C++ programme. *Nearest–cell* was modified to output families in both plain text and HTML format to standard output. The web service collects this output and builds a web page.

The web page presents results in condensed format with families denoted by representative entries. Javascript is used to allow families to be expanded and collapsed, and by default, they are presented in collapsed format. This allows users to quickly look at representative hits without being inundated with the full result set.

[5] `http://code.google.com/p/cdhit/source/detail?r=dd555b4174ff48acb97b0cc33a047647406ef785`

2.5 Results and Discussion

Nearest–cell is offered as a publicly available web service[6] and has been integrated into the *fast_dp* pipeline (Winter et al., 2011) at the Diamond Light Source. Figure 2.8, reproduced from Ramraj et al. (2012), shows a sample result with *Nearest–cell* running as part of *fast_dp* at Diamond.

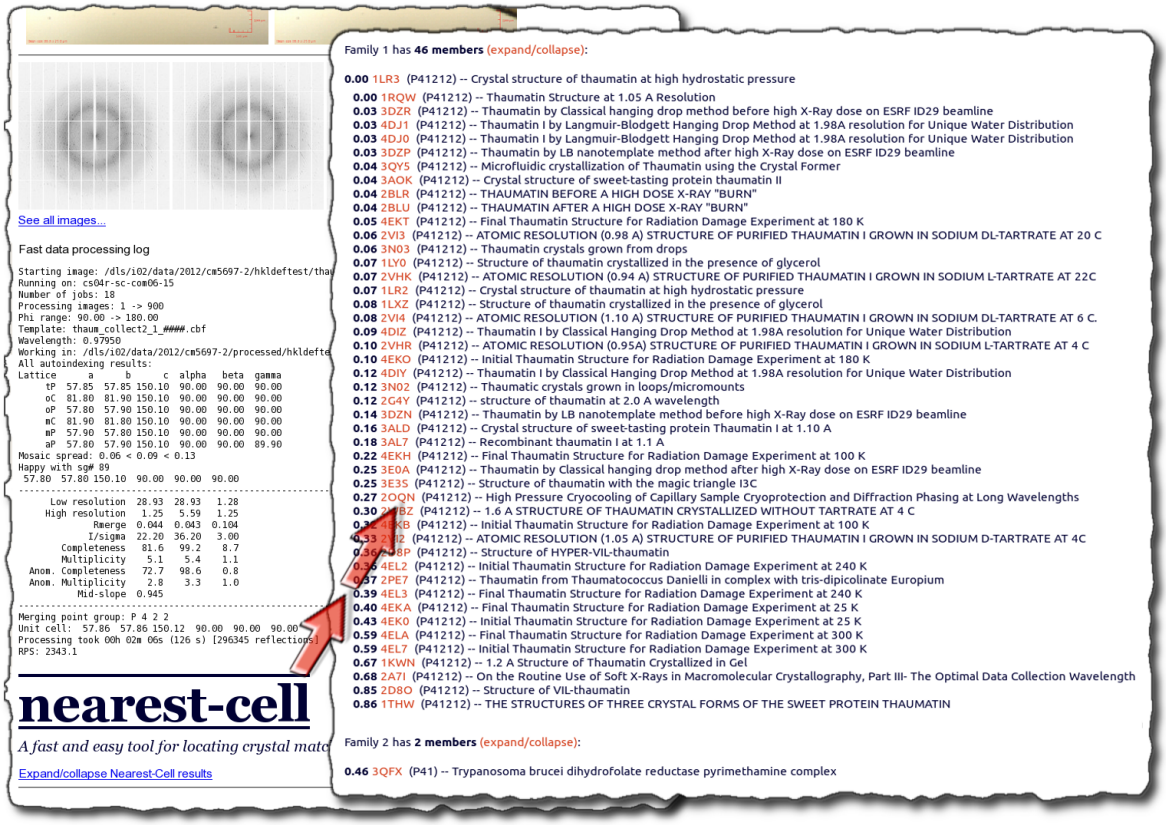


Figure 2.8: Screenshot showing results from the *Nearest–cell* web service. Taken from Ramraj et al. (2012).

For each family, the number of hits is shown, with the ability to expand or collapse the results. When the result is expanded, the family members are shown with their RMSDs, PDB IDs, space group and title (taken from the PDB). Furthermore, the PDB ID is a hyperlink that redirects to the entry's page on the PDBe website (`http://www.pdbe.org`). Family 1 has been expanded to show the PDB hits within;

[6] `http://www.strubi.ox.ac.uk/nearest-cell/nearest-cell.cgi`

note that 1LR3 is the representative, and is a perfect match (RMSD = 0.0) to the input. Note also, that most of the members of family 1 are in the same space group and are crystal structures of thaumatin. One could conclude from this result that the input crystal is thaumatin as well, or a closely–related structure[7] without knowing anything more than the unit cell dimensions and a putative space group. Therefore, whilst *Nearest–cell* can be used to survey relationships between crystals in the PDB on a broad scale, it can very quickly reveal whether an incorrect protein product (*e.g.* an impurity) has been crystallised in an experimental setup.

At Diamond, *Nearest–cell* runs immediately after the primary indexing solution[8], which means it can run ahead of slower refinement steps and can provide feedback further upstream of the final data.

Nearest–cell runs very quickly. Even on a single core on a modern computer[9], where the parallelised section shown in figure 2.5 is limited to single–threaded operation, it takes just over **one second** to compute when the input cell is provided in *P1*[10]. If the input cell is specified at a higher symmetry, the call to PHENIX to reduce the cell to *P1* takes approximately five seconds and is therefore the rate–limiting factor. Nevertheless, even with the call to PHENIX, this quick execution speed makes it an ideal candidate for bioinformatics pipelines such as *fast_dp*, and indeed, speed was a very important factor during the algorithm design process!

Whilst similar approaches to *Nearest–cell* do exist (Allen et al., 1979; Mighell, 2002), McGill et al. (2013) propose an alternate method that relies on the $\boldsymbol{G^6}$ representation of the reduced cell (Rigault et al., 1980; Andrews et al., 2012), which is different from the Niggli cell reduction algorithm (Santoro et al., 1970; Mighell et al., 1980; Grosse-Kunstleve et al., 2003) implemented in PHENIX and proposed

[7]Based on the fact that similarly–structured proteins will crystallise in a similar manner.

[8]The "first guess" about the crystal parameters, before slower refinement steps and analysis are conducted.

[9]We assume the average processor clock speed of today's computer to be 2.4 GHz.

[10]Note that this involves superposing the input cell on *every* known *P1* cell in the PDB!

and refined by Buerger (1957) and Krivy et al. (1976), respectively. The paper by McGill et al. (2013) suggests that *Nearest–cell* is somewhat imprecise and prone to generating false–positives. Firstly, a reduced cell will be unique, but under special circumstances, can be unstable; certain variations in input cell dimensions can lead to abrupt changes in the reduced cell's angles[11]. The second, more philosophical argument is that a unit cell is a representation of a crystal lattice, but that lattices can be represented equally well by infinite different unit cells, and therefore it is more useful to search for matches to the actual lattice obtained by the indexing solution. Their method, SAUC, along with WebCSD (Thomas et al., 2010), utilise this alternate representation of the unit cell to essentially reduce the number of false positives that could be generated by the *Nearest–cell* method and to find unit cells that represent the same crystal lattice, but that may be obscured by the simplicity of the *P1*–cell matching approach. Furthermore, methods that use $\boldsymbol{G}^6$ cells are apparently more stable against experimental error, and are more forgiving than legacy methods (Andrews et al., 1980; Milne et al., 1980). However, *Nearest–cell* was designed with speed and accuracy in mind; the loss of precision with the *P1* cell method is not a weakness when newly–indexed cells are matched as they come off the beamline, and for small cells, more comprehensive methods may not add any new information (Mighell, 2001). The use case which *Nearest–cell* is designed to answer revolves around finding any close match, rather than all close matches with high precision, and for this purpose, *Nearest–cell* is the fastest solution. The Family Clustering Algorithm makes the output results more user–friendly.

[11]For interest's sake, the solution suggested by Andrews et al. (1980) is to use just cell lengths for comparisons; this means two reduced cells will match as long as their lengths are within a specified tolerance, even if the cell angles are vastly different.

2.5.1 Caveats in PDB data

When mining data out, it was immediately evident that, while the PDB formats themselves are standardised, individual depositions varied in where data were recorded. The problem was especially glaring for space group conventions, as deposited data often varied from PHENIX naming scheme. This required a translator to be written which mapped non–standard space groups in the PDB to their PHENIX counterparts; this work was done manually and is hard–coded into *Nearest–cell*. Whilst hard–coding is not an ideal long term solution, it at least allows *Nearest–cell* to operate on all known inputs. Table 2.2 shows the manual non–standard space group mapping.

PDB SG	**Standard SG**	**PHENIX SG**
A1	P1	*P1*
B2	B112	C2
C1211	C121	C2
C21	C121	C2
I1211	I121	C2
I21	I121	C2
P21212A	P21212	P21212
R3	R3	R3:R
H3	R3	R3:H
R32	R32	R32:R
H32	H32	R32:H
*C4212**	*C4212**	*P422**
*F422**	*F422**	*I422**

Table 2.2: Table of non–standard space groups found in the PDB, with their mappings to standard space groups and space groups that PHENIX is able to manipulate. The last two entries do not map directly; rather, they require the unit cell axis lengths to be altered as discussed in text.

The last two, C4212 and F422 map to P422 and I422 (respectively) in PHENIX, but this mapping requires the unit cell lengths, a and b to be changed by a factor of $\sqrt{2}$. To convert a unit cell in C4212 to P422: $a = a/\sqrt{2}$ and $b = b/\sqrt{2}$. To map F422 to I422: $a = a \times \sqrt{2}$ and $b = b \times \sqrt{2}$. When these space groups are encountered,

Nearest–cell performs the conversion ahead of reduction to *P1*.

Note also that PHENIX often uses space group naming conventions that differ from the standard, the clearest example being the standard R3 space group denoted as "R3:R" within PHENIX. Furthermore, H3 maps directly to its R counterpart (R3:H). A similar argument can be made for R32 and H32. In both cases, the H counterparts are hexagonal representations of the trigonal (R) space groups; in the latter, the unit cell is $1/3$ the size. PHENIX performs the unit cell conversions in these cases automatically.

2.5.2 Embedding Fortran MATFIT as a function library

Embedding MATFIT into C++ code creates conceptual and implementation hurdles. Conceptually, two-dimensional arrays in Fortran are indexed column-first, while the reverse is true in C++. Preprocessor macros were written in C++ to mimic Fortran-style arrays.

In terms of implementation and in order to save time, rather than rewriting MATFIT in C++, it was compiled as a shared library and linked into *Nearest–cell*. In order to use the Fortran function correctly, it had to be declared as an external C-style function within the C++ source with careful attention to pointers and memory access.

2.5.3 Philosophy behind the Family Clustering Algorithm

The clustering method was developed when it became apparent that some input unit cells produced too many hits in the PDB to be useful to a crystallographer at a beamline (a myoglobin unit cell produces well over 250 hits). To reduce the number of hits visible to the user, a clustering approach was necessary. After initial investigations into unit–cell based clustering proved unsuccessful, it was decided

that protein sequence could be used instead; there are a myriad different sequence clustering methods.

While CD–HIT was chosen for its speed and intuitiveness, PDB entries with multiple chains or subunits would often have individual components as part of different clusters, which removes the ability to analyse the PDB entry as a whole and makes it conceptually difficult to consider the crystal unit cell as a whole.

The Family Clustering algorithm reconciles sequence and structure to produce families of related proteins with one representative from each family presented to the user. It uses CD–HIT clusters and then checks that chains from different PDB entries are clustered in the same way, even if across multiple families (Ramraj et al., 2012). In order to gain true insight into the actual contents of the crystal, the original space group is used in order to determine the number of copies of a particular sequence. This data is in PHENIX and a Python script was written to extract this data into the PRAXIS database (it was run once, as this information does not change).

Within the command-line interface, the full family can be displayed through an argument to *Nearest–cell*. This option is enabled by default for the web interface, and Javascript on the Python side arranges family members into hidden layers that can be toggled on or off by using the "expand/collapse" option next to a family representative (see figure 2.8). Thus, this clustering method enables results to be grouped in a sensible manner both for casual surveys of the PDB as well as high–throughput science on a synchrotron beamline.

2.5.4 Recent improvements

Subsequent to the writing of this chapter, both PRAXIS and *Nearest–cell* (including the Family Clustering algorithm) were rewritten in Python and the database back–end was switched to `sqlite3`, which creates the database as a single file with-

out any need for a PostgreSQL–style server–client setup. This greatly increases portability and was done to transport the application to Diamond as smoothly and quickly as possible. The web service was also modified to use this new Python version. The architecture, functionality and use cases remain identical, however.

Chapter 3

Disorder Prediction: RONN

3.1 Overview

The elegance of the *Nearest–cell* solution for crystal matching shows that homologous structures will crystallise in nearly identical fashion, and that the reduced P1 cell can both locate PDB matches and reciprocally predict the contents of a crystal to ensure that the desired protein has been crystallised. This is because tight crystal packing promotes maximum intermolecular interaction, which is the driving force for crystal growth; essentially, molecules that fit comfortably together will crystallise the best (Rupp, 2009). "Bumps" in one protein molecule will tend to bury into an accommodating depression in another, and since homologous structures contain similar bumps and depressions, they inform similar crystal growth patterns. By extension, the physics that drives tight crystal packing is confounded by the presence of flexible disordered regions in between stable structural motifs. These regions do not usually adopt a preferential conformation, which can discourage tight crystal packing. When disordered regions are present, packing interactions might be mechanically blocked by the flexible regions (McPherson, 1999), and it becomes necessary to remove the disordered regions. Crystal constructs are often designed

to express only the essential functional domains and not the whole structure, which might contain these disordered linkers or termini. In both cases, it is often helpful to demarcate the disordered regions in order to isolate just the ordered structure, thereby increasing crystallisation success rate and reduce the number of constructs that need to be made and tried (Bergfors, 2009).

A robust understanding of disorder is therefore central to our understanding of proteins and protein–protein interactions (PPIs). However, given that disordered regions are often cleaved to form crystals, they are naturally absent in the final structure solution and cannot be studied directly (Harp et al., 1998). Their ubiquity and evolutionary persistence in both structural and functional applications makes them far more significant than simply a nuisance to crystal growth, because disordered regions and proteins represent an entirely new way of looking at the structure–function paradigm (Wright et al., 1999). This is due to their propensity for adopting transient structure, or guide the folding of the structured regions; thus, the paradigm remains intact but does not fully explain the existence of disorder. The transient structure is of great importance to the proper functionality, binding and recognition of numerous proteins as described in section 3.1.1.

Whilst the impact of disorder on crystallography is a practical one, the impact of disorder on the modern understanding of the structure–function paradigm is more profound and fundamental. Given disorder's importance and the inability to study it directly through crystallography and other biophysical methods, *in silico* prediction is the only way to identify and characterise it. Disorder prediction algorithms are today's approach to a more complete understanding of protein structure and function, and improvement of experimental throughput and success rates of crystallisation (Yang et al., 2005; Dosztányi et al., 2007, 2010).

3.1.1 The Role of Disorder in Biological Processes

Given a disordered region's flexibility, it is often able to bind to multiple partners and features in biological processes. For example, the enzyme calcineurin is activated by Ca2+-calmodulin complexes binding to its disordered region. The flexible disordered region allows the calmodulin complex to enter, and then tightly binds to it (Dunker et al., 2001b). Calmodulin is able to regulate calcineurin due to the latter's disordered region, and promotes calcineurin's specificity and reversibility properties.

Disorder also plays an important role in transcriptional regulation. For instance, the spliceosome protein, p14, has been experimentally shown to contain regions of disorder that directly contribute to the catalytic activity of the splicing factor of which it is part (Vucetic et al., 2007).

Vucetic et al. (2007) go on to describe other cellular components that contain marked disorder regions, including nuclear proteins, chromosomal proteins, cytoskeletal elements and organelles such as the Golgi apparatus and mitochondrion. Interestingly, they present another list of cellular components that contain a high degree of order, and there is no overlap between the two lists, nor is there any trend in the kinds of proteins that are in one list or the other. This seems to indicate that disorder is one of the principal driving forces for functionality in certain proteins, yet classical ordered structures also exist in important cellular roles. Thus, disorder might be viewed as an additional mechanism of gaining functionality which expands our understanding of classical order-based functionality.

Disorder is implicated in the kinetics of a reaction as well. When disordered proteins or regions participate in protein-protein interactions (PPIs), they undergo a mechanical conformational change into ordered states. Known as induced folding or disorder–to–order transition, this process has been revealed to influence reaction rate (Mészáros et al., 2007). This may be due to disorder providing specificity

without too strong of a binding interaction, allowing a shorter entry or exit time for the substrates. Furthermore, short disordered regions can serve as recognition sites for small molecule binding or structural elements on the interaction partner proteins, thereby increasing the propensity for correct binding (Mészáros et al., 2007).

3.2 Disorder Prediction Algorithms

In order to predict and study disorder, we must first define the term and its scope in a computational sense (*i.e.* the properties of disorder that can be exploited by prediction algorithms). Disordered regions are incredibly diverse in their properties, judging by the numerous *in silico* prediction approaches available, and their often different outputs for the same input protein sequence (Obradovic et al., 2003; Vucetic et al., 2003; Dosztányi et al., 2007, 2010). Disordered proteins may adopt vastly different conformations depending on their surroundings and may even adopt order when involved in PPIs (Dunker et al., 2001a). These regions may also display other properties, such as secondary structure conformation, solvent accessibility (or hydrophobicity; Vucetic et al., 2003). While this indicates that a secondary structure predictor could also predict disorder, note that the absence of stable tertiary structure denotes disorder, but no such statement can be made about secondary structure. This is exemplified in cases where disordered regions display transient secondary structure during binding with substrates (Fuxreiter et al., 2004; Dosztányi et al., 2010). Dosztányi et al. (2010) go on to warn that caution must be taken when applying secondary structure or solvent accessibility prediction methods to disorder because these methods are exclusively based on *ordered* proteins, and are therefore attempting to answer a fundamentally different question. Secondary structure prediction can, however, be used to complement other

disorder prediction approaches, as is the case with the PONDR family of predictors (discussed further in section 3.2.1).

At the amino–acid sequence level, disordered regions are often composed of large quantities of small, charged amino acids, the most common being lysine, glutamate, proline, serine, glutamine, arginine and alanine (KEPSQRA), i.e. they are regions of low-complexity (Yang et al., 2005; Esnouf et al., 2006). Conversely, regions rich in tyrosine, tryptophan or phenylalanine usually display stable tertiary structure (Esnouf et al., 2006).

Computational approaches also sometimes subdivide the disorder prediction problem by categorising disorder into different "flavours" ("collapsed", "extended" and "intermediate"), distinguished by function, amino acid composition and where the disorder occurs along the entire protein sequence (N–terminal, C–terminal or in the middle; Vucetic et al., 2003). "Collapsed" disorder resembles the molten globule form of proteins without the hydrophobic interactions needed to form a stable tertiary structure; however, as stated previously, it may contain transient secondary structural elements (Dosztányi et al., 2010). The "extended" form, as its name implies, exists as a linear thread-like structure, while the "intermediate" form is somewhere between the two other forms. While it is true that different disordered motifs may be associated with different biological functions, this categorisation system feels somewhat artificial; for example, it seems unlikely for a disordered region to exist as a free–floating, long, thread–like structure for any appreciable length of time, and Vucetic et al. (2003) themselves acknowledge this.

These different properties of proteins have resulted in over fifty disorder predictors and predictor variants available today (He et al., 2009). However, only a select few are reliable in their prediction, and the best predictors all rely on a well-curated training data set. They do vary, however, in the features and motifs they examine in order to predict disordered regions.

In some cases, as in the PONDR family and IUPred, multiple predictors examining different properties are often bolted together to give one overall prediction. This method helps reduce noise from the individual component predictors and is known as "meta–prediction" (Obradovic et al., 2003; Xue et al., 2010).

3.2.1 A Survey of Selected Disorder Predictors

One common requirement for all disorder prediction approaches is a database of protein sequences and structures, such as the PDB (which contains indirect information about disordered regions even though it is a database of structure), and DisProt, which contains disordered regions longer than 30 residues (Dosztányi et al., 2010). Le Gall et al. (2007) surveyed the PDB and found intrinsic disorder of various lengths, some greater than 70 residues simply by examining the sequence information in the PDB (i.e. which residues were visible in crystals and accounted for and which were not, the latter case pointing strongly to disorder). While some of these cases could point to errors in refinement of the experimental data, overall, this approach is simple and the data is easily available.

There are several methods that attempt to identify disorder using protein sequence information; this is considered to be the classical approach and usually relies on machine learning techniques such as neural networks or support vector machines (SVMs) (Esnouf et al., 2006; Dosztányi et al., 2010). Table 3.1 summarises these approaches, and is adapted from studies by Radivojac et al. (2007) and He et al. (2009).

Method	Publication Year	Description
PONDR	1997-2011	Meta–predictor that uses differently trained underlying predictors for length and location within a sequence. The latest iteration is called PONDR-FIT (2011).
GlobPlot	2003	Relative propensity of amino acid to be ordered or disordered.
DisEMBL	2003	Predicts different flavours of disorder.
DISOPRED, DISOPRED2	2003-2004	Neural network and SVM trained on whole disordered sequences.
FoldUnfold	2004	Measures conformational entropy and interaction energy to predict folded state.
IUPred	2005	Measures inter–residue interactions to predict folded state.
RONN	**2005**	**Neural network trained on functional alignments of known disordered windows.**
FoldIndex	2005	Analyses ratio of charge vs. hydropathy.
The POODLE family	2007	Meta–predictors based on SVMs to predict disordered regions of different lengths.
MoreRONN	**2013**	**Neural network trained on functional alignments against representatives of disordered sequence families.**

Table 3.1: A table of selected predictors and meta–predictors. Each method is fundamentally different in its approach. There are many other meta–predictors that have not been included here since they combine the predictors discussed here. RONN and *MoreRONN* are highlighted since they were the methods used and developed, respectively, in this project.

Disorder predictors such as Globplot (Linding, 2003), the PONDRs (VL-XT, VSL2) (Romero et al., 1997) or FoldIndex (Prilusky et al., 2005) utilise individual amino acid propensities. Propensity is calculated as the difference between existing as a random coil (RC) versus existing as a regular secondary structural element with hydrogen bonds (SS) as shown below in equation 3.1. If RC is greater than SS,

then P is positive and indicates disorder. This calculation is easy to perform, but on the downside, it is a single property that is attempting to explain all of disorder's complexities, which seems rather simplistic (Esnouf et al., 2006; Dosztányi et al., 2010).

$$P = RC - SS \tag{3.1}$$

IUPred (Dosztányi et al., 2005) goes one step further by estimating *pairwise* interaction energies between adjacent residues, the theory being that unfolded disordered regions will have different interaction energy values than globular, folded ones, but IUPred is unable to account for transient order. However, given this approach, this set of algorithms that are based on physical and chemical properties tends to perform well on either fully ordered or disordered sequences (Esnouf et al., 2006).

The RONN predictor on the other hand uses non-gapped sequence alignment between "windows" of amino acid sequences for the known and unknown folding states of protein sequences (known folding states are obtained from the PDB and DisProt and are used to train the algorithm). RONN, therefore, tends to perform better on partially-disordered sequences (Yang et al., 2005; Esnouf et al., 2006).

Meta–predictors such as the PONDR family attempt to capture the strengths of various underlying predictors, wherein each is tuned to a different parameter. The hope here is that the combined meta–prediction will reflect the strength of all underlying approaches. The newest predictor, PONDR-FIT (Xue et al., 2010), is a combination of 6 underlying predictors, including the older PONDRs, as well as FoldIndex and IUPred.

3.3 RONN

The Regional Order Neural Network (RONN) algorithm was developed in 2005 at the Division of Structural Biology at the University of Oxford and is an artificial neural network that has been trained on known disordered sequence motifs (Yang et al., 2005). It forms the basis of the disorder prediction research discussed in this thesis, and is the progenitor of the *MoreRONN* algorithm discussed in chapter 5.

3.3.1 Artificial neural networks: An Overview

The term "neural network" is classically applied to a network or circuit of biological neurons; artificial neural networks, or ANNs, are computational algorithms that simulate neural operation with the use of artificial or simulated neurons. Connections between artificial neurons are modelled as weights, where positive weights are akin to biologically excitatory connections, while negative weights are inhibitory. ANNs consist of one input layer, any number of hidden layers, and one output layer, which forms the "prediction". ANNs are "trained" using a data set that reflects the kind of patterns that the network is designed to predict (Durbin et al., 1998).

RONN is built as a simple single–layer, feed–forward neural network, where information is passed from input layer to the single hidden layer, to the final output. It consists of a training component where it is trained against known **prototype** disordered sequence motifs (derived from the PDB and DisProt, discussed in section 3.3.3), and a prediction component which uses its training to predict regions of disorder in novel amino–acid sequences (Yang et al., 2005). Thus, while other approaches such as PONDR often rely on meta–prediction to improve quality, RONN is trained purely on known disordered primary sequences and uses only this information in its predictions.

Intrinsic to any ANN is a scoring model for assigning edge weights between neu-

ral connections. In RONN, this weight represents the degree of similarity between two disordered sequences. It is calculated using the bio–basis function (BBF), and is defined by Thomson et al. (2002) as follows:

$$f(x,y) = \exp\left(\alpha \frac{s(x,y) - \beta_y}{\beta_y}\right) \tag{3.2}$$

The function $s(x,y)$ returns the BLOSUM62 similarity score between two windows, x and y, where the resulting score is a sum of the BLOSUM62 scores per pair of residues, as defined in equation 3.3 below:

$$s(x,y) = \sum_n \text{BLOSUM62}(x_n, y_n) \tag{3.3}$$

The individual BLOSUM62 scores are shown in figure 1.5.

β_y is the **bias**, or maximum score of the yth sequence, and it is equivalent to $s(y,y)$, since the maximum BLOSUM62 score between two sequences can only occur if the sequences are identical (self–scoring). If the BBF is considered to be a distribution, then the constant α controls the half–width of the curve. Further discussion on α can be found in section 5.3.2, as this constant was tested heavily during the development of *MoreRONN* (see chapter 5).

Since the BBF is used to generate edge weights, the ANN is known as the **bio–basis function neural network** (**BBFNN**; Thomson et al., 2002; Yang et al., 2005).

3.3.2 Measuring predictor performance

The most common metrics for measuring quality are accuracy, precision, sensitivity and specificity. It is convenient to define them using a truth table (see figure 3.1).

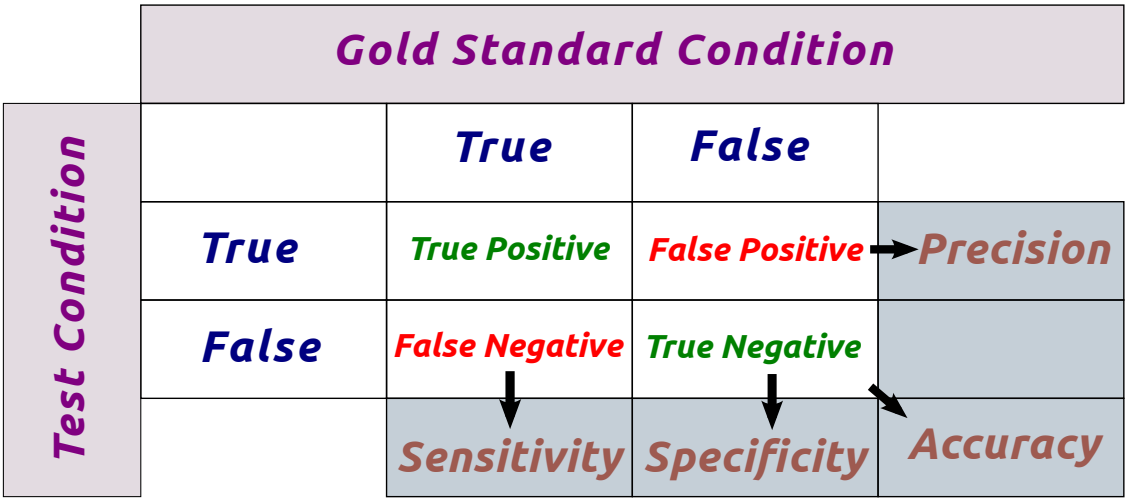


Figure 3.1: A truth table is a useful graphical representation of the quality metrics for a prediction. It compares a gold standard (i.e. which amino acids *should* be predicted as disordered) versus the "test outcome" (the actual prediction). From this, the four metrics, accuracy, precision, sensitivity, and specificity can be calculated as discussed in the text.

A truth table consists of four quadrants that describe the binary (true/false) value of a test prediction against what the prediction should be (the gold standard). If a value is correctly predicted as being true, it is a *true positive* (TP). If it is incorrectly predicted as being true, it is a *false positive* (FP). Similarly, if a value is correctly predicted as being false, it is a *true negative* (TN), and if it is incorrectly predicted as false, it is a *false negative* (FN). From this truth table, the four metrics of interest can be calculated as described below. This truth table is used in all statistical applications to calculate the quality of the disorder predictions in this and subsequent chapters. Accuracy, precision, sensitivity and specificity are defined as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \tag{3.4}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.5}$$

$$\text{Sensitivity} = \frac{TP}{TP + FN} \tag{3.6}$$

$$\text{Specificity} = \frac{TN}{FP + TN} \tag{3.7}$$

Each metric, therefore, captures one aspect of the story of the quality of a prediction. However, individual metrics cannot provide a complete picture of the quality of a predictor on their own. A fifth metric, the Matthews correlation coefficient (MCC), tries to solve this problem. It is a single measure of the quality of binary classifications and can also be calculated from the truth table as follows:

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{3.8}$$

Whilst MCC is better than a single metric, it is still influenced by the relative class frequencies; therefore, Yang et al. (2005) preferred to use a metric called **probability excess** (PE) that is more robust.

3.3.2.1 Probability Excess

The probability excess measurement provides a more complete picture of predictor quality because it is resistant to large differences in class size and lends itself well to geometric/graphical representation (see section 3.3.6). PE is therefore the measure of choice and is used throughout this thesis.

Probability excess was defined by Foster (2004) as a metric for text characterisation and is defined as:

$$\text{PE} = \text{Sensitivity} + \text{Specificity} - 1.0 \tag{3.9}$$

Since it takes both sensitivity and specificity into account, the PE is a robust measure that better reflects the true quality of a prediction. It is also independent of the frequency of the distribution of data, which makes it an unbiased metric for measurement of disorder predictor quality. A predictor with a PE of 1.0, or 100% is a perfect predictor, while a PE of 50% could be achieved by a predictor that is 75% sensitive and 75% specific.

3.3.3 Training, Validation and Testing

The data set is derived from the PDB and contains sequences of known ordered or disordered states. Curation of a new data set is discussed in section 3.4.1. The data are randomly divided into ten subsets. The division of the data set for training as shown in figure 3.2. One subset of ordered and disordered sequences is divided in two, and the halves are used for training and validation against purely the *disordered* **prototypes** from the other nine subsets.

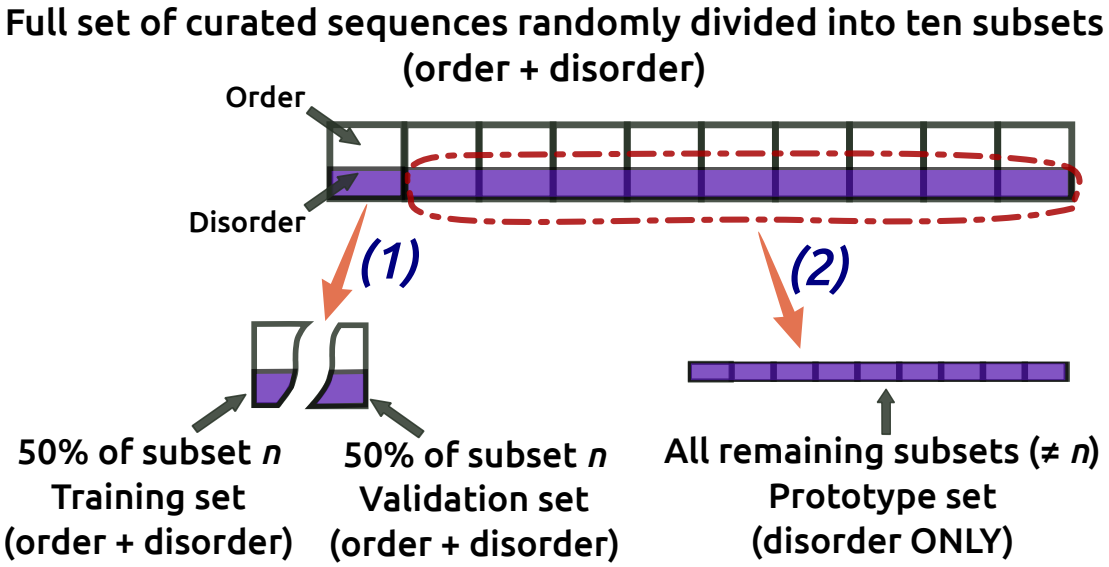


Figure 3.2: Division of data for one round of training and validation. Training and validation sets contain both ordered and disordered sequences (1), but are compared purely against disordered prototypes that are not included in the training and validation sets (2).

This process is repeated ten times (once per subset), ensuring that each subset is unique[1]. Between any two rounds of training and validation, 8/9 of the prototypes will be the same, since the disordered sequences in a subset are used as prototypes for the other nine rounds of training and validation. The ten sets of results from training and validation are then statistically combined at the prediction stage (see section 3.3.4).

Training is performed by comparing fixed–length **windows** of each training sequence to prototype windows. Windows are generated by sliding along the length of a sequence. Yang et al. (2005) found the optimal window length to be 19 residues, and this number was used as the starting point in the study and improvement of RONN. Throughout this chapter, unless otherwise specified, the window length can be assumed to be 19 amino acids.

The goal for the training stage is to assign a per–round weight for each prototype sequence out of N sequences (ω_1 to ω_n). During a round, each training window is matched against each prototype window and the maximum BBF score is used to represent the prototype sequence. Equation 3.10 is a classifier that captures this concept, where $f(x, y)$ is the bio–basis function defined in equation 3.2, and ω_n is the weight associated with the n^{th} prototype.

$$y = \sum_{n=1}^{N} \omega_n f(x, y) + \omega_0 + \epsilon \tag{3.10}$$

In the equation, y is known; it is the assigned disorder/order state of a prototype. In the RONN trainer, the value 0 is used for order, and 1 is used to indicate disorder. The value ϵ is an error term and is removed. The ω_0 is known as a "bias term" (Yang et al., 2005) and affects how each the results of each round are biased. It is

[1]The formal term used in machine learning to describe this method is "ten–fold cross–validation." However, we omit the use of the word "fold" here lest it be confused with protein folding. Instead, the word "round" is used.

calculated along with the other weights (ω_1 to ω_n) in a linear algebra system:

$$\mathbf{F} = \begin{pmatrix} 1 & f_{11} & f_{12} & \cdots & f_{1N} \\ 1 & f_{21} & f_{22} & \cdots & f_{2N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & f_{M1} & f_{M2} & \cdots & f_{MN} \end{pmatrix} \tag{3.11}$$

where M is the number of ordered and disordered training sequences[2] and N is the number of prototypes. Note that the ω_0 bias term is calculated in the first column of the matrix by seeding the value of 1, which is an arbitrary choice. The other ω values represent the best BBF scores between the training and prototype sequences[3]. If the desired weight vector is denoted $\mathbf{w} = (\omega_0 \quad \omega_1 \quad \omega_2 \quad \cdots \quad \omega_N)^T$ and the known disorder/order state vector $\mathbf{y} = (y_1 \quad y_2 \quad \cdots \quad y_M)^T$, the solution to $\mathbf{w}$ is:

$$\mathbf{w} = \mathbf{F}^{-1}\mathbf{y} \tag{3.12}$$

Calculating the true inverse ($\mathbf{F}^{-1}$) is a slow and non–trivial problem for large matrices. Therefore, the Moore–Penrose pseudoinverse is used instead as it is a close approximation and is much quicker to calculate. This pseudoinverse (yielding $\mathbf{w}^*$ instead of $\mathbf{w}$) is calculated thus:

$$\mathbf{w}^* = \left[(\mathbf{F}^T\mathbf{F})^{-1}\mathbf{F}^T\right]\mathbf{y} \tag{3.13}$$

In the RONN trainer source code, this formula is implemented using the method described in Numerical Recipes (Press, 2007) to find the Moore–Penrose pseudoinverse and breaks the calculation of $\mathbf{w}^*$ broadly into two halves, $(\mathbf{F}^T\mathbf{F})^{-1}$ and $(\mathbf{F}^T\mathbf{y})$, before combining them for the final multiplication. In their study of fast algorithms

[2] See section 3.3.3.1 for a discussion on the distribution of training sequences.

[3] Note again, that the score of the best–matching prototype *window* is used to represent the whole prototype sequence.

to compute pseudoinverses, Courrieu (2008) note that the Moore–Penrose pseudoinverse reduces to the normal pseudoinverse described above when the matrix has full rank as in this case, so this algorithm is acceptable for use here without requiring any modifications. Figure 3.3 shows the complete training workflow.

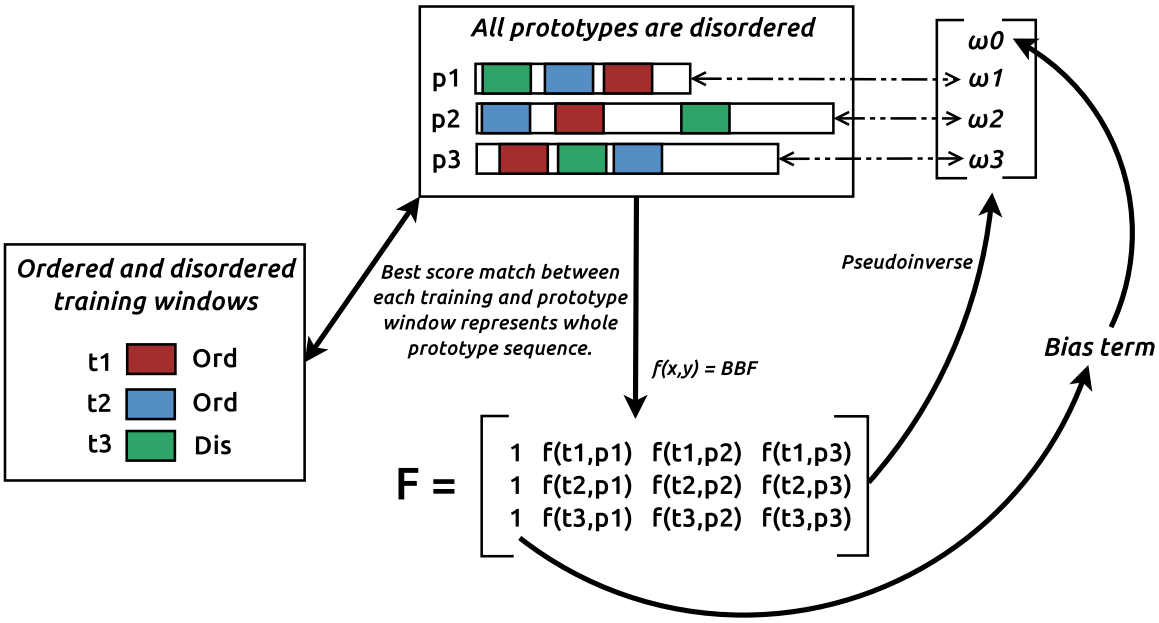


Figure 3.3: The RONN trainer at work for one round. The goal is to obtain a weight for each disordered prototype sequence by ten–round training and cross–validation with selected ordered and disordered training windows. The best matching prototype windows for the training windows ($t1$ to $t3$) are highlighted in the prototype sequences.

Once weights for each database sequence in each round are calculated, a **validation** step takes places with the remaining 5% of the sequences. A **density estimator** calculates mean and standard deviation for order and disorder for each round to provide, in effect, a measure of confidence for that round.

3.3.3.1 Balancing training data with the "class–ratio" factor

In order for optimal training to occur, it is important that the two classes of training or validation sequences (ordered and disordered) be in some sensible ratio. In the original RONN trainer code, a ratio of **4:1** (order:disorder) was chosen for each round, that is, for every disordered training sequence, four ordered sequences

are trained against each prototype. This was deemed to be the optimal ratio through trial and error and is controlled in the RONN source code by a "class–ratio" factor that adjusts the ratio as required.

Since small changes in the "class–ratio" factor do not affect the training very much, it is not discussed in any more detail. The key is the ratio of training sequences, and subsequent results are reported with the ratio, where required. Indeed, the "class–ratio" factor is somewhat of an inconvenient parameter here, but encouragingly, it becomes nearly negligible for *MoreRONN* (see chapter 5).

3.3.4 Prediction

The predictor uses the ten–round trained data in similar fashion to the trainer, but in this case, each window of the query sequence is BBF–scored against each prototype window. The weight for each prototype (derived from training) is then multiplied by this score. The ten predictions are then combined with the calculated probability densities and averaged to produce the final per–window prediction. Finally, these are overlapped to produce averaged per–residue predictions of disorder. Figure 3.4[4] shows a prediction for a given window in a query sequence. For any given residue, a predicted score ≥ 0.5 indicates disorder.

The predictor also contains a cost coefficient which is designed to offset training bias in the per–residue prediction scores. The cost coefficient was set to **0.53**, implying that a residue that attains a score of 0.53 is adjusted to 0.5, which is the cutoff for disorder. The cost function varies the prediction score only by a small margin, and it is another nuisance parameter, like the "class–ratio" factor (see section 3.3.3.1) that was arbitrarily adjusted in order to produce the highest PE scores. Ef-

[4]Copyright 2004, Springer, Intelligent Data Engineering and Automated Learning – IDEAL 2004, Lecture Notes in Computer Science Volume 3177, 2004, pages 108–116, "Prediction of Natively Disordered Regions in Proteins using a Bio–basis Function Neural Network", Thomson R and Esnouf R, Figure 1; with kind permission from Springer Science and Business Media.

forts were undertaken in the development of *MoreRONN* (see chapter 5) to remove this parameter; for the time being however, it remains at 0.53 for the results in this chapter to maintain consistency with the published RONN and is not discussed further.

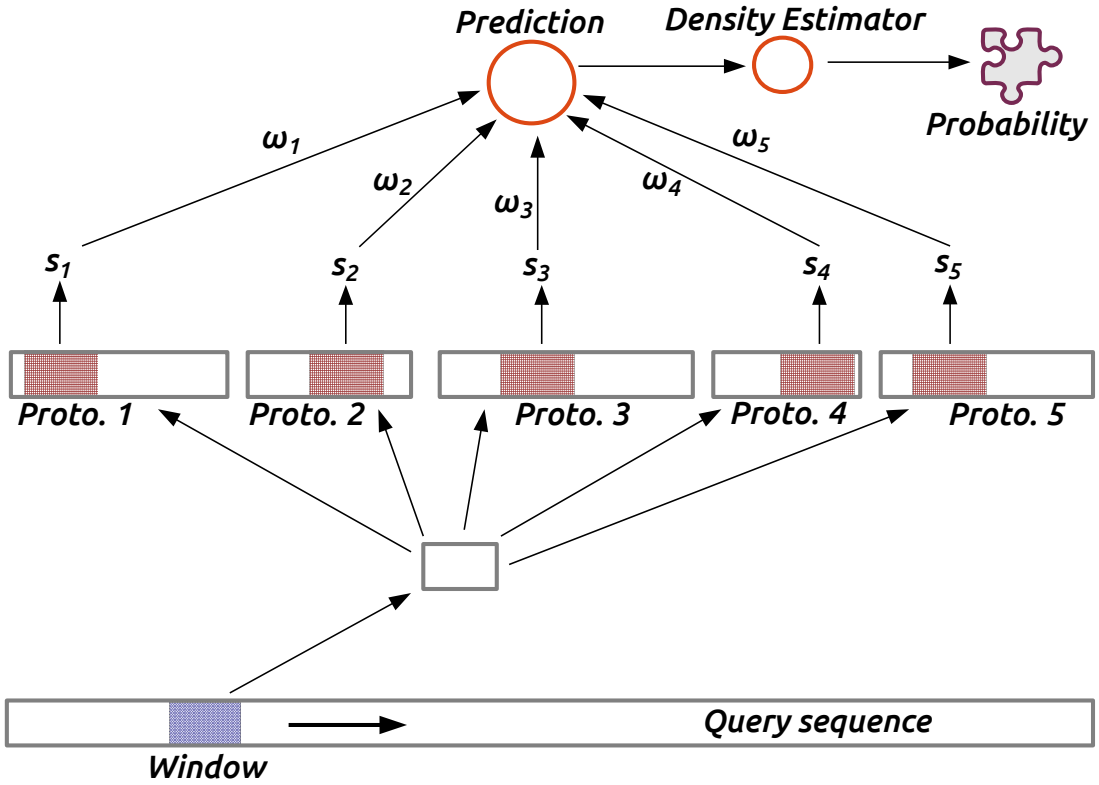


Figure 3.4: RONN prediction for a window in the query sequence. The figure assumes that there are only five prototype sequences. Scores s_1 to s_5 are the highest BBF scores for the query window against the respective prototype windows, which are then used to represent the whole prototype sequence. They are multiplied by the weights for the prototypes (ω_1 to ω_5) which were derived during training. Figure adapted from Thomson et al. (2004).

3.3.5 Testing the predictor

Yang et al. (2005) defined two test sets, an 80–protein and a 203–protein[5] blind set with known regions of disorder. These two sets contain sequences that were not used in training (i.e. RONN is "blind" to them).

[5]The original set contained 205 sequences; however, two sequences were found to have large regions of order cleaved during experimentation by fungal proteases and were therefore removed from testing.

The test sets were run through a testing engine written in Perl by Yang et al. (2005) that measured probability excess. The same test sets were then fed into PONDR, DisEMBL and other disorder predictors to rate the predictors against each other. At the time of publication, Yang et al. (2005) argued that RONN was the class–leading predictor.

This testing framework was used as–is for all subsequent tests on RONN and other predictors.

3.3.6 Pyramid Plots

When RONN was published, it was the best-in-class predictor, with a PE of 43.8% when tested on 80 proteins, and a PE of 48.7% when tested on 205 proteins. Yang et al. (2005) developed a graphical representation of PE, known as a **pyramid plot**, as shown below in figure 3.5.

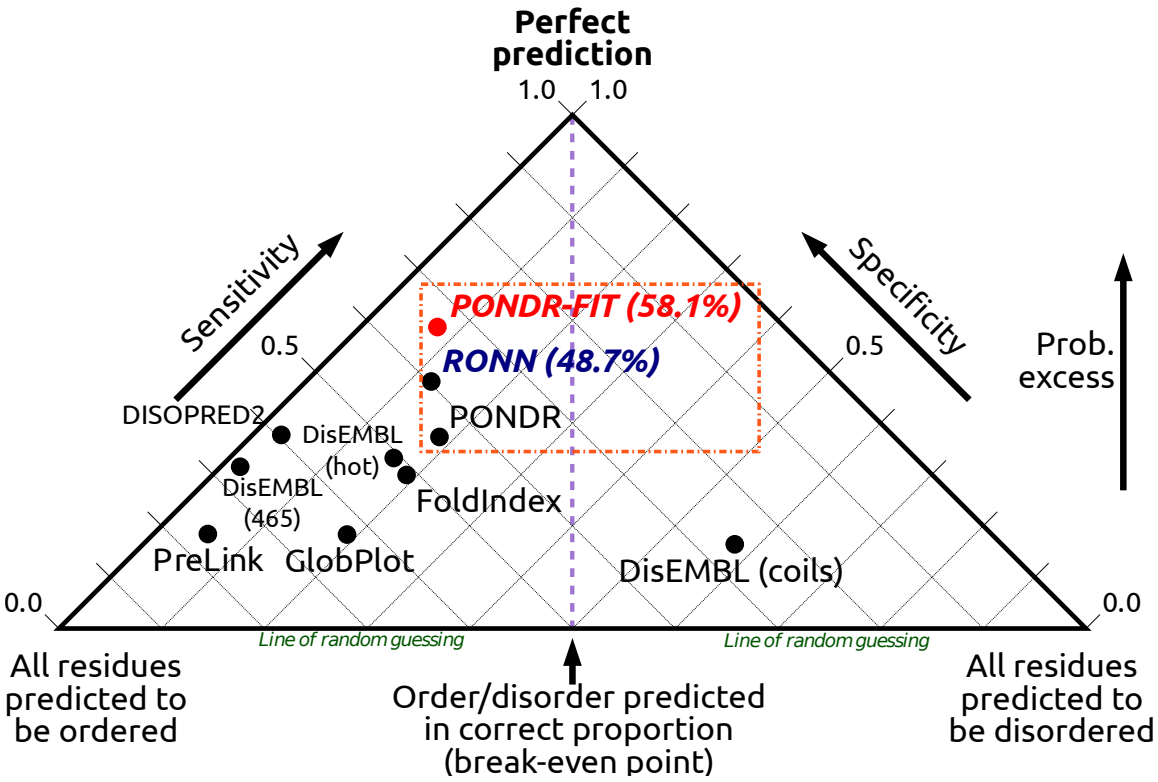


Figure 3.5: Pyramid plot showing the probabilities excess of various disorder predictors, as tested by Yang et al. (2005) against the 203–protein blind test set. PONDR, RONN and PONDR-FIT are highlighted as they are the most balanced predictors (i.e. their data points are closer to the middle axis of the pyramid). At the time of publication, RONN was the best-in-class predictor with an adjusted PE of 48.7%. PONDR-FIT was tested against the same 203–protein test set and outperforms RONN. Note that most predictors have high specificity since more training information is available for order, rather than disorder.

In this graph, sensitivity is plotted against specificity, with the perfect prediction (PE = 1.0) at the top. A predictor that is balanced between sensitivity and specificity will feature closer to centre of the pyramid figure. Notionally, the best predictors should be closer to the top of the pyramid and in the middle[6]. The *"Line of Random Guessing"* on this figure marks a PE of 0%; any predictor situated along this line would do no better than flipping a fair coin to determine whether a residue was ordered or disordered. Conversely, an apex predictor situated at the very top of the pyramid would have a sensitivity and specificity of 100.0%, which means it would be perfect.

[6]The author is fond of the term *"apex predictor"* to describe the highest predictor on the pyramid.

The same two test sets were run with the latest version of PONDR, dubbed PONDR-FIT, which was released in 2010. PONDR-FIT is more sensitive and specific than RONN (and indeed, at the time, the current best–in–class predictor with a PE of 58.1% when tested against the 203–protein test set), as indicated by its position on the pyramid plot in figure 3.5. The PE for RONN has been adjusted for the 203–protein set, as opposed to the original set of 205 proteins.

Given that RONN was trained on just 530 disordered sequences curated from the PDB nearly a decade ago, it is not surprising that it has lost its top spot. Given the increase in structural data available today, it is logical to curate a new training set to boost RONN's performance.

3.4 Methods: Improving and Updating RONN

Since the RONN database was last updated in 2007, the PDB has grown more than twofold, from approximately 42,000 entries (Berman et al., 2007) to its current size of approximately 97,000. Advances in experimental methods and equipment have improved the quality of depositions and therefore, it is a safe hypothesis that a training set curated from the latest PDB will be more comprehensive, of higher quality, and more diverse, and is therefore likely to improve RONN's prediction ability immediately without any modifications to the algorithm itself. An automatic curation pipeline was therefore developed to select high–quality PDB entries for re–training RONN.

While it may seem counter–intuitive to be able to obtain disorder data from crystal depositions in the PDB, which is a database of ordered structure, there is a method of inferring disorder which relies on the *absence* of ATOM records.

Definition 3.4.1. *While the PDB is a repository of structure, disordered regions can be inferred by examining differences between SEQRES and ATOM sequences. The SEQRES sequence is assumed to be present in the crystal. Missing ATOM records, therefore, indicate that these regions are present but invisible, implying disorder.*

Therefore, by applying definition 3.4.1 to the PDB, it is possible to obtain disorder sequences for training RONN.

Apart from disordered regions buried within structured regions in a crystal, there are also proteins that are fully disordered (i.e. they exist as random coils in their natural state). The DisProt database is a curated repository of these fully disordered proteins and is a valuable additional and distinct source of training data (Sickmeier et al., 2007).

3.4.1 Assembling the new data set: Crystals, NMR and DisProt

The data set contains the entire DisProt database and known disordered sequences curated from the PDB. The PRAXIS pipeline was modified to parse ATOM records into a sub–repository similar to the manner in which SEQRES records were handled.

3.4.1.1 PRAXIS 2.0**: Modifying the pipeline**

PRAXIS 2.0 builds on version 1.0 with the only difference being the addition of an ATOM record parser that creates a mirror on disk similar to the SEQRES mirror. This program is called `fasta-mirror-creator-atom` and uses the same *rsync* output as the SEQRES mirror creator (see section 2.3 and figure 2.4) to generate the ATOM records. As with the rest of the PRAXIS workflow, this step is automated and runs weekly. The updated PRAXIS is shown in figure 3.6.

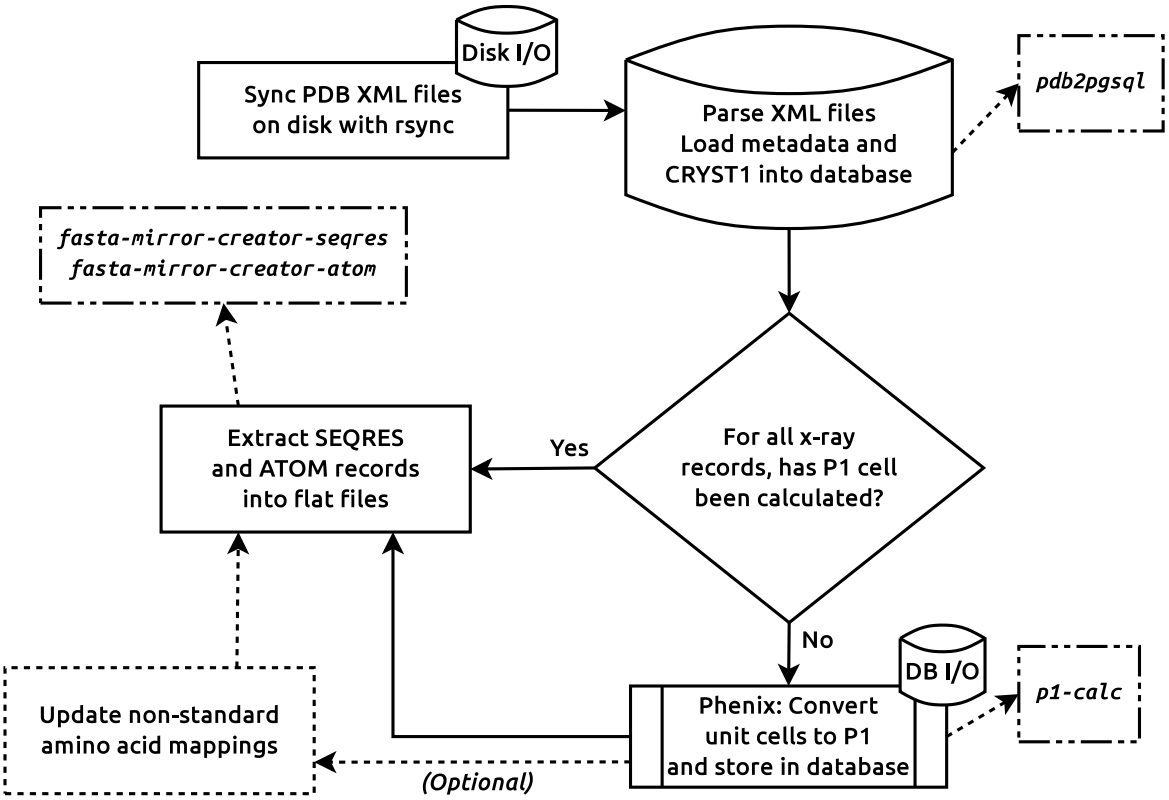


Figure 3.6: The updated PRAXIS pipeline, showing the addition of the `fasta-mirror-creator-atom` program, which creates the ATOM record mirror on disk. Internally this program is very similar to `fasta-mirror-creator-seqres`.

3.4.1.2 Coarse filtering quality metrics

A set of quality cutoffs were defined and applied to the PRAXIS database in order to select higher–quality crystallographic PDB records for further data preparation. The cutoffs are shown in table 3.2 and the PostgreSQL query used to obtain this data is shown in listing 2 in appendix A.1. Note that these cutoffs were an arbitrary starting point and select for vaguely reliable structures.

Metric	Parameters
Method	`"X-RAY DIFFRACTION"`
Resolution	< 2.5 Å
RMS bond length deviation	< 0.015 Å
RMS bond angle deviation	$< 2.0°$
R_{work}	< 0.25

Table 3.2: Coarse cutoff values used to filter appropriately high–quality crystals for disorder data set curation.

This query yielded **35,444** filtered X–ray entries, and the decision was made to add the entire set of NMR records (**9,060** of them) as well[7], bringing the grand total of coarsely–filtered PDB files to **44,504**. Note that these filters do not select for protein samples specifically, as there are cases where protein–ligand or protein–nucleic acid interactions are crystallised and it is often difficult to infer the contents of a crystal or NMR record from the PDB headers alone. In hybrid cases, the protein component may contain regions of disorder and it is useful to rescue its sequence for possible inclusion in training. Therefore, no assumption is made as to the contents of the crystals or NMR models at this stage.

3.4.1.3 Isolating high quality ATOM chains with `disxml`

A C++ program called `disxml` was written to examine ATOM chain geometry and to isolate ordered and disordered fragments in one–letter, FASTA format, similar to the manner in which SEQRES sequences are represented. Figure 3.7 describes the operation of the program.

[7]NMR records can highlight locally disordered regions.

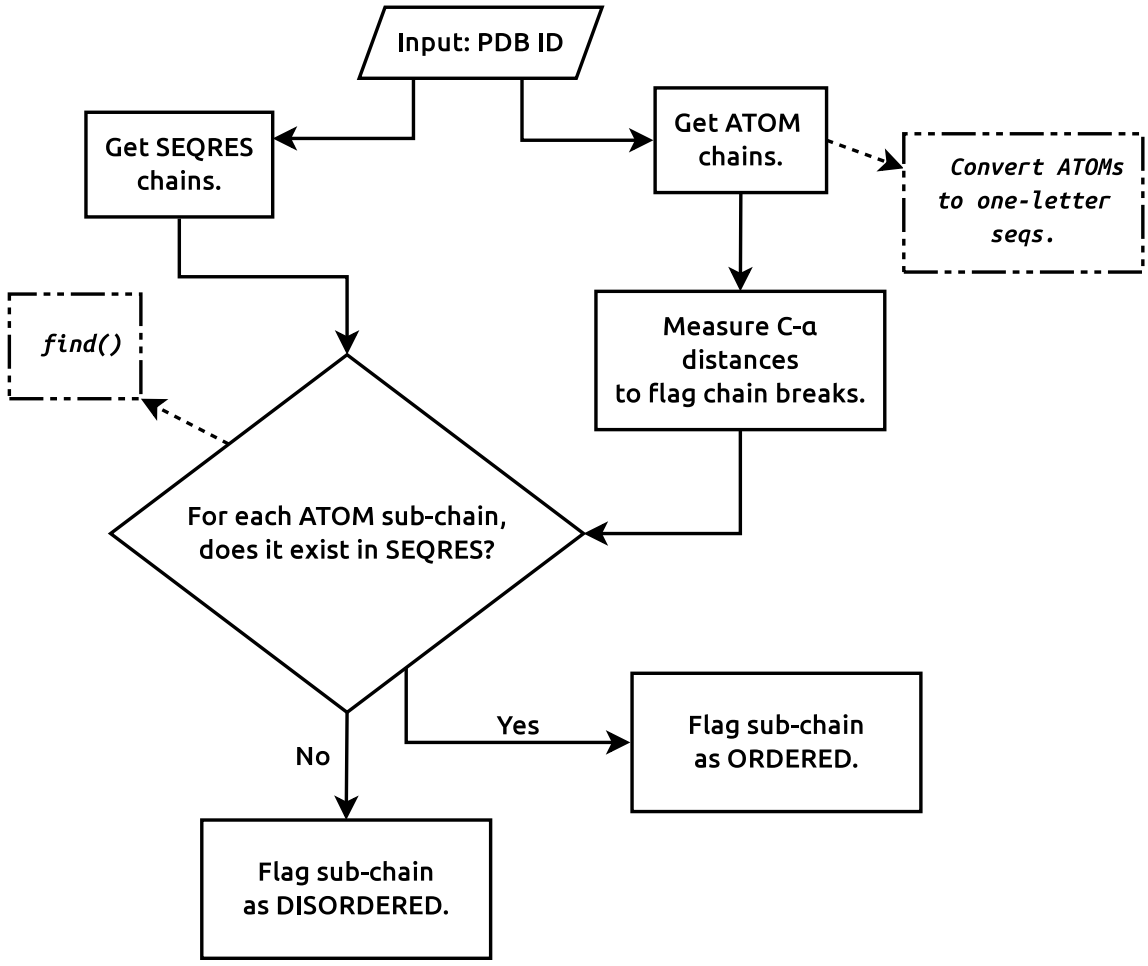


Figure 3.7: An overview of `disxml`'s operation.

Note that, since `disxml` needs to convert three–letter amino acid codes into one–letter codes, this automatically eliminates any chain that is not a polypeptide. This implicit filter enables `disxml` to work with hybrid protein–nucleic acid data, for example, where the polypeptides and nucleic acids are assigned different chain identifiers.

The input to `disxml` is a PDB ID. The program then retrieves the corresponding flat–file SEQRES and ATOM records generated by PRAXIS[8]. It then performs a simple geometric distance check between neighbouring Cα atoms for each chain. The

[8]For NMR data, PRAXIS only generates ATOM records for the first model.

squared distance between two Cα atoms, A and B, is given by:

$$\text{distance}^2 = (B_x - A_x)^2 + (B_y - A_y)^2 + (B_z - A_z)^2 \tag{3.14}$$

In peptide chains, consecutive Cα atoms are not more than 4.2Å apart, and the "ideal" distance is approximately 3.85Å. For `disxml`, this cutoff is relaxed to 4.3Å, which means that the condition (distance$^2 \leq 18.49$Å^2) must be satisfied[9]. In cases where this condition is not met, `disxml` flags a chain break between the two offending residues.

Once all the peptide chains have been processed into fragments delineated by chain breaks, `disxml` performs a `find()` operation on each fragment against the SEQRES sequence for the chain. As the C++ standard `find()` function returns the index of the first occurrence of a sub–sequence (in this case, the chain fragment) in a string, `disxml` builds a list of locations where chain fragments occur in the SEQRES. It then flags residues in between chain fragments as contiguous disordered fragments. Figure 3.8 illustrates `disxml`'s output for chain A of the PDB record for 1VRT (Ren et al., 1995). The `disxml`–generated FASTA headers contain a 'D' or 'O' for disorder or order, respectively, and the start index at which the fragment occurs in the SEQRES sequence.

[9]Square roots were computationally costly until the advent of more advanced SSE processor instruction sets; calculation time is now barely noticeable for all but the largest computations. However, in keeping with the ethos of this thesis in terms of extracting the best possible performance out of the hardware, we work here with squared distances.

1VRT - Chain A SEQRES sequence (disordered regions highlighted):

```
>1VRT E_1 C_A T_p 560 XRAY 1.85 0.17989 0.247 HIV-1 REVERSE
TRANSCRIPTASE <POL_HV1H2> [Human immunodeficiency virus 1]
PISPIETVPVKLKPGMDGPKVKQWPLTEEKIKALVEICTEMEKEGKISKIGPENPYNTPVFAIKKKDSTKW
RKLVDFRELNKRTQDFWEVQLGIPHPAGLKKKKSVTVLDVGDAYFSVPLDEDFRKYTAFTIPSINNETPGI
RYQYNVLPQGWKGSPAIFQSSMTKILEPFRKQNPDIVIYQYMDDLYVGSDLEIGQHRTKIEELRQHLLRWG
LTTPDKKHQKEPPFLWMGYELHPDKWTVQPIVLPEKDSWTVNDIQKLVGKLNWASQIYPGIKVRQLCKLLR
GTKALTEVIPLTEEAELELAENREILKEPVHGVYYDPSKDLIAEIQKQGQGQWTYQIYQEPFKNLKTGKYA
RMRGAHTNDVKQLTEAVQKITTESIVIWGKTPKFKLPIQKETWETWWTEYWQATWIPEWEFVNTPPLVKLW
YQLEKEPIVGAETFYVDGAANRETKLGKAGYVTNRGRQKVVTLTDTTNQKTELQAIYLALQDSGLEVNIVT
DSQYALGIIQAQPDQSESELVNQIIEQLIKKEKVYLAWVPAHKGIGGNEQVDKLVSAGIRKVL
```

Output from `disxml`*:*

```
>1vrt C_A D_0 IDX_1
PIS
>1vrt C_A D_1 IDX_444
GAANRETKLGK
>1vrt C_A D_last IDX_540
KGIGGNEQVDKLVSAGIRKVL
>1vrt C_A O_0 IDX_4
PIETVPVKLKPGMDGPKVKQWPLTEEKIKALVEICTEMEKEGKISKIGPENPYNTPVFAIKKKDSTKWRK
LVDFRELNKRTQDFWEVQLGIPHPAGLKKKKSVTVLDVGDAYFSVPLDEDFRKYTAFTIPSINNETPGIR
YQYNVLPQGWKGSPAIFQSSMTKILEPFRKQNPDIVIYQYMDDLYVGSDLEIGQHRTKIEELRQHLLRWG
LTTPDKKHQKEPPFLWMGYELHPDKWTVQPIVLPEKDSWTVNDIQKLVGKLNWASQIYPGIKVRQLCKLL
RGTKALTEVIPLTEEAELELAENREILKEPVHGVYYDPSKDLIAEIQKQGQGQWTYQIYQEPFKNLKTGK
YARMRGAHTNDVKQLTEAVQKITTESIVIWGKTPKFKLPIQKETWETWWTEYWQATWIPEWEFVNTPPLV
KLWYQLEKEPIVGAETFYVD
>1vrt C_A O_1 IDX_455
AGYVTNRGRQKVVTLTDTTNQKTELQAIYLALQDSGLEVNIVTDSQYALGIIQAQPDQSESELVNQIIEQ
LIKKEKVYLAWVPAH
```

Start position

Figure 3.8: Ordered and disordered fragments as calculated by `disxml` on 1VRT chain A. Disordered fragments are highlighted in bold face. The last disordered fragment is highlighted to show its starting index in SEQRES.

A quality–control filter was implemented in `disxml` to ensure the ATOM records were pristine. For instance, `find()` can return false if there is a point mutation or an incorrect annotation in the ATOM or SEQRES. In such cases, `disxml` outputs an appropriate warning without outputting any sequence information, which makes it easy to flag and cull the PDB entry. There is another special case where `disxml`

might find a chain break in the ATOM record which is absent in the SEQRES, *i.e.* a case of **abutting** ATOM sequences. In such cases, `disxml` outputs a different warning since the data are very–nearly acceptable, allowing future data collection methods to isolate these PDB entries for manual curation. However, for the purposes of RONN training set curation, none of these special cases were considered.

The set of 44,504 coarse–filtered PDB entries was fed to `disxml` to obtain ordered and disordered fragments. The full DisProt database (version 5.8) was appended to this list of chains and then CD–HIT was run to remove redundant sequences at 90% identity, with a minimum acceptable sequence length of 19 residues (corresponding to window size[10]). Lastly, sequences that were in the two test sets were removed (to preserve RONN's "blindness" to them). The data size results are shown in table 3.3.

Coarse–filtered PDB entries	44,504
`disxml`–filtered ordered sequence fragments	105,528
`disxml`–filtered disordered sequence fragments	106,550
Fully–disordered sequences and disordered regions in DisProt 5.8	1,358
Total number of curated sequence fragments before CD–HIT	213,436
Disordered sequences after CD–HIT (90% identity) — test seqs.	**2848**
Ordered sequences after CD–HIT (90% identity) — test seqs.	**22290**

Table 3.3: Data set sizes after `disxml` curation, addition of DisProt and CD–HIT redundancy removal. As a final step, sequences appearing in the blind test sets were removed from the training data. The total fragment size is the sum of DisProt and the `disxml`–filtered ordered and disordered fragments.

Therefore, the curation of the new RONN dataset provided 2848 disordered prototypes, which is a fivefold increase from the original (published) RONN's prototype set (Yang et al., 2005), which contained just 530 disordered sequences.

[10]Any sequence smaller than the window size is useless as it will not factor in training.

3.5 Training RONN with new data

Simply plugging this new training set into the original Java–based RONN trainer code proved unsuccessful as the code was unable to handle the larger data set due to small size limits on statically declared variables such as arrays (more serious numerical instability problems were encountered later; these are discussed in section 3.7).

3.5.1 Improving the RONN Trainer and Predictor

The RONN predictor (implemented in C++) was improved to remove restrictions on trained data size and query sequence length by using dynamically–allocated STL `vector` data structures instead of statically-allocated arrays, and to enable easy multithreading through libraries such as OpenMP (Dagum et al., 1998) for quicker computation. This allows the predictor to dynamically expand its data structures to accommodate larger input sets without any ceiling on the number of sequences allowed.

The decision was made to convert the RONN trainer's Java code to C++ in order to make it more easily understandable and maintainable. Upon examination of the training code, it was noted that it would scale linearly with the size of the input data set, thus the original one–hour training time (530 disordered sequences) with RONN 3.1 (the published version) would balloon to approximately five hours with the new data set (2848 disordered sequences) and the original code. As it is the author's personal preference to avoid proprietary, high–level programming languages that require an additional virtual machine layer to execute properly (Oracle Java™), the decision was made to re-implement the trainer from scratch in C++ using clear variable and function names, and to document it clearly. This would also give access to easy multithreading through OpenMP and reduce the training time

by parallelising the main training loop.

The validation stage was left intact in Java as it does not involve the slow pseudoinverse computation present in the trainer. However, the statically–allocated Java arrays were converted to dynamic `ArrayList` containers which scale in size, similar to the use of STL `vector` in the predictor.

The new C++ trainer and predictor were tested with the old published 530–sequence training set and the identical PE values were obtained, verifying the correctness of the new code[11].

3.6 Preliminary Results and Discussion

With the window size of 19 residues carried over from the old predictor, the new training data produced a marked improvement in RONN PE (see table 3.4), but it was still not performing as well as PONDR-FIT. However, this new RONN (dubbed RONN 4.0) was much more sensitive than both RONN 3.1 and PONDR-FIT for both test sets; PONDR-FIT's stronger specificity score, however, gave it a higher PE. The new data set did not produce any increase in specificity in RONN; in fact, the specificity got slightly worse!

[11] There was a small bug in the Perl testing framework wherein the last residue in a query sequence was not scored. Since the last residue only occurs in the last window, it does not affect PE very much, but all previously published data reported in this thesis have been corrected with the fixed testing framework. These corrected numbers were corroborated by the new trainer and predictor code.

Predictor	Sensitivity	Specificity	PE
RONN 3.1	58.5%	85.3%	43.8%
PONDR-FIT	61.0%	**93.7%**	**54.7%**
RONN 4.0	**66.0%**	85.2%	51.1%

(a) Differences in sensitivity and specificity across the three predictors for the 80–protein test set.

Predictor	Sensitivity	Specificity	PE
RONN 3.1	54.4%	90.1%	44.5%
PONDR-FIT	65.7%	**92.5%**	**58.1%**
RONN 4.0	**68.5%**	82.2%	50.7%

(b) Differences in sensitivity and specificity across the three predictors for the 203–protein test set.

Table 3.4: It is noteworthy that RONN 4.0 is the most sensitive of the three predictors (published RONN 3.1, PONDR-FIT and updated RONN 4.0) for both test sets, but its lower specificity, compared to PONDR-FIT's, pulls its PE down.

RONN 4.0 was uploaded to the existing RONN web server as an interim solution for users, since its higher sensitivity compared to RONN 3.1 would still result in clearer boundary predictions between ordered and disordered regions. For crystallographers interested in designing constructs for expression, this boundary point is critical as it marks the location where the protein under study can be cleaved in order to improve expression. Therefore, whilst the lower specificity was a disappointing result, the markedly better sensitivity was certainly a welcome bonus, given that it was obtained with essentially one change to the predictor: the new training data set. It is also interesting to note that the order to disorder training/validation ratio has gone from 4:1 in the old RONN, to **2:1** in RONN 4.0. This implies that order and disorder are more in balance by a factor of two from the old code, which is very encouraging.

3.6.1 Window sizes

Windows lengths between 13 and 25 residues were tested by Yang et al. (2005); a length of 19 residues was determined to give the highest PE. Thus, 19 was chosen as the starting point for the new data set in order to directly compare against the old results. However, there is no biological reason for the 19-residue window to be the standard size. In fact, 19 residues seem rather long in characterising disorder. Indeed, disordered sequences in nature can be very short (8–9 residues), yet RONN is unable to provide reliable predictions when trained on window sizes below 17 residues (Yang et al., 2005)! Therefore, whilst it could have been interesting to train RONN 4.0 on various window lengths to observe any trends that might have emerged, the decision was made to focus on improving the prediction algorithm itself[12].

3.6.2 Progress so far

The sequence collection pipeline described in this chapter represents a scalable, robust method of assessing the quality of putative training data in the PDB and gathering new data in the future. While the performance gain with RONN and the new data set was measurable, it was disappointing as it was still far from PONDR-FIT's prediction scores. Instead of an in–depth analysis of other tuning parameters (window size, cost coefficient, "class–ratio" factor), it was decided that the probability excess difference between RONN and PONDR-FIT was large enough that it was now necessary to delve into the heart of the prediction engine itself in order to obtain any significant gain in PE, especially in terms of specificity. However, the

[12]Through the course of his D.Phil. studies, and especially with the advances made with *MoreRONN*, the author has come to appreciate the fact that a solid algorithm will remain relatively robust regardless of nuances in the data or choice of parameters. Therefore, changing the "brain" of the predictor is likely to yield much more satisfying results than worrying about parameters such as window size or "class–ratio" factor.

work undertaken in this chapter has, at the very least, provided a much larger data set of ordered and disordered sequences. A more comprehensive "knowledge base" is key for any machine learning algorithm, and this new, high–quality data set now allows us to now focus our attention to the prediction algorithm itself.

The next section and subsequent two chapters explore three separate attempts to modify the prediction algorithm to take full advantage of the newly–curated data set. These attempts culminate in the development of a new predictor, *MoreRONN*, which partitions the data set differently for training and is much more resistant to parameter tweaks; indeed, many of the extra parameters were removed as a result of the new prediction method.

3.7 Training RONN with Declustered Windows

3.7.1 Towards a New Predictor

Whereas the new data set curated from the PDB improved RONN's predictive power (see table 3.4), the limits of the algorithm were apparent as PONDR-FIT continued to maintain the top score. Since the quality of the training data had been improved dramatically by the work described in chapter 3 (and the data does not drastically change in a short period of time), it appeared that PEs around 50% were the ceiling for the current RONN algorithm even when trained with the most up-to-date PDB and DisProt data. Whilst PONDR-FIT is a meta–predictor and therefore combines the strengths of multiple underlying prediction methods, our hypothesis was that RONN could be improved to compete with, and possibly outperform the former at the algorithmic level to make the best possible use of the new data set.

One advantage of meta–predictors such as PONDR-FIT is that weaknesses (noise) of the underlying predictors can be masked in the final meta–prediction, either statistically or by virtue of one prediction method compensating for the weaknesses in another. For example, one internal predictor that is trained to find short regions of disorder ($\leq$ 30 residues) might find a region of disorder that is predicted to contain alpha helices by another internal algorithm that predicts secondary structure; in such cases statistically–weighted predictions that bias one predictor over another might mask the actual truth, that the stretch of query sequence adopts transient secondary structure[13] (Dosztányi et al., 2010). Even within a single class of predictor, this problem is apparent. For example, in the sequence–based disorder prediction space, predictors designed for long disordered regions are often poor at identifying short regions of disorder, $\leq$ 30 residues; this is, in fact, the main obser-

[13]Indeed, almost every predictor is a binary classifier and therefore, transient structure is a problem.

vation that kick–started the development of the PONDR family (He et al., 2009), and remains an observed weakness in RONN due to the fixed window size. By incorporating an additional short–region predictor, sensitivity for short stretches of disorder might be increased and the overall meta–prediction improved, but this adds a layer of complexity for the user. For example, IUPred utilises a different energy calculation for long ($\geq$ 30 residues) versus short predictions, and therefore, the desired method of prediction must be specified at run–time (Dosztányi et al., 2005). PONDR, IUPred and most meta–predictors, through their evolution, have introduced numerous parameters for the user to interact with to try and find the best combination that is suitable for the query protein (Vucetic et al., 2003; Liu et al., 2006; He et al., 2009; Xue et al., 2010). However, these user–selected run–time parameters are problematic as they require "expert" knowledge of disorder prediction (and in some cases, the properties of the query protein itself!). Furthermore, the parameters often need to be fine–tuned repeatedly in order to generate optimal predictions. Therefore, while meta–prediction scores can often outdo single predictors (as PONDR-FIT does against RONN, for instance), arriving at the seemingly magic combination of parameters to produce optimal predictions is not an exact science.

With RONN, however, the goal has always been to develop a single predictor that is capable of predicting disordered regions regardless of disorder length, amino–acid chemistry or other properties, using as few input parameters as possible, enabling non–expert and automated pipeline use (Yang et al., 2005; Esnouf et al., 2006). That said, it must be remembered that a pure disorder predictor such as RONN can, and is, implemented into meta–predictors such as MetaDisorder (Kozlowski et al., 2012), therefore it benefits meta–predictors and the field of disorder prediction as a whole when pure predictors such as RONN are improved.

Having realised the limited scope of the newly curated data set to yield further

improvement (as discussed in chapter 3), it became necessary to reconsider the RONN algorithm itself and attempt to improve it in order to increase the quality of the prediction while maintaining the simplicity of the RONN method as much as possible[14]. Three methods of altering RONN were explored in detail, hereafter referred to as the **declustered**, **CD–HIT** and ***MCL*** methods. The declustered method is discussed in this chapter, while the CD–HIT method is explored in chapter 4. Finally, the *MCL* method, which forms a core part of the novel predictor called MoreRONN, is described in chapter 5.

3.7.2 Revisiting training weights

The RONN neural network performs a tenfold cross–validation during training. In each fold of randomly–assigned ordered and disordered sequences[15], 90% of the full–length disordered sequences are designated as prototypes. The network assigns a weight (ω) to each prototype sequence during training; a larger weight implies that the sequence contains at least one "general solution" of disordered motifs, since weight increases as the ratio of strong BBF hits to training windows increases. The weights are then used during prediction when each window of the query sequence is scored against the prototypes. Figure 3.4 shows the operation of the neural network during prediction.

The RONN trainer assigns a weight (ω) to each disordered prototype sequence by sliding along the prototype windows and picking the window with the highest BBF score against the training window to represent the entire prototype sequence.

[14]It was decided right at the beginning that the goal was to improve the pure disorder prediction technique as much as possible in order to provide a real value addition to the understanding of disorder. Simply resorting to developing a meta–predictor was not deemed to be novel enough, as myriad meta–predictors already exist!

[15]The ratio of ordered:disordered sequences in each fold is controlled by the "class–ratio factor" described in section 3.3.3.1.

The equation to calculate the BBF score is:

$$f(x,y) = \exp\left(\alpha \frac{s(x,y) - \beta_y}{\beta_y}\right) \tag{3.15}$$

All other lower–scoring windows (and weights) are discarded as a result, even if their scores are close to the highest. For instance, from 2848 prototype sequences in the updated training set, there are 107,413 19-residue windows and therefore, the potential for 107,413 scores to be used in training, but only 2848 top scores (one per prototype sequence) are stored and used.

Put another way, essentially, all 19–residue windows in a single disordered prototype sequence are treated as a **cluster**, represented by the best **representative** BBF score to any window in the cluster[16]. Therefore, the first place we began to explore improving the predictor was to keep all of the ~100,000 window weights in training, thereby **declustering** them from their source sequences (hence the name of this method). The aim was to have many more training weights and hope that the neural network could be trained to be sensitive to a small set of strong hits in a sea of weak hits. This would give the predictor the full known range of disordered prototype motifs to compare a query against, and the weights would, hopefully, be more reflective of the real distribution of disorder in the training set (and ideally, real life!).

3.7.3 Training with declustered windows

In order to test the hypothesis that training weights for all windows would "teach" the predictor more effectively, the input prototype set for each fold was expanded to make a large set of fixed–length prototype windows and passed to the trainer, as this involved the least amount of C++ code changes in the trainer in order

[16]Thus, our new data set can be interpreted as containing 2848 disorder clusters.

to obtain a preliminary result, despite the larger input file sizes. Hereafter, the original RONN trainer is called the **sequence trainer**, as it generates one weight per prototype *sequence*, while this version is referred to as the **declustered–window trainer**, as it generates one weight per fixed–length prototype *window*.

A Python script was written to extract the windows from the sequences. The relevant code snippet is shown in listing 3 in appendix A.

Unfortunately, the training code was unable to run due to duplicate windows in the data set. A key step in RONN is the generation of weights by a pseudo–inversion on the the matrix **F** described in equation 5.1 and reproduced below:

$$\mathbf{w^*} = \left[(\mathbf{F}^T\mathbf{F})^{-1}\mathbf{F}^T\right]\mathbf{y} \tag{3.16}$$

where the pseudoinverse is in square brackets.

Matrix **F** was now populated with BBF scores for each prototype, meaning that it contained $N \approx$ 100,000 columns (instead of 2848 in the sequence trainer). The determinant of any matrix (see equation 2.3 for a simplified example) evaluates to 0 if the matrix contains duplicate entries; such a **singular** matrix cannot be inverted to give training weights. Due to the duplication of windows in the new, larger **F** matrix in the declustered–window trainer, the pseudoinverse C++ code was unable to run. Incidentally, it is serendipitous that such an identical window collision did not occur in the sequence trainer. This would require a window motif to be selected as the highest scoring window in more than one sequence. Both cases did not occur, otherwise that training run would have also failed! In this declustered–window trainer, however, since all windows from all prototype sequences were kept, duplicate windows were generated across sequences (and kept for training) that would otherwise be discarded in the sequence trainer, thereby creating the singular **F** matrix.

Duplicate windows were easily removed, however, with a trivial alteration to

the Python snippet in listing 3. The Python `list` data structure was changed to a `set` structure, that is, simply by changing line 1 marked by "***" in listing 3 from `list()` to `set()`[17]. In computer science, by definition, a set cannot contain duplicate entries, and therefore, the use of the data structure guaranteed that an attempt to insert a duplicate window into the `set` would fail without requiring any extra coding effort.

While this resolved the singular matrix problem, the pseudoinverse C++ code now ran very slowly as the manipulation of a $\approx$100,000^2 matrix naturally takes more time than a 2848^2 matrix. The size is squared because the pseudoinverse was calculated using a technique known as Gauss–Jordan elimination which requires double the memory footprint of the matrix itself; the matrix values are overwritten as the elimination proceeds. To illustrate this point, a matrix of size 2848 $\times$ 2848 needs just over 32MB to reside in RAM[18]. A matrix that is 100,000 $\times$ 100,000 large, however, needs **40GB** RAM, which is, on average, five times larger than the standard memory capacity in a modern desktop computer! Furthermore, since the Gauss–Jordan elimination was forcibly single–threaded[19], it was not suitable for quickly computing the much larger **F** matrix in the declustered–window trainer. As a side note, the code implementations found in Numerical Recipes are generally not well optimised, often contain bugs and are not distributed under a suitably free license (Snyder, 1991). This, coupled with the unreasonably slow execution of the training code, made it necessary to completely replace the pseudoinverse calculation with a more robust implementation.

[17]Line 5 requires a small change from `append()` to `add()`; the latter is the synonymous function call for a Python `set`.

[18]We assume each BBF score is represented as a 4–byte floating point value.

[19]Since the Numerical Recipes solution for Gauss–Jordan elimination relies on matrix values being overwritten, it is not conducive to parallelization due to race conditions and concurrent write access by multiple threads. One thread might hold a copy of a matrix value that then gets overwritten by another thread within a short time, or two threads might try to write to the same matrix location, which can lead to undefined behaviour and unstable answers that change each time the code is run.

3.7.4 Improving trainer performance and stability with EIGEN

Given that the pseudoinverse C++ code was directly translated from the original Java source code (see section 3.5.1), it was not optimised to handle the large data set[20]. Whilst other pseudoinverse implementations perform faster and with better numerical stability (Courrieu, 2008), it would save time to find a well-tested third–party library instead. This would *i*) reduce the size and complexity of the RONN codebase and *ii*) ensure that the mathematics and implementations were tested and proven correct through a peer–review or collaborative process[21].

The EIGEN linear algebra library for C++ includes common dense and sparse matrix data structures, and multithreaded, highly optimised computation and matrix manipulation functions (Guennebaud et al., 2010). The code to integrate EIGEN into a C++ application is very straightforward, and the matrix and vector data structures are nearly perfect drop–in replacements for C++ Standard Template Library (STL) data structures such as `vector`. EIGEN also contains numerous memory management and processor optimisations, and its code base is frequently updated. All factors considered, EIGEN seemed like a logical choice for improving the RONN trainer.

Initially, C++ vectors and matrices (implemented as two–dimensional `vector` structures in the bug–fixed RONN training code) were replaced by EIGEN `Vector` and `Matrix` structures, respectively. The pseudoinverse code was otherwise kept intact. This iteration of the training code was tested on the old sequence trainer data set to ensure the same PEs were obtained (they were), but once again, this code

[20]The original Java–based trainer imposed hard–coded limits on array and matrix sizes, both of which were surpassed in the current window trainer. This hearkens back to a programming philosophy of managing memory usage tightly, a restriction that is all but absent in modern computers, unless the data size truly exceeds the physical limitations of the machine. The moral here is that, nowadays, RAM is cheap and abundant, and so dynamically allocated memory can be used for all but the most time– and size–critical tasks. This results in scalable, portable and reusable code.

[21]The author's favourite argument for choosing libraries distributed under a free software license rather than a proprietary one.

was unable to compute the pseudoinverse on the declustered–window trainer set since the Numerical Recipes implementation of the pseudoinverse was a bottleneck. Therefore, efforts were undertaken to try and replace the existing pseudoinverse code with one of EIGEN's built-in matrix decompositions and solvers.

3.7.5 Matrix Decompositions

The key equation at the heart of the BBFNN is:

$$\mathbf{Fw} = \mathbf{y} \tag{3.17}$$

In our case, since **F** and **y** are known, the equation rearranges to the more familiar form:

$$\mathbf{w} = \mathbf{F}^{-1}\mathbf{y} \tag{3.18}$$

The calculation of $\mathbf{F}^{-1}$ is performed by the pseudoinverse method in the sequence trainer, as shown in equation 3.16. However, there are many other ways to find the solution to **w**. Equation 3.17 is simply describing a system of equations of the form:

$$Ax = b \tag{3.19}$$

where A and b are matrices (or where b is a vector in a special case, such as ours). In order to find a solution to x (**w** in our case), a **decomposition** can be used to factorise matrix A into a product of matrices. There are numerous decompositions; EIGEN includes a comprehensive set where many are optimised for speed and memory management. Basic matrix–matrix and matrix–vector operations are multithreaded via OPENMP which help further speed up computation. A partial list of EIGEN decompositions (as of version 3.2 released on 24 July, 2013) is shown in

table 3.5 and is adapted from EIGEN documentation (Guennebaud et al., 2010).

Decomposition	Speed	Reliability and accuracy	Optimisations
PartialPivLU	Fast	Stable*	Yes
FullPivLU	Slow	Proven	None
ColPivHouseholderQR	**Fast**	**Good**	***Coming soon***
FullPivHouseholderQR	Slow	Proven	None
LLT	Very fast	Stable*	Blocking
LDLT	Very fast	Good	*Coming soon*

Table 3.5: A partial list of decompositions provided by EIGEN 3.2. The decomposition name is the EIGEN class name. If reliability is "Proven", it means the decomposition is thorough to ensure maximum numerical stability, whereas for "Stable*" decompositions, the matrix must be invertible and EIGEN assumes this is the case without checking, which speeds up the algorithm. If reliability is "good", it means the solutions are numerically stable in most cases. Optimisations are tricks used by EIGEN to parallelise the decomposition to speed it up. ColPivHouseholderQR is highlighted because it was the chosen decomposition.

These decompositions were short–listed as the best candidates as they provide a reasonable balance between speed and accuracy. However, after personal communication with various members of the EIGEN Internet Relay Chat (IRC) support channel[22], the LLT and PartialPivLU decompositions were rejected as the numerical stability was not deemed good enough for a matrix of the size in this application. Decompositions without optimisations either currently or in the future (FullPivLU and FullPivHouseholderQR) were also rejected as speed was important for this application. This left LDLT and ColPivHouseholderQR (HOUSEHOLDER–QR with column pivoting) which have an acceptable feature set, and ColPivHouseholderQR was arbitrarily chosen for the first attempt.

A HOUSEHOLDER–QR transformation or reflection takes a vector and reflects it about some plane (Householder, 1958). The QR–factorisation decomposes the

[22]Channel "#eigen" on `irc.freenode.net`

matrix thus:

$$A = QR \tag{3.20}$$

where Q is an orthogonal matrix ($Q^TQ = I$) and R is an upper triangular matrix. P is a permutation matrix. The ColPivHouseholderQR decomposition is calculated thus:

$$AP = QR \tag{3.21}$$

where P is a permutation matrix that is usually chosen so that the diagonal elements of R do not increase (*i.e.* $|r_{11}| \geq |r_{22}| \geq \cdots |r_{nn}|$) (Householder, 1958; Guennebaud et al., 2010). Column pivoting is useful for improving numerical accuracy without the added overhead of performing a full pivot (FullPivHouseholderQR).

EIGEN's decompositions all include a `solve()` function to solve the $Ax = b$ equation. Therefore, this single function call was all that was required to obtain the final weight vector (**w**), which is effectively the set of eigenvalues for matrix **F**.

This version of the trainer with HOUSEHOLDER–QR decomposition was tested and verified to produce identical weights and prediction scores as the old RONN sequence trainer, marking a milestone departure from the old algorithm and completely removing the legacy pseudoinverse–based training code.

3.7.6 Declustered windows with EIGEN: Results

With the ColPivHouseholderQR decomposition in place, training completed successfully in half a day. This newly trained predictor was tested with the 80– and 203–protein test sets as before. However, the PE for both test sets was a disap-

pointing **0.0%** (sensitivity = 0.0%, specificity = 100%)[23], that is, all residues were predicted as being ordered. Figure 3.9 shows the position of this declustered predictor on the pyramid plot based on the truth table. The initial assumption was that this was a bug in the code, especially given the suite of changes that were made, but after lengthy and comprehensive checks on the code correctness[24], the 0.0% PE was accepted as correct.

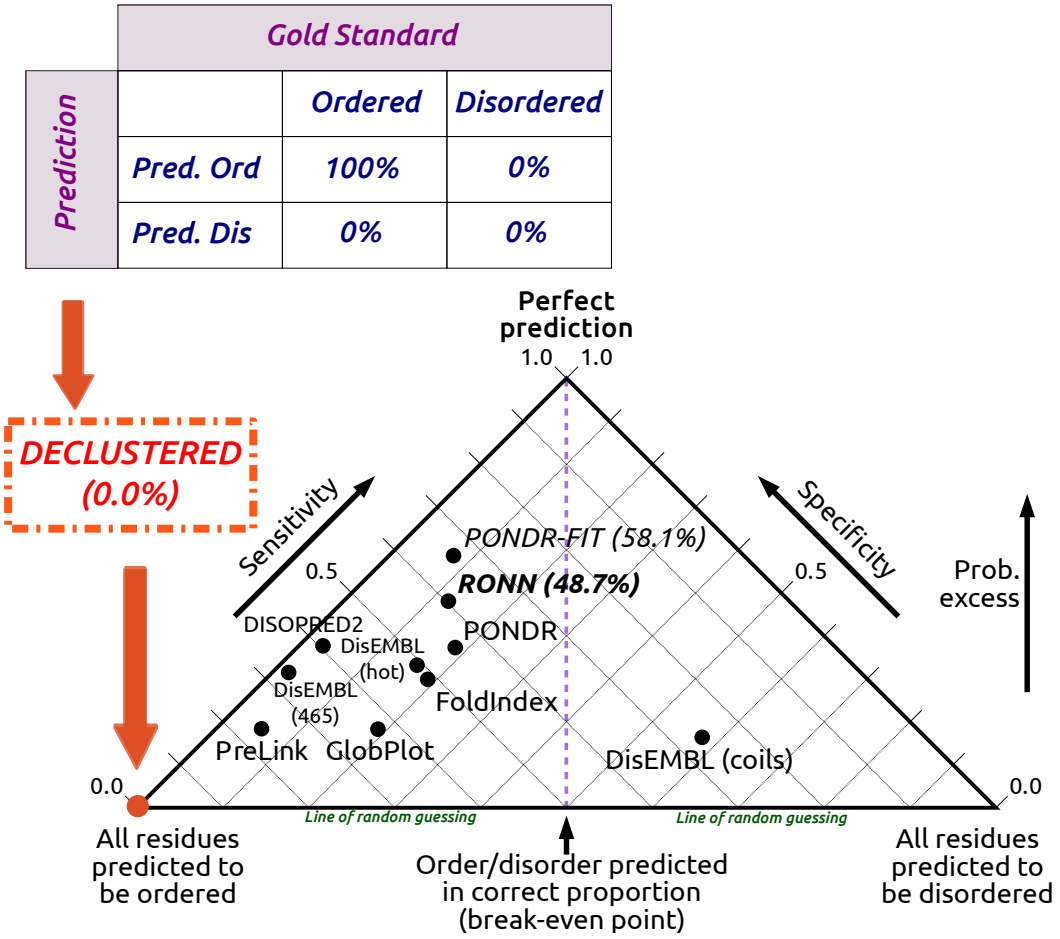


Figure 3.9: Probability excess result for the declustered predictor, showing the truth table and the predictor's position at the bottom–left of the pyramid plot.

We rationalised that there were so many poor scores that were retained in the matrix and weight calculations, whereas they would have been discarded in the

[23]This predictor was doing no better than random guessing.

[24]Testing this EIGEN–optimised code with the old sequence trainer data sets produced identical PEs as before, so the possibility of a bug in the new implementation could safely be eliminated.

old sequence trainer. This resulted in a large amount of noise that completely outweighed the few positive scoring window pairs (signal), thereby reducing the sensitivity to 0.0%[25].

Thus, it was realised that an almost accidental *strength* of the RONN sequence trainer was that only high-scoring window weights were kept, implying that each sequence was actually an **arbitrary** cluster of windows that reduced noise. Therefore, having achieved an important primary result[26] with declustered windows, we concentrated our efforts on clustering techniques, which led to subsequent training methods (discussed in chapters 4 and 5). For these methods, once the concept of clustering had been accepted as valid, the RONN approach to clustering (*i.e.* keeping windows within their source sequences) hinted at the possibility of improvement by reorganising the clusters.

[25]Note that the specificity, on the other hand, was a perfect 100%. Since all residues were predicted as ordered, all truly ordered residues were, therefore, predicted correctly. This, again, highlights the weakness in choosing just one statistical metric that does not paint the full picture. The robustness of probability excess is clearly on display here.

[26]A vastly improved and numerically–stable trainer that operates with a smaller memory footprint, and peer–reviewed linear algebra code in the form of EIGEN.

Chapter 4

Using CD–HIT to Optimise RONN

4.1 Re-partitioning the Data Set

While identical windows had been removed and the code base markedly improved, the poor results indicated that the windows had to be clustered somehow to prevent low-scoring hits from weighing the prediction down. In the original RONN sequence trainer (see chapter 3), windows were preserved within their parent sequences with the top–scoring representative window for each sequence, whilst in the window trainer, all windows were allowed to mix together into one large data set.

There was no need to preserve windows within their source disordered sequences. It was also not optimal to keep every window score, as recently discovered, as a large majority of score matches were very poor. With the removal of identical windows allowing the algorithm implementation to complete, we now concentrated on partitioning the data set differently to more suitably reflect the

diversity of the sequence space. To explore this hypothesis, a sequence clustering method such as CD–HIT seemed a logical way forward in order to cluster windows at a desired percent identity, perform the matching against every window as before, and assign the highest scoring window's weight as the weight for the cluster. It would be a proof–of–concept that would be easy to implement in order to test the validity of partitioning the data in this way. It represents a hybrid method between the sequence and the window trainer, whereby the number of weights is sharply reduced, but the weight for any given cluster of related prototype windows is more representative of the commonality between the windows in each cluster, as opposed to being an arbitrary arrangement within a longer sequence. Indeed, clustering as a method of removing redundancies is a technique used for curating data sets used in other predictors (Dosztányi et al., 2010), which further supported our hypothesis.

This method would also give a new tunable parameter, that is, the number of clusters (and therefore the number of training weights) desired, which could be altered at run–time to test larger or smaller numbers of weights.

4.1.1 Incorporating CD–HIT into the RONN trainer

The trainer was modified to run `cdhit-modified` (discussed in section 2.4.2.1) to cluster all the windows beforehand. Instead of assigning a BBF score to a sequence, the best BBF score to any member of the cluster was assigned to the entire cluster. The modified trainer is shown in figure 4.1.

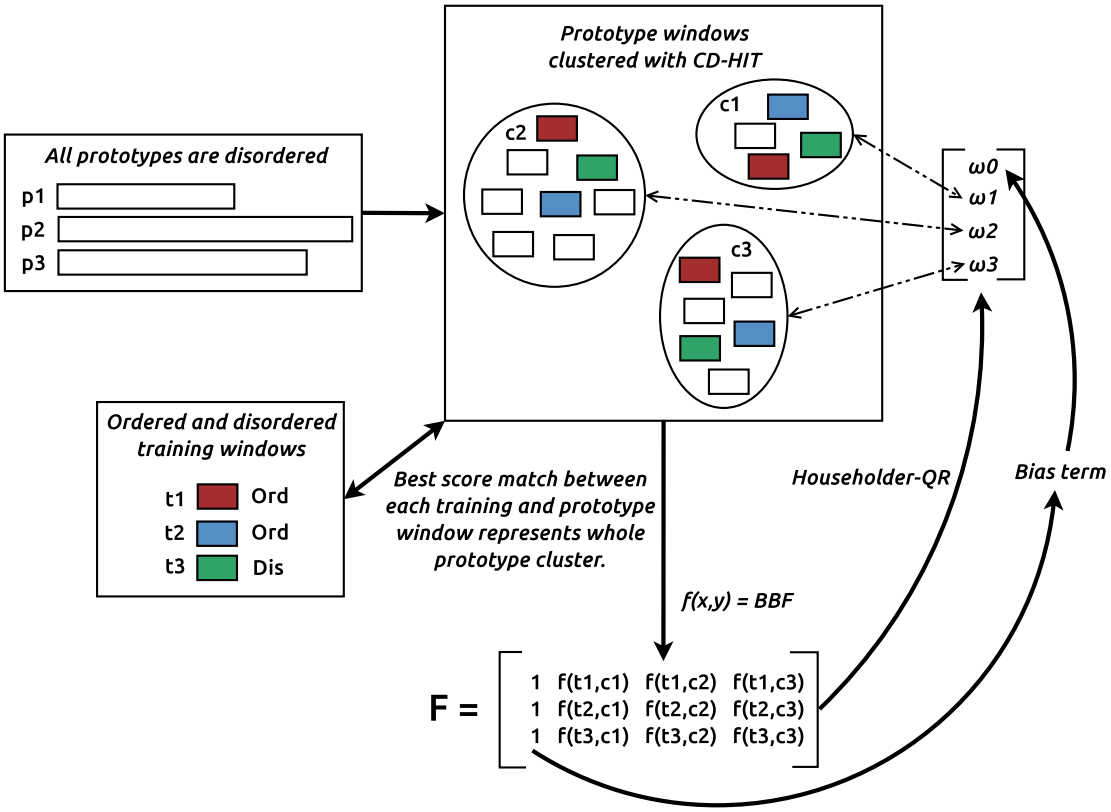


Figure 4.1: The CD–HIT trainer at work for one fold. The goal is to obtain a weight for each disordered prototype cluster in this case. The best matching prototype windows for the training windows ($t1$ to $t3$) are highlighted in the clusters. The weights are obtained by HOUSEHOLDER–QR decomposition of the **F** matrix.

A Python script was written to convert CD–HIT output and training results to a FASTA–delimited cluster file as shown in listing 4.2.

```
>Cluster_0 | 0.864122
AAAAAEAKKKAVKESSIRS
AAAAALPDAPIITPINSEW
AAAAEAKKKAVKESSIRSV
AAAAIHPNNYRRVIRALEI

[...]

>Cluster_2376 | 0.748644
VVGLSILLRGTVHEKLKWA
>Cluster_2377 | -3.430216
YFLPVGMTPLVVHGEIVEK
```

Figure 4.2: The new output format for training on CD–HIT clustered 19–residue windows, clustered at 40% identity. The cluster is identified with a FASTA–style header that includes the cluster number, with count beginning from 0, and the training weight assigned to the cluster. One such file is generated per cross–validation fold, amounting to ten files in total.

Since CD–HIT identity was now a tunable parameter, it effectively controlled the number of clusters (and therefore, number of weights) desired. The hypothesis was that the optimal distribution of disordered motifs would produce the best-weighted predictor and boost the probability excess scores when tested with the 80-protein and 203-protein test sets. The predictor was appropriately modified to take in this new training file format. The new prediction engine is shown in figure 4.3[1].

[1]Copyright 2004, Springer, Intelligent Data Engineering and Automated Learning – IDEAL 2004, Lecture Notes in Computer Science Volume 3177, 2004, pages 108–116, "Prediction of Natively Disordered Regions in Proteins using a Bio–basis Function Neural Network", Thomson R and Esnouf R, Figure 1; with kind permission from Springer Science and Business Media.

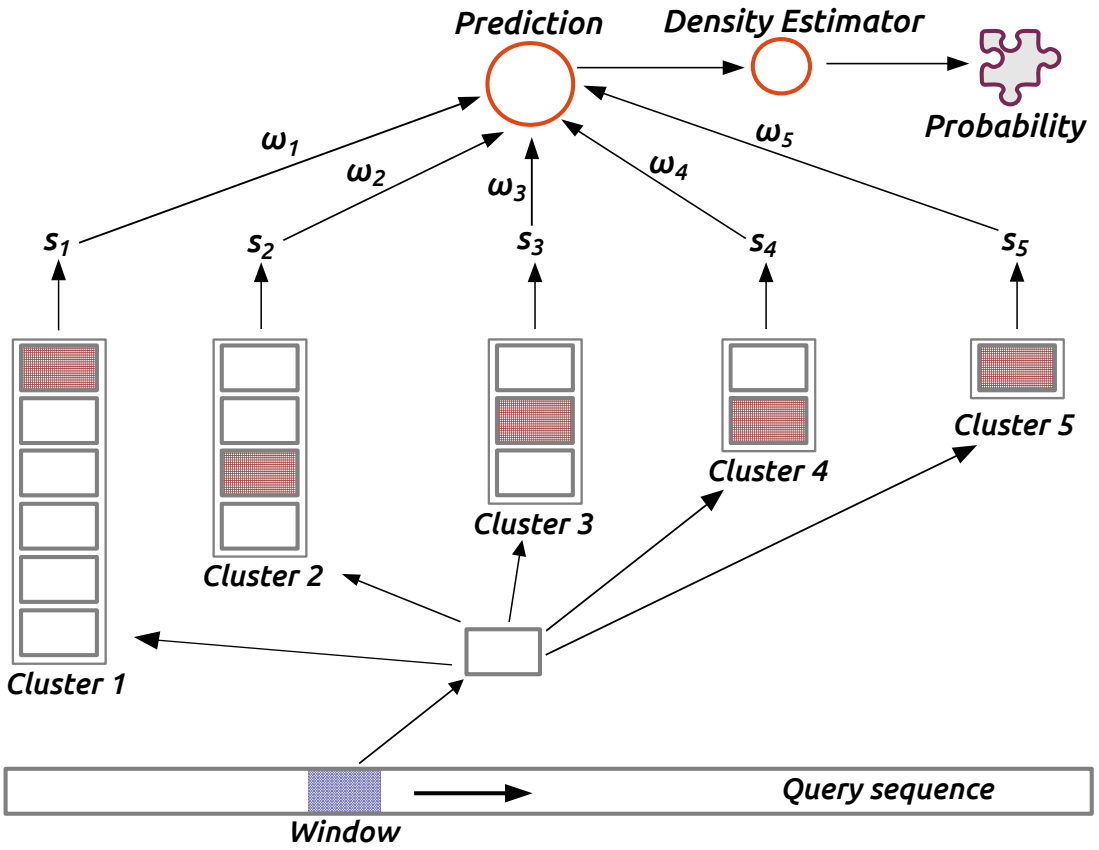


Figure 4.3: Prediction for a window in a query sequence based on CD–HIT clusters. The figure assumes that there are only five prototype clusters. Scores s_1 to s_5 are the highest BBF scores for the query window against any window in the respective cluster, which is then used to represent the whole cluster. They are multiplied by the weights for the clusters (ω_1 to ω_5) which were derived during training. Figure adapted from Thomson et al. (2004).

Note that the clusters are generally in descending order of size, as output by CD–HIT. For each cluster, the best matching window represents the whole cluster, and is multiplied by the training weight (ω) for the cluster. This process is repeated for each of the ten folds. The original sequence trainer (figure 3.3) is performing a similar job in that prototype sequences are essentially arbitrarily–clustered windows. With the CD–HIT method, however, redundancy (and therefore, over–training) is reduced since related windows will be clustered together, meaning that the clusters as a whole are able to represent the breadth of available disordered sequence space.

4.1.1.1 CD–HIT optimised RONN results

Table 4.1 shows how CD–HIT identity changes the number of clusters, and the remarkable recovery in probability excess scores.

CD–HIT Identity (%)	Number of Clusters	PE (%)
40	**2378**	**45.5**
50	9525	38.6
60	12048	20.4
70	20905	0.0
80	32454	0.0
90	44528	0.0
95	101082 (number of input windows)	0.0

Table 4.1: The effect of CD–HIT identity on cluster number when tested on 19–residue windows against the 80–protein test set. The value in bold–face (40%) produces the best PE. Note that for a 19–residue window, a single residue change gives an identity of (18/19 = 93%), so there can be no clustering at 95%.

This is a headline result! The CD–HIT optimisation at the minimum possible CD–HIT clustering identity (40%) brought the PE back up to 45.5% from the declustered case, which was remarkable for a raw, untuned predictor on a novel hypothesis, especially given that it is comparable to RONN 4.0's score of 51.1% (see chapter 3). A new disorder predictor was born at this point — eventually called *MoreRONN* to acknowledge its ancestry.

Based on this trend, it is possible that lower identity values might produce even higher PEs with fewer clusters, but CD–HIT cannot cluster below 40% identity and therefore, this could not be tested. However, the results did suggest significantly reducing cluster counts.

While this CD–HIT method seemed like a roundabout route to obtain similar PE scores to the old sequence–trained predictor without increasing prediction quality, it more than refined the concept, since the PE had held steady without dropping. This method of data partitioning now represented a fundamental conceptual difference in this predictor, *MoreRONN*, compared to RONN. Firstly, in the CD–HIT model,

only the highest–scoring window will be rescued, thereby increasing sensitivity of the prediction[2]. The entirety of the available sequence space is represented in this new clustering method, and over–training is reduced because highly redundant windows will be clustered together. Secondly, as CD–HIT cluster counts increase with increasing identity cutoffs, there is a point at which the PE drops, implying that there is an ideal number of clusters (and therefore weights) that can improve the sensitivity of the predictor.

However, as noted earlier, this CD–HIT partitioning method was a proof–of–concept, since it scores sequence identity, as opposed to the BBF's measure of similarity. It represented a pragmatic solution to test this data partitioning concept as quickly as possible in order to find out if there was any mileage at all in clustering.

4.1.2 Refining the clustering paradigm

The results of the CD–HIT clustering method showed that clustering over all possible overlapping windows addresses over–representation during training, thereby increasing sensitivity. The CD–HIT method in particular, however, has a few drawbacks[3]:

1. CD–HIT cannot cluster below 40% identity, which would reduce the cluster numbers even further, and possibly increase PE, based on the trend seen in table 4.1.

2. CD–HIT is weaker at clustering short-length sequences, especially when all sequences are of the same length (Li et al., 2006). The window lengths here were possibly too short for CD–HIT to produce the best clusters with ideal sequence distribution.

[2]Chapter 5 discusses surprisingly good results even with *one* cluster!

[3]It is not quite correct to refer to them as "weaknesses", as identity scoring does have applications elsewhere. For very long sequences, for example, identity might be a good proxy for similarity; this is, in fact, CD–HIT's primary use case.

3. The way CD–HIT measures sequence relatedness is by identity (Li et al., 2006), which is fundamentally different from the similarity measure of the BBF used in RONN (Yang et al., 2005).

While items 1 and 2 were intrinsic to CD–HIT and therefore not modifiable, item 3 was important to note as it deviated from the BBF scoring model used in the trainer and predictor. For the benefit of scientific rigour, the scoring model for clustering would ideally be consistent, even though it was quickly becoming apparent that the key to higher scores was to find an optimal number of clusters rather than tweaking parameters and scoring models. However, standardising the clustering method to use a similarity score can affect item 1 as well. Whilst CD–HIT generates clusters based on identity, a similarity–based clustering method might incorporate more windows into a given cluster[4], since the similarity space is larger than the identity space for a given amino acid (and therefore, amino–acid sequence). Thus, we would expect fewer clusters to represent the ensemble well. Biologically speaking, similarity–based clustering would group disordered windows by the closeness between the amino acid properties (as defined by the BLOSUM62 matrix, for example), rather than a simple binary (identical–or–not) text matching approach.

Since CD–HIT is free software under the GNU General Public License version 2, and since it is very quick at clustering large sequence sets, the initial thought was to modify its internal scoring model to incorporate the BBF instead of using an alternative, slower clustering method. After personal communication with the CD–HIT developers to discuss changing the algorithm to work on similarity rather than identity, it became clear that there would be far too much work involved. Therefore, a search was conducted to find a flexible-score clustering method in order to apply the BBF at the clustering stage which would, hopefully, run in a reasonable amount of time and allow finer–grained control over cluster numbers.

[4] Please refer to section 1.6.1.3 for a more detailed introduction to identity and similarity.

Chapter 5

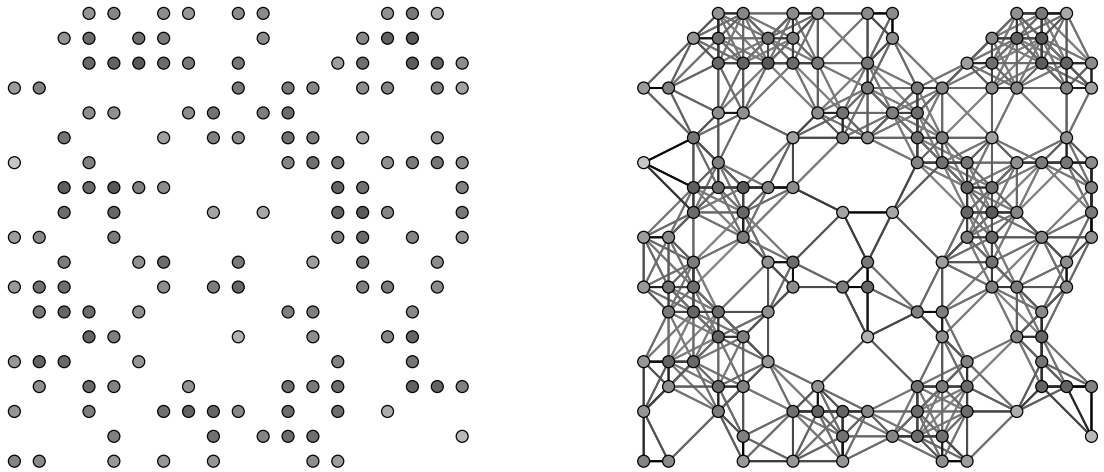
MoreRONN: A new disorder predictor

5.1 Graph clustering

Results from CD–HIT clustering of prototypes were encouraging but were unable to break through the PE barrier set by PONDR-FIT. However, the scores obtained with the CD–HIT method were supporting the use of prototype clustering as the correct approach to boost sensitivity. In order to find a suitable replacement for CD–HIT without the same limitations (see section 4.1.2), a change in clustering approach was required. In abstract terms, the CD–HIT algorithm is an example of a graph clustering method where each window (or node) is connected to a related window, based on CD–HIT identity. The graph is generated and *grows* on–the–fly, as interconnectedness is bootstrapped during the running of the algorithm through pairwise identity comparisons. CD–HIT is fast because nodes and node representatives assigned to one cluster are assumed not to belong to any other clusters, thereby greatly reducing the number of comparisons that need to be made between

windows[1]. This is a classic example of a **greedy** algorithm that sacrifices precision for speed, but works well in most use cases.

The Markov Clustering Algorithm (*MCL*), on the other hand, is a more comprehensive graph clustering method, whereby node interconnectedness is specified in advance in order to build a graph[2]. Depending on whether interconnectedness is expressed as a binary term or as a weight, the graph is unweighted or weighted (van Dongen, 2000; Enright et al., 2002). Figure 5.1 illustrates this point, as nodes are first scored against each other to generate either weighted or unweighted edges.



(a) Arbitrary nodes **(b)** Graph building

Figure 5.1: The *MCL* algorithm first builds a collection of nodes (5.1a) into a graph (5.1b) by pairwise–scoring the nodes to determine edge weights. Darker edges represent stronger links (closer nodes). Figure adapted from van Dongen (2000).

The algorithm then *reduces* the number of edges until a suitable number of clusters is obtained (see figure 5.2). It does this iteratively using a method called flow simulation, where flow is promoted where the current is strong (edge weights are stronger), and demoted where the current is weak (nodes are more distantly related) (van Dongen, 2000). Cluster size can be fine–tuned with the **inflation factor**, a parameter that is specified to *MCL* at run–time (see section 5.1.1).

[1]The clustering algorithm described in chapter 7 borrows inspiration from CD–HIT.

[2]The name "*MCL*" is given to both the clustering algorithm and the central executable. The technique was developed by Stijn van Dongen and is currently maintained at the European Bioinformatics Institute, near Cambridge.

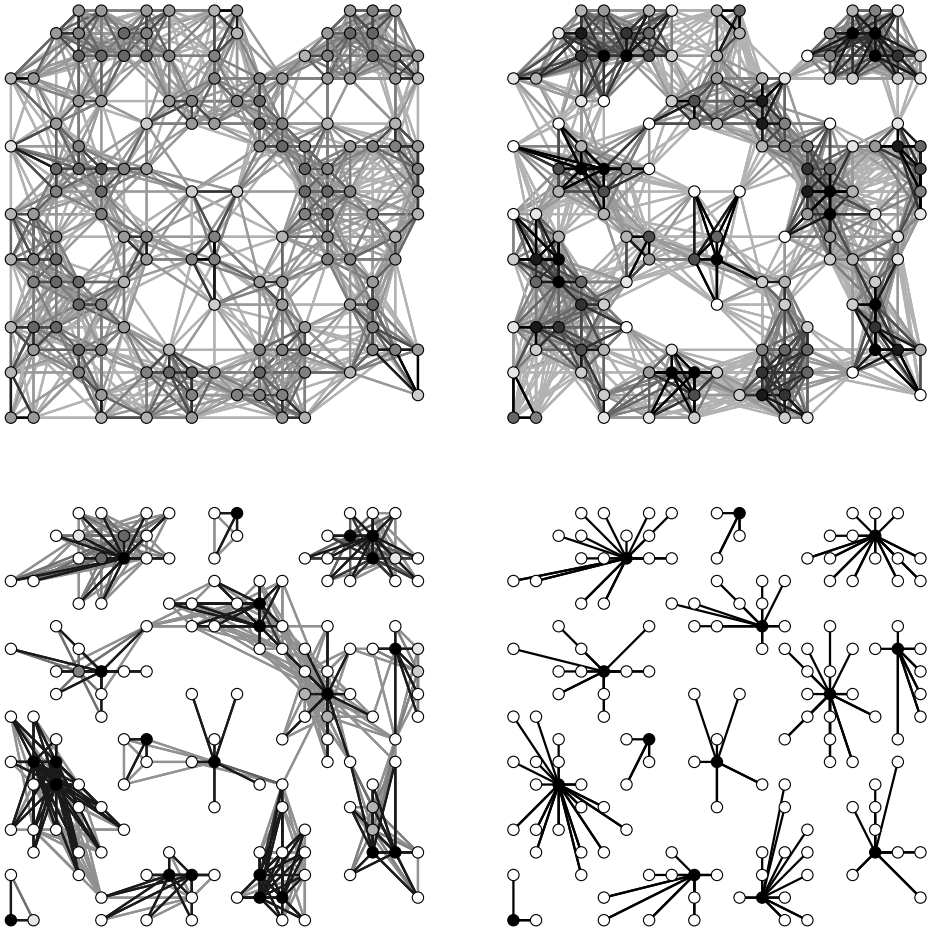


Figure 5.2: Using flow simulation, graph edges are iteratively reduced into clusters notionally anchored to a representative node. The bottom right node represents the limit obtained for this particular *MCL* process, that is, where flow is constant. The degree of shading denotes total incoming flow; a dark bond between a white node and a grey node indicates that flow is strongest towards the grey node, and weak towards the white node. Cluster size is controlled to a large extent by the inflation factor. Figure taken from van Dongen (2000).

Naturally, the advanced pairwise scoring step takes time compared to bootstrapped scoring in CD–HIT, but clustering is more precise. Furthermore, *MCL* offers a distinct advantage over CD–HIT in that edge weight scoring is completely customisable, and therefore, it is straightforward for our use case to apply the BBF scoring model to the graph building step[3]. *MCL* is also highly parallelizable, which compensates for its comprehensive clustering approach when deployed across mul-

[3]A similarity–based score in lieu of CD–HIT identity.

tiple cores.

5.1.1 The *MCL* Inflation Factor

Flow in *MCL* is simulated through the input graph by transforming it into a Markov graph (van Dongen, 2000). A Markov graph is essentially a state diagram with transition weights between states; for all nodes, the weights of outgoing arcs must sum to 1. An example Markov graph with three states is shown in figure 5.3.

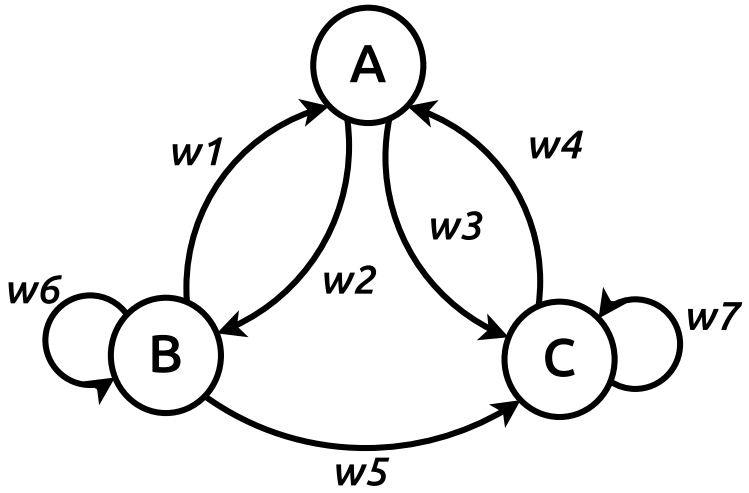


Figure 5.3: A simple Markov graph with three states, where transition weights between states are denoted $w1$ to $w7$. All outgoing weights for each node must sum to 1; for example, for node A: $w2 + w3 = 1$. Note that self–transitions are possible in the general case. Figure adapted from Bender-deMoll (2001).

In the *MCL* process, the Markov stochastic matrix associated with the graph is alternatively **expanded** and **inflated** until flow is constant. Expansion refers to computation of the powers of the Markov matrix, *i.e.* the usual discrete Markov process (Norris, 1998). It allows flow to become more homogeneous and dissipate across the entire graph[4]; with just expansion, the entire matrix would converge to the same equilibrium state and no contrasts or differences would remain (van Dongen, 2000). In order to maintain contrast, the inflation factor was introduced

[4]Both stronger and weaker edge weights tend towards the average.

to perform the opposite operation, that is, to strengthen strong weights and weaken weak ones. This has the effect of breaking the graph into different components, or clusters, where edges between clusters are eliminated during the inflation process, whilst edges within clusters are strong enough to withstand inflation. Therefore, supplying a higher inflation factor to *MCL* means that edges within clusters must be stronger in order for the cluster to stay together; this directly leads to smaller cluster sizes (and therefore, more clusters).

Thus, *MCL* represents a workable, drop–in replacement for CD–HIT in that it allows BBF scoring between windows and control of cluster size and number of clusters with the inflation factor.

5.1.2 Preparing Data for *MCL*

The *MCL* preprocessing code was custom–developed by the author entirely in C++ and is divided into a few stages. A driver program (`driver_master_preproc`) formats the data set (see chapter 3) into windows, creates the BBF pairwise score input file for *MCL* input and then invokes the *MCL* suite. Finally, it formats the final *MCL* clustering data into a CD–HIT–style output document, since this format is already readable by the training code. Figure 5.4 illustrates this data preparation routine.

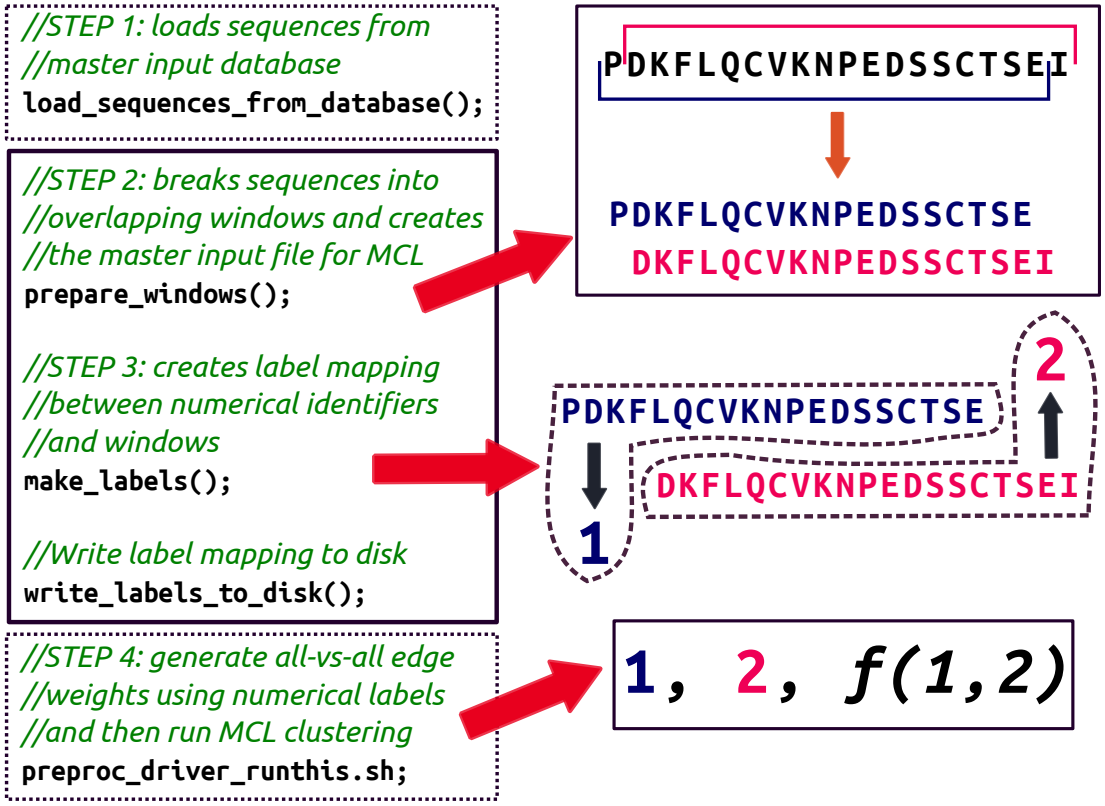


Figure 5.4: A simplified view of the `driver_master_preproc` pipeline, showing a sample data preparation as described by steps 1 to 3. Window size is 19 residues. The BBF function, $f(x, y)$, is used to generate weights. The shell script system call to `preproc_driver_runthis.sh` is required to parallelise the MCL stages properly, and is discussed in more detail in section 6.2.2.

The *MCL* executable requires a 3–column tab–separated file as input (edge x label, edge y label, $f(x, y)$)[5] and a mapping file between the numeric edge labels and their textual identifiers (in this case, the windows themselves). Steps 2 and 3 show the mapping between the windows in the preprocessor code that generate these two files required by *MCL*. Step 4 consists of a system call to a BASH script called `preproc_driver_runthis.sh`, which generates the edge weights and runs all external *MCL*–specific steps. It is discussed in detail in section 6.2.2 in chapter 6 in the context of order prediction. Note that window size is maintained at 19 residues as a lay–over from RONN and CD–HIT tests. Experimentation with different window sizes is discussed later in this chapter in section 5.2.

[5]Where $f(x, y)$ is the BBF score (see equation 3.2 in chapter 3).

Once the `driver_master_preproc` pipeline runs, it produces a cluster file that is then suitable for input for the trainer as before. Since the file mirrors the CD–HIT format, the training can proceed seamlessly without any changes to the code downstream. The training process is shown in figure 5.5.

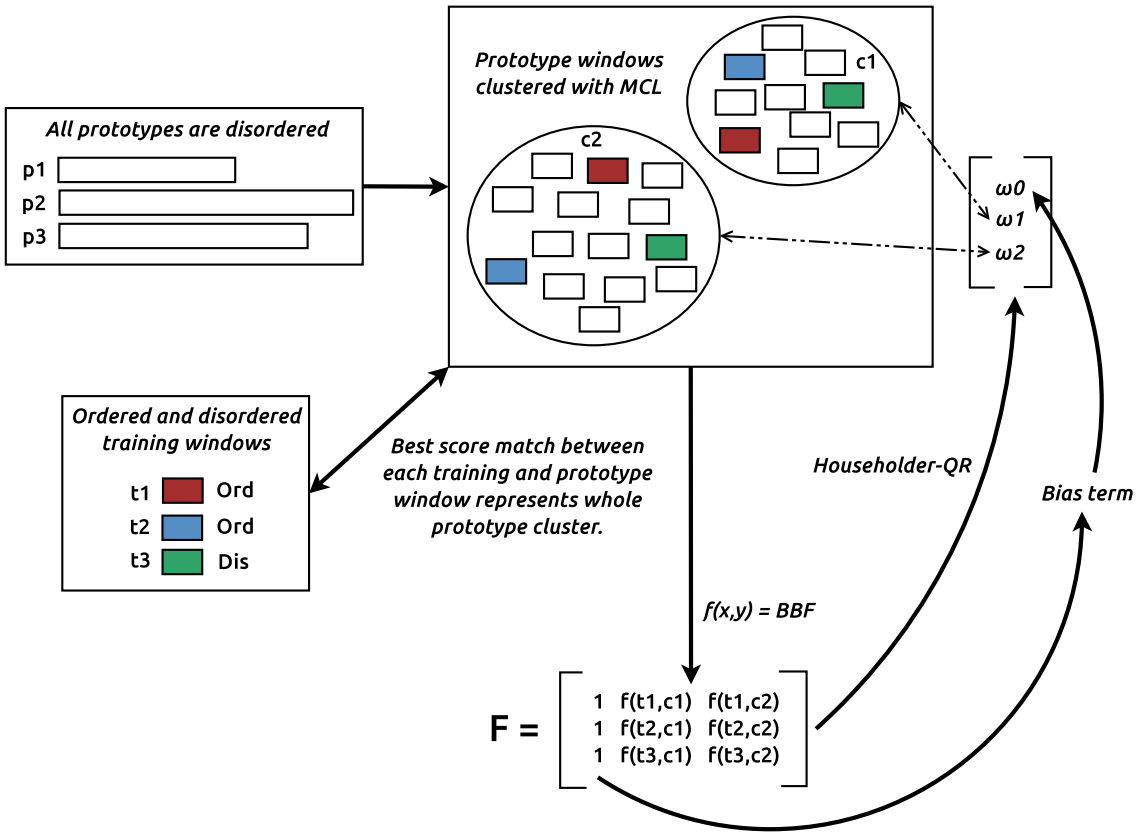


Figure 5.5: The *MCL*–powered *MoreRONN* trainer at work for one round. The goal is to obtain a weight for each disordered prototype cluster as before with the CD–HIT version. The best matching prototype windows for the training windows ($t1$ to $t3$) are highlighted in the clusters. Note that *MCL* is able to produce fewer clusters than CD–HIT, leading to fewer ω weights.

Prediction is identical to the CD–HIT method since the cluster output from *MCL* is formatted to CD–HIT–style output. Figure 5.6[6] illustrates prediction for one round, using *MCL*.

[6]Copyright 2004, Springer, Intelligent Data Engineering and Automated Learning – IDEAL 2004, Lecture Notes in Computer Science Volume 3177, 2004, pages 108–116, "Prediction of Natively Disordered Regions in Proteins using a Bio–basis Function Neural Network", Thomson R and Esnouf R, Figure 1; with kind permission from Springer Science and Business Media.

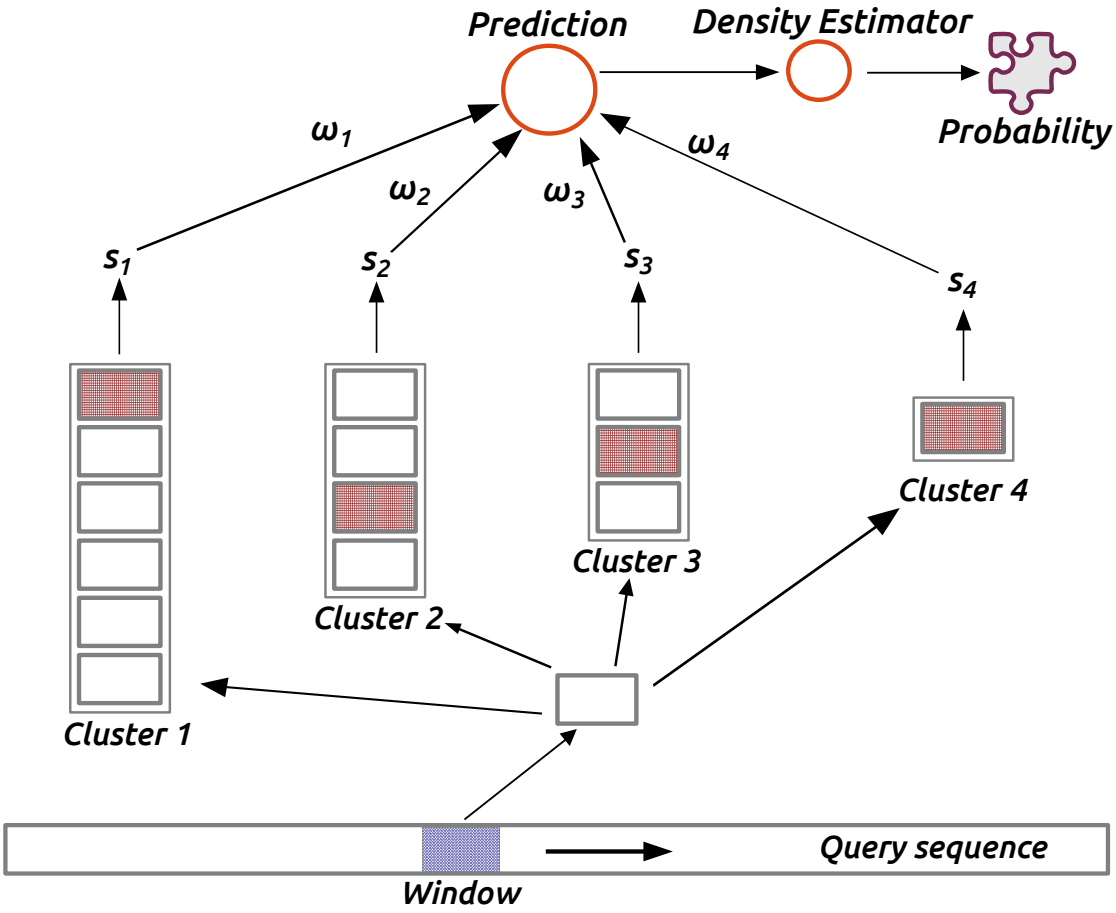


Figure 5.6: Prediction for a window in a query sequence based on *MCL* clusters. This workflow is identical to that of the CD–HIT–trained *MoreRONN*, but note again that there are fewer *MCL* clusters. The figure assumes that there are only five prototype clusters. Scores s_1 to s_5 are the highest BBF scores for the query window against any window in the respective cluster, which is then used to represent the whole cluster. They are multiplied by the weights for the clusters (ω_1 to ω_5) which were derived during training. Figure adapted from Thomson et al. (2004).

Upon running the `driver_master_preproc` pipeline, immediately, there was a problem with the large number of edge–weight inputs, causing slow performance and non–results (PE = 0%). The cause was determined to be an artefact of the BBF itself; the worst matching windows will still score relatively high with the BBF cutoff at 0.85 (where 1.0 is a perfect match) due to the BLOSUM62 substitution score and the short length of the windows. Therefore, given that most of the windows matched poorly against each other, this caused an overload of edge weights in the 0.85 range, confounding *MCL*. The twofold solution was to first reject any

edge weight below an arbitrary cutoff, and then to make use of built–in **contrast expansion** functions in *MCL* to resolve close weights more clearly. As there were 96,021 19–residue disordered windows to be clustered, this gave a possible ~9 billion edge–weights (including self–matches), which was then doubled within *MCL* (as part of its implementation), producing 18.4 billion edge weights, which was far too many for *MCL* to handle. The 0.92 minimum BBF cutoff produced just over 75 million edge weights (culling 99.5% of the edges), by comparison[7]. To improve contrast, a $-log_{10}$ transformation[8] was applied to every edge weight that was higher than the BBF score cutoff (arbitrarily chosen to be **0.92**).

5.2 Results with *MCL* clustering

Since *MCL* is much more comprehensive than CD–HIT's greedy approach, the clustering steps took much longer to run than the CD–HIT method. The very first run, using 19–residue windows, generated 432 clusters; once trained, there was an immediate, dramatic improvement in performance with a PE of 65.7% for the 80–protein blind test set[9]! This represents an 8% improvement on PONDR-FIT. At this point, a lengthy round of optimisation was conducted, and the results are shown in table 5.1.

[7]Note that self–matches were always performed to avoid omitting windows completely, in cases where all of a window's unique matches might be below the cutoff. The self–match would give a BBF score of 1.0, safely including every window in at least one pair.

[8]A solution suggested by Stijn van Dongen, author of *MCL*.

[9]Whilst rare, accidentally correct parameter combinations are heartening.

Window size	MCL Inflation	Clusters	Min. BBF score	PE (%)
19	2.8	1	0.95	48.9
19	1.1	6	0.94	59.9
19	1.2	20	0.94	58.9
19	1.5	22	0.92	61.3
19	1.53	72	0.93	61.3
19	1.4	310	0.94	52.3
19	3.4	408	0.95	64.4
19	3.5	432	0.95	65.7*
19	4	708	0.95	58.4
19	1.6	1150	0.93	47.4
19	1.4	1868	0.945	44.3
19	1.4	6210	0.955	39.4
15	**1.5**	**135**	**0.92**	**66.5**
15	3.5	236	0.9	60.6
15	3.6	357	0.9	62.7
15	3.8	414	0.9	66.4
9	1.8	38	0.92	63.4
9	1.9	232	0.92	62.5
9	2.5	421	0.91	64.6
9	2.6	558	0.91	64.3

Table 5.1: Initial optimisation rounds conducted on *MCL*–enhanced *MoreRONN* with the $-log_{10}$ contrast expansion function applied in each case. Note that the reported minimum BBF score cutoff is before the contrast expansion. The predictor was tested against the 80–protein blind set, and the initial result (65.7% PE) is marked with an asterisk. The best result obtained is highlighted in bold face.

The first result of PE 65.7% is marked with an asterisk, whilst the top score obtained (66.5%) is highlighted in bold face. Window lengths were altered and tested with various *MCL* inflation factors, which produced different numbers of clusters. The BBF α–value was kept the same (0.1) as in RONN. The peak window length was determined to be **15 residues**, which gave a PE of 66.5% when tested against the 80–protein test set[10].

Whereas the clustering and training steps are now in sync insofar as the use of BBF scoring, the main contributing factor to the better PE, based on table 5.1

[10]This corresponds to a sensitivity of 74.1% and specificity of 92.4%.

was *MCL*'s ability to produce a small number of clusters. In contrast, the lowest CD–HIT identity (40%) gave 2378 clusters and a PE of 45.5% for the same test set (see table 4.1). A pattern quickly emerges in the 19–residue window test runs; smaller clusters (smaller inflation parameters) with a lower BBF cutoff to introduce as many edge weights as possible tend to produce better PE scores[11]. Therefore, the decision was made to experiment with smaller window sizes to see if the predictor maintained its strong predictive power. Note that this is biologically very important, as a shorter–window predictor that was equally good or better than the current best would lend support to the hypothesis that disorder can be encoded within very short regions!

As window sizes were adjusted, the inflation parameter was targeted to produce similar numbers of clusters. The optimal 15–residue window size (with a lower BBF cutoff) was a desirable result both for reducing the disorder window size and for including as many graph edges as possible for *MCL*[12]. Biologically, this implies that disorder can indeed be encoded at shorter lengths, and perhaps novel predictors of the future can be trained on even shorter window lengths, which opens the door to exciting possibilities. The 66.5% PE version of *MoreRONN* (bold face in table 5.1) is compared to RONN and PONDR-FIT in table 5.2 and in the pyramid plot in figure 5.7.

[11] It is a bell–curve, however.

[12] The data set contains 107,413 disordered 15–residue windows, as compared to 96,021 19–residue windows.

Predictor	Sensitivity	Specificity	PE
RONN 3.1	58.5%	85.3%	43.8%
PONDR-FIT	61.0%	**93.7%**	54.7%
RONN 4.0	66.0%	85.2%	51.1%
MoreRONN	**74.1%**	92.4%	**66.5%**

Table 5.2: Differences in sensitivity and specificity for *MoreRONN* as compared with the other three predictors for the 80–protein test set. Again, top scores in each column are highlighted in bold face. Note that PONDR-FIT is slightly more specific than *MoreRONN*, but the latter's increased sensitivity outweighs this slight weakness and is the principal contributor to its higher PE score.

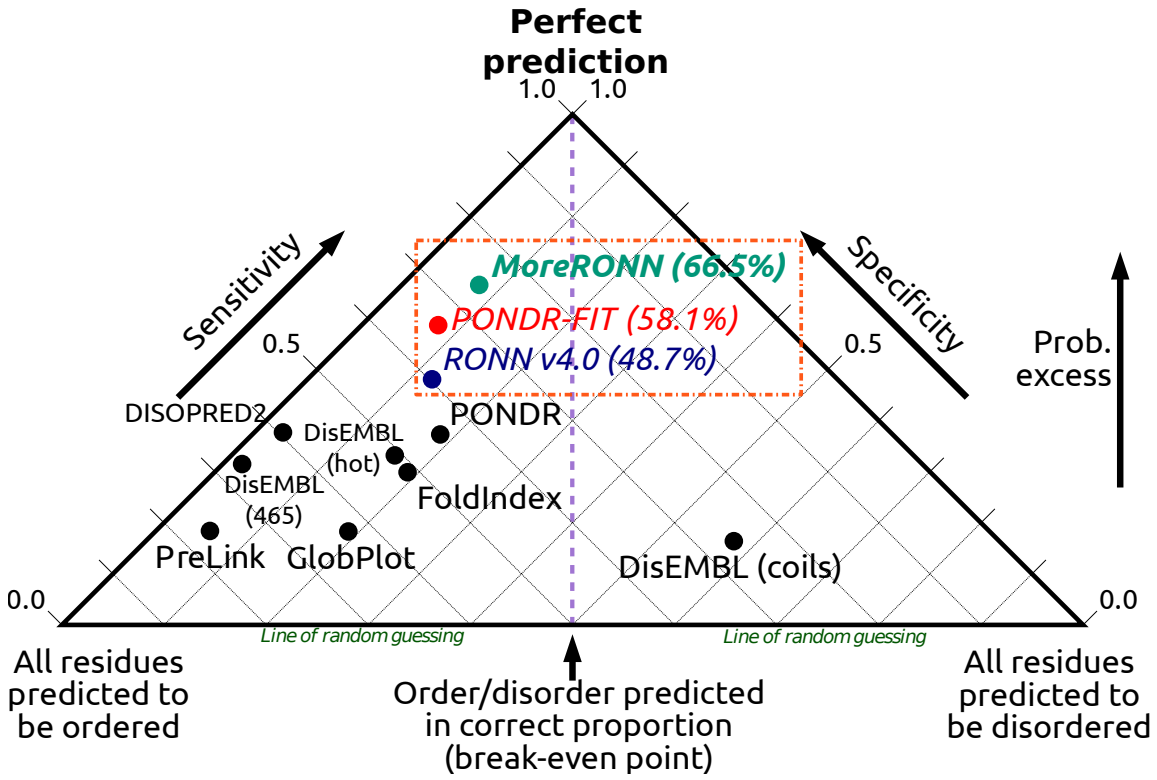


Figure 5.7: Pyramid plot showing *MoreRONN*'s improved probability excess as compared with the other predictors for the 80–protein blind test set. Note that *MoreRONN*'s data point is closer to the centre axis of the pyramid, showing that it is a more balanced predictor than either RONN or PONDR-FIT, and has higher sensitivity than all other predictors.

For 15–residue windows, however, the 66.4% PE result with 414 clusters, whilst slightly lower, is arguably better for real–world and novel predictions as the BBF cutoff is more lenient (0.9 as opposed to 0.92), meaning more edge–relationships were factored into the clustering and therefore, the clusters are better represen-

tations of disordered sequence families. As window size came down, the cutoffs became more important to balance between sensible cluster sizes and run time. Indeed, for 9 residue windows, data sets with cutoffs below 0.91 caused *MCL* to fail in its execution.

Note that the "class–ratio" factor has been omitted thus far in the discussion. This is because it no longer made an impact with *MCL*; indeed, the ratio of order to disorder in the training windows did very little to alter the results, and therefore, the ratio was kept as close to 1:1 as possible[13]. This is a *very* positive result as it eliminates a user–editable parameter and indicates that the *MoreRONN* algorithm is much more robust.

MoreRONN's increased sensitivity directly contributes to its decisiveness at transition points between order and disorder, as shown in figure 5.8.

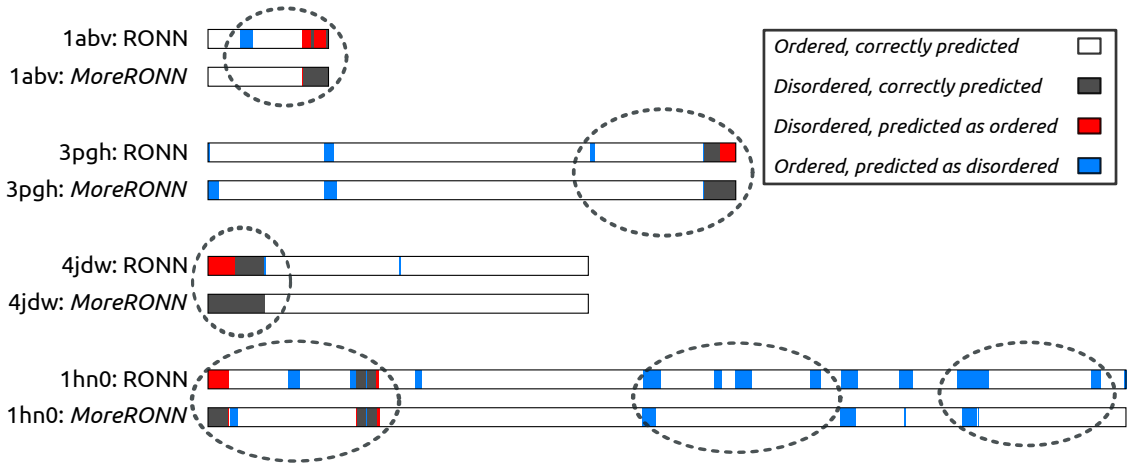


Figure 5.8: Four examples from the 80–protein test set, clearly showing *MoreRONN*'s decisiveness at transition points between order and disorder. The ovals indicate regions where the prediction improves dramatically (blue and red regions are reduced).

MoreRONN shows a particular improvement in predictions at the termini compared to RONN, which is important for construct design for crystallography; these regions can be cleaved to improve the chance of tight crystal packing, and knowing

[13]For interest's sake, the "class–ratio" factor for a 1:1 ratio is 22, and was kept at this value for the remainder of *MoreRONN* benchmarking.

where disorder ends and order begins, or vice versa, is key to ensuring that the correct number of residues is cleaved. To test *MoreRONN* in a wet lab setting, it was made available internally within The Division of Structural Biology. Figure 5.9 from the Mancini group shows RONN and *MoreRONN* predictions overlapped with that of a secondary structure predictor, TALOS+, which is a neural network that uses NMR chemical shift data to predict secondary structure (Shen et al., 2009). The query protein in figure 5.9 is an ubiquitin–binding domain (the sequence has been withheld at the request of the Mancini group). Note how *MoreRONN* is more sensitive than RONN at the N–terminus (to the left of the graph), and that it predicts a dip in per–residue scores towards the middle, correlating with a TALOS+ secondary structure prediction in the area. Even though *MoreRONN*'s per–residue predictions do not indicate ordered residues in this area, the dip still indicates that the *MoreRONN* prediction for this region is tending towards order[14].

At this point, since 15–residue windows produced the best scores, all subsequent training and prediction tests should be assumed to be run with 15–residue windows unless otherwise specified.

[14]Since the per–residue scores are ≥ 0.5, the *MoreRONN* result alone does not indicate the penchant for order in the middle of the protein. Combined with the TALOS+ result, however, the dip in *MoreRONN* plot suggests a different story!

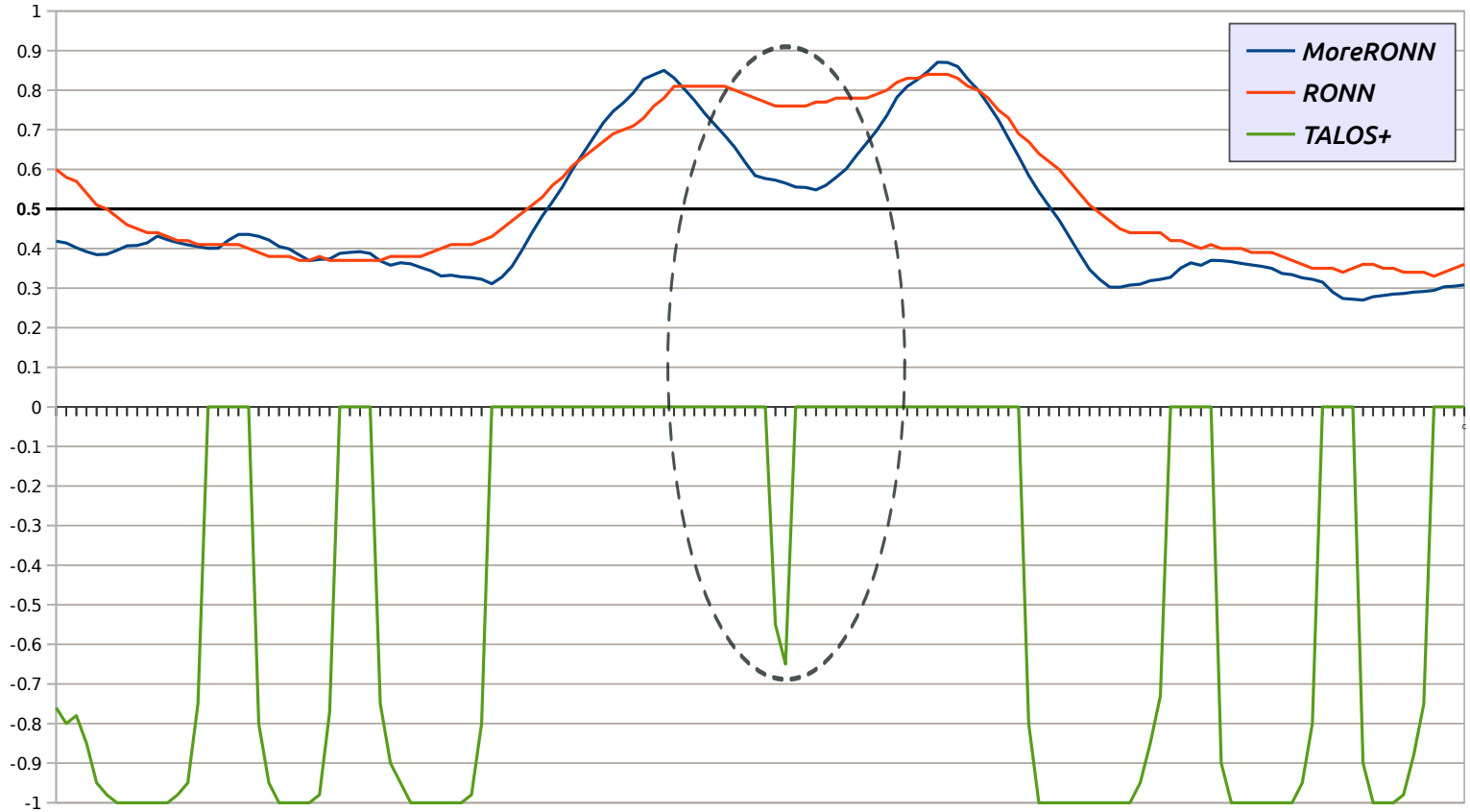


Figure 5.9: Overlapped RONN, *MoreRONN* and TALOS+ predictions for an ubiquitin–binding domain. Negative spikes in TALOS+ predictions should correlate with predicted regions of order (per–residue scores < 0.5) in the disorder predictions. *MoreRONN*'s prediction for the stretch of order in the middle of the protein denoted by the dashed oval, whilst still not predicting order, still displays a telltale dip that is almost absent in RONN. Also note *MoreRONN*'s sensitivity at the N–terminus, compared to RONN's. The sequence runs from N to C left–to–right, but amino acid labels have been omitted as per the request of the Mancini group.

5.2.1 *MoreRONN* with a single cluster

One *MoreRONN* result remains to be addressed; that is, the remarkably high PE (48.9%) obtained with just one cluster. In this case, the clustering method (CD–HIT or *MCL*) and associated parameters are moot points; as clustering collapses, the neural network also collapses; essentially, the best score to any prototype above a certain threshold indicates disorder. The PE was obtained with a sensitivity of **50.1%** and a specificity of **98.8%**; since disorder is only controlled by a single weight, sensitivity drops, but this also implies that more ordered residues are correctly predicted as being ordered, thereby raising specificity. This simplification still yields predictions comparable to RONN and this version of *MoreRONN* still outperforms most predictors!

5.3 Improving Predictor Computation Performance

Having achieved the best possible quality of training, optimising predictor speed and cleaning up its code became top priority. The "quick fix" to get the cluster–based predictions working involved extracting windows and storing them individually for input into the trainer, since this would require the least amount of code change to the trainer. However, since 89% of the windows are redundant, the ~tenfold increase in file size (and disk I/O) is unnecessary. Therefore, a new database file format was designed which fits all cross–validation and training weight information required by the predictor into a single file without redundant copying of windows. The ten means and standard deviations calculated by the probability density function are appended at the end in ten **==PDF==** data blocks. This effectively reduces the disk footprint by 90% and the predictor runs much more quickly with the reduced I/O. The new database file format is shown in figure 5.10.

```
>Cluster_0|0.864122 0.877913 0.855288 0.845104
0.916817 0.886930 0.897303 0.875089 0.737355 0.905719
AAAAAEAKKKAVKES        9
AAAAALPDAPIITPI        1
AAAAEAKKKAVKESS        7
AAAAIHPNNYRRVIR        0
AAAAITWVPKPNVEV        2

[...]

>Cluster_178|-3.43021 -1.04093 0.147678 -2.52621
4.27698 0.248925 -2.43968 -1.35250 0 1.834476
YFLPVGMTPLVVHGE        8

===PDF_0===
0.3000422698663727
0.5282472479487903
0.03154020860087205
0.07996617618159942

[...]

===PDF_9===
0.31276037479218716
0.5440082271694008
0.028971699196707355
0.08305549483297422
```

Figure 5.10: New single–file database format for *MoreRONN*. The cluster FASTA–header now contains all ten training weights delimited by spaces, one per cross–validation round. Training weights have been truncated for typesetting clarity. The cross–validation index for each window is now printed next to the window; the data are separated out during prediction into their separate rounds.

Cross-validation indices are assigned randomly after the clustering step[15], as the database file is being assembled. *MoreRONN* then uses this new format for training. The PE of 66.5% that was reported in figure 5.7 and table 5.2 was derived from the cross-validation indices in this new database format, along with the bug fixes

[15]Initially, clustering was run once for each round. However, it makes more sense to cluster the entire (100%) of the windows first and then partition them later. This can cause slight changes in the results as discussed in the text.

to the training code discussed in section 5.3.1. Rearranging the indices resulted in fluctuations anywhere between 66.0% and 66.5%; this is because, whilst each part of the tenfold training and cross–validation was run separately, merging the clusters is not trivial, and causes fluctuations. The score can be optimised by trying various random number seeds to alter the partitioning of the data, but as the PE is not dramatically altered, and since there is no change in the science, it is left as a fine tuning exercise, and 66.5% is reported as the best probability excess with this new database format. The key point is that *MoreRONN* now exists as a single executable file with a single flat–file database, meaning it has a small footprint, is more portable, and executes much faster than RONN.

5.3.1 Predictor Optimisations and Bug Fixes

The following bug fixes and optimisations were carried out in the predictor to make it more friendly for users:

1. A progress bar was added to keep users informed about the state of the predictions. On a modern computer, *MoreRONN* (which is single–threaded) predicts at a rate of 150 residues per second[16].

2. *MoreRONN* now handles multiple FASTA–delimited query sequences and produces a single output, where predictions are appended and delimited by their FASTA headers.

3. *MoreRONN* is now optimised to be easily convertible into a dæmon for a web–service. It is also much more easily ported to different operating systems and architectures than its predecessor. Given that it is single–threaded, multiple

[16]This is not a trivial result, but the detailed mechanics are outside the scope of this thesis. In summary, before the database was consolidated, the old predictor running on the *MoreRONN* data sets was approximately ten times slower. The new data set, coupled with a reordering of data loading steps in the predictor, quickly and cheaply improved performance.

instances can be run to serve simultaneous prediction requests through a web portal.

The current version of *MoreRONN* is 4.6. The program is run in exactly the same way as RONN; the user simply need provide a FASTA–delimited sequence, or a set of sequences, and invoke as shown below:

```
[user:machine ~]$ moreronn_46 query.faa
```

The prediction is written to standard output, which can be redirected to a file by the user. Note that the legacy RONN executable accepts sequences without FASTA headers through the use of an extra argument during invocation of the program, but this feature has been removed in *MoreRONN*. The current version aborts if it is unable to detect a header in a provided input file. It was decided that enforcing the FASTA format is a better design decision, especially for multiple sequence input or a web service interface[17].

5.3.2 Bias terms and α in the bio–basis function

At this point, whilst preparing to deploy *MoreRONN* as a public web service, an important omission was noticed in the neural network. Recall from section 3.3.3 that the bio–basis scores for each prototype–training window pair populate a matrix:

$$\mathbf{F} = \begin{pmatrix} 1 & f_{11} & f_{12} & \cdots & f_{1N} \\ 1 & f_{21} & f_{22} & \cdots & f_{2N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & f_{M1} & f_{M2} & \cdots & f_{MN} \end{pmatrix} \tag{5.1}$$

[17] If a header is not provided with single–sequence input to the web service, it is easy to generate one at run–time. If multiple sequences were provided, they would need to be demarcated somehow anyway.

The first "column of ones" corresponds to ω_0, which is the bias term in the main classifier function:

$$\hat{y} = \sum_{n=1}^{N} \omega_n f(x, y) + \omega_0 \tag{5.2}$$

The bias term is not necessary if the matrix is positive–definite (Kecman, 2001), but if it is not possible to tell whether this is the case, the bias term should be included (Huang et al., 2004). In this application, whilst delving into the linear algebra was outside the scope of the thesis and therefore the positive–definite state of the matrix was not verified[18], it was deemed necessary to include the bias term to remain consistent with Yang (2004), the principal publication that discusses RONN's mathematics. This proved to be a simple alteration of the EIGEN matrix to seed the column. In the output database format, a new FASTA–header was created with the ten bias terms, and the predictor was altered to add ω_0 for each round as per the equation.

The addition of *MCL* to *MoreRONN* improved sensitivity greatly, but introduced two new parameters in the form of the minimum BBF cutoff and the $-log_{10}$ contrast expansion function. The high cutoffs in table 5.1 were problematic, in that a slight change to the cutoff produced much larger fluctuations in numbers of edges, thus requiring *MCL* inflation factors to be very finely tuned to keep cluster sizes under control. In other words, it was difficult to control the granularity of the BBF in its current form. This opened the possibility of altering the α constant, which had been set to 0.1 thus far, and seemed to be an arbitrary choice[19]. However, α does have an effect on the BBF function in that it controls the half–life. Larger α values shorten the half–life, and therefore force more specific matching between two windows. In order to determine α's effects on clustering, the obvious starting point in this appli-

[18]The reason for omitting this term, if there is one, is also lost.
[19]In another application of the BBF, Yang et al. (2006) use 10.0 as the α value.

cation was to set α to 1.0 to provide a larger spread of BBF scores, thereby removing the need for the contrast function and lowering the BBF cutoff substantially. Recall that the BBF function is:

$$f(x,y) = \exp\left(\alpha \frac{s(x,y) - \beta_y}{\beta_y}\right) \tag{5.3}$$

If minimum BBF cutoff = 0.92 and α = 0.1, then:

$$0.92 = \exp\left(0.1 \frac{s(x,y) - \beta_y}{\beta_y}\right) \tag{5.4a}$$

$$-0.0833 = 0.1\left(\frac{s(x,y)}{\beta_y} - 1\right) \tag{5.4b}$$

$$-0.833 = \left(\frac{s(x,y)}{\beta_y} - 1\right) \tag{5.4c}$$

$$\therefore \quad \exp(\text{new } \alpha \times -0.833) = \left(\frac{s(x,y)}{\beta_y} - 1\right) \tag{5.4d}$$

$$\exp(1.0 \times -0.833) = \mathbf{0.434} \tag{5.4e}$$

Thus, **0.434** is the new minimum BBF cutoff when α = 1.0, and since the self–match score (β_y) will still be 1, the contrast between good and bad window matches is sharply expanded[20].

Immediately, this brings into question the two α values in *MoreRONN*, namely the clustering α and the training/prediction α. A series of tests was conducted with α = 0.1 and α = 1.0 for clustering, with varying constants for training[21], using the training data that produced the best scores thus far. Results are shown in table 5.3.

[20]Note that the number of clusters stayed constant with this change, remaining at 135.
[21]It should be assumed that the trainer and predictor α's are always in sync.

Clustering α= 0.1	
Training α	**PE (%)**
0.005	64.2
0.01	64.2
0.02	64.2
0.05	64.3
0.1	*66.4**
0.2	64.6
0.5	64.8
1	65.6
5	64.8
10	45.5

Clustering α= 1.0	
Training α	**PE (%)**
0.005	63.9
0.01	63.9
0.02	63.9
0.05	64
0.1	64.1
0.2	64.3
0.5	64.9
1	*65.6**
2	66.8
3	67.5
4	**67.7**
5	67.6
6	67
10	65.6

Table 5.3: Varying α values for training as tested against the 80–protein test set. Asterisk values in italics are baseline values, *i.e.* when both α's are the same. The best score is highlighted in bold face.

The predictor is now even better with a clustering α = 1.0 and a training α = 4.0, giving a PE of **67.7%**! In clustering the goal is to produce larger clusters of related sequences. Keeping clustering α small allows more distant matches to score highly and to be clustered together. However, in training and prediction, it is important to isolate the best matches and reduce noise; in these cases, training, validation and prediction α values should be set higher than clustering α (but equal to each other), which reflects the trend observed in table 5.3. The improvement in *MoreRONN* PE is illustrated by the pyramid plot in figure 5.11. The axes have been adjusted to show *MoreRONN*'s improvement more clearly.

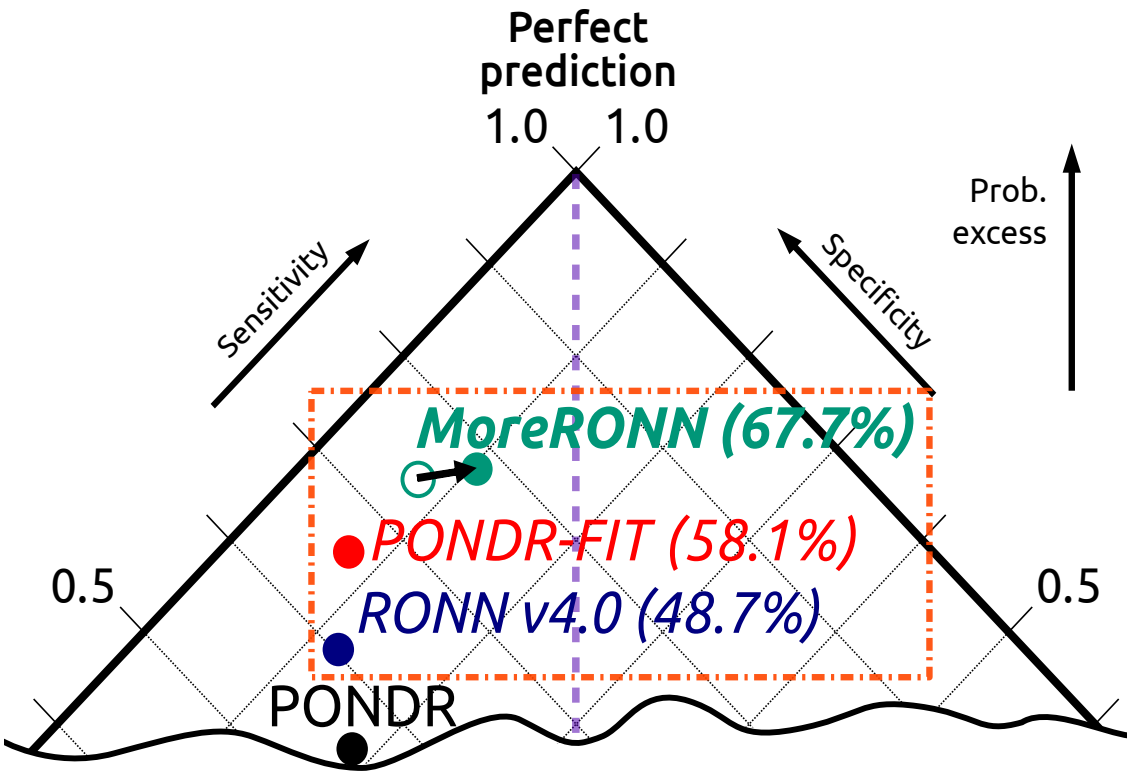


Figure 5.11: Top section of the pyramid plot showing the improvement in *MoreRONN* PE with the new clustering α value of 1.0, tested on the 80–protein blind test set. The open green circle represents the old PE (66.5%). Note that *MoreRONN*'s data point is even closer to the centre axis of the pyramid than before, showing that its sensitivity has improved further. The pyramid's axes have been adjusted for clarity.

The hypothesis here is that the increased contrast available to *MCL* with a higher clustering α outweighed the benefits of applying the $-log_{10}$ function in the previous iteration. Note that the scores are much flatter when clustering $\alpha = 1.0$, which supports this theory and also showcases the improved robustness of the *MCL*–powered neural network. This version of the predictor is known as *MoreRONN* version 4.6, and barring any further development, will be the published and released version.

5.4 Discussion

MoreRONN is now best–in–class while maintaining its pure prediction based solely on known disordered sequences without differentiating between different

classes or lengths of disordered regions. The window size has been reduced from 19 to 15 residues, and all signs point to further improvement with 9 residues, and perhaps even smaller. The trend is that sensitivity increases with the smaller windows, but the predictions tend to be noisier, which reduces overall PE at a certain point. However, the scope is there, especially with the changes to α values providing better contrast between high and low BBF scores, to improve on the 9–residue predictor. After all, smaller window lengths leading to improved PE scores has a biological significance; it lends support to the fact that disorder can be encoded in fewer residues!

It is interesting that the application of *MCL* to *MoreRONN* immediately improved robustness and flattened out the algorithm's performance across myriad different inflation factors, cluster sizes and all other parameters. It also removed the need for the uneasy "class–ratio" factor; indeed, the even distribution of order and disorder in the training and validation sets is a sign of *MoreRONN*'s robustness. Another positive side effect is the best scores are now obtained with a balanced cost function (0.5 as opposed to 0.53; see section 3.3.4), effectively removing it. This implies that the new algorithm is resistant to training bias without stopgap measures. After all, an algorithm that performs well in the general case without relying too heavily on fine–tuning parameters is a more scientifically valid solution[22]!

Both CD–HIT and *MCL* provided a new tunable parameter to control how many clusters were produced, which controlled how many weights were generated. Effectively, this reduced the main scientific question to "how many clusters?" It was clear that too many clusters weighed down the predictions as in the declustered case (see chapter 3.7), and that strong disorder predictions could be obtained with remarkably few ($\sim$100) clusters. Indeed, running *MCL* with just one cluster produced a PE comparable to RONN, although it is noteworthy to mention that all these factors

[22]In the author's humble opinion.

are entirely dependent on the quality and size of the data set. The achievements detailed in this thesis simply try to stretch the predictor to the limit when it comes to using the PRAXIS–curated data set effectively. On that note, however, *MoreRONN* may now be so good that it can be used to triage the data set itself — perhaps being able to highlight errors in training and prototype sequences which can then be backed up from the literature or annotated in DisProt.

MoreRONN has high scientific value for predicting disorder in novel sequences, where meta–predictors such as PONDR-FIT might be expected to struggle. The results from the Mancini group prediction (figure 5.9) corroborate this thought, and initial expression results from other groups at STRUBI are showing high levels of expression with *MoreRONN*–designed constructs as opposed to those designed with RONN[23].

5.4.1 Recent improvements

As with any evolving software tool, *MoreRONN* was æsthetically improved subsequent to the writing of this chapter to improve the readability of the results. Whilst per–residue score output is easy to plot, it is often helpful to survey the disordered state of the protein "at–a–glance." For this purpose, *MoreRONN* now writes out a header as shown in the snippet in figure 5.12.

[23]Unfortunately, this work is in a very early stage and as such, data have been withheld.

```
>TEST_SEQUENCE
>MFNSMTPPPISSYGEPCCLRPLPSQGAPSVGTEGLSGPPFCHQANLM
>#####======----------=====######====---------=
>$
>PRSAVKLTKKRALSISPLSDASLDLQTVIRTSPSSLVAFINSRCTSP
>##======#####==========----------
>$
[...]
```

Figure 5.12: *MoreRONN*'s new header output allows the user to quickly survey the protein to determine which residues are in clearly ordered (' ', a blank space character) or clearly disordered regions ('#', for a per–residue score > 0.6). It also shows "slightly" disordered residues ('=') with scores between 0.5 and 0.6, or ordered residues that are "nearly" disordered with scores between 0.4 and 0.5 ('–'). The dollar symbols ('$') are used for padding and readability. Each header line starts with the standard FASTA carat ('>'). Per–residue scores are output below this header as before.

Of note are the residues in the 0.4 to 0.6 borderline range; the ability to recognise them instantly with this new header output is helpful for construct design or decisions about crystallisability. One can now quickly determine which residues should be cleaved off to potentially improve solubility of a construct or promote tight crystal packing.

Chapter 6

Attempts at Order Prediction

6.1 Hypothesis and testing strategy

Since disordered regions are nested in between regions of order, it can be argued that an algorithm that identifies structured regions would work equally well in predicting disorder, since the regions that are *not* flagged as having structure can be marked as disordered. Since established structure assignment, prediction, or domain detection methods uncover protein regions that are structured, this is predicting the inverse of disorder. Therefore, structure assignment and prediction methods can complement disorder prediction; some flavours of PONDR embed DSSP secondary structure assignments into their engines to bolster their predictive capabilities (Peng et al., 2006). However, simply inverting the results obtained from secondary structure predictor output, while computationally valid and a useful additional metric, is biologically oversimplified on its own because disordered regions can still adopt structure under certain conditions, and fully disordered proteins sometimes exist as a well–defined three–dimensional (coil) structure (Dosztányi et al., 2007). The lesson here is that structure prediction methods cannot account for disordered regions that adopt transient, temporal order; it can only usefully yield

naturally disordered regions that remain disordered.

Having recognised the value and weakness of secondary structure identification tools when applied to disorder, the reverse possibility is that disorder prediction methods can be applied to the study of ordered regions. After all, the PDB, from where most of the training data is derived, is a repository of *structure*, and using this extra information could provide better sequence coverage and improve the prediction[1]! Whilst the goal with RONN and *MoreRONN* has always been to build the best possible single–engine[2] *disorder* predictor, in theory it is possible to train the engine to predict *ordered* motifs[3]. For clarity, this hypothetical predictor is referred to as *MoreRONN*-O to differentiate it from the regular *MoreRONN*, or *MoreRONN*-D, which is trained using disordered prototypes. The "holy grail" neural network would be trained with both classes simultaneously, thereby applying a comprehensive knowledge base of both order and disorder for a prediction[4]; let us refer to this predictor as *MoreRONN*-C. This would represent the next fundamental change to the neural network, where multiple classes of data are used simultaneously for training. Since multi–class neural networks are complex to design and implement (Lippmann, 1987; Schetinin et al., 2003), a sensible intermediate testing step would involve combining *MoreRONN*-D's and *MoreRONN*-O's predictions, similar to a meta–prediction, except here the underlying prediction engines are identical (this predictor is referred to as *MoreRONN*-OD). This represents a philosophy in-between a single–engine predictor such as *MoreRONN* and the meta–prediction approach of PONDR, which consists of fundamentally different underlying predictors. In order to build up to *MoreRONN*-C, therefore, it is first necessary to build *MoreRONN*-O, and move on to *MoreRONN*-OD if *MoreRONN*-O's predictions are on

[1]The amount of training data available naturally limits the predictive capability of the neural network.

[2]Not a meta–predictor.

[3]The regions in–between predicted order would be considered disordered.

[4]This contrasts with meta–prediction where separate underlying predictions are statistically combined.

par with *MoreRONN*-D's results (see chapter 5). This chapter describes the methods applied and the results obtained in attempting to build *MoreRONN*-O.

6.2 Building the Ordered Data Set

The same curated data set (see chapter 3) was used with *MoreRONN*, except the trainer was trivially altered to use ordered windows for *MCL* clustering instead of disordered sequences. There were 22,290 ordered sequences[5], corresponding to 3,556,942 15–residue, ordered windows.

Since *MCL* took approximately 15 minutes to cluster 107,413 disordered windows, and since 12 minutes were devoted to reading the windows from disk and populating data structures[6], the estimate for 3.5 million ordered windows was 6.5 hours for data preparation alone. Since this data is entirely new and different from the disordered data, no estimates could be made about *MCL* inflation factor, minimum BBF cutoff and cluster sizes, either; this would potentially require numerous training runs amounting to weeks of waiting. Therefore, a random subset of ~2500 full–length ordered sequences (around 10% of the total ordered set) was chosen as a prototyping benchmark, since this was similar to the number of disordered sequences (2848) in the data set and would therefore take a similar amount of time to cluster and train. Should this data set provide reasonable predictions, it would motivate training against the complete data set.

6.2.1 Prototyping Sets and Preliminary Results

With an input of ~2500 full–length ordered sequences, 357,623 15–residue prototype windows were generated for *MCL* clustering input, which is three times

[5]A tenfold increase compared to the number of disordered sequences
[6]The single–threaded, rate–limiting step.

larger than the number of disordered windows, implying that the average ordered sequence is thrice as long as the average disordered sequence. Clustering α was kept at 0.1 with the $-log_{10}$ contrast function in place, since the tests on α values had not yet been conducted at this time (see section 5.3.2). With an *MCL* inflation factor of 3.8 and a minimum BBF cutoff of 0.90, 620 clusters were produced, compared to 135 for *MoreRONN*-D.

This small–set *MoreRONN*-O achieved a maximum PE of **40.7%** for the 80-protein set, which was much lower than *MoreRONN*-D's numbers[7], but was at least in the same range as RONN. The predictor's truth table and position on the pyramid plot are shown in figure 6.1.

[7] *MoreRONN* lived up to its name instead of its potential?

Prediction		Gold Standard	
		Ordered	Disordered
	Pred. Ord	81.2%	40.5%
	Pred. Dis	18.8%	59.5%

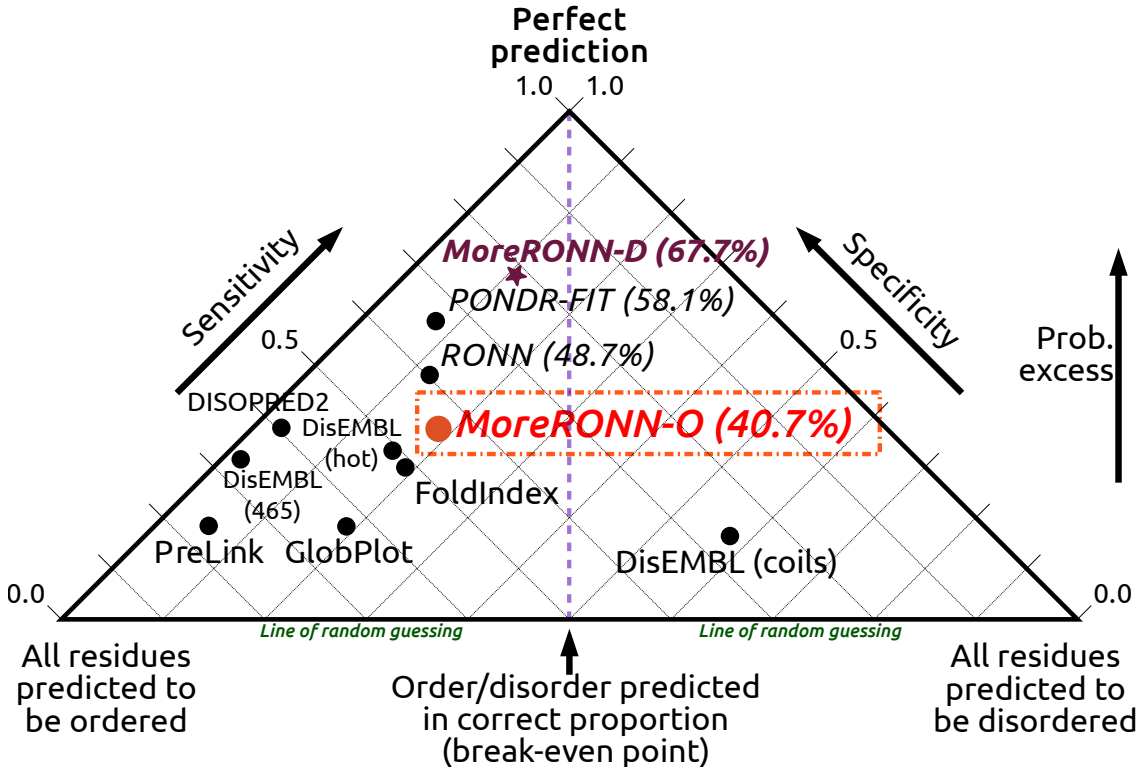


Figure 6.1: *MoreRONN*-O's best result with 10% of the ordered set. Both sensitivity (59.5%) and specificity (81.2%) have dropped compared to *MoreRONN*-D, but the predictor is still as balanced as PONDR-FIT, based on its distance from the centre vertical axis of the pyramid.

While this result was initially disheartening, we considered the possibility that a bulk of uniqueness in order had been discarded by choosing a small subset. Thus, it was still worth attempting to train with the full ordered data set by modifying *MCL* to handle the data efficiently.

6.2.2 Modifying *MCL* to cope with data size

The `driver_master_preproc` pipeline for *MCL* data preparation is shown again in figure 6.2. Steps 1–3 did not have much scope for speed or efficiency increase, but step 4 was heavily modified to minimise disk I/O as much as possible.

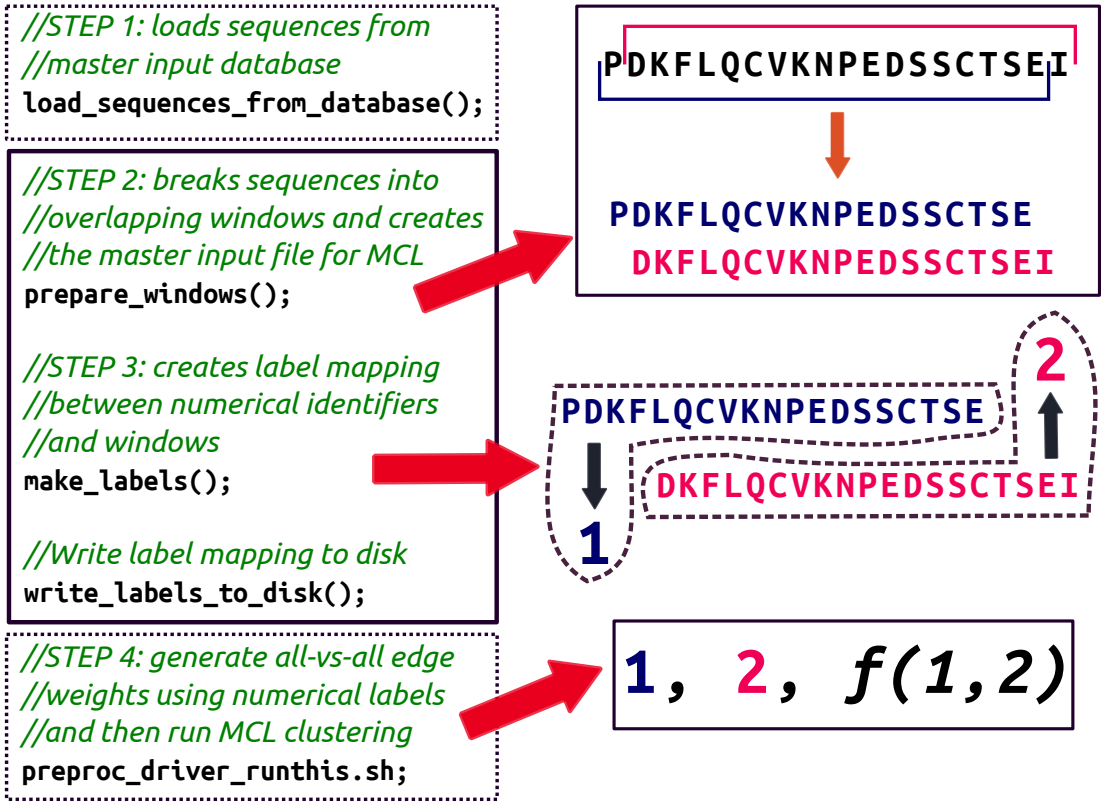


Figure 6.2: A simplified view of the `driver_master_preproc` pipeline, showing a sample data preparation as described by steps 1 to 3. Window size is 19 residues. The BBF function, $f(x, y)$, is used to generate weights. The shell script system call to `preproc_driver_runthis.sh` is required to parallelise the *MCL* stages properly.

The `preproc_driver_runthis.sh` shell script essentially creates the edge–weighted input graph for *MCL* by performing pairwise BBF comparisons and accepting edge pairs with scores greater than the specified minimum BBF cutoff. The graph input file for *MCL* is in a special binary format that is created by a helper program called *mcxload*; this file is then read in by the *mcl* executable to perform the actual clustering. Figure 6.3 shows the initial implementation of the *MCL* graph generator for *MoreRONN*-D.

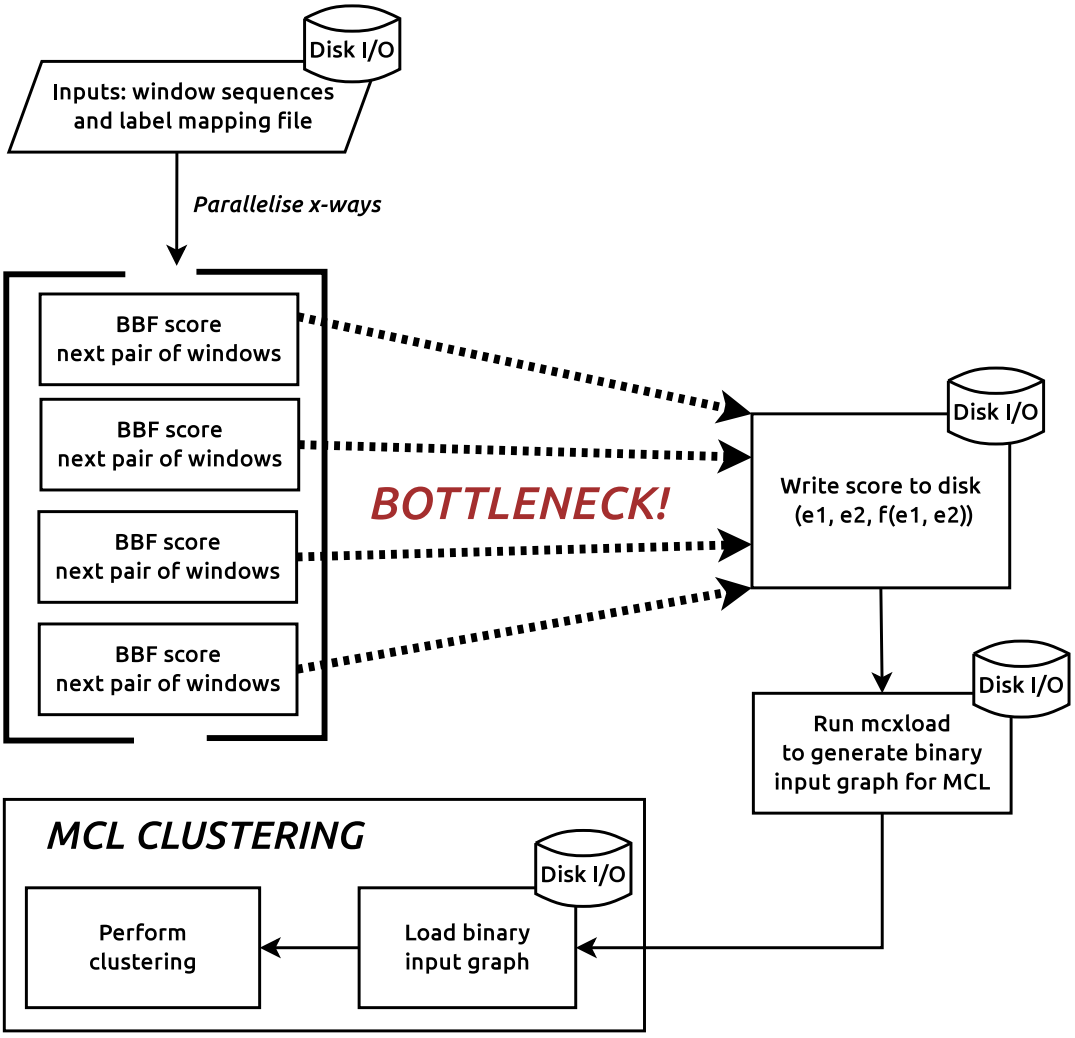


Figure 6.3: The first implementation of the *MCL* input generator. Disk I/O steps are necessarily single–threaded, but the BBF scoring function is parallelised. However, this solution does not give perfectly parallel speedups since there is a bottleneck where scores need to get written to a single edge–weight file for *mcxload*.

Whilst this method performed acceptably for *MoreRONN*-D, *MoreRONN*-O was unable to execute pairwise BBF scoring for the ~3.5 million ordered windows since this method relies on multiple, slow disk I/O steps; even though the BBF scores are calculated in parallel, only a single thread can write to disk at any given time, leading to a bottleneck. Furthermore, there are numerous save/load cycles with intermediate, temporary files which each add extra overhead.

The first attempt at optimisation involved partitioning the data and running mul-

tiple instances of the preprocessor in parallel. The *MCL* suite contains a programme, *mcx collect*, which, as its name implies, collects the edge–weight files generated by *mcxload* and assembles the full graph for *MCL*. Thus, by splitting the initial scoring work into smaller chunks and by piping the output to *mcxload* as a data stream, the hope was that this would run more efficiently, since the large bottleneck would be removed. Figure 6.4 shows the modified workflow.

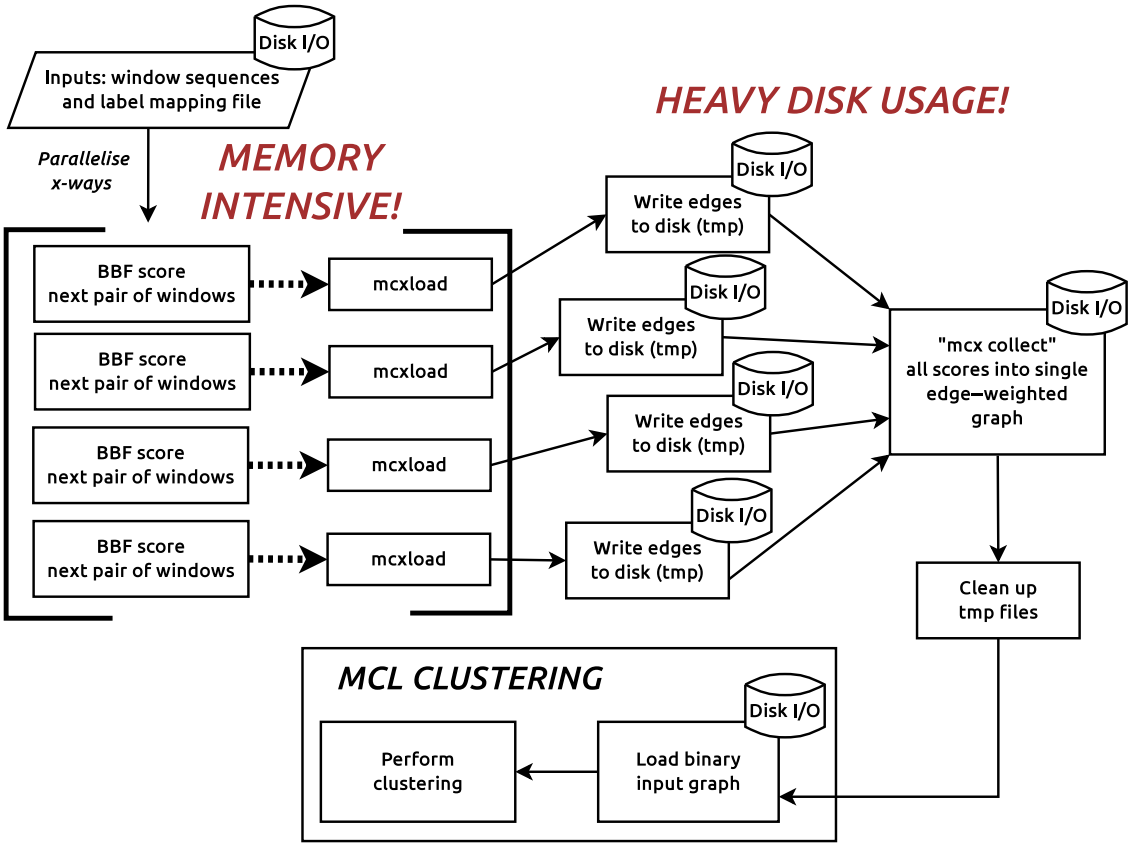


Figure 6.4: The second implementation of the *MCL* input generator. Each BBF scoring thread now gets its own file descriptor and *mcxload* process to write a portion of the graph to disk. The result is increased memory and disk requirements, but faster parallel performance at the scoring stage.

In this implementation *mcxload* waits for all the scores to be read in before writing its portion of the graph to disk, thereby greatly reducing disk I/O at this stage, but increasing memory requirement since the scores need to be held in memory before being dumped to disk. The program *mcx collect* then collates all the

edge weights before the temporary *mcxload* files can be cleaned up. This solution requires a temporarily file to be written by each thread nearly–simultaneously (assuming the pairwise scoring is divided equally between threads), but speeds up pairwise score calculation. Both disk space and memory requirements increase as a result.

To benchmark this method, it was tested with *MoreRONN*-D. The correctness of the method was validated by constant training and prediction scores when run on the 80–protein and 203–protein test sets and the data loading stage now took just 7 minutes instead of 12 with the original method (figure 6.3), implying that the pairwise scoring in memory was providing enough of a speed–up that the extra disk I/O was not yet showing its weakness. Given that this would translate to just under 4 hours to load *MoreRONN*-O's data, it was decided that this method should be tested on the ordered windows.

Whilst the parallel components did complete, *mcx collect* had still not finished assembling the final input file after nearly ten hours. With the help of Stijn van Dongen, *MCL* author, the diagnosis was that the intermediate files were being written and read in human–readable ASCII format, which is much slower than reading and writing binary data. Stijn modified *mcxload* to read binary data directly from the BBF scoring function and to write the temporary files out into binary format as well. He also modified *mcx collect* to read and write in binary, which would hopefully allow it to run. Conceivably, this would speed up all disk I/O stages and increase the data transfer rate between each BBF scoring function and its *mcxload* process.

It was decided between Stijn and the author[8] that the BBF scoring function would output each pairwise score in a 12–byte packet, as shown in figure 6.5. Note that this is identical to the specification used in clustering work (see figure 7.8 in chapter 7)!

[8]On a sunny day's trip to the EBI, situated on the outskirts of the "other" place.

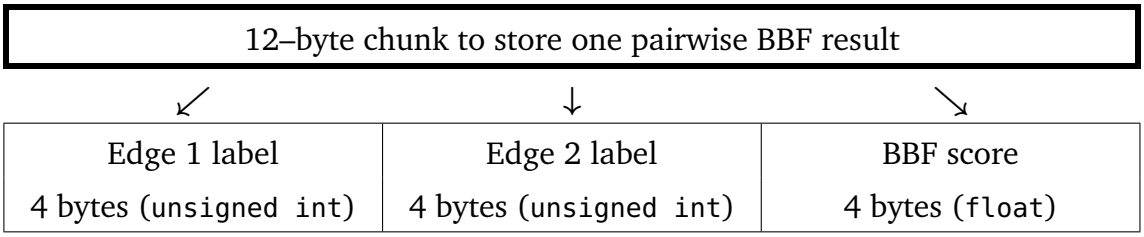


Figure 6.5: Binary format for pairwise scoring. The BBF score is a single–precision floating–point number.

The final workflow is shown in figure 6.6. At peak usage with all temporary files present on disk, approximately 400GB free space was required.

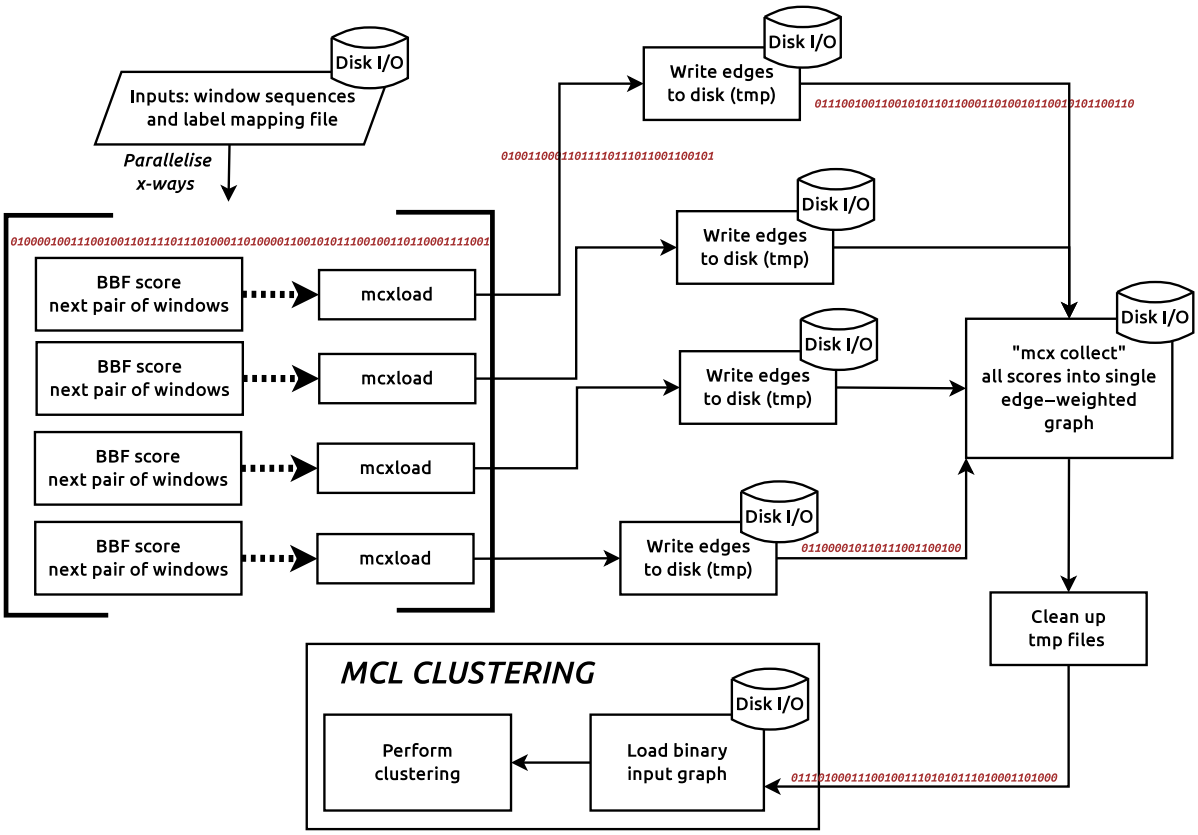


Figure 6.6: The final implementation of the *MCL* input generator. Each BBF scoring thread now sends data to *mcxload* in a custom binary format, and all disk I/O is in binary format as well. The result is a smaller disk footprint and faster disk I/O rates. Parallelization is also slightly improved since the overhead of transferring data in binary is smaller than transferring in ASCII format. RAM requirements remain unaffected.

These modifications finally allowed data to be loaded in approximately 2 hours, which was a large improvement over the estimated 4 hours with the previous iter-

ation. However, a final bottleneck occurred with the actual clustering, where *MCL* was unable to produce more than one cluster. The inflation factors and minimum BBF cutoffs were varied once more to attempt to induce clustering, without any result. Unfortunately, due to time constraints, work had to be stopped at this stage, but the updated data preparation workflow was much quicker and very helpful in generating different–sized clusters for *MoreRONN*-D; indeed, much of the data described in chapter 5 was derived with the help of this final version of the data preparation pipeline, since disordered data loading now took just over 3 minutes (compared to 12 minutes, initially); added to the actual clustering time of approximately 3 minutes, we were now able to produce a different disorder cluster data set every 6 minutes (compared to the initial engine, which took 15 minutes for a full clustering run)!

6.3 Differences between Order and Disorder Predictors

Despite being unable to obtain clustering (and therefore, training and testing) results for the full ordered data, the results from the small subset do allow some analysis and speculation to be conducted. It is interesting, for instance, that the algorithm maintains its predictive stability even when the sequence profile of the training data is changed completely. Hypothetically, complete–set training runs with various input parameters should remain stable as well[9].

The ~20% drop in predictive power with the comparably–sized ordered data is more problematic, however. It implies that there is a fundamental difference in the the amount of information that can be gained from a similar number of ordered

[9]Hopefully, the prediction numbers would stabilise at a much higher probability excess with the full ordered data set.

sequences, and in fact, there is *less complete* information present in the ordered subset. Perhaps this implies, as stated previously, that the full available data set would boost numbers because it covers a larger ordered sequence space. If this is the case, however, it raises an interesting argument about the complexity of ordered and disordered sequences. While it is known that disordered sequences include those of "low complexity", *MoreRONN* supports this argument by showing that it needs relatively few families of disordered windows of a given length in order to make strong predictions. This also implies that distant hits (lower BBF scores) to disordered sequences may be good enough to detect disorder, but the same is not true of order. In terms of the neural network, after a certain number of high scoring cluster weights, extra lower weights begin to adversely affect prediction quality. With order, however, many more sequences (and therefore, families of windows) are required to capture the available order space, and the low probabilities excess for the order–trained *MoreRONN* indicate that our subset is not nearly comprehensive enough.

Even more intriguingly, these results give rise to a thought experiment about why order might be more difficult to predict than disorder. Consider figure 6.7 below.

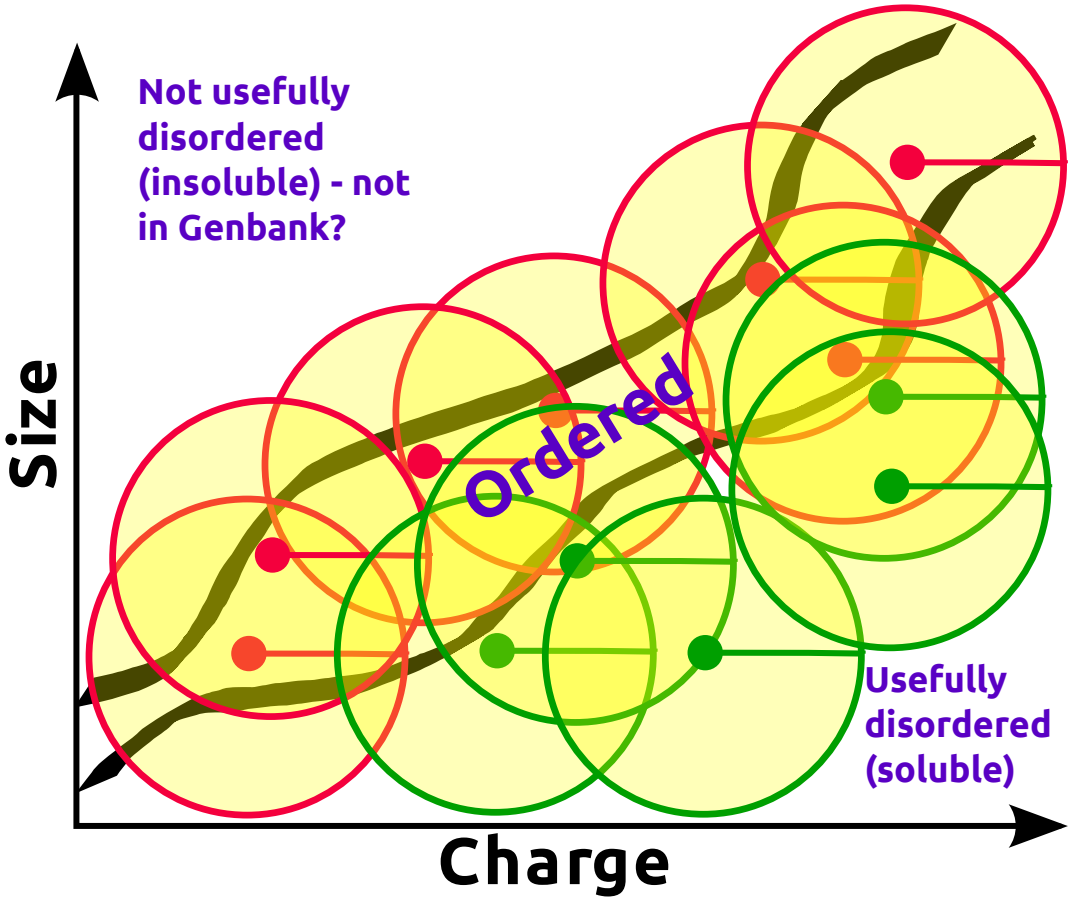


Figure 6.7: A conceptual charge vs. size distribution for protein sequences. Small, coloured circles are *MoreRONN* representative prototype windows surrounded by clusters of related windows (bigger circles). Red clusters are ordered, green clusters are disordered.

The graph plots average charge of a protein sequence versus its size. Side chains of amino acids in disordered sequences are, by nature, usually short and polar (see chapter 1). They are represented in figure 6.7 by the small green circles. As amino acid composition changes and hydrophobic amino acids are present, the complexity of the sequence increases and often induces order. These sequences are represented in figure 6.7 by the small, filled red circles. Note that the top left of the figure is empty, as agglutinating, fully hydrophobic disordered sequences are unlikely to be biologically useful. Therefore, we categorise them as not usefully disordered, while the soluble disordered sequences are present and participate in numerous cellular processes (see chapter 1). The larger circles represent clusters of related sequences

centred around a representative.

When *MoreRONN* is trained on disordered windows, due to their low complexity, clusters tend to encompass most of the known disordered sequence space, and therefore the predictions are of high quality. However, when ordered windows are used in training, the clusters, and therefore the predictions, are much noisier and the probability excess is lower. As shown in the graph, the known ordered sequence space when using the small[10] subset is much narrower, and therefore, clusters might be a lot noisier. This also implies that, when the full ordered set training can be made to work, the best predictions would require more clusters than with the disordered data. In the figure, this would correspond to smaller cluster radii for the ordered data which would reduce overlap and spillage into the disordered regions, thereby increasing sensitivity of the prediction.

There is a more subtle hint in these observations, however. If this hypothesis is correct, it is possible that the natural state of proteins is to be disordered; strategic evolutionary introduction of order–promoting amino acids generates structure. Therefore, perhaps order is the complex boundary between soluble and agglutinating disordered regions? This would explain why *MoreRONN* requires a much larger ordered training set to improve its predictions, and why tuning the size of clusters, or conceptually, altering the radii of the big circles in figure 6.7, makes an impact on the prediction quality. It is important, therefore, to find a suitable method of clustering the full set of ordered windows. If the quality of *MoreRONN*-O's predictions improves, then it is worth exploring a joint predictor (*MoreRONN*-C) that uses both ordered and disordered sequences for training. This joint predictor can then predict using both ordered and disordered sequence spaces and ideally produce the best possible prediction, affected only by the size and quality of the training data.

[10] ~2500 fully–ordered sequences.

Chapter 7

Clustering and Sampling of Structural Fragments in the PDB

7.1 Introduction

One important sub–plot that has recurred throughout this thesis has been the efficient clustering of large data sets. In *Nearest–cell*, the Family Clustering algorithm was implemented to reconcile differences between crystal forms of related protein species (see chapter 2 and Ramraj et al., 2012). In *MoreRONN*'s development, *MCL* was implemented in order to cluster disordered prototype windows (see chapter 5); whereas with ordered sequences, the limits of clustering algorithms were reached (see chapter 6).

Why is clustering important in bioinformatics? Biologically speaking, it allows us to divide a set of data into evolutionarily–related structural or functional families which helps us categorise and understand more easily. Computationally speaking,

it lets us manage large volumes of data for algorithms in order to compute things quickly. In some cases, it can also help machine learning algorithms produce better results because it removes redundancy in the data. This was very evident in disorder prediction when *MCL* was applied to reduce redundancy in prototype windows, resulting in increased sensitivity (see chapter 5). However, the PDB is primarily a repository of structure (3D atom positions), and we have yet to effectively utilise this large resource of order for our work. An attempt was made in order prediction (chapter 6), but the amount of data proved to be too large for *MCL* to produce sensible clusters, so we will need to explore different ways to cluster this information. Furthermore, an ordered prototype is positively identified in the PDB by ATOM information, whereas disorder can only be indirectly inferred. In this chapter, we turn our attention from clustering short sequence windows to the possibility of clustering short structural fragments in the PDB.

Clustering and profiling short regions is important because of biology's tendency to encode a wealth of information in this short space. As we have already seen with disorder prediction, short windows of 9 residues still produced excellent prediction results (see chapter 5); this implies that there is enough information present in these short windows to train *MoreRONN* effectively[1]. A more tangible example can be found in "major histocompatibility complexes" (MHCs), which are built around the recognition of short (8–10 residue) oligopeptides in a binding groove (Janeway et al., 2001). They are found on almost every nucleated cell in the human body and their main role is to identify foreign (pathogenic) proteins and initiate a complex series of immune response events in the host to fight off the invading species. The remarkable point here is that even these short oligopeptides possess enough information to identify a foreign *species*; this further supports the theory that biology is able to encode complex information within very short pieces of sequence.

[1]Indeed, this was our driving hypothesis underlying the reduction in the original RONN window size from 19 residues.

It is therefore a useful exercise to profile all known structural fragments in the PDB and attempt to cluster them into families. Unfortunately, this work requires significant computational resources and a finely tuned clustering approach to produce results at all, and has, to the best of our knowledge, never been attempted before. Arguably, *MCL* could be used once more, however, our inability to obtain suitable clusters on the full set of *ordered* prototypes was the roadblock that prevented order prediction results from being obtained (see chapters 5 and 6), and we estimated a similarly sized structure fragment data set, but with many, many more edge weights!

Therefore, in this chapter, we explore the design and development of a bespoke structural fragment clustering and sampling algorithm which is designed to be highly parallelizable, thus working quickly on very large fragment data sets[2]. We also hypothesise about retroactive application of this algorithm to the order prediction work (see chapter 6), in lieu of *MCL*. A short introduction to structural classification follows.

7.2 Structural Classification

The amount of structural data in the PDB (stored in ATOM records) is vast, and attempts have been made to group tertiary structure motifs into databases such as SCOP (Lo Conte et al., 2000) and CATH (Pearl, 2003). In terms of assigning structure for a single protein, however, various techniques have been developed, but the simplest quantitative and classic benchmark method is known as DSSP (Kabsch et al., 1983).

DSSP is an algorithm developed by Kabsch et al. (1983) that identifies secondary structural motifs by calculating hydrogen bond electrostatics from a set of PDB

[2]A "fragment" in this chapter should be assumed to be a short–length oligopeptide derived from contiguous residues in a PDB entry.

ATOM records. The energy of a hydrogen bond between an amide hydrogen and a carbonyl oxygen can be determined using Coulomb's law, shown in equation 7.1 below.

$$E_H = \frac{q_1 q_2}{4\pi\epsilon_0}\left(\frac{1}{r_{ON}} + \frac{1}{r_{CH}} - \frac{1}{r_{OH}} - \frac{1}{r_{CN}}\right) \tag{7.1}$$

Figure 7.1 illustrates a hydrogen bond and the location of the partial charges. The charges are $+q_1$ on the carbonyl C, $-q_1$ on the carbonyl O, $+q_2$ on the amide H and $-q_2$ on the amide N.

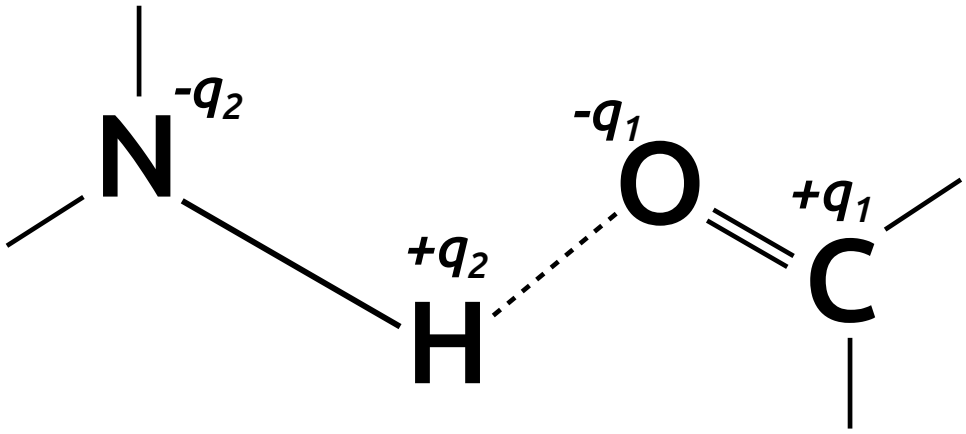


Figure 7.1: A hydrogen bond in the peptide backbone, showing partial charges.

q_1 is approximately 0.42 electron charges and q_2 is approximately 0.20 electron charges, giving a value of 27.888 kcal Å mol^{-1} for the constant term. DSSP assigns secondary structural motifs using a code derived from 7.1.

DSSP Code	Definition
H	α helix
I	π helix
G	$^3/_{10}$ helix
E	Extended strand, participates in β–ladder
B	Residue in isolated β–bridge
T	Hydrogen bonded turn
S	Bend
C	Others (loops, irregular stretches)

Table 7.1: Table of DSSP codes. The codes are arranged in decreasing order of priority, that is, if DSSP encounters ambiguous stretches of structure, it will assign the highest priority label.

DSSP, therefore, identifies and assigns secondary structure to whole proteins from their experimental (ATOM) data in a standardised way; while its output has been used to seed and train secondary–structure predictors such as *PDBeFold* (formerly SSM; Krissinel et al., 2004), DSSP is *not* a secondary–structure predictor in itself. Furthermore, the energetics calculations are best suited to whole proteins and are not useful for identifying or clustering short structural fragments. An example DSSP output is shown in figure 7.2.

```
  #  RESIDUE AA STRUCTURE BP1 BP2  ACC
    1    2 A T              0   0   77
    2    3 A T  E     -A   34   0A  21
    3    4 A a  E     -A   33   0A   0
    4    5 A b        -     0   0    0
    5    6 A P  S    S+     0   0   52
    6    7 A S  S  > S-     0   0   48
    7    8 A I  H  > S+     0   0  123
    8    9 A V  H  > S+     0   0   98
    9   10 A A  H  > S+     0   0    6
   10   11 A R  H  X S+     0   0   55
```

```
B = residue in isolated β-bridge
E = extended strand, in β-ladder
G = 3-helix (3/10 helix)
H = α-helix
I = 5-helix (π helix)
T = hydrogen bonded turn
S = bend
```

Figure 7.2: Example output from DSSP. A portion of secondary structure assignment is shown on the left, with the legend on the right. Figure taken from Joosten et al. (2011b).

7.3 Methods

7.3.1 Quality Metrics for Data Filtration

As with training data for *MoreRONN*, it was first necessary to define a set of metrics to obtain high–quality curated ATOM data from the PDB. Since we are interested in small structural fragments, we outlined two levels of curation: **global** and **local**. The former denotes total quality of the deposited protein structure and is similar in philosophy and implementation to the curation method used to obtain training data for *RONN* and *MoreRONN* (see chapters 3, 4 and 5). The latter is a set of rules we developed to judge the quality of individual residues; stretches of high–quality residues can then be selected as fragments for clustering.

7.3.1.1 Global Quality Metrics

Global quality parameters are shown in table 7.2 below and were applied to PRAXIS to obtain a filtered set of structures.

Metric	**Parameters**
Method	`"X-RAY DIFFRACTION"`
Resolution	$\leq$ 2.0 Å
RMS bond length deviation	$<$ 0.01 Å
RMS bond angle deviation	$<$ 2.0°
R_{work}	$\leq$ 0.22

Table 7.2: Table of global quality metrics applied to the PRAXIS database to obtain the first pass of high–quality PDB depositions.

The PostgreSQL query described in listing 2 was used with the cutoff values shown in table 7.2 above. Out of approximately 90,000 crystal depositions in the PDB, **15,704** structures were isolated in this global filtering step.

7.3.1.2 Local Quality Metrics

Since the goal was to obtain high quality short fragments (6 residues), the global filtering step would not be able to adequately capture quality variation at the local structural level (*e.g.* poorly–ordered loops). Indeed, global quality metrics such as R_{work} simply cannot identify local errors in structure (Tickle, 2012). Therefore, it was necessary to develop a method for evaluating the quality of each residue within a selected structure. It would then be trivial to construct fragments of the desired length using contiguous stretches of high–quality residues.

The *PDB_REDO* software suite (Joosten et al., 2009a, 2011b), which is designed to rebuild and optimise PDB structures, provides a very straightforward way of filtering high–quality residues after the global quality filtering step. It consists of scripts, databases and several external programs, including the major software packages CCP4 (Collaborative Computational Project, number 4, 1994) and *WHAT_IF*/YASARA (Krieger et al., 2002). *PDB_REDO* re–refines PDB structures, notably with REFMAC maximum–likelihood method (Murshudov et al., 1997; Winn et al., 2001; Winn et al., 2003) in its newest iteration, REFMAC5 (Murshudov et al., 2011), to better fit the experimental data using modern techniques, validating the refined model and recording the progress and results of its re–refinement efforts. Whilst re–refinement may seem unnecessary given that electron density map models are now created with automated methods such as ARP/wARP (Perrakis et al., 1999), Resolve (Terwilliger, 2003) or Buccaneer (Cowtan, 2006), the final refinement steps are still conducted by the user. Therefore, human–error is introduced due to time constraints, lack of experience or other intangibles, and as a result, even modern–day depositions can contain errors. *PDB_REDO* aims to solve this problem in an automated way and, along with optimised structures, maintains a set of structural databases[3] as shown in figure 7.3, taken from Joosten et al. (2011b). It

[3]This data storage and curation ethos is similar to the motivation for developing PRAXIS,

also maintains a record of problems that it encounters in a separate database called "WHY_NOT."

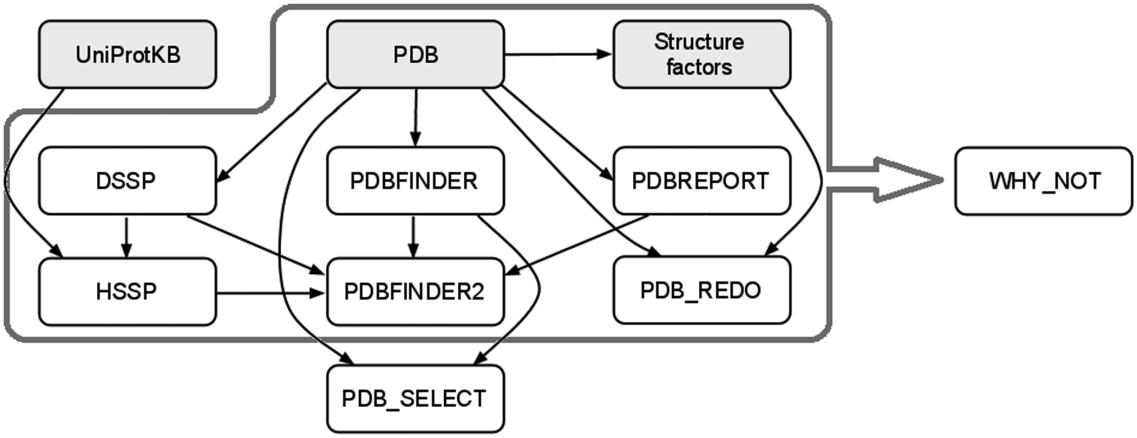


Figure 7.3: Database dependency within *PDB_REDO*. Local copies of the databases shown in white boxes are generated by *PDB_REDO*, and databases surrounded by the grey box are indexed in WHY_NOT entries, which are generated if problems are encountered during the rebuilding process. Note that the driving paradigm for this thesis, the PDB, is also central to *PDB_REDO*'s data pipeline! Figure taken from Joosten et al. (2011b).

The result of this work is that modelling errors are corrected and quality of structures[4] is (often significantly) improved. An example, taken from Joosten et al. (2011a), is shown in figure 7.4. Here, *PDB_REDO* flips a peptide using a custom embedded algorithm called *pepflip*, in order to improve the fit to the electron density map, shown as a light blue mesh.

whereby oft–utilised data are kept in a quickly–retrievable way.

[4]Usually old depositions, some of which are $>$ 30 years old!

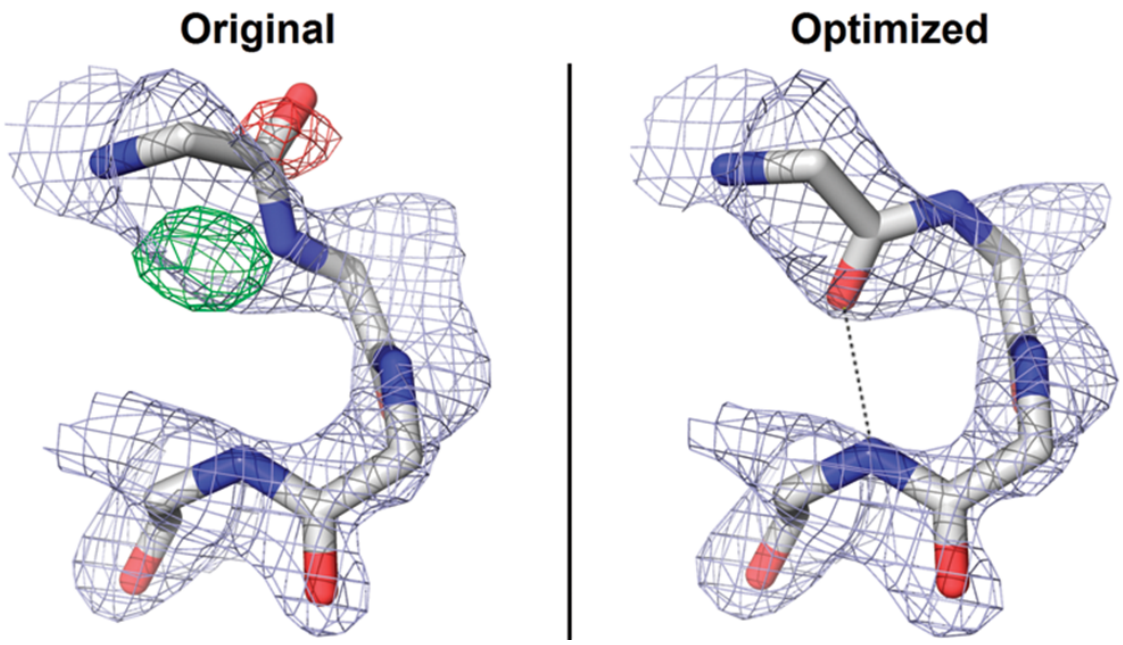


Figure 7.4: *PDB_REDO* flips a peptide using *pepflip* to improve fit to the electron density map (light blue mesh). This introduces a hydrogen bond (dotted line) into the model that better reflects a stable protein structure. Figure taken from Joosten et al. (2011a).

PDB_REDO consists of 9 steps for rebuilding a model and relies on numerous external programs, but essentially, the end result consists of a conservatively–optimised model, a fully–optimised model[5] and associated data files. For this clustering application, the three most important features that *PDB_REDO* provides are:

1. The PDBREPORT database generated by *WHAT_CHECK* (Vriend, 1990; Hooft et al., 1996), which flags low quality residues, or unusual bond lengths or angles;

2. Per–residue quality metrics calculated by the program EDSTATS, part of CCP4 (Collaborative Computational Project, number 4, 1994); and

3. The rebuilt structures themselves.

[5]Through personal communication with Robbie Joosten, it was determined that the fully–optimised structures should be used for this particular case, as they are considered to be *PDB_REDO*'s final output.

The rebuilt structures are used in lieu of the original PDB depositions to retrieve fragments for clustering, as they correlate with the quality measurements by *WHAT_CHECK* and EDSTATS. By only using *PDB_REDO* structures, we also ensure that every structure is subject to the same refinement paradigm, thereby standardising the data set, in essence.

Table 7.3 records the values that were used as cutoffs for the four key EDSTATS metrics. The machine names are used in code, subsequent text and figures in this chapter to refer to these defined values.

Local Metric (per–residue)	**Cutoff**	**Machine Name**
RMS Bond Length	$\leq 4.0\ \sigma$	—
RMS Bond Angle	$\leq 4.0\ \sigma$	—
Real–space Residual (RSR)	≤ 0.12	RSR_CUTOFF
Real–space Correlation Coefficient (RSCC)	≥ 0.9	RSCC_CUTOFF

Table 7.3: Table of local quality metrics applied to *PDB_REDO*–optimised structures to obtain individual high–quality residues. The RSR and RSCC cutoff values are often referred to by their machine names in this chapter; the machine names are the names of the variables used in the Python implementation of the local filtering algorithm.

Having now designed the local clustering pipeline and obtained a local copy of the *PDB_REDO* database, Python scripts were written to implement local filtering. The pipeline is divided into two parts; the first part is described in figure 7.5 and is the important step in selecting high–quality residues.

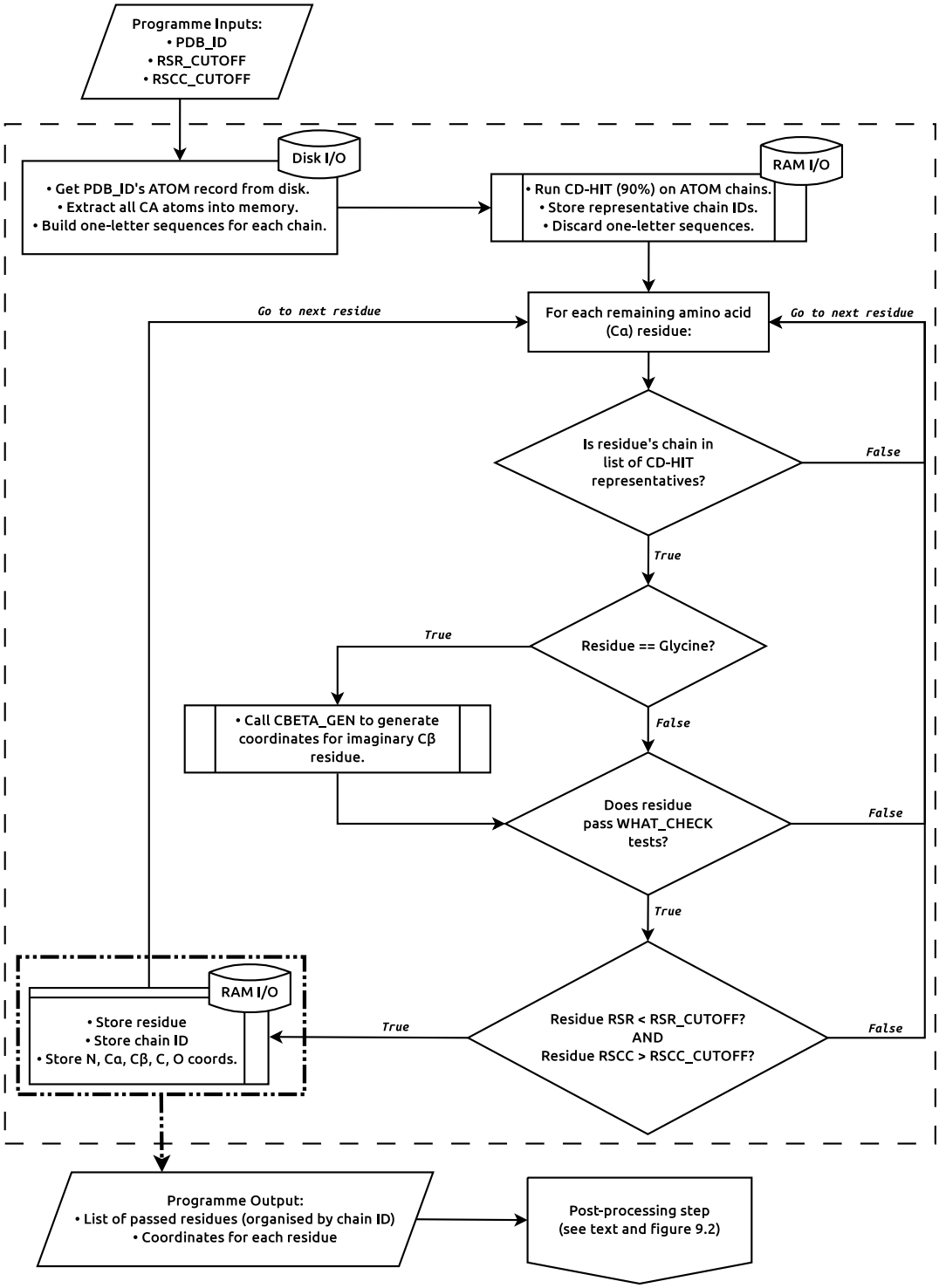


Figure 7.5: Flowchart describing the local filtering algorithm. The CD–HIT filtering (at 90% identity) ensures that only non–redundant chains are evaluated. Contrived Cβ coordinates for glycines are generated using a FORTRAN program called *CBETA_GEN*. The algorithm stores the coordinates for each amino acid that passes the filtering criteria. Amino acids are stored in memory with mappings to their chain ID. Once all amino acids in a given PDB record are evaluated, they are reassembled into fragments in the post–processing step (discussed in text and figure 7.6).

Residues that passed this filtering step were then post–processed into fragments on a chain–by–chain basis, with geometric chain breaks signalling breaks between fragments. The post–processing step is shown in figure 7.6. A geometric chain break is identified by calculating the distance between the Cα atoms of a pair of residues, and if this distance is greater than 4.0Å, the chain is considered to be broken and a new fragment begins at the latter residue.

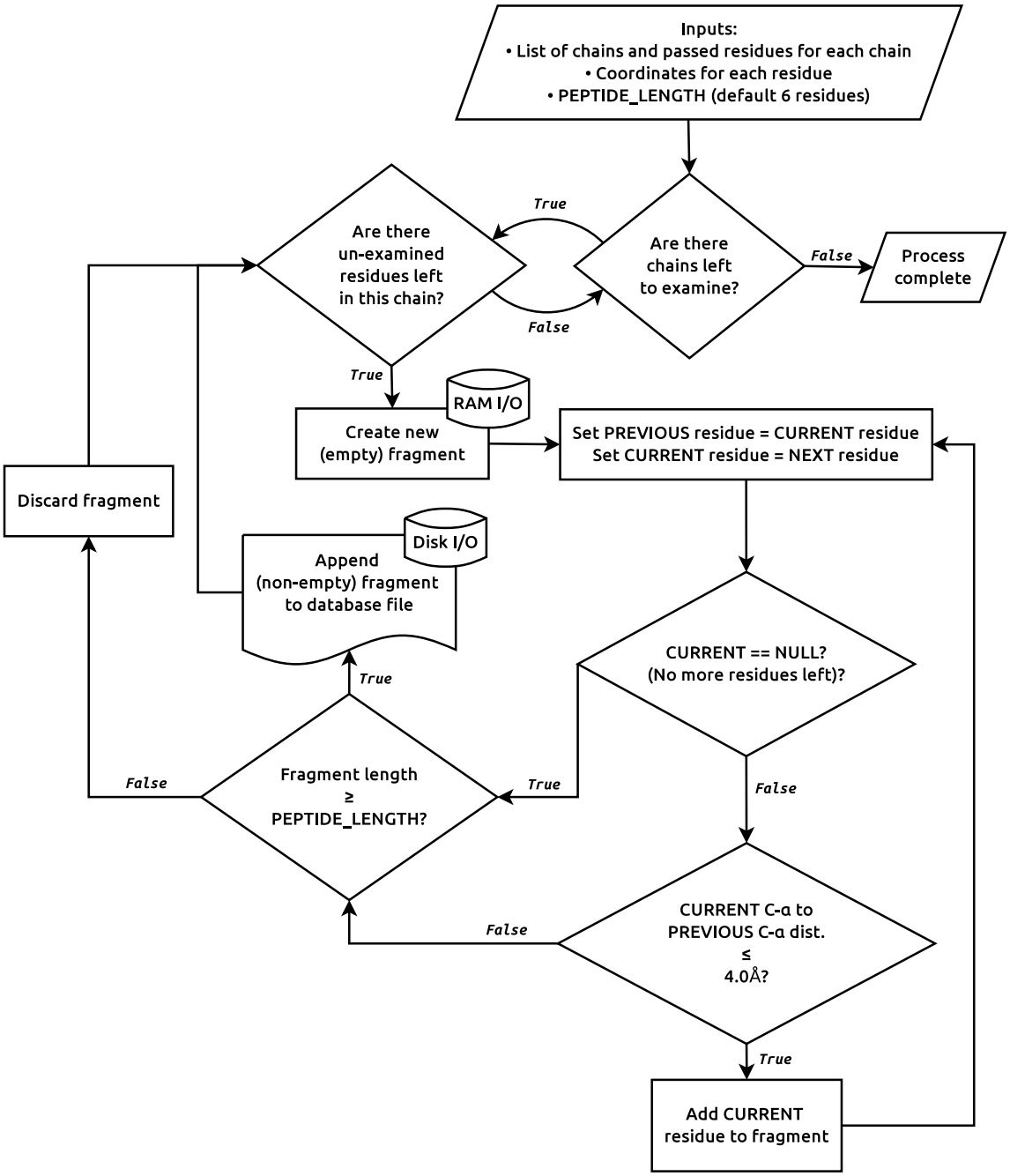


Figure 7.6: Post–processing to reassemble fragments continues from where figure 7.5 leaves off. A fragment is built with contiguous stretches of residues as long as a consecutive pair of Cα atoms is not more than 4.0Å apart (indicative of a chain break). "CURRENT", "PREVIOUS" and "NEXT" are pointers to residues. The algorithm depends on CURRENT to evaluate to NULL in order to determine when the chain has been fully processed, that is, when CURRENT equals NULL, it is pointing beyond the chain and there are no residues left to evaluate. Fragments are written to the final database file in the custom format described in figure 7.7. There is an explicit sanity check to ensure that the fragment is neither empty nor smaller than the desired fragment clustering size (PEPTIDE_LENGTH) before it is written to disk. PEPTIDE_LENGTH can be specified by the user at run–time and is set to 6 residues by default.

Fragments were written to a custom FASTA–formatted database file on–the–fly (see figure 7.7). The coordinate lines are of fixed–width and coordinates are truncated to three decimal places (see figure 7.7a for the database file specification). From this database, overlapping structural fragment windows of a desired length can be extracted for clustering. We concentrated on clustering 6–residue windows; After local filtering, **114,306** structural fragments were obtained, which corresponded to **2,653,478** 6–residue windows for clustering[6]

[6]A length of six residues was chosen because it would allow direct incorporation into CALPHA; this was the primary driving reason. The conformational space is smaller for shorter fragments and therefore a similar number of fragments gives greater coverage. Longer fragments provide greater uniqueness of coverage but with a similar number of fragments, the sampling is necessarily more sparse. Therefore, this question should be revisited in the future, since the pre–filtering and clustering algorithm presented here should be able to handle clustering of longer fragments without presenting significant computational challenges.

```
The FASTA header format is as follows:
    >(PDB ID)_(Chain ID)_(Fragment start index)_(Fragment length)

Coordinate data format is as follows (all coordinates are truncated to three decimal places):
    (ResNum) (3-letter-code); Nx Ny Nz; CAx CAy CAz; CBx CBy CBz; Cx Cy Cz; Ox Oy Oz
```

(a) Specifications for FASTA header and coordinate lines in database file.

```
>1iz9_A_2_89
    2 LYS;  -5.967   37.590   60.565;  -4.515   37.472   60.567;  -4.031   36.082   60.189;  -3.818   38.498   59.691;  -4.296   38.851   58.612
    3 ALA;  -2.657   38.947   60.151;  -1.841   39.895   59.385;  -0.706   40.427   60.236;  -1.287   39.187   58.149;  -0.934   38.012   58.214
    4 PRO;  -1.211   39.895   57.017;  -0.726   39.251   55.792;  -0.748   40.378   54.747;   0.696   38.689   55.920;   1.503   39.228   56.666
    5 VAL;   0.984   37.605   55.211;   2.347   37.108   55.103;   2.494   35.614   55.483;   2.787   37.337   53.670;   1.961   37.374   52.751
    6 ARG;   4.091   37.488   53.483;   4.646   37.720   52.157;   5.728   38.788   52.217;   5.230   36.431   51.617;   6.092   35.810   52.254
    7 VAL;   4.749   36.033   50.450;   5.191   34.823   49.794;   3.990   33.922   49.496;   5.890   35.183   48.487;   5.279   35.759   47.588
    8 ALA;   7.178   34.851   48.405;   7.969   35.088   47.210;   9.377   35.504   47.590;   7.989   33.826   46.354;   8.126   32.729   46.880
[...remaining residues in this fragment have been omitted...]
    *                         *                         *                         *                         *
    *                         *                         *                         *                         *
>1izc_A_230_15
  230 PRO;  45.778   66.500   60.377;  46.828   66.707   61.397;  46.118   67.540   62.476;  47.398   65.408   62.001;  48.581   65.358   62.340
  231 THR;  46.559   64.378   62.139;  46.993   63.057   62.607;  45.800   62.088   62.800;  48.018   62.463   61.659;  48.982   61.843   62.096
  232 PHE;  47.799   62.646   60.371;  48.731   62.181   59.353;  48.091   62.248   57.965;  50.011   63.016   59.402;  51.124   62.471   59.339
  233 VAL;  49.857   64.335   59.496;  51.047   65.199   59.601;  50.683   66.704   59.635;  51.896   64.775   60.792;  53.127   64.634   60.661
  234 GLU;  51.270   64.538   61.939;  52.009   64.141   63.136;  51.101   64.010   64.353;  52.729   62.813   62.915;  53.880   62.636   63.336
  235 ALA;  52.047   61.883   62.252;  52.599   60.561   61.997;  51.558   59.687   61.312;  53.843   60.658   61.114;  54.871   60.015   61.378
  236 MET;  53.742   61.463   60.066;  54.853   61.646   59.136;  54.412   62.407   57.893;  56.033   62.386   59.794;  57.207   62.095   59.484
  237 THR;  55.732   63.318   60.690;  56.779   64.005   61.441;  56.235   65.198   62.255;  57.492   63.058   62.363;  58.732   63.085   62.471
  238 LYS;  56.728   62.233   63.061;  57.333   61.245   63.924;  56.278   60.524   64.744;  58.238   60.275   63.134;  59.371   60.005   63.543
  239 PHE;  57.728   59.778   62.011;  58.441   58.862   61.147;  57.502   58.410   60.003;  59.726   59.486   60.599;  60.813   58.888   60.682
  240 SER;  59.623   60.696   60.055;  60.762   61.298   59.357;  60.341   62.483   58.479;  61.852   61.712   60.343;  63.028   61.567   60.038
  241 THR;  61.450   62.187   61.520;  62.371   62.554   62.597;  61.607   63.244   63.758;  63.115   61.310   63.109;  64.341   61.337   63.346
  242 ALA;  62.382   60.207   63.269;  63.006   58.974   63.731;  61.954   57.892   63.970;  64.068   58.488   62.717;  65.161   58.092   63.105
  243 ALA;  63.727   58.503   61.430;  64.690   58.142   60.376;  64.039   58.201   58.993;  65.918   59.074   60.412;  67.062   58.616   60.440
  244 GLN;  65.667   60.370   60.404;  66.764   61.376   60.405;  66.201   62.806   60.380;  67.715   61.211   61.594;  68.938   61.200   61.437
    *                         *                         *                         *                         *
    *                         *                         *                         *                         *
```

(b) Snippet from final database file. The FASTA header and Cα coordinate column are highlighted in bold.

Figure 7.7: The database file format specification is shown in figure 7.7a. There are two kinds of lines in the file: the FASTA header and the coordinate data. The header contains PDB ID, chain ID, starting point of the fragment in the original ATOM record, and fragment length, respectively. Each piece of information is separated by an underscore ("_") character. The coordinate data lines are essentially condensed ATOM coordinate lines delimited by semicolons (";"), with three coordinates (x, y, z) for each backbone atom in the amino acid. The line structure is fixed–width and is described in figure 7.7a. Figure 7.7b shows a snippet of actual data. Note that the first residue's index (2, 230) and the length of the fragment (89, 15 residues) are stored as metadata in the respective FASTA headers, permitting cross–referencing back to the original ATOM records.

7.3.2 Devising a clustering algorithm

7.3.2.1 Finding structurally similar fragment pairs

Given that there were approximately 2.5 million fragment windows to cluster, it did not seem likely that *MCL* would work, given that *a*) it was unable to produce sensible clusters for a similar–sized ordered prototype set for *MoreRONN* (see chapter 6); and *b*) many more edge weights were expected. With the 15–residue ordered windows, there was probably not enough range in the BLOSUM62–derived BBF scores for *MCL* to resolve the distance between two windows; in the current fragment clustering case, given that the fragments were just 6 residues in length, we felt certain that the resolution (regardless of scoring model) would be even worse[7]. Therefore, *MCL* was not an option. It was therefore necessary to devise a fast clustering algorithm that could cope with the amount of data present. Developing a bespoke method is also advantageous in that the implementation can be carefully designed in order to be as memory–efficient and parallelizable as possible.

Just like with the BBF, scores were used as measures of similarity in the sequence clustering case. Here, MATFIT superposition and the resulting RMSD can be used as a similarity metric between two structural fragments with accepted window pairs having an RMSD lower than a pre–defined cutoff, which was set to **0.15**. Given that each pair needed to be superposed, this meant running MATFIT 3.52×10^{12} times[8], which translates to approximately 2 months of running time[9]! Furthermore, the current implementation of MATFIT (built by Ramsay, 1990) is not designed for shared–memory parallelization, negating the possibility of using OPENMP to speed

[7]Note that this is before the revelations about changing the BBF cutoffs and α values in the BBF function for clustering and training, but the greater number of edge weights expected still pointed towards a different clustering method.

[8]Since the RMSE of the superposition of fragment A on fragment B will be the same as the superposition of B on A, the formula to calculate maximum number of matches is $n(n-1)/2$, where n=2,653,478.

[9]Each MATFIT call takes $\sim$1.5μsec to run.

up the process. However, given that a majority of superpositions would not pass the RMSD cutoff criterion, a pre–filtering step was devised to eliminate as many distant pairs as possible, thereby drastically reducing the number of system calls to MATFIT. The pre–filtering is based on centre–of–gravity of Cα atoms and is described in definition 7.3.1 below.

Definition 7.3.1. *In order to minimise the number of calls to* MATFIT*, we consider the following method of pre–filtering pairs of fragments that will definitely fail the RMSE cutoff criteria.*

For fragments A and B, where $\vec{a_n}$ and $\vec{b_m}$ are atom positions of the n^{th} and m^{th} Cα atom in fragments A and B, respectively:

If N = length of peptides to be superposed (in this case, $N = 6$), the RMS error (RMSE) is defined ad calculated by MATFIT *in the following manner:*

$$RMSE = \sqrt{\frac{1}{N} \sum_{0 \leq n < N} |\vec{a_n} - \vec{b_m}|^2} \tag{7.2}$$

However, it is an invariant that MATFIT *will **always** superpose centres of gravity (COGs). Therefore, we define a new superposition which only preserves the distances from Cα atoms to the COG within each fragment. Due to the invariant, this superposition **must** give an equal or better (smaller RMS) result than* MATFIT*. However, the benefit is that most of this calculation is done **within** a fragment, as opposed to between fragments.*

Considering equation 7.7 above, for each $(\vec{a_n} - \vec{b_m})$, the best result is when the COGs for A and B are superposed and $\vec{a_n}$ and $\vec{b_m}$ are in the same direction. Thus, $(\vec{a_n} - \vec{b_m})$ reduces to $|distA_n - distB_m|$, where $distA_n$ and $distB_m$ are the distance between the COG of A and the n^{th} Cα in A, and the distance between the COG of B and the m^{th} Cα in B, respectively.

We define a cutoff, denoted RMSE_CUTOFF, such that we only accept RMSE's below this value. Therefore:

$$RMSE\_CUTOFF \geq \sqrt{\frac{1}{N} \sum_{0 \leq n < N} |distA_n - distB_m|^2} \tag{7.3}$$

$$\therefore RMSE\_CUTOFF^{\,2} \times N \geq \sum_{0 \leq n < N} |distA_n - distB_m|^2 \tag{7.4}$$

*If all COG–to–Cα distance pairs, **except one** (denoted z), are zero, this reduces to:*

$$RMSE\_CUTOFF^{\,2} \times N \geq (distA_z - distB_z)^2 \tag{7.5}$$

$$\therefore RMSE\_CUTOFF \times \sqrt{N} \geq (distA_z - distB_z) \tag{7.6}$$

By setting each residue pair as the non–zero pair (z) in turn, this check can be performed along the length of the fragment. The values $distA_z$ and $distB_z$ can be pre–calculated to save even more time.

Therefore, in order to perform the check described in equation 7.6, the COG for each fragment, and the distance between each Cα atom in a fragment and the respective COG, need to be pre–calculated[10]. Furthermore, the implementation of this algorithm relies on the fact that the first and last residues ($z = 1$ and $z = 6$ in the 6–residue case) will be the most discriminating since they are furthest from the COG. Therefore, rather than looping from $z = 1$ to 6, the code forcibly matches in the order: $z = 1, 6, 2, 5, 3, 4$.

Whilst this first–order test would likely eliminate most of the pairs, a second–order quadratic test was also implemented. Again, since the first and last residues are furthest from the COG, these two residue distances are used as shown in definition 7.3.2. By pre–calculating and storing fragment lengths (*i.e.* the distance from the first residue to the last residue) and the $(length)^2/2$ values, this quadratic test does not add any extra execution time overhead. In the code, this test runs after the first–order tests, although the ordering of tests is arbitrary.

From a possible 3.52×10^{12} calls to MATFIT, 3.32×10^{12} pairs were eliminated through first–order and quadratic tests, thus requiring only 220,471,421,503 MATFIT calls. Out of these, only **43,393,718,660** pairs had RMS deviations under the 0.15Å cutoff (corresponding to a MATFIT efficiency of 19.7%) and were therefore selected for clustering. Note that this is a rejection rate of 98.7% from the original number of pairs.

[10] In contrast to MATFIT, this task can easily be parallelised.

Definition 7.3.2. *From definition 7.3.1:*

$$RMSE = \sqrt{\frac{1}{N} \sum_{0 \leq n < N} |\vec{a_n} - \vec{b_m}|^2} \tag{7.7}$$

If we require an RMS error $\leq$ RMSE_CUTOFF, then:

$$RMSE\_CUTOFF^{\,2} \times N \geq \sum_{0 \leq n < N} |distA_n - distB_m|^2 \tag{7.8}$$

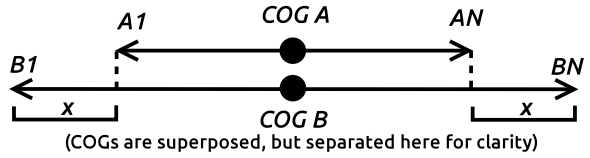
Assume atoms $z = 2 \rightarrow N - 1$ can be superposed freely. With this superposition:

1. *Result will be at least as good as* MATFIT.
2. *Superposition will have atoms 1 and N of each fragment collinear, with their (pairwise) COGs superposed. So:*

$$RMSE\_CUTOFF^{\,2} \times N \geq |distA_1 - distB_1|^2 + |distA_N - distB_N|^2 \tag{7.9}$$

*Since distance of A_1 from COG of A (1 & N) is the **same** as the distance of A_N from COG of A (1 & N), and since COGs are superposed:*

$$RMSE\_CUTOFF^{\,2} \times N \geq |distA_1 - distB_1|^2 + |distA_N - distB_N|^2 \tag{7.10}$$



$$\therefore \qquad x = |length_A/2 - length_B/2| \tag{7.11}$$

$$RMSE\_CUTOFF^{\,2} \times N \geq |length_A/2 - length_B/2|^2 + |length_A/2 - length_B/2|^2 \tag{7.12}$$

$$\frac{RMSE\_CUTOFF^{\,2} \times N}{2} \geq |length_A/2 - length_B/2|^2 \tag{7.13}$$

$$\geq (length_A)^2/4 + (length_B)^2/4 - ((length_A \times length_B))/2 \tag{7.14}$$

$$RMSE\_CUTOFF^{\,2} \times N \geq (length_A)^2/2 + (length_B)^2/2 - (length_A \times length_B) \tag{7.15}$$

Given that the indices of the window pair would need to be stored as well, only a pre–defined binary file format could produce a "reasonable" file size (where "reasonable" still requires access to high–performance machines and a large storage grid). We defined a binary file format where each score can be stored in 12 byte chunks, as shown in figure 7.8.

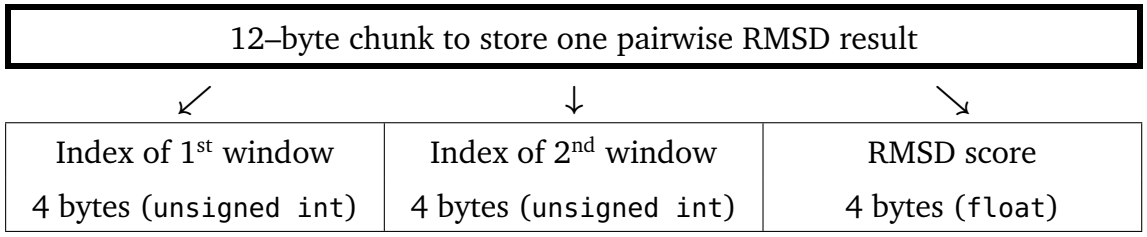


Figure 7.8: Each window–vs–window record can be stored in a 12–byte chunk. The window indices are unsigned integers from 1 to (2,653,478), and the RMSD is a single–precision floating–point number.

Note that we were able to use a 12–byte chunk only because we had fewer than 2^{32} windows, which is the maximum number of 4–byte indices that can be represented in memory[11]. However, this trade–off was deemed acceptable since we were sacrificing portability and future–proofing for speed and efficient packing on disk, which was the priority. With this storage scheme, **485GB** of disk space were required to store the final set of ~43 billion pairs.

RMSE_CUTOFF was set to **0.15Å** because whole–structure rebuilding algorithms such as SuperPose are usually set to 2.0Å (Maiti et al., 2004), and therefore, it makes sense to impose tighter restrictions on small fragments. Furthermore, from experience, α–helix RMSDs tend to be in the 0.07Å to 0.12Å range. To ensure correctness of algorithm and implementation, tests were run on smaller data sets (10k, 20k, 50k and 250k source fragments).

[11] If window indices start at 0, the largest unsigned index that can be represented in 4 bytes is 4,294,967,295 ($2^{32}-1$).

Number of source fragments	Fragments selected for clustering
10,000	3500
20,000	7278
50,000	22,950
250,000	147,366
2,653,478 (full set)	2,419,663

Table 7.4: Table showing number of fragments that actually make it to the clustering stage when RMSE_CUTOFF = 0.15Å is applied.

The results of fragment culling are shown in table 7.4. It explains that fragments that did not pair at least once with RMSE_CUTOFF set to 0.15Å were eliminated from clustering. For example, from 10,000 source fragments, only 3500 were part of at least one pair that matched closer than 0.15Å!

7.3.2.2 Clustering

The design and implementation of the clustering approach follows the ethos of the fragment–pair gathering stage described above (see section 7.3.2.1); due to the large memory requirements, data structures had to be packed in memory as tightly as possible[12]. Furthermore, the clustering algorithm had to be as parallelizable as possible in order to make efficient use of multiple–processor architectures.

The bespoke clustering algorithm has two steps. The first stage serves as a bootstrapping tool to calculate the amount of memory required for the second, actual clustering, step. This is done simply by counting the number of fragment **neighbours** for each window and writing these out to a two–column intermediate file, where the first column is the window index and the second column is the number of times the window occurs in a pair record. This gives us some extra information as well:

[12]Dynamically–allocated data structures such as STL `vector` have the advantage that the number of elements need not be known at the beginning; however, static allocation improves memory–usage and performance significantly. Due to the amount of data being manipulated here, we revert to manually–optimised, static programming practices, saving about a factor of 50 in memory!

1. Since the second column represents the number of neighbours for the window, the total of the second column $\times$ 4 bytes is the amount of RAM that is required to store everything in memory for clustering.

2. The number of lines in the file is equal to the number of windows that had at least one structurally similar fragment within the 0.15Å RMS cutoff.

3. Since only windows that have at least one neighbour are recorded to the binary file, it is straightforward to calculate the percentage of windows that are singletons, *i.e.* they were never incorporated into any pairs.

The results obtained from this first pass with the data at hand are shown below in table 7.5.

Statistic	**Value**
Total number of windows before first pass	2,653,478
Number of windows with at least one neighbour	2,484,482 (93.6%)
Number of window pairs	43,393,718,660
Total number of neighbours to store in memory	86,787,437,320

Table 7.5: Statistics of fragment pairing data elucidated after the first pass. 93.6% of windows had at least one structurally similar neighbour and therefore made it into the clustering stage. Note that the total number of neighbours (~86 billion) is exactly twice the number of window pairs (~43 billion).

After this bootstrapping step, the actual clustering begins. Figure 7.9 shows the clustering workflow. Clustering depends on a large contiguous array (`ixarray`) that stores neighbour indices in memory. Since each neighbour index can be represented in 4 bytes, the total storage required for ~86 billion indices was approximately 350GB! This necessitated the use of a special large memory machine in the Wellcome Trust Centre which had 1TB of memory with 64 hyper–threaded cores, and was therefore capable of storage and fast, parallelised manipulation of this massive data structure; the machine's specifications can be found in appendix B.1[13].

[13]Thanks to Dr Zamin Iqbal for providing access to this machine, called "*banyan*."

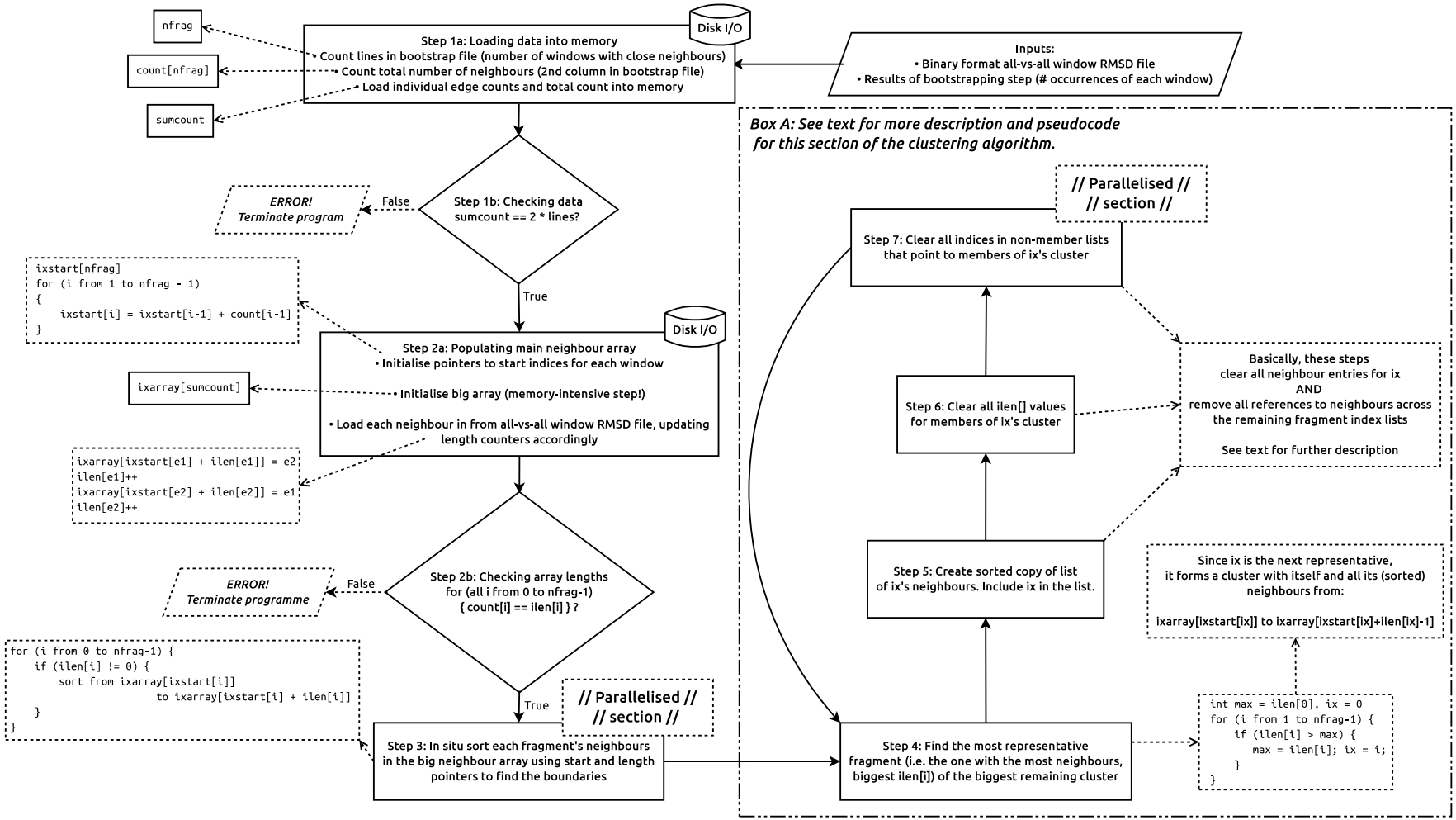


Figure 7.9: Flowchart showing the major steps in the clustering algorithm. The steps forming the loop inside Box A would be slow to execute and were therefore devised specifically so they could be parallelised with garbage collection to manage memory and make execution even faster. The details of Box A are discussed in text. Steps 3 and 7 were parallelised using OpenMP.

Steps 1 to 3 in figure 7.9 consist of preprocessing and validation stages to ensure the integrity of the data being loaded. They also prepare the large data structures for the actual clustering loop that is shown in Box A. For clarity, only a broad overview of the clustering loop is shown in Box A, however, the details are interesting as they were developed specifically with OPENMP parallelization in mind[14]. Therefore, a more detailed description of Box A is described in algorithms 7.1 and 7.2.

[14]Since OPENMP does not require any special threading code, it keeps the algorithm design and implementation clean and easy to read, with the added bonus that the code will execute perfectly in a single–threaded setup.

Algorithm 7.1 Pseudo–code describing the special optimisations and garbage collection steps that were devised for the fragment clustering algorithm specifically for efficient execution in parallel environments. The procedures correspond to the steps in Box A in figure 7.9 and variable names have been kept consistent.

1: **procedure** STEP4-FIND-MAXIMUM
2: int $nclus \leftarrow 0$; ▷ Current cluster number, for output
3: int $max \leftarrow ilen[0]$, $ix \leftarrow 0$; ▷ ix is the next representative fragment
4: **for** $i = 1 \rightarrow nfrag - 1$ **do**
5: **if** $ilen[i] > max$ **then**
6: $max \leftarrow ilen[i]$;
7: $ix \leftarrow i$; ▷ ix now points to window with the most neighbours

Postcondition: ix is the next representative window and it forms a cluster with itself and all its neighbour indices, which had been sorted beforehand in **step 3**. Therefore, ix's neighbours are found between $ixarray[ixstart[ix]]$ and $ixarray[ixstart[ix] + ilen[ix] - 1]$. We need to clear all the entry pairs for these neighbours (i.e. $ilen[i] \leftarrow 0$ ***AND*** remove all references to these neighbours from the rest of the entry pairs that are not part of this cluster (whilst updating all the $ilen[i]$ values for these entries). This is done along with retaining the sort order and garbage collecting the removed entries, which speeds up the algorithm with each iteration, reduces memory access and allows it to be perfectly parallelizable! The subsequent procedures go through this clearing process.

8: **procedure** STEP5-RETRIEVE-CLUSTER-MEMBERS ▷ Including ix
9: int $ixelim$[safely large init value]; ▷ Stores indices to be eliminated.
10: **for** $i = 0 \rightarrow ilen[ix] - 1$ **do**
11: $ixelim[i] \leftarrow ixarray[ixstart[ix] + i]$;
12: $ixelim[ilen[ix]] \leftarrow ix$; ▷ Append ix itself.
13: int $nelim \leftarrow$ size of $ixelim[]$
14: sort $ixelim[]$ ▷ from $ixelim[0]$ to $ixelim[nelim - 1]$

Postcondition: We can now eliminate all pairs where the indices stored in $ixelim[]$ occur.

Algorithm 7.2 Algorithm 7.1 continued

1: **procedure** STEP6-REMOVE-CLUSTER-MEMBER-PAIRS($ixelim[]$)
2: **for** $i = 0 \rightarrow nelim$ **do** ▷ $nelim ==$ size of $ixelim[]$
3: $ilen[i] \leftarrow 0$; ▷ Ensures these windows never get selected as representatives in Step 4.

4: **procedure** STEP7-PARALLEL-REMOVE-OTHER-PAIRS-WITH-CLUSTER-MEMBERS
▷ This is the slow and heavy loop that uses $ilen[]$ values and is designed to be perfectly parallelizable. It garbage collects as it goes along and updates $ilen[]$ values at the end.
5: **#pragma omp parallel for** ▷ OPENMP parallel for–loop
6: **for** $j = 0 \rightarrow nfrag - 1$ **do** ▷ Go through all windows
7: $i \leftarrow ixstart[j]$;
8: $k \leftarrow 0$; ▷ Pointer into $ixelim[]$
9: $l \leftarrow i$; ▷ Pointer after garbage collection
10: **while** $i < ixstart[j] + ilen[j]$ **and** $k < nelim$ **do**
11: **if** $ixarray[i] < ixelim[k]$ **then**
12: **if** $i \neq l$ **then**
13: $ixarray[l] \leftarrow ixarray[i]$;
14: **else if** $ixarray[i] > ixelim[k]$ **then**
15: $k \leftarrow k + 1$;
16: **else** ▷ $ixarray[i] == ixelim[k]$, so no more indices to eliminate
17: $i \leftarrow i + 1$;
18: $k \leftarrow k + 1$;
19: **end if**
20: **end while**
21:
22: ▷ If the while loop terminated because $k == nelim$, then there are no more window indices to eliminate. However, we still need to shuffle the remain indices down in $ixarray[]$ to compact it and remove unnecessary padding, unless $i == l$, where there is no shuffling to do.
23:
24: **if** $k == nelim$ **and** $i \neq l$ **then**
25: **while** $i < ixstart[j] + ilen[j]$ **do**
26: $ixarray[l] \leftarrow ixarray[i]$;
27: $i \leftarrow i + 1$;
28: $l \leftarrow l + 1$;
29: **end while**
30: **end if**
31: $ilen[j] \leftarrow ilen[j] + (l - i)$;
32: **end for**
33:
34: **go to** STEP4-FIND-MAXIMUM ▷ Find the next cluster representative

The way the algorithm is designed, the garbage collection happens on–the–fly as new clusters are assembled; by ensuring that no two threads will be writing to the same locations in memory, we paved the way for OpenMP to be used seamlessly to parallelise the clustering algorithm.

The algorithm is implemented in hybrid Python-C++ code, where the OpenMP parallel processing elements are written in C++, and the file input and output are handled by Python[15].

7.3.3 Cluster rebalancing

Due to the greedy, "first–come, first–serve" nature of the clustering algorithm, there will be many fragment windows which have been included into one cluster but which are more structurally similar to a representative of another cluster, particularly for large clusters such as those describing α–helices. Since it is scientifically sensible for cluster members to be closest to their representative and not the representative of another cluster, a post–processing rebalancing step was implemented to redistribute cluster members as required. Figure 7.10 describes the rebalancing process. Note that representatives do not change.

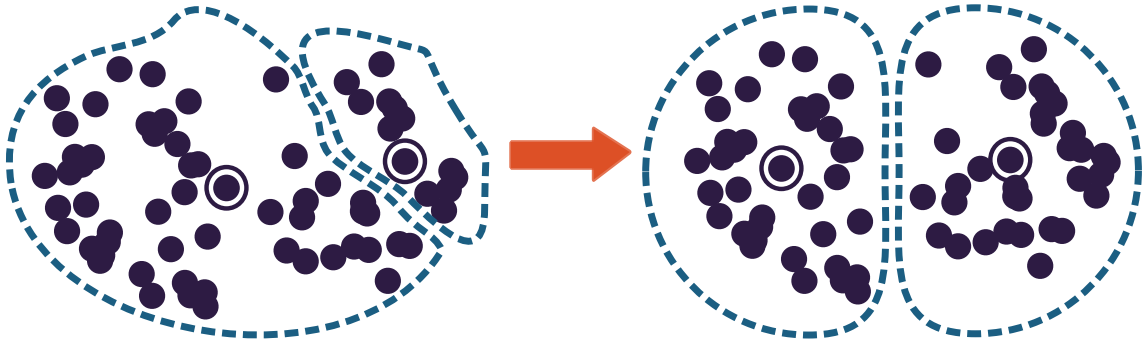


Figure 7.10: Schematic description of the rebalancing step. Representatives are highlighted as "bull's–eyes", and do not change.

Quite simply, this involves checking the RMSD of fragment windows of each

[15]This is simply because working with file I/O is much easier in Python than in C/C++.

cluster against all other representatives. Since these RMSDs have already been pre–calculated and stored as part of the data collection step ahead of clustering (see figure 7.8), the data can simply be read from disk instead of calculating RMSDs again.

The algorithm proceeds as described in algorithm 7.3 and is implemented in Python. The clusters are read into memory in step 1, and then rebalanced in step 2 by reading RMSDs from the binary file. In order to avoid reading this large file repeatedly, the data structures generated in step 2 are written to disk, so they can be accessed for diagnosis, if required. In Python, this is easily achieved by using the "`pickle`" module, which is used to serialise data structures and objects. Compared to the 350GB edge weight file, the data structures take just 100MB on disk and are therefore much more lightweight and faster to manipulate.

Algorithm 7.3 Cluster rebalancing

```
1: procedure STEP1-LOAD-CLUSTERS-GET-MAX-INDEX(input_cluster_filename)
2:     nfrag = 0;
3:     clusters[][];                    ▷ Cluster storage in memory is 2D array/vector
4:     for each line in input_cluster_filename do
5:         if line.startsWith('>') = TRUE then      ▷ FASTA header. Initialise cluster.
6:             current_cluster ← line;
7:             Initialise clusters[current_cluster];
8:         else                          ▷ Populate clusters and check for largest index.
9:             clusters[current_cluster].append(line)          ▷ Add index to cluster.
10:            if line > nfrag then
11:                nfrag ← line;                    ▷ Update largest fragment index

12: procedure STEP2-REDISTRIBUTE-CLUSTERS(nfrag, clusters[][])
13:    ▷ Initialise 5 working data structures that will be populated as the large
       binary edge–weight file is being read; these will be stored to disk for easy re-
       trieval.
14:    Init. isrep[nfrag] to 0;                  ▷ Binary flag if index is rep. or not.
15:    Init. oldrepfrag[nfrag] to −1;                          ▷ Old cluster rep. index
16:    Init. newrepfrag[nfrag] to −1;                          ▷ New cluster rep. index
17:    Init. oldreprms[nfrag] to 999.99;                  ▷ RMS of fragment to old rep.
18:    Init. newreprms[nfrag] to 999.99;                  ▷ RMS of fragment to new rep.
19:
20:    final_clusters[][];                       ▷ Final rebalanced clusters container
21:    for each edge–weight pair(e1, e2, rmsd) in binary file do
22:        if isrep[e1] = 1 then                                        ▷ Edge 1 is a rep
23:            if e1 = oldrepfrag[e2] then                      ▷ Edge 1 is edge 2's rep
24:                oldreprms[e2] ← rmsd;            ▷ Store original RMSD for edge 2
25:            if newreprms[e2] > rmsd then                  ▷ We found a lower RMS!
26:                newreprms[e2] ← rmsd;
27:                newrepfrag[e2] ← e1;
28:        if isrep[e2] = 1 then                                        ▷ Edge 2 is a rep
29:            if e2 = oldrepfrag[e1] then                      ▷ Edge 2 is edge 1's rep
30:                oldreprms[e1] ← rmsd;            ▷ Store original RMSD for edge 1
31:            if newreprms[e1] > rmsd then                  ▷ We found a lower RMS!
32:                newreprms[e1] ← rmsd;
33:                newrepfrag[e1] ← e2;
34:    Dump isrep, oldrepfrag ,newrepfrag, oldreprms, newreprms to disk.
35:    for i = 0 → nfrag do                                  ▷ Assemble new clusters.
36:        if isrep[newrepfrag[i]] = 1 then                                  ▷ i is a rep.
37:            Initialise final_clusters[newrepfrag[i]];
38:            final_clusters[newrepfrag[i]].append(newrepfrag[i]);   ▷ Self–add i
39:        ▷ Add i to its new cluster. If the clusters have not changed oldreprms
       and newreprms will match, preserving the original grouping.
40:        final_clusters[newrepfrag[i]].append(i);
```

7.4 Results and Discussion

The initial all–vs–all RMSD calculation takes most of the time, whilst the clustering step is extremely quick by comparison. The breakdown of time taken for the full data set is shown in table 7.6 below. Predictably, the slow down was primarily due to portions of the clustering that could not be parallelised (file input and output, or calls to MATFIT, which is not designed for shared–memory parallelization methods such as OPENMP). The last column in the table shows a high level overview of memory and disk usage to highlight the bottlenecks and hardware limitations that were overcome with the computing resources available at the Wellcome Trust building.

	Stage	Time	Hardware Usage
1.	Generating all–vs–all RMSD	5.5 days	485GB disk
2.	1st pass, generating bootstrap file	14.15 hours	27MB disk
3.	2nd pass, populating data structures	36.14 hours	347GB **memory**
4.	Parallelised clustering (64 cores)	**2.18 hours**	20MB disk
5.	Cluster rebalancing	16.15 hours	120MB disk

Table 7.6: Breakdown of time taken for each stage of clustering the full data set. Single–threaded execution accounted for the slower running times of steps 1, 2, 3 and 5. The actual clustering algorithm (step 4) is extremely quick, although single–threaded, it would take close to 6 days to complete! The token 27MB disk space taken by step 2 is to write out the first pass bootstrap file that is then used to populate the data structures in step 3. Similarly, 20MB disk space is required at the end of step 4 to write the final clusters out to disk. The rebalancing step has almost exactly the same overhead as the first pass. It writes out rebalanced clusters to disk, along with 100MB of data structures for analysis and debugging.

Interestingly, steps 2 and 3 had very different running times, although they both involved opening and reading the same binary–format all–vs–all RMSD file. Step 3 needed extra time for reserving space in memory for the large arrays and for physically loading the data in, as opposed to simply counting occurrences of windows in step 2. Since disk space is cheap, the hardware requirement for step 1 was easily met; however, the limiting factor for manipulating data sets of this size is, of

course, step 3, which requires an extremely large memory footprint. Step 1 was by far the slowest stage, despite the pre–filtering in place to avoid unnecessary calls to MATFIT[16]. It is estimated that, if the filtering were absent, step 1 would have taken approximately two months to complete[17]!

The RMSE_CUTOFF variable (set to 0.15Å) shows its importance here. A slight alteration could have large repercussions downstream in terms of hardware requirements! Our estimates on smaller data sets, therefore, made it very helpful to spec the memory requirements to cluster the full set. Similarly, the use of a pre–defined binary file format for storing all–vs–all RMSD window pairs ensured that the required disk and memory requirements could be precisely estimated, and also improved the speed of reading the file in in subsequent steps. ASCII human–readable files, by comparison, have a much larger footprint[18] and are much slower to process. These exercises in stingy coding, so to speak, and careful attention to parallelization requirements were especially useful since the clustering algorithm could then be designed with the size of the data in mind right from the beginning, and most importantly, we could be confident about obtaining some sort of result instead of running into issues as we did with *MCL*!

The final number of clusters, and therefore, representative sample fragments, is shown in table 7.7.

[16]Although outside the scope of this thesis, it is the author's belief that a free (and open–source), thread–safe library implementation of MATFIT would be an extremely useful contribution for structural biology software developers, as it would enable parallelization without workarounds or caveats in a wide variety of applications.

[17]3.52×10^{12} pairs $\times$ 1.5μsec per MATFIT call = ~61 days.

[18]In modern ASCII, each character needs one byte for representation, so an edge weight label of 1234567, for example, would require 7 bytes; in contrast, a binary representation of this label takes 4 bytes. This saving makes a large impact with the data sizes discussed here.

Number of fragments	Number of clusters
3500	658
7278	1343
22,950	4178
147,366	23,617
2,419,663	**198,130**

Table 7.7: Number of clusters obtained for the different test sets. The full set is highlighted in bold face.

The immediate question that comes to mind is: "how many sample (representative) fragments are required to cover X % of the PDB?" Since the data for the full set are large, the results from the smallest data set (3500 fragments) are shown in figure 7.11. Note that this output is obtained from the clustering program. The sizes of the top 20 clusters are shown, before and after rebalancing. The representative coverage is also shown; for example, the top 14 representatives, after rebalancing, cover 50% of the input fragment set. The table is sorted in decreasing order of *rebalanced* cluster size.

IDX	OSIZE	TOTAL OLD	ORIG. %	NSIZE	TOTAL NEW	NEW %
1	1158	1158	33.086	729	729	20.829
2	220	1378	39.371	227	956	27.314
3	160	1538	43.943	165	1121	32.029
4	83	1621	46.314	121	1242	35.486
5	48	1669	47.686	112	1354	38.686
6	64	1733	49.514	94	1448	41.371
7	**31**	**1764**	**50.400**	67	1515	43.286
8	11	1775	50.714	67	1582	45.200
9	30	1805	51.571	38	1620	46.286
10	15	1820	52.000	32	1652	47.200
11	33	1853	52.943	27	1679	47.971
12	6	1859	53.114	25	1704	48.686
13	18	1877	53.629	25	1729	49.400
14	20	1897	54.200	**23**	**1752**	**50.057**
15	19	1916	54.743	22	1774	50.686
16	8	1924	54.971	20	1794	51.257
17	11	1935	55.286	19	1813	51.800
18	16	1951	55.743	18	1831	52.314
19	9	1960	56.000	18	1849	52.829
20	4	1964	56.114	17	1866	53.314

Figure 7.11: Top 20 clusters from clustering output for 3500 fragments, adapted from the clustering program's output. Pre– and post–rebalancing results are shown. "OSIZE" and "NSIZE" refer to old and new cluster sizes, respectively. The "TOTAL OLD" and "TOTAL NEW" columns are cumulative numbers of fragments. The percentages show cumulative coverage of the 3500–fragment set, and 50% coverage before and after rebalancing is highlighted in bold face.

Similar results were obtained for the other data sets; for the full run, the top cluster contained **568,677** members, corresponding to a coverage of 23.9%. The number of representatives required for 50% coverage are shown in table 7.8. Note that for the full set, the 50% coverage means that just over 7,000 fragment samples are required to represent half of the structure fragments in the PDB!

Number of fragments	Frags. required for 50% coverage
3500	14
7278	35
22,950	202
147,366	1,569
2,419,663 (full set)	**7,151**

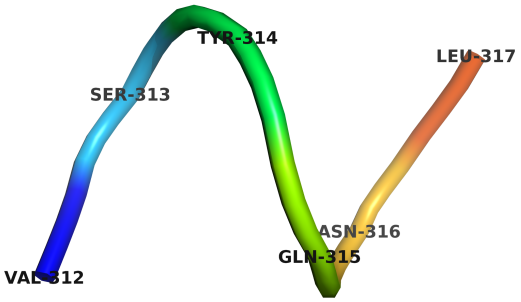
Table 7.8: Results for all test sets showing the number of fragments required after rebalancing to cover 50% of the input data.

Therefore, out of 198,130 clusters (see table 7.7), the top 7,151 contain enough information to represent half of the PDB. This result is not surprising, since the expectation is that the large majority of common fragments (mainly α–helices) would be represented within the top few clusters (note that cluster sizes are sorted in decreasing order during the clustering and rebalancing steps).

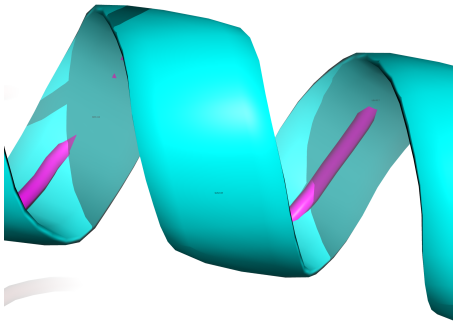
An interesting qualitative result can also be obtained by modelling fragments in PyMol (DeLano, 2002), to visualise fragment structures within that representative's cluster. A Python script was written to write the coordinates of fragments into the standard ATOM format. These files were then loaded into PyMol to render the fragments. The structures of the top five sample fragments from the clustering (and rebalancing) of the full data set are shown in figure 7.12. Source ATOM files for these structures and PyMol rendering information are provided in appendix B.2. Helices are drawn by PyMol as broad spirals[19]. A superposition of the top representative on its source protein structure is shown in figure 7.12b to illustrate where

[19]Note that the representative of the largest cluster is not identified as a helix by itself within PyMol even though it is one, as seen in figure 7.12b. Its members are identified as helices, however.

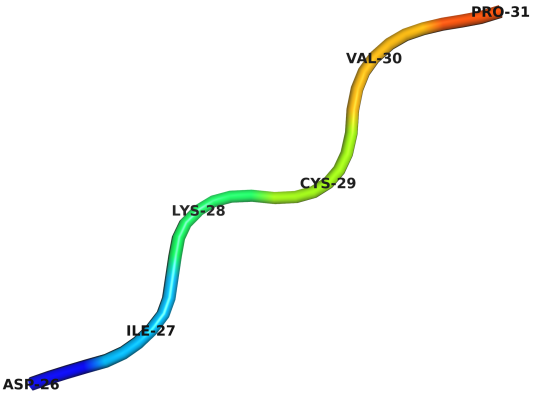
the fragment came from originally. Note that all coordinates are fully optimised by *PDB_REDO*. The second largest family consists of wavy loop regions that are found in β–sheets in the same structure, 1XDW. (the representative in figure 7.12c is also shown in figure 7.12d as part of the whole structure. Note that it is neatly embedded into a β–strand denoted as a yellow arrow, and that parallel strands is visible in the background. The third representative is α–helical and comes from PDB entry 3AG3 (Muramoto et al., 2010; figure 7.12e). The fourth and fifth families (figure 7.12f) consist of fragments involved in turns and short loops, and come from PDB entries 3RA6 (Chan et al., 2011) and 2OB0 (Walker et al., 2006), respectively.



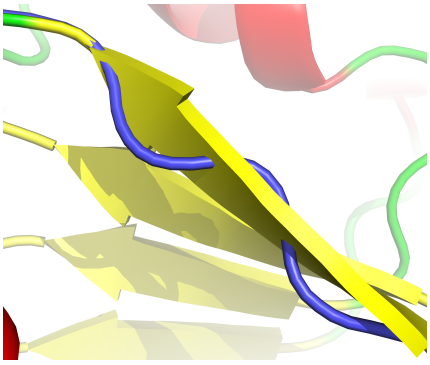
(a) Representative for the largest cluster. Members are helical, but the representative is "not quite", according to PyMol.



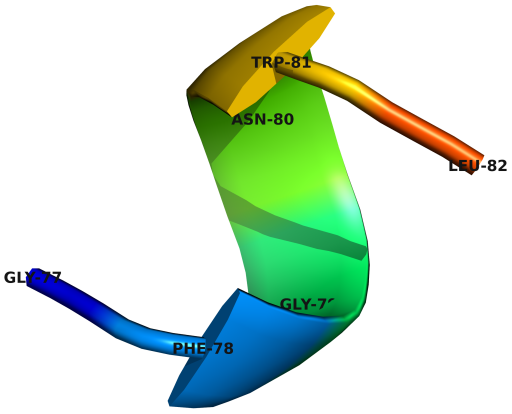
(b) The representative (magenta) shown with the source α–helix structure (blue).



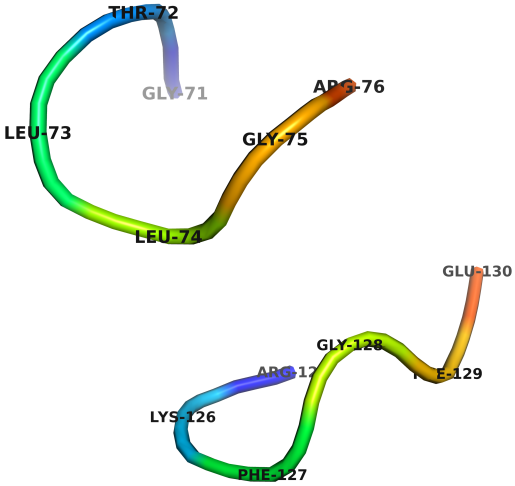
(c) The representative for the second largest cluster, a β–strand.



(d) The representative (blue) shown within the source structure as part of a β–strand (yellow arrow). Parallel strands are in the background.



(e) Representative for the third cluster. Another classic α–helix.



(f) Representatives for the fourth (top) and fifth (bottom) largest clusters.

Figure 7.12: A qualitative survey of the fragments representing the five largest clusters. The two most common fragment motifs (represented in 7.12a and 7.12c, respectively) are shown within their source structures (figures 7.12b and 7.12d, respectively). Where applicable, the N–termini are shown in blue; the C–termini are in orange. See appendix B.2 for more details.

7.4.1 Applications

The fragment clustering method is fast (once the data are loaded) and scalable, and provides a way to survey short fragments in the PDB. The most obvious application for this is in a structure–rebuilding tool such as CALPHA (Esnouf, 1997), which uses an internal database of fragments to rebuild the Cα backbone of low–resolution structures. By sampling the PDB in the manner described here, we have effectively reduced 2.5 million fragments to just under 200,000 representatives. Work is currently being done to update CALPHA's database with the representatives obtained through the work described in this chapter, in the database format described in figure 7.7, with the difference being that each fragment is of a fixed length (6 residues). The updated CALPHA will then contain a high–quality repository of fragments, and can possibly be utilised for fast, initial structure rebuilds as part of an automated structure determination pipeline. Furthermore, with the advances in cryo-EM structure solution methods, this database has a natural application for backbone and loop building applications.

The clustering algorithm devised in this chapter is a drop–in replacement for *MCL* in clustering ordered windows for *MoreRONN*-O (see chapter 6), where the similarity measure is the BBF score between two windows. Whilst it is a greedy algorithm and not as precise as *MCL*, the added precision is a moot point when the data set is large and there is granularity between the elements; the work with the contrast function in the BBF (see section 5.3.2) has gone a long way towards resolving closely–related windows, enabling the use of a greedy approach.

Chapter 8

General Conclusions and Future Directions

8.1 General conclusions

The PDB is now growing at a rate of approximately 10,000 new structures per year. The techniques of crystallography, NMR and electron microscopy are constantly improving, as are the computational tools and power of computing resources available. Large facilities such as synchrotrons drive ever–more sophisticated methodologies. Even for older PDB entries, pipelines like *PDB_REDO* are now sophisticated enough to enable the improvement and standardisation of structure quality. However, the PDB and other experimentally–derived, biological databases are often just treated as archives, where the data are not looked at any further on a global, database–wide scale. Now that these databases are large, bioinformatics should move to a phase of extensive analysis.

There are many good bioinformatics tools that are often built on great insight but with little data. These tools are often developed against small data sets (and

validated against small data sets), embodying empirical rules without scaling well to the full database. In particular, modern computers are only slowly getting faster *per core*; the majority of computational performance increase is a result of machines containing many more cores. Typically, 16 or 32 cores are readily available on a small, relatively inexpensive symmetric multiprocessing (SMP) box[1]. Most bioinformatics code was written before these advances and, therefore, simply cannot exploit these new architectures and processor enhancements[2]. In the case of the PDB, given its size and the aforementioned problems with legacy bioinformatics code, there has been little global analysis of trends. This thesis has sought to develop novel methods and code for today's multi–core machines to enable such a global analysis.

8.1.1 *Nearest–cell*

The initial work with *Nearest–cell* focused on developing a pipeline, PRAXIS, for farming crystal structures from the PDB. These data were then applied to the *Nearest–cell* algorithm itself, in order to facilitate fast PDB–wide unit cell matching. *Nearest–cell* was cited in McGill et al. (2013) and compared to the superior $\boldsymbol{G}^6$ method, but this new approach is not yet usable by the public. In contrast, *Nearest–cell* remains a fast and usable tool, where conceptually, its internal engine could be modified in the future to work in the $\boldsymbol{G}^6$ space instead of relying on PHENIX to perform the cell reduction. However, it is clear that the problem of matching crystal forms in the PDB has been considered by others, and *Nearest–cell* offers a publicly–accessible interface for both occasional users and crystallographers on a beamline.

[1] SMP architectures describe two or more processing units connected to the same pool of memory, *i.e.* a personal computer with multiple processors inside.

[2] There are "embarrassingly parallel" ways of forcing a bioinformatics routine to run across multiple cores through brute–force data partitioning, but this is not elegant or sustainable.

8.1.2 RONN

The classic RONN algorithm was pushed to its limits and beyond through complete rewrites of its linear algebra modules to improve numerical stability. The integration of the EIGEN library enables the use of high–quality, peer–reviewed linear algebra solvers that are optimised for modern processors[3] and parallelised as much as possible. The resulting RONN implementation is robust and scales easily to accommodate larger prototype data sets, and can be finely–tuned to the capabilities of the host hardware[4], a fact that is important when deploying the predictor as a web service in order to optimise running time as much as possible. However, despite the size of the new data set, only modest improvements were achieved with the prediction itself, which suggests that the algorithm is the limiting factor.

8.1.3 *MoreRONN*

With the development of *MoreRONN*, we found a new way of partitioning the same prototype and training data through the use of *MCL* clustering, which led to a nearly 20% improvement in probability excess over RONN. The resulting algorithm is very forgiving when it comes to input parameter combinations, to the point that even a single cluster produced remarkably high PE scores. Attempts were made to extend it to clustering and training on ordered prototypes (*i.e.* the development of *MoreRONN*-O), but the limits of *MCL* were reached before useful results could be obtained.

[3] EIGEN can even be optimised for certain specific processor architectures, such as modern Intel chips that have the latest generation of vector compute algorithms and extensions built directly into the hardware.

[4] Using compiler flags that optimise the output executable to take full advantage of processor extensions.

8.1.4 Structure fragment clustering

In order to avoid the pitfalls with *MCL*, the size of the data set and large–scale clustering considerations were addressed from the outset through various pre–filtering steps and careful design of the clustering algorithm to parallelise effectively. The demanding requirements on memory and disk space necessitated the use of different machines for different stages of the pre–filtering and clustering! The ~2.5 million fragment input was successfully reduced to just under 200,000 fragments in the PDB, implying that there are in fact a few very common motifs (such as α–helices), followed by a long tail of variable and rarer fragments. Notably, as sample size increases, cluster growth slows down which opens up the tantalising possibility that we might be close to getting comprehensive coverage of fragments in the PDB. This has future implications for model building, especially in the realm of electron microscopy, as this method is slated to dominate within a few years. However, application work is still required in this domain.

8.2 Future directions

The work in this thesis sought to make efficient use of the entire PDB to develop robust, scalable bioinformatics techniques to analyse and learn about protein structure, with emphasis on high–throughput applications such as beamline pipelines. To that end, *Nearest–cell* is already in regular use. Improvements in the web service are planned in order to collect metrics on usage.

As with many doctoral research projects, time constraints have left incomplete threads, including raising some profound questions and challenges for future work. Several threads of further work are immediately apparent:

1. Further refinement of *MoreRONN*, including application of the technique to

proteins–of–interest within the Division of Structural Biology.

2. The development of an explicit order predictor, using a variant of the clustering engine developed in chapter 7, and then the combination of both predictors into a single, novel mixed predictor with improved edge–case determination.

3. An ultimate goal is to allow the reliable annotation of order/disorder states within UniProt and the development of a new public database with this information.

4. The analysis and application of the database of structural fragments to building protein structure models. Some of this work is being undertaken at present, with the integration of the new fragment database into CALPHA.

8.2.1 Further refinement and application of *MoreRONN*

MoreRONN and its predecessor, RONN, are unique among disorder predictors in that they use a neural network based directly (and purely) on pairwise sequence similarity, thereby avoiding preconceived bias from ascribing amino acid properties. The fundamental advance in *MoreRONN* has been to cluster the prototype data differently, through the use of *MCL*. Disorder data are indirectly inferred from the PDB and DisProt, and continued data expansion and curation are required as more sequences and structures are deposited.

The window length of 15 residues, whilst providing the best predictions, seems relatively long compared to other biological motifs such as peptides within MHCs. It also results in smoothed predictions at the expense of some precision in determining transition points. For short regions of disorder this can be particularly noticeable. Therefore, optimisation of *MoreRONN* at shorter window lengths is an important future task to try and determine the biological limits of encoding disorder. This will

also increase sensitivity of *MoreRONN* for short regions of disorder, such as loops and inter–domain linkers. For instance, disordered prototypes < 15 residues are currently discarded, as they are shorter than the window size. However, this likely causes valuable prototype information to be lost as it is likely that short motifs that might occur in 5–residue stretches might very well be absent in the longer 15–residue windows. One way of including these shorter disordered stretches might be to cluster windows of each length smaller than 15 residues separately. As we saw with the results of the single clustering, a measurable signal might still be generated, and the addition of these shorter windows might increase sensitivity. The trainer and predictor would have to be modified accordingly to work with these shorter window clusters, but there is nothing intrinsic to the algorithm or implementation of RONN or *MoreRONN* that prevents this.

MoreRONN executes very quickly, and thus is perfectly suited to analyse large families of sequences of interest to other members of the Division of Structural Biology and the broader structural biology community as a whole. There may also be potential for multiple–alignments ahead of the prediction to improve predictions for families of related sequences to look for conserved (and hence, biologically significant) features. This approach is analogous to the method used by secondary structure tools as such JPRED (Cole et al., 2008) to improve prediction, but JPRED is weaker when predicting with single inputs rather than multiple inputs; conversely, one of *MoreRONN*'s strengths is its ability to reliably predict disorder on a single, and possibly novel sequence.

8.2.2 Developing an order predictor and a combined predictor

MoreRONN derives the bulk of its training data from the indirect inference of disorder in the PDB. However, this is perverse since the PDB is a well-curated repository of structure and so the bulk of data are being discarded. Furthermore, since

disorder is a binary property (i.e. a residue is either ordered or disordered), a perfect order predictor is equivalent to a perfect disorder predictor.

The preliminary work outlined in chapter 6 showed some promise but had to be cut short due to time limits. Nevertheless, as an indicative result, an order predictor based on only 10% of the PDB data produced disorder predictions nearly as good as RONN (40.7%, as opposed to 51.1% for RONN 4.0). This result was very encouraging, but at the time this work was conducted, we had no protocols in place for very large–scale clustering. The work on short–fragment clustering (chapter 7) has, however, led to the development of a very fast, very scalable (although necessarily somewhat crude) clustering algorithm. It will likely be possible to integrate this clustering algorithm into *MoreRONN*-O to overcome the size of the data to be clustered[5]. Note that the number of ordered windows to be clustered (~3.5 million) is on the same scale as the number of fragments that were clustered with the method described in chapter 7 (~2.6 million), which is encouraging.

The development of an order predictor, however, is only intended as a stepping stone to the ultimate goal: a combined order–disorder predictor (*MoreRONN*-C). Many machine-learning techniques (such as support vector machines) focus on discrimination of the edge cases between two states to develop good predictors. For disorder prediction, the ability to move to the next level of accuracy by changing machine-learning engine depends on developing independent order and disorder predictors of comparable quality. As outlined above, this seems possible within the *MoreRONN* framework, at first glance. Mathematically, it simply requires a change to the first column of the **F** matrix (see equation 5.1) to define order and disorder as states –1 and +1, instead of 0 and 1 as in RONN and *MoreRONN*[6]. Developing such

[5]The only difference is in the scoring, whereby the RMSD in the clustering case is replaced by the BBF score in the order prediction case.

[6]In rigorous terms, 1 denotes disorder, and 0 denotes *lack thereof*, or an unknown, in RONN and *MoreRONN*. With defined order and disorder states, –1 might depict order, +1 might depict disorder, and scores around 0 (perhaps between –0.5 and +0.5) might denote unknowns. In both cases, unknowns simply result from lack of prototype information to fit the window under examination.

a combined predictor would be a milestone result as no such combined predictor exists, preliminary results suggest that it can be done, and it has the potential to raise in-silico disorder prediction to a level of accuracy and precision that make it a reference tool for all protein scientists. By initially developing the two–layer neural network (*MoreRONN*-OD), this route also opens up exploration of other machine learning techniques.

8.2.3 Annotation of UniProt

Whether using the *MoreRONN* disorder predictor or an improved combined predictor, the facility now exists to be able to develop a database that annotates and maintains predicted order/disorder states for all proteins in the UniProt database and make this freely available. For example, it may enable comparison of predicted disordered regions with annotations in Pfam to identify potential conserved disordered regions or ordered domains.

8.2.4 Exploiting the database of diverse structural fragments

Even though only a preliminary analysis of the outputs of the structural clustering of the PDB could be conducted, the qualitative results corroborated with the initial hypothesis that common motifs would indeed be clustered together and would form the largest clusters. As well as providing a view onto how elements of structure are clustered and how sequence might determine structure[7], this work can be exploited practically. By providing a small database ($<$ 200,000 fragments) giving near–complete ($>$ 99%) coverage of the structural variation within the PDB (see table 7.7), we hope to enable the development of new, fast tools for structure solution and refinement. In particular, for cases where only low–resolution X–ray

[7]Only backbone coordinates were analysed through superposition, but ATOM fragments also have sequence information.

data are available (as is often the case with leading-edge research into molecular complexes and machines), the use of such fragments may offer a practical route to reliable structure refinement.

With limited data, the key challenge is to improve the observations–to–parameters ratio, meaning that reduced representations of structure are required. A comprehensive database of structure fragments offers a practical solution to this problem, for example, representing the position of 30 main–chain/Cβ atoms in 6–residue fragments with only a fragment index and six translational and rotational degrees of freedom. Therefore, collaboration with experimental crystallographers (at Diamond Light Source, for instance) provides an opportunity to develop tools to meet these challenges.

8.3 Computing environment and outreach

The infrastructure for research computing within the Wellcome Trust Centre for Human Genetics is unparalleled within the University. Much of the computation carried out within the Centre requires large amounts of memory, and for applications such as the clustering challenges outlined above, it is one of the few places worldwide, outside large facilities and supercomputing centres, that can support large–memory computing. Indeed, this year it will deploy a set of machines with $>$ 1.5TB integrated memory with high–performance, high–capacity storage. This has enabled, without obstacles, the large–scale work described in this thesis to be conducted; as a result, software tools have been designed and built with modern, parallel–processing capable machines in mind. Whilst analysis and methods development require large resources, to make an impact in research requires these advances to be embodied in lightweight, fast tools suitable for web services or local installation. The goal in my research has always been to develop such robust,

public–friendly tools and resources. *Nearest–cell* is already published, the work with *MoreRONN* is being prepared for publication, and another publication is expected from the work on structural fragment clustering.

8.4 Concluding remarks

Robust computational structural biology has never been more important than now, as the quantity of data is growing at rates that demand carefully designed and implemented software tools that are "future–proof." The work presented in this thesis has enabled the author's personal goal of developing and delivering reference–grade structural biology software, with an emphasis on the "general solution", that will have a long shelf life and reach a large community. In particular, it is hoped that these tools can reduce analysis time at the beamline (*Nearest–cell*), the cost of designing constructs and growing crystals (*MoreRONN*), and the quality of structure building and refinement (fragment clustering), especially since preliminary use cases have shown promise. However, all three tools have also led to a slightly better understanding of the fundamentals of protein structure, which will hopefully impact an even larger audience and future generations of biological scientists.

Appendix A

Code snippets

A.1 SQL queries

Listing 2 below shows the query that was run on the PRAXIS database to retrieve suitably high–quality PDB entries for re–training RONN (see section 3.4.1.2). PRAXIS stores parsed data from PDB XML files in a table called "`pdb_data`".

Listing 3 shows the basic code required to extract windows from a query sequence as discussed in chapter 3.7.

```
/* Get values from the "pdb_data" table and cast "resolution_high"
to a decimal number since it is stored as a string in pdb_data */
SELECT pdb_id, r_working, angle_dev, geom_dist_dev, resolution_high,
to_number(resolution_high, '9.999') FROM pdb_data

/* We do not want null or invalid data. This can happen if the
source XML file does not contain data in the accepted format. */
WHERE (r_working IS NOT NULL AND angle_dev IS NOT NULL
AND geom_dist_dev IS NOT NULL)
AND (r_working != -99 AND angle_dev != -99 AND geom_dist_dev != -99)
AND resolution_high != ''

/* Check against our filtering criteria */
AND method='X-RAY DIFFRACTION'

/* Check the numerical cast of resolution_high
against the desired filtering cutoff value */
AND to_number(resolution_high, '9.999') < 2.5

/* Apply remaining cutoffs */
AND (geom_dist_dev < 0.015) AND (angle_dev < 2.0)
AND (r_working < 0.25)
```

Listing 2: PostgreSQL query to filter PDB entries for RONN training.

```
list_of_windows = list() #Array to hold all the windows ***see text
for prototype_sequence in list_of_sequences:

    windows = generateWindows(prototype_sequence)
    list_of_windows.append(windows) #*** see text

# a) Write list_of_windows to prototype input file.
# b) Write one window per line.

# Use this new window input file in training.

## Continue with training as before. ##
```

Listing 3: Python snippet to extract windows. "`generateWindows()`" is a Python generator function that slides along the sequence and returns an array containing each window of the desired residue length. The lines marked by "***" are discussed in the text in section 3.7.3.

Appendix B

Miscellaneous information

B.1 The fragment clustering machine

Access to the machine ("*banyan*") on which fragment clustering was performed (see section 7.3.2.2) was kindly provided by Dr Zamin Iqbal (personal communication). This machine's specifications are listed below. The principal requirement was a RAM capacity greater than 350GB, which *banyan* was able to provide.

Processor	Intel(R) Xeon(R) CPU E5-4650
Clock speed	2.70GHz
Cache size	20, 480 KB
Memory	1TB
Cores	32 physical, 64 hyperthreaded

B.2 Rendering fragments with PyMol

To generate the images of the fragments shown in figure 7.12, a Python script extracts fragment windows from clustering output, which is in the custom database format described in figure 7.7a, and stores them as standard ATOM files, which can then be read in by PyMol to render the model. A FASTA–header is written at the

top; please see figure 7.7a for the specification. The index of this fragment window is also written as part of the header; this provides an easy way to test whether the correct fragments are being retrieved. All images were exported in PNG format at 2400 $\times$ 2400 pixels using the option "`ray 2400,2400`" in PyMol.

Figure 7.12a is the representative of the largest cluster, which contains 568,677 members. It is a fragment from PDB entry 1XDW (Martins et al., 2005), chain A. The ATOM record is shown below.

```
>1xdw_A_211_120| FRAGMENT START INDEX: 1484171
ATOM      1   N  VAL A  312   -157.222 145.600-627.120  1.00 23.03           N
ATOM      2   CA VAL A  312   -158.356 146.409-627.489  1.00 23.03           C
ATOM      3   CB VAL A  312   -158.901 147.219-626.295  1.00 23.03           C
ATOM      4   C  VAL A  312   -157.991 147.303-628.674  1.00 23.03           C
ATOM      5   O  VAL A  312   -158.784 147.468-629.614  1.00 23.03           O
ATOM      6   N  SER A  313   -156.771 147.862-628.650  1.00 23.03           N
ATOM      7   CA SER A  313   -156.363 148.713-629.761  1.00 23.03           C
ATOM      8   CB SER A  313   -155.016 149.410-629.472  1.00 23.03           C
ATOM      9   C  SER A  313   -156.295 147.949-631.116  1.00 23.03           C
ATOM     10   O  SER A  313   -156.712 148.461-632.164  1.00 23.03           O
ATOM     11   N  TYR A  314   -155.736 146.755-631.100  1.00 23.03           N
ATOM     12   CA TYR A  314   -155.746 145.920-632.268  1.00 23.03           C
ATOM     13   CB TYR A  314   -155.001 144.639-631.994  1.00 23.03           C
ATOM     14   C  TYR A  314   -157.175 145.604-632.722  1.00 23.03           C
ATOM     15   O  TYR A  314   -157.471 145.611-633.916  1.00 23.03           O
ATOM     16   N  GLN A  315   -158.037 145.238-631.798  1.00 23.03           N
ATOM     17   CA GLN A  315   -159.421 144.964-632.164  1.00 23.03           C
ATOM     18   CB GLN A  315   -160.237 144.354-631.007  1.00 23.03           C
ATOM     19   C  GLN A  315   -160.068 146.256-632.715  1.00 23.03           C
ATOM     20   O  GLN A  315   -160.862 146.218-633.680  1.00 23.03           O
ATOM     21   N  ASN A  316   -159.752 147.398-632.122  1.00 23.03           N
ATOM     22   CA ASN A  316   -160.330 148.632-632.592  1.00 23.03           C
ATOM     23   CB ASN A  316   -159.836 149.815-631.779  1.00 23.03           C
ATOM     24   C  ASN A  316   -159.927 148.846-634.084  1.00 23.03           C
ATOM     25   O  ASN A  316   -160.709 149.284-634.902  1.00 23.03           O
ATOM     26   N  LEU A  317   -158.676 148.562-634.409  1.00 23.03           N
ATOM     27   CA LEU A  317   -158.165 148.835-635.769  1.00 23.03           C
ATOM     28   CB LEU A  317   -156.657 148.660-635.807  1.00 23.03           C
ATOM     29   C  LEU A  317   -158.869 147.890-636.755  1.00 23.03           C
ATOM     30   O  LEU A  317   -159.333 148.300-637.845  1.00 23.03           O
```

Since PyMol did not characterise it as an α–helix, even though it looks helical, the whole structure was included in figure 7.12b to show that this fragment is

indeed nestled within an α–helix[1]!

The representative of the second–largest cluster (figure 7.12c is a wavy loop region that forms part of a β–sheet in the same structure, 1XDW. The ATOM records are shown below.

```
>1xdw_A_23_62| FRAGMENT START INDEX: 1483909
ATOM      1   N  ASP A   26   -148.602 144.299-639.744  1.00 23.03           N
ATOM      2   CA ASP A   26   -147.187 144.575-639.944  1.00 23.03           C
ATOM      3   CB ASP A   26   -146.911 144.947-641.403  1.00 23.03           C
ATOM      4   C  ASP A   26   -146.833 145.695-638.997  1.00 23.03           C
ATOM      5   O  ASP A   26   -146.945 146.874-639.348  1.00 23.03           O
ATOM      6   N  ILE A   27   -146.410 145.340-637.791  1.00 23.03           N
ATOM      7   CA ILE A   27   -146.199 146.351-636.743  1.00 23.03           C
ATOM      8   CB ILE A   27   -146.609 145.867-635.337  1.00 23.03           C
ATOM      9   C  ILE A   27   -144.760 146.849-636.683  1.00 23.03           C
ATOM     10   O  ILE A   27   -143.801 146.081-636.813  1.00 23.03           O
ATOM     11   N  LYS A   28   -144.604 148.137-636.434  1.00 23.03           N
ATOM     12   CA LYS A   28   -143.311 148.647-636.046  1.00 23.03           C
ATOM     13   CB LYS A   28   -142.775 149.594-637.106  1.00 23.03           C
ATOM     14   C  LYS A   28   -143.475 149.381-634.726  1.00 23.03           C
ATOM     15   O  LYS A   28   -144.385 150.206-634.580  1.00 23.03           O
ATOM     16   N  CYS A   29   -142.578 149.110-633.783  1.00 23.03           N
ATOM     17   CA CYS A   29   -142.603 149.737-632.470  1.00 23.03           C
ATOM     18   CB CYS A   29   -142.568 148.686-631.382  1.00 23.03           C
ATOM     19   C  CYS A   29   -141.410 150.674-632.317  1.00 23.03           C
ATOM     20   O  CYS A   29   -140.264 150.267-632.506  1.00 23.03           O
ATOM     21   N  VAL A   30   -141.651 151.922-631.941  1.00 23.03           N
ATOM     22   CA VAL A   30   -140.539 152.814-631.682  1.00 23.03           C
ATOM     23   CB VAL A   30   -140.467 153.980-632.697  1.00 23.03           C
ATOM     24   C  VAL A   30   -140.684 153.339-630.280  1.00 23.03           C
ATOM     25   O  VAL A   30   -141.822 153.350-629.742  1.00 23.03           O
ATOM     26   N  PRO A   31   -139.558 153.762-629.653  1.00 23.03           N
ATOM     27   CA PRO A   31   -139.651 154.338-628.332  1.00 23.03           C
ATOM     28   CB PRO A   31   -138.200 154.258-627.795  1.00 23.03           C
ATOM     29   C  PRO A   31   -140.104 155.787-628.328  1.00 23.03           C
ATOM     30   O  PRO A   31   -140.551 156.305-627.291  1.00 23.03           O
```

Figure 7.12d shows this figure within the context of the whole structure. Note that it is clearly part of a β–strand.

The representative for the third–largest cluster is a α–helix from PDB entry 3AG3 (Muramoto et al., 2010). The ATOM record is shown below.

[1]For an as–yet–undetermined reason, PyMol assigns the fragment as a helix when it has more neighbouring atoms available for context.

```
>3ag3_A_72_41| FRAGMENT START INDEX: 2380414
ATOM      1   N  GLY A   77     57.882 317.338 210.667  1.00 23.03           N
ATOM      2   CA GLY A   77     56.874 317.007 211.648  1.00 23.03           C
ATOM      4   C  GLY A   77     57.262 315.854 212.556  1.00 23.03           C
ATOM      5   O  GLY A   77     57.587 316.036 213.734  1.00 23.03           O
ATOM      6   N  PHE A   78     57.242 314.641 212.008  1.00 23.03           N
ATOM      7   CA PHE A   78     57.623 313.519 212.781  1.00 23.03           C
ATOM      8   CB PHE A   78     57.312 312.222 212.006  1.00 23.03           C
ATOM      9   C  PHE A   78     59.088 313.571 213.222  1.00 23.03           C
ATOM     10   O  PHE A   78     59.419 313.191 214.370  1.00 23.03           O
ATOM     11   N  GLY A   79     59.960 314.064 212.348  1.00 23.03           N
ATOM     12   CA GLY A   79     61.379 314.178 212.701  1.00 23.03           C
ATOM     14   C  GLY A   79     61.580 315.002 213.955  1.00 23.03           C
ATOM     15   O  GLY A   79     62.196 314.587 214.939  1.00 23.03           O
ATOM     16   N  ASN A   80     61.088 316.229 213.923  1.00 23.03           N
ATOM     17   CA ASN A   80     61.234 317.075 215.087  1.00 23.03           C
ATOM     18   CB ASN A   80     60.689 318.474 214.791  1.00 23.03           C
ATOM     19   C  ASN A   80     60.562 316.574 216.348  1.00 23.03           C
ATOM     20   O  ASN A   80     61.124 316.675 217.419  1.00 23.03           O
ATOM     21   N  TRP A   81     59.336 316.084 216.225  1.00 23.03           N
ATOM     22   CA TRP A   81     58.622 315.537 217.385  1.00 23.03           C
ATOM     23   CB TRP A   81     57.199 315.145 217.005  1.00 23.03           C
ATOM     24   C  TRP A   81     59.355 314.332 218.006  1.00 23.03           C
ATOM     25   O  TRP A   81     59.574 314.288 219.237  1.00 23.03           O
ATOM     26   N  LEU A   82     59.762 313.403 217.154  1.00 23.03           N
ATOM     27   CA LEU A   82     60.180 312.092 217.625  1.00 23.03           C
ATOM     28   CB LEU A   82     59.732 311.006 216.646  1.00 23.03           C
ATOM     29   C  LEU A   82     61.681 311.952 217.883  1.00 23.03           C
ATOM     30   O  LEU A   82     62.079 311.190 218.789  1.00 23.03           O
```

The fourth and fifth clusters are represented by fragments from PDB entries 3RA6 (Chan et al., 2011) and 2OB0 (Walker et al., 2006), respectively. The ATOM records are shown below.

```
>3ra6_A_65_13| FRAGMENT START INDEX: 478288
ATOM      1   N  GLY A   71      5.498-301.626   9.083  1.00 23.03           N
ATOM      2   CA GLY A   71      6.930-301.339   9.179  1.00 23.03           C
ATOM      4   C  GLY A   71      7.451-301.382  10.602  1.00 23.03           C
ATOM      5   O  GLY A   71      8.513-301.954  10.867  1.00 23.03           O
ATOM      6   N  THR A   72      6.702-300.784  11.525  1.00 23.03           N
ATOM      7   CA THR A   72      7.093-300.766  12.930  1.00 23.03           C
ATOM      8   CB THR A   72      6.167-299.859  13.771  1.00 23.03           C
ATOM      9   C  THR A   72      7.119-302.182  13.517  1.00 23.03           C
ATOM     10   O  THR A   72      8.047-302.525  14.247  1.00 23.03           O
ATOM     11   N  LEU A   73      6.122-303.003  13.179  1.00 23.03           N
ATOM     12   CA LEU A   73      6.062-304.390  13.663  1.00 23.03           C
ATOM     13   CB LEU A   73      4.757-305.070  13.231  1.00 23.03           C
ATOM     14   C  LEU A   73      7.254-305.210  13.163  1.00 23.03           C
ATOM     15   O  LEU A   73      7.682-306.156  13.827  1.00 23.03           O
ATOM     16   N  LEU A   74      7.775-304.833  11.992  1.00 23.03           N
ATOM     17   CA LEU A   74      8.984-305.425  11.411  1.00 23.03           C
ATOM     18   CB LEU A   74      8.883-305.378   9.887  1.00 23.03           C
ATOM     19   C  LEU A   74     10.274-304.723  11.858  1.00 23.03           C
ATOM     20   O  LEU A   74     11.352-305.010  11.342  1.00 23.03           O
ATOM     21   N  GLY A   75     10.159-303.804  12.813  1.00 23.03           N
ATOM     22   CA GLY A   75     11.312-303.133  13.403  1.00 23.03           C
ATOM     24   C  GLY A   75     11.917-302.038  12.543  1.00 23.03           C
ATOM     25   O  GLY A   75     13.029-301.579  12.819  1.00 23.03           O
ATOM     26   N  ARG A   76     11.195-301.605  11.513  1.00 23.03           N
ATOM     27   CA ARG A   76     11.705-300.602  10.578  1.00 23.03           C
ATOM     28   CB ARG A   76     11.257-300.922   9.149  1.00 23.03           C
ATOM     29   C  ARG A   76     11.231-299.200  10.949  1.00 23.03           C
ATOM     30   O  ARG A   76     10.104-299.036  11.409  1.00 23.03           O
```

```
>2ob0_C_113_27| FRAGMENT START INDEX: 708020
ATOM      1   N  ARG C  125     36.998 380.250 -82.052  1.00 23.03           N
ATOM      2   CA ARG C  125     35.554 380.658 -81.973  1.00 23.03           C
ATOM      3   CB ARG C  125     35.415 382.133 -81.630  1.00 23.03           C
ATOM      4   C  ARG C  125     34.833 380.275 -83.228  1.00 23.03           C
ATOM      5   O  ARG C  125     33.660 379.839 -83.182  1.00 23.03           O
ATOM      6   N  LYS C  126     35.498 380.309 -84.375  1.00 23.03           N
ATOM      7   CA LYS C  126     34.894 379.928 -85.628  1.00 23.03           C
ATOM      8   CB LYS C  126     35.813 380.144 -86.808  1.00 23.03           C
ATOM      9   C  LYS C  126     34.477 378.438 -85.570  1.00 23.03           C
ATOM     10   O  LYS C  126     33.513 378.059 -86.208  1.00 23.03           O
ATOM     11   N  PHE C  127     35.209 377.588 -84.857  1.00 23.03           N
ATOM     12   CA PHE C  127     34.889 376.132 -84.768  1.00 23.03           C
ATOM     13   CB PHE C  127     36.174 375.250 -84.631  1.00 23.03           C
ATOM     14   C  PHE C  127     33.988 375.763 -83.592  1.00 23.03           C
ATOM     15   O  PHE C  127     33.835 374.555 -83.224  1.00 23.03           O
ATOM     16   N  GLY C  128     33.410 376.774 -82.945  1.00 23.03           N
ATOM     17   CA GLY C  128     32.467 376.519 -81.884  1.00 23.03           C
ATOM     19   C  GLY C  128     33.091 376.375 -80.519  1.00 23.03           C
ATOM     20   O  GLY C  128     32.449 375.926 -79.594  1.00 23.03           O
ATOM     21   N  PHE C  129     34.347 376.737 -80.353  1.00 23.03           N
ATOM     22   CA PHE C  129     34.886 376.807 -79.052  1.00 23.03           C
ATOM     23   CB PHE C  129     36.407 376.808 -79.056  1.00 23.03           C
ATOM     24   C  PHE C  129     34.453 378.132 -78.375  1.00 23.03           C
ATOM     25   O  PHE C  129     34.288 379.127 -79.049  1.00 23.03           O
ATOM     26   N  GLU C  130     34.292 378.101 -77.075  1.00 23.03           N
ATOM     27   CA GLU C  130     33.935 379.273 -76.296  1.00 23.03           C
ATOM     28   CB GLU C  130     32.636 379.008 -75.540  1.00 23.03           C
ATOM     29   C  GLU C  130     34.976 379.542 -75.278  1.00 23.03           C
ATOM     30   O  GLU C  130     35.554 378.620 -74.718  1.00 23.03           O
```

Bibliography

Adams, P. D., Afonine, P. V., Bunkóczi, G., Chen, V. B., Davis, I. W., Echols, N., Headd, J. J., Hung, L.-W., Kapral, G. J., Grosse-Kunstleve, R. W., McCoy, A. J., Moriarty, N. W., Oeffner, R., Read, R. J., Richardson, D. C., Richardson, J. S., Terwilliger, T. C., and Zwart, P. H. (2010). "PHENIX: a comprehensive Python-based system for macromolecular structure solution." *Acta crystallographica. Section D, Biological crystallography*, 66(Pt 2), pp. 213–21 (cited on p. 36).

Adrian, M., Dubochet, J., Lepault, J., and McDowall, A. (1984). "Cryo-electron microscopy of viruses" (cited on p. 16).

Allen, F. H., Bellard, S., Brice, M., Cartwright, B. A., Doubleday, A., Higgs, H., Hummelink, T., Hummelink-Peters, B., Kennard, O., Motherwell, W., et al. (1979). "The Cambridge Crystallographic Data Centre: computer-based search, retrieval, analysis and display of information". *Acta Crystallographica Section B: Structural Crystallography and Crystal Chemistry*, 35(10), pp. 2331–2339 (cited on p. 52).

Altschul, S. F., Gish, W., and Miller, W. (1990). "Basic local alignment search tool". *Journal of molecular biology* (cited on p. 24).

Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. J. (1997). "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs." *Nucleic acids research*, 25(17), pp. 3389–402 (cited on p. 26).

Andrews, L. and Bernstein, H. (2012). "The Geometry of Niggli Reduction I: The Boundary Polytopes of the Niggli Cone". *arXiv*, 1203 (cited on p. 52).

Andrews, L., Bernstein, H., and Pelletier, G. (1980). "A perturbation stable cell comparison technique". *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, 36(2), pp. 248–252 (cited on p. 53).

Bahar, M., Ballard, C., Cohen, S. X., Cowtan, K. D., Dodson, E. J., Emsley, P., Esnouf, R. M., Keegan, R., Lamzin, V., Langer, G., Levdikov, V., Long, F., Meier, C., Muller, A., Murshudov, G. N., Perrakis, A., Siebold, C., Stein, N., Turkenburg, M. G. W., Vagin, A. A., Winn, M., Winter, G., and Wilson, K. S. (2006). "SPINE workshop on automated X-ray analysis: a progress report". *Acta Crystallographica Section D*, 62(10), pp. 1170–1183 (cited on p. 33).

Bender-deMoll, S. (2001). *Information, Uncertainty, and Meaning*. [Online; accessed 15-February-2014] (cited on p. 116).

Bergfors, T. M. (2009). *Protein Crystalization*. Internat'l University Line (cited on p. 60).

Berman, H. M., Henrick, K., and Nakamura, H. (2003). "Announcing the worldwide Protein Data Bank." *Nature structural biology*, 10(12), p. 980 (cited on p. 12).

Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N., and Bourne, P. E. (2000). "The Protein Data Bank." *Nucleic acids research*, 28(1), pp. 235–42 (cited on pp. 12, 13).

Berman, H. M., Bourne, P. E., and Research Collaboratory for Structural Bioinformatics (2007). *PDB Newsletter*. Piscataway, NJ (cited on p. 79).

Bernstein, F. C., Koetzle, T. F., Williams, G. J., Meyer, E. F., Brice, M. D., Rodgers, J. R., Kennard, O., Shimanouchi, T., and Tasumi, M. (1977). "The Protein Data Bank: a computer-based archival file for macromolecular structures." *Journal of molecular biology*, 112(3), pp. 535–42 (cited on pp. 12, 13, 34).

Böck, A., Forchhammer, K., Heider, J., Leinfelder, W., Sawers, G., Veprek, B., and Zinoni, F. (1991). "Selenocysteine: the 21st amino acid". *Molecular Microbiology*, 5(3), pp. 515–520 (cited on p. 5).

Bragg, W. L. (1913). "The diffraction of short electromagnetic waves by a crystal". In: *Proceedings of the Cambridge Philosophical Society*. Vol. 17. 43, p. 4 (cited on p. 37).

Brändén, C.-I. and Jones, T. A. (1990). "Between objectivity and subjectivity". *Nature* (cited on p. 19).

Brändén, C.-I. and Tooze, J. (1991). *Introduction to protein structure*. Garland New York (cited on pp. 8–10).

Brünger, A. T. (1992). “Free R value: a novel statistical quantity for assessing the accuracy of crystal structures”. *Nature*, 355, pp. 472–475 (cited on p. 17).

Buerger, M. (1957). “Reduced cells”. *Zeitschrift fur Kristallographie*, 109, pp. 42–60 (cited on p. 53).

Cajocari, D. and Wikipedia (2014). *Amino acid — Wikipedia, The Free Encyclopedia*. [Online; accessed 11-February-2014] (cited on p. 4).

Chan, C.-H., Yu, T.-H., and Wong, K.-B. (2011). “Stabilizing salt-bridge enhances protein thermostability by reducing the heat capacity change of unfolding”. *PLoS One*, 6(6), e21624 (cited on pp. 187, 206).

Charif, D. and Lobry, J. (2007). “SeqinR 1.0-2: a contributed package to the R project for statistical computing devoted to biological sequences retrieval and analysis.” In: *Structural approaches to sequence evolution: Molecules, networks, populations*. Ed. by U. Bastolla, M. Porto, H. Roman, and M. Vendruscolo. Biological and Medical Physics, Biomedical Engineering. ISBN : 978-3-540-35305-8. New York: Springer Verlag, pp. 207–232 (cited on p. 6).

Clark, J., DeRose, S., et al. (1999). *XML path language (XPath) version 1.0* (cited on p. 42).

Cole, C., Barber, J. D., and Barton, G. J. (2008). “The Jpred 3 secondary structure prediction server.” *Nucleic acids research*, 36(Web Server issue), W197–201 (cited on p. 196).

Collaborative Computational Project, number 4 (1994). “The CCP4 suite: programs for protein crystallography.” *Acta crystallographica. Section D, Biological crystallography*, 50(Pt 5), p. 760 (cited on pp. 159, 161).

Courrieu, P. (2008). “Fast computation of Moore-Penrose inverse matrices”. *arXiv preprint arXiv:0804.4809*, 8(2), pp. 25–29 (cited on pp. 74, 98).

Cowtan, K. (2006). “The Buccaneer software for automated model building. 1. Tracing protein chains.” *Acta crystallographica. Section D, Biological crystallography*, 62(Pt 9), pp. 1002–11 (cited on p. 159).

Csaba, G., Birzele, F., and Zimmer, R. (2009). “Systematic comparison of SCOP and CATH: a new gold standard for protein structure analysis.” *BMC structural biology*, 9(1), p. 23 (cited on p. 29).

Dagum, L. and Menon, R. (1998). “OpenMP: an industry standard API for shared-memory programming”. *Computational Science & Engineering, IEEE*, 5(1), pp. 46–55 (cited on p. 87).

Dayhoff, M., Eck, R., and Park, C. (1972). “A model of evolutionary change in proteins”. *In Atlas of protein sequence and structure*, 5, pp. 89–99 (cited on p. 21).

Dayhoff, M., Schwartz, R., and Orcutt, B. (1978). “A model of evolutionary change in proteins”. *In Atlas of protein sequence and structure*, 5 suppl. 3, pp. 345–352 (cited on p. 21).

DeLano, W. L. (2002). “The PyMOL molecular graphics system” (cited on p. 186).

Donovan, J. and Copeland, P. (2010). “The efficiency of selenocysteine incorporation is regulated by translation initiation factors”. *Journal of molecular biology*, 400(4), pp. 659–664 (cited on p. 5).

Dosztányi, Z., Csizmok, V., Tompa, P., and Simon, I. (2005). “IUPred: web server for the prediction of intrinsically unstructured regions of proteins based on estimated energy content.” *Bioinformatics (Oxford, England)*, 21(16), pp. 3433–4 (cited on pp. 66, 93).

Dosztányi, Z., Mészáros, B., and Simon, I. (2010). “Bioinformatical approaches to characterize intrinsically disordered/unstructured proteins”. *Briefings in Bioinformatics*, 11(2), pp. 225–243 (cited on pp. 60, 62–64, 66, 92, 106).

Dosztányi, Z., Sándor, M., Tompa, P., and Simon, I. (2007). “Prediction of protein disorder at the domain level.” *Current protein & peptide science*, 8(2), pp. 161–71 (cited on pp. 60, 62, 139).

Dunker, A., Lawson, J., Brown, C., Williams, R., Romero, P., Oh, J., Oldfield, C., Campen, A., Ratliff, C., Hipps, K., Ausio, J., Nissen, M., Reeves, R., Kang, C., Kissinger, C., Bailey, R., Griswold, M., Chiu, W., Garner, E., and Obradovic, Z. (2001a). “Intrinsically disordered protein.” *Journal of molecular graphics & modelling*, 19(1), pp. 26–59 (cited on pp. 10, 11, 62).

Dunker, A. and Obradovic, Z. (2001b). “The protein trinity–linking function and disorder.” *Nature biotechnology*, 19(9), pp. 805–6 (cited on p. 61).

Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press (cited on pp. 21, 22, 67).

Eddy, S. R. (2004). “Where did the BLOSUM62 alignment score matrix come from?” *Nature biotechnology*, 22(8), pp. 1035–6 (cited on p. 23).

Edgar, R. C. (2004). “MUSCLE: multiple sequence alignment with high accuracy and high throughput.” *Nucleic acids research*, 32(5), pp. 1792–7 (cited on p. 26).

Enright, A. J., van Dongen, S., and Ouzounis, C. A. (2002). “An efficient algorithm for large-scale detection of protein families.” *Nucleic acids research*, 30(7), pp. 1575–84 (cited on pp. 28, 114).

Esnouf, R. M. (1997). “Polyalanine reconstruction from Calpha positions using the program CALPHA can aid initial phasing of data by molecular replacement procedures.” *Acta crystallographica. Section D, Biological crystallography*, 53(Pt 6), pp. 665–72 (cited on pp. 39, 48, 189).

Esnouf, R. M. (1992). “Protein Structure from Simulated NMR Databases”. PhD thesis. University of Oxford, pp. 1–250 (cited on pp. 6, 7).

Esnouf, R. M., Hamer, R., Sussman, J. L., Silman, I., Trudgian, D., Yang, Z.-R., and Prilusky, J. (2006). “Honing the in silico toolkit for detecting protein disorder.” *Acta crystallographica. Section D, Biological crystallography*, 62(Pt 10), pp. 1260–6 (cited on pp. 63, 64, 66, 93).

Foster, G. (2004). *Performance Measures for Text Categorization* (cited on p. 70).

Fuxreiter, M., Simon, I., Friedrich, P., and Tompa, P. (2004). “Preformed structural elements feature in partner recognition by intrinsically unstructured proteins.” *Journal of molecular biology*, 338(5), pp. 1015–26 (cited on p. 62).

Gonnet, G. H., Cohen, M. a., and Benner, S. a. (1992). *Exhaustive matching of the entire protein sequence database* (cited on p. 22).

Grosse-Kunstleve, R. W., Sauter, N. K., and Adams, P. D. (2003). “Numerically stable algorithms for the computation of reduced unit cells”. *Acta Crystallographica Section A Foundations of Crystallography*, 60(1), pp. 1–6 (cited on p. 52).

Guennebaud, G., Jacob, B., et al. (2010). *Eigen v3* (cited on pp. 98, 100, 101).

Harp, J. M., Timm, D. E., and Bunick, G. J. (1998). “Macromolecular Crystal Annealing: Overcoming Increased Mosaicity Associated with Cryocrystallography”. *Acta Crystallographica Section D Biological Crystallography*, 54(4), pp. 622–628 (cited on p. 60).

He, B., Wang, K., Liu, Y., Xue, B., Uversky, V. N., and Dunker, A. K. (2009). "Predicting intrinsic disorder in proteins: an overview." *Cell research*, 19(8), pp. 929–49 (cited on pp. 63, 64, 93).

Henikoff, J. G., Greene, E. A., Pietrokovski, S., and Henikoff, S. (2000). "Increased coverage of protein families with the blocks database servers." *Nucleic acids research*, 28(1), pp. 228–30 (cited on p. 22).

Henikoff, S. and Henikoff, J. G. (1991). "Automated assembly of protein blocks for database searching." *Nucleic acids research*, 19(23), pp. 6565–72 (cited on p. 22).

Henikoff, S. and Henikoff, J. G. (1992). "Amino acid substitution matrices from protein blocks." *Proceedings of the National Academy of Sciences of the United States of America*, 89(22), pp. 10915–9 (cited on p. 22).

Higgins, D. and Taylor, W. (2000). *Bioinformatics: sequence, structure, and databanks: a practical approach*. Oxford University Press, Inc. (cited on p. 29).

Higgins, D. and Sharp, P. (1988). "CLUSTAL: a package for performing multiple sequence alignment on a microcomputer." *Gene*, 73(1), pp. 237–44 (cited on p. 26).

Holm, L. and Sander, C. (1998). "Removing near-neighbour redundancy from large protein sequence collections." *Bioinformatics*, 14(5), pp. 423–429 (cited on p. 27).

Hooft, R., Vriend, G., Sander, C., and Abola, E. (1996). "Errors in protein structures". *Nature* (cited on p. 161).

Householder, A. (1958). "Unitary triangularization of a nonsymmetric matrix". *Journal of the ACM (JACM)*, 2, pp. 339–342 (cited on pp. 100, 101).

Huang, T. M. and Kecman, V. (2004). "Bias Term b in SVMs Again". In: *Proceedings of 12th European Symposium on Artificial Neural Networks - ESANN 2004*. April, pp. 441–448 (cited on p. 132).

Jacob, F. (1977). "Evolution and tinkering". *Science*, 196(4295), pp. 1161–1166 (cited on p. 21).

Janeway, C. A., Travers, P., Walport, M., and Capra, J. D. (2001). *Immunobiology: the immune system in health and disease*. Vol. 2. Churchill Livingstone (cited on p. 154).

Jones, T. A., Zou, J. Y., Cowan, S. W., and Kjeldgaard, M. (1991). “Improved methods for building protein models in electron density maps and the location of errors in these models”. *Acta Crystallographica Section A Foundations of Crystallography*, 47(2), pp. 110–119 (cited on pp. 18, 19).

Joosten, R. P., Joosten, K., Cohen, S. X., Vriend, G., and Perrakis, A. (2011a). “Automatic rebuilding and optimization of crystallographic structures in the Protein Data Bank.” *Bioinformatics (Oxford, England)*, 27(24), pp. 3392–8 (cited on pp. 160, 161).

Joosten, R. P., Long, F., Murshudov, G. N., and Perrakis, A. (in press). “The PDB_REDO server for macromolecular structure model optimization”. *IUCr J*, 1 (cited on p. 19).

Joosten, R. P., Salzemann, J., Bloch, V., Stockinger, H., Berglund, A.-C., Blanchet, C., Bongcam-Rudloff, E., Combet, C., Da Costa, A. L., Deleage, G., Diarena, M., Fabbretti, R., Fettahi, G., Flegel, V., Gisel, A., Kasam, V., Kervinen, T., Korpelainen, E., Mattila, K., Pagni, M., Reichstadt, M., Breton, V., Tickle, I. J., and Vriend, G. (2009a). “PDB_REDO: automated re-refinement of X-ray structure models in the PDB.” *Journal of applied crystallography*, 42(Pt 3), pp. 376–384 (cited on pp. 19, 20, 159).

Joosten, R. P., te Beek, T. A. H., Krieger, E., Hekkelman, M. L., Hooft, R. W. W., Schneider, R., Sander, C., and Vriend, G. (2011b). “A series of PDB related databases for everyday needs.” *Nucleic acids research*, 39(Database issue), pp. D411–9 (cited on pp. 157, 159, 160).

Joosten, R. P., Womack, T., Vriend, G., and Bricogne, G. (2009b). “Re-refinement from deposited X-ray data can deliver improved models for most PDB entries.” *Acta crystallographica. Section D, Biological crystallography*, 65(Pt 2), pp. 176–85 (cited on p. 20).

Kabsch, W. and Sander, C. (1983). “Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features.” *Biopolymers*, 22(12), pp. 2577–637 (cited on pp. 29, 155).

Kabsch, W. (1976). “A solution for the best rotation to relate two sets of vectors”. *Acta Crystallographica Section A: Crystal Physics*, (6), pp. 922–923 (cited on pp. 30, 39).

Kabsch, W. (1978). “A discussion of the solution for the best rotation to relate two sets of vectors”. *Acta Crystallographica Section A: Crystal Physics*, (9) (cited on p. 39).

Karlin, S. and Altschul, S. (1990). “Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes”. *Proceedings of the National Academy of Sciences*, 87(March), pp. 2264–2268 (cited on p. 25).

Karush, F. (1950). “Heterogeneity of the Binding Sites of Bovine Serum Albumin”. *Journal of the American Chemical Society*, 72(6), pp. 2705–2713 (cited on p. 11).

Kecman, V. (2001). *Learning and soft computing: support vector machines, neural networks, and fuzzy logic models*. MIT press (cited on p. 132).

Keegan, R. M. and Winn, M. D. (2007). “Automated search-model discovery and preparation for structure solution by molecular replacement”. *Acta Crystallographica Section D*, 63(4), pp. 447–457 (cited on p. 33).

Koshland, D. E. (1958). “Application of a Theory of Enzyme Specificity to Protein Synthesis.” *Proceedings of the National Academy of Sciences of the United States of America*, 44(2), pp. 98–104 (cited on p. 11).

Kozlowski, L. P. and Bujnicki, J. M. (2012). “MetaDisorder: a meta-server for the prediction of intrinsic disorder in proteins.” *BMC bioinformatics*, 13(1), p. 111 (cited on p. 93).

Krieger, E., Koraimann, G., and Vriend, G. (2002). “Increasing the Precision of Comparative Models with YASARA NOVA - a Self-Parametrizing Force Field”. *Proteins: Structure, Function and Genetics*, 402(September 2001), pp. 393–402 (cited on p. 159).

Krissinel, E. and Henrick, K. (2004). “Secondary-structure matching (SSM), a new tool for fast protein structure alignment in three dimensions.” *Acta crystallographica. Section D, Biological crystallography*, 60(Pt 12 Pt 1), pp. 2256–68 (cited on pp. 29, 157).

Krivy, I. and Gruber, B. (1976). “A unified algorithm for determining the reduced (Niggli) cell”. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, 32(2), pp. 297–298 (cited on p. 53).

Larkin, M. A., Blackshields, G., Brown, N. P., Chenna, R., McGettigan, P. A., McWilliam, H., Valentin, F., Wallace, I. M., Wilm, A., Lopez, R., Thompson, J. D.,

Gibson, T. J., and Higgins, D. G. (2007). “Clustal W and Clustal X version 2.0.” *Bioinformatics (Oxford, England)*, 23(21), pp. 2947–8 (cited on p. 26).

Le Gall, T., Romero, P. R., Cortese, M. S., Uversky, V. N., and Dunker, A. K. (2007). “Intrinsic disorder in the Protein Data Bank.” *Journal of biomolecular structure & dynamics*, 24(4), pp. 325–42 (cited on p. 64).

Lemieux, R. U. and Spohr, U. (1994). “How Emil Fischer was led to the lock and key concept for enzyme specificity.” *Advances in carbohydrate chemistry and biochemistry*, 50, pp. 1–20 (cited on p. 10).

Li, W., Jaroszewski, L., and Godzik, A. (2002). “Sequence clustering strategies improve remote homology recognitions while reducing search times”. *Protein Engineering Design and Selection*, 15(8), pp. 643–649 (cited on p. 27).

Li, W. and Godzik, A. (2006). “Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences.” *Bioinformatics*, 22(13), pp. 1658–9 (cited on pp. 111, 112).

Li, W., Jaroszewski, L., and Godzik, A. (2001). “Clustering of highly homologous sequences to reduce the size of large protein databases”. *Bioinformatics*, 17(3), pp. 282–283 (cited on p. 27).

Linding, R. (2003). “GlobPlot: exploring protein sequences for globularity and disorder”. *Nucleic acids research*, 31(13), pp. 3701–3708 (cited on p. 65).

Lippmann, R. (1987). “An introduction to computing with neural nets”. *ASSP Magazine, IEEE* (cited on p. 140).

Liu, J., Perumal, N. B., Oldfield, C. J., Su, E. W., Uversky, V. N., and Dunker, A. K. (2006). “Intrinsic disorder in transcription factors.” *Biochemistry*, 45(22), pp. 6873–88 (cited on p. 93).

Lo Conte, L., Ailey, B., Hubbard, T. J., Brenner, S. E., Murzin, A. G., and Chothia, C. (2000). “SCOP: a structural classification of proteins database.” *Nucleic acids research*, 28(1), pp. 257–9 (cited on pp. 14, 29, 155).

Maiti, R., Van Domselaar, G. H., Zhang, H., and Wishart, D. S. (2004). “SuperPose: a simple server for sophisticated structural superposition.” *Nucleic acids research*, 32(Web Server issue), W590–4 (cited on p. 173).

Martins, B. M., Macedo-Ribeiro, S., Bresser, J., Buckel, W., and Messerschmidt, A. (2005). “Structural basis for stereo-specific catalysis in NAD+-dependent (R)-

2-hydroxyglutarate dehydrogenase from Acidaminococcus fermentans". *FEBS Journal*, 272(1), pp. 269–281 (cited on p. 204).

McGill, K. and Asadi, M. (2013). "The Geometry of Niggli Reduction III: SAUC–Search of Alternate Unit Cells". *arXiv*, 1307, pp. 1–10 (cited on pp. 52, 53, 192).

McLachlan, A. D. (1972). "A mathematical procedure for superimposing atomic coordinates of proteins". *Acta Crystallographica Section A*, 28(6), pp. 656–657 (cited on p. 39).

McLachlan, A. (1982). "Rapid comparison of protein structures". *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, (1965), pp. 1980–1982 (cited on p. 30).

McPherson, A. (1999). *Crystallization of biological macromolecules*. Vol. 586. Cold Spring Harbor Laboratory Press Cold Spring Harbor, NY (cited on p. 59).

Mészáros, B., Tompa, P., Simon, I., and Dosztányi, Z. (2007). "Molecular principles of the interactions of disordered proteins." *Journal of molecular biology*, 372(2), pp. 549–61 (cited on pp. 61, 62).

Mighell, A. (2001). "Lattice symmetry and identification-The fundamental role of reduced cells in materials characterization". *Journal of Research-National Institute of Standards and Technology*, 106(6), pp. 983–995 (cited on p. 53).

— (2002). "Lattice Matching (LM)-Prevention of Inadvertent Duplicate Publications of Crystal Structures". *Journal of Research-National Institute of Standards and Technology*, 107(5), pp. 425–429 (cited on p. 52).

Mighell, A. and Rodgers, J. (1980). "Lattice symmetry determination". *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, 36(2), pp. 321–326 (cited on p. 52).

Milne, G. W. and Heller, S. R. (1980). "NIH/EPA chemical information system". *Journal of chemical information and computer sciences*, 20(4), pp. 204–211 (cited on p. 53).

Muramoto, K., Ohta, K., Shinzawa-Itoh, K., Kanda, K., Taniguchi, M., Nabekura, H., Yamashita, E., Tsukihara, T., and Yoshikawa, S. (2010). "Bovine cytochrome c oxidase structures enable O2 reduction with minimization of reactive oxygens and provide a proton-pumping gate". *Proceedings of the National Academy of Sciences*, 107(17), pp. 7740–7745 (cited on pp. 187, 205).

Murshudov, G. N., Skubák, P., Lebedev, A. A., Pannu, N. S., Steiner, R. A., Nicholls, R. A., Winn, M. D., Long, F., and Vagin, A. A. (2011). "REFMAC5 for the refinement of macromolecular crystal structures." *Acta crystallographica. Section D, Biological crystallography*, 67(Pt 4), pp. 355–67 (cited on p. 159).

Murshudov, G. N., Vagin, A. A., and Dodson, E. J. (1997). "Refinement of macromolecular structures by the maximum-likelihood method." *Acta crystallographica. Section D, Biological crystallography*, 53(Pt 3), pp. 240–55 (cited on p. 159).

Needleman, S. and Wunsch, C. (1970). "A general method applicable to the search for similarities in the amino acid sequence of two proteins". *Journal of molecular biology*, 48(3), pp. 443–453 (cited on p. 23).

Norris, J. R. (1998). *Markov chains*. 2008. Cambridge university press (cited on p. 116).

Obradovic, Z., Peng, K., Vucetic, S., Radivojac, P., Brown, C. J., and Dunker, A. K. (2003). "Predicting intrinsic disorder from amino acid sequence." *Proteins*, 53 Suppl 6(February), pp. 566–72 (cited on pp. 62, 64).

Panjikar, S., Parthasarathy, V., Lamzin, V. S., Weiss, M. S., and Tucker, P. A. (2009). "On the combination of molecular replacement and single-wavelength anomalous diffraction phasing for automated structure determination". *Acta Crystallographica Section D*, 65(10), pp. 1089–1097 (cited on p. 33).

Pearl, F. M. G. (2003). "The CATH database: an extended protein family resource for structural and functional genomics". *Nucleic acids research*, 31(1), pp. 452–455 (cited on pp. 14, 29, 155).

Pearson, W. and Miller, W. (1992). "Dynamic programming algorithms for biological sequence comparison". *Methods in enzymology*, 210 (cited on pp. 22, 23).

Peng, K., Radivojac, P., Vucetic, S., Dunker, A. K., and Obradovic, Z. (2006). "Length-dependent prediction of protein intrinsic disorder." *BMC bioinformatics*, 7, p. 208 (cited on p. 139).

Perrakis, A., Morris, R. J., and Lamzin, V. S. (1999). "Automated protein model building combined with iterative structure refinement". *Nature Struct. Biol.* 6, pp. 458–463 (cited on p. 159).

Pietrokovski, S., Henikoff, J. G., and Henikoff, S. (1996). "The Blocks database–a system for protein classification." *Nucleic acids research*, 24(1), pp. 197–200 (cited on p. 22).

Press, W. H. (2007). *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press (cited on p. 73).

Price, N. and Stevens, L. (1999). *Fundamentals of Enzymology: The Cell and Molecular Biology of Catalytic Proteins*. New York: Oxford University Press, New York. (cited on p. 10).

Prilusky, J., Felder, C. E., Zeev-Ben-Mordehai, T., Rydberg, E. H., Man, O., Beckmann, J. S., Silman, I., and Sussman, J. L. (2005). "FoldIndex: a simple tool to predict whether a given protein sequence is intrinsically unfolded." *Bioinformatics (Oxford, England)*, 21(16), pp. 3435–8 (cited on p. 65).

Punta, M., Coggill, P. C., Eberhardt, R. Y., Mistry, J., Tate, J., Boursnell, C., Pang, N., Forslund, K., Ceric, G., Clements, J., Heger, A., Holm, L., Sonnhammer, E. L. L., Eddy, S. R., Bateman, A., and Finn, R. D. (2012). "The Pfam protein families database." *Nucleic acids research*, 40(Database issue), pp. D290–301 (cited on pp. 14, 26).

Radivojac, P., Iakoucheva, L. M., Oldfield, C. J., Obradovic, Z., Uversky, V. N., and Dunker, A. K. (2007). "Intrinsic disorder and functional proteomics." *Biophysical journal*, 92(5), pp. 1439–56 (cited on p. 64).

Ramachandran, G., Ramakrishnan, C., and Sasisekharan, V. (1963). "Stereochemistry of polypeptide chain configurations". *Journal of Molecular Biology*, 7(1), pp. 95–99 (cited on p. 6).

Ramraj, V., Evans, G., Diprose, J. M., and Esnouf, R. M. (2012). "Nearest-cell: a fast and easy tool for locating crystal matches in the PDB." *Acta crystallographica. Section D, Biological crystallography*, 68(Pt 12), pp. 1697–700 (cited on pp. 34, 39, 47, 51, 56, 153).

Ramsay, J. O. (1990). "Matfit: A fortran subroutine for comparing two matrices in a subspace". *Psychometrika*, 55(3), pp. 551–553 (cited on pp. 30, 48, 168).

Read, R. J., Adams, P. D., Arendall, W. B., Brunger, A. T., Emsley, P., Joosten, R. P., Kleywegt, G. J., Krissinel, E. B., Lütteke, T., Otwinowski, Z., Perrakis, A., Richardson, J. S., Sheffler, W. H., Smith, J. L., Tickle, I. J., Vriend, G., and Zwart, P. H. (2011). "A new generation of crystallographic validation tools for the protein data bank." *Structure*, 19(10), pp. 1395–412 (cited on p. 7).

Ren, J., Esnouf, R., Garman, E., Somers, D., Ross, C., Kirby, I., Keeling, J., Darby, G., Jones, E., Stuart, D., and Stammers, D. (1995). "High resolution structures

of HIV-1 RT from four RT–inhibitor complexes". *Nature structural & molecular biology* (cited on p. 84).

Rigault, G. and Taylor, C. (1980). *Metric tensor and Symmetry operations in crystallography*. i. Published for the International Union of Crystallography by University College Cardiff Press, Cardiff, Wales (cited on p. 52).

Romero, P., Obradovic, Z., Kissinger, C., Villafranca, J., and Dunker, A. (1997). "Identifying disordered regions in proteins from amino acid sequence". *Proceedings of International Conference on Neural Networks (ICNN'97)*, pp. 90–95 (cited on p. 65).

Rupp, B. (2009). *Biomolecular crystallography*. Garland Science (cited on p. 59).

Santoro, A. and Mighell, A. D. (1970). "Determination of reduced cells". *Acta Crystallographica Section A*, 26(1), pp. 124–127 (cited on p. 52).

Schetinin, V., Schult, J., Scheidt, B., and Kuriakin, V. (2003). "Learning Multi-Class Neural-Network Models from Electroencephalograms". In: *Knowledge-Based Intelligent Information and Engineering Systems*. Ed. by V. Palade, R. J. Howlett, and L. Jain. Lm. Springer Berlin Heidelberg, pp. 155–162 (cited on p. 140).

Shen, Y., Delaglio, F., Cornilescu, G., and Bax, A. (2009). "TALOS+: a hybrid method for predicting protein backbone torsion angles from NMR chemical shifts." *Journal of biomolecular NMR*, 44(4), pp. 213–23 (cited on p. 126).

Sickmeier, M., Hamilton, J. a., LeGall, T., Vacic, V., Cortese, M. S., Tantos, A., Szabo, B., Tompa, P., Chen, J., Uversky, V. N., Obradovic, Z., and Dunker, A. K. (2007). "DisProt: the Database of Disordered Proteins." *Nucleic acids research*, 35(Database issue), pp. D786–93 (cited on p. 80).

Sievers, F., Wilm, A., Dineen, D., Gibson, T. J., Karplus, K., Li, W., Lopez, R., McWilliam, H., Remmert, M., Söding, J., Thompson, J. D., and Higgins, D. G. (2011). "Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega." *Molecular systems biology*, 7(539), p. 539 (cited on p. 26).

Smith, T., Waterman, M., et al. (1981). "Identification of common molecular subsequences". *J. mol. Biol*, 147(1), pp. 195–197 (cited on p. 23).

Snyder, W. V. (1991). *Why not to use Numerical Recipes?* (Cited on p. 97).

Stuart, D., Levine, M., Muirhead, H., and Stammers, D. (1979). “Crystal structure of cat muscle pyruvate kinase at a resolution of 2.6 Å”. *Journal of molecular biology* (cited on p. 29).

Terwilliger, T. (2003). “SOLVE and RESOLVE: automated structure solution, density modification and model building”. *Journal of synchrotron radiation*, pp. 49–52 (cited on p. 159).

Thomas, I. R., Bruno, I. J., Cole, J. C., Macrae, C. F., Pidcock, E., and Wood, P. A. (2010). “WebCSD: the online portal to the Cambridge Structural Database”. *Journal of applied crystallography*, 43(2), pp. 362–366 (cited on p. 53).

Thompson, J. D., Higgins, D. G., and Gibson, T. J. (1994). “CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice”. *Nucleic acids research*, 22(22), pp. 4673–4680 (cited on p. 26).

Thomson, R. and Yang, Z. R. (2002). “A novel basis function neural network”. In: *Neural Information Processing, 2002. ICONIP'02. Proceedings of the 9th International Conference on*. Vol. 1. IEEE, pp. 441–446 (cited on p. 68).

Thomson, R. and Esnouf, R. (2004). “Prediction of natively disordered regions in proteins using a bio-basis function neural network”. *Intelligent Data Engineering and Automated Learning - IDEAL 2004*, pp. 108–116 (cited on pp. 76, 109, 120).

Tickle, I. (2012). “Statistical quality indicators for electron-density maps”. *Acta Crystallographica Section D: Biological Crystallography*, 68, pp. 454–467 (cited on pp. 18, 19, 159).

Van Dongen, S. M. (2000). “Graph clustering by flow simulation”. PhD thesis. Universiteit Utrecht, p. 173 (cited on pp. 114–116).

Vriend, G. (1990). “WHAT IF: a molecular modeling and drug design program”. *Journal of molecular graphics*, 8(3), pp. 52–56 (cited on pp. 19, 161).

Vucetic, S., Brown, C. J., Dunker, A. K., and Obradovic, Z. (2003). “Flavors of protein disorder.” *Proteins*, 52(4), pp. 573–84 (cited on pp. 62, 63, 93).

Vucetic, S., Xie, H., Iakoucheva, L. M., Oldfield, C. J., Dunker, A. K., Obradovic, Z., and Uversky, V. N. (2007). “Functional anthology of intrinsic disorder. 2. Cellular components, domains, technical terms, developmental processes, and coding sequence diversities correlated with long disordered regions.” *Journal of proteome research*, 6(5), pp. 1899–916 (cited on p. 61).

Walker, J. R., Schuetz, S., Antoshenko, T., Wu, H., Bernstein, G., Loppnau, P., Weigelt, J., Sundstrom, M., Arrowsmith, C. H., Edwards, A. M., Bochkarev, A., Plotnikov, A. N., and Structural Genomics Consortium (2006). *Human MAK3 homolog in complex with Acetyl–CoA (publication unavailable)* (cited on pp. 187, 206).

Westbrook, J., Ito, N., Nakamura, H., Henrick, K., and Berman, H. M. (2005). "PDBML: the representation of archival macromolecular structure data in XML." *Bioinformatics (Oxford, England)*, 21(7), pp. 988–92 (cited on p. 13).

Wikipedia (2014). *Bragg's Law — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-February-2014] (cited on p. 37).

Wilson, I., Skehel, J., and Wiley, D. (1981). "Structure of the haemagglutinin membrane glycoprotein of influenza virus at 3Å resolution". *Nature* (cited on p. 8).

Winn, M., Murshudov, G., and Papiz, M. (2003). "Macromolecular TLS refinement in REFMAC at moderate resolutions." *Methods in enzymology*, 374, p. 300 (cited on p. 159).

Winn, M., Isupov, M., and Murshudov, G. (2001). "Use of TLS parameters to model anisotropic displacements in macromolecular refinement". *Acta Crystallographica Section D*, D57, pp. 122–133 (cited on p. 159).

Winter, G. and McAuley, K. E. (2011). "Automated data collection for macromolecular crystallography". *Methods*, 55(1), pp. 81–93 (cited on pp. 33, 51).

Wright, P. and Dyson, H. (1999). "Intrinsically unstructured proteins: re-assessing the protein structure-function paradigm." *Journal of molecular biology*, 293(2), pp. 321–331 (cited on pp. 11, 60).

Wu, Y., Vadrevu, R., Kathuria, S., Yang, X., and Matthews, C. R. (2007). "A tightly packed hydrophobic cluster directs the formation of an off-pathway submillisecond folding intermediate in the alpha subunit of tryptophan synthase, a TIM barrel protein." *Journal of molecular biology*, 366(5), pp. 1624–38 (cited on pp. 8, 9).

Wüthrich, K. (2001). "The way to NMR structures of proteins". *Nat Struct Biol*, 8(11), pp. 923–925 (cited on p. 15).

Xue, B., Dunbrack, R. L., Williams, R. W., Dunker, A. K., and Uversky, V. N. (2010). "PONDR-FIT: a meta-predictor of intrinsically disordered amino acids." *Biochimica et biophysica acta*, 1804(4), pp. 996–1010 (cited on pp. 64, 66, 93).

Yang, Z. R. (2004). “Biological applications of support vector machines.” *Briefings in bioinformatics*, 5(4), pp. 328–38 (cited on p. 132).

Yang, Z. R., Dry, J., Thomson, R., and Charles Hodgman, T. (2006). “A bio-basis function neural network for protein peptide cleavage activity characterisation.” *Neural networks : the official journal of the International Neural Network Society*, 19(4), pp. 401–7 (cited on p. 132).

Yang, Z. R., Thomson, R., McNeil, P., and Esnouf, R. M. (2005). “RONN: the bio-basis function neural network technique applied to the detection of natively disordered regions in proteins”. *Bioinformatics*, 21(16), pp. 3369–3376 (cited on pp. 60, 63, 66–68, 70, 72, 76–78, 86, 90, 93, 112).