APPENDIX A: DOMAIN INSERTION

A.1 Introduction

Taking advantage of an evolutionary basis of domain classification, here I describe the nature and characteristics of domain insertions in protein structures, a phenomenon that is different from the usual pattern of sequential arrangement of domains in multi-domain proteins.

Domains constitute the basic structural, functional and evolutionary unit of proteins (Holm and Sander, 1996; Murzin *et al.*, 1995; Orengo *et al.*, 1997). Proteins can comprise a single domain or a combination of domains. It is well established that multi-domain proteins with widely diversified architecture and functions are generated from a limited repertoire of domain families (Bork *et al.*, 1996; Chothia, 1992). Structural assignments to complete genomes revealed that almost two-thirds of prokaryotic proteins and 80% of eukaryotic proteins are multi-domain proteins (Teichmann *et al.*, 1998). In 1973, Donald Wetlaufer introduced the classification of domains into continuous and discontinuous (Wetlaufer, 1973). A continuous domain is formed by one part of a polypeptide chain, while a discontinuous domain is formed by two or more parts of a single polypeptide chain. Thus, discontinuous domains are essentially formed by one-dimensionally non-contiguous segments of a polypeptide. While most multi-domain proteins have continuous domains, some proteins exhibit non-contiguous arrangement of their domains (Wetlaufer, 1973). In this work, I focus on insertions (Russell, 1994), which are the cases of one domain being inserted into another domain (Figure 67).

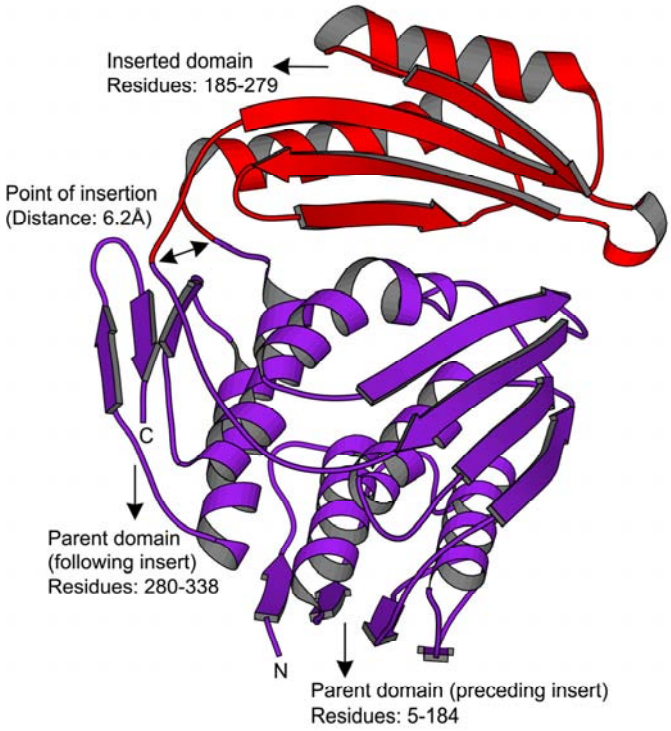

Figure 67. Domain insertion in Escherichia coli enzyme RNA 3'-terminal phosphate cyclase (PDB 1qmhA). The E. coli *enzyme RNA 3'-terminal phosphate cyclase consists of two domains, of which one is contained within the other. The parent domain (residues 5-184, 280-338, coloured purple) consists of three repeated folding units; each unit has two α-helices and a four-stranded β-sheet. The folding unit resembles the C-terminal domain of bacterial translation initiation factor 3 (IF3). Between an α-helix and a β-strand of the third IF3-like repeat of the parent domain, there is a smaller inserted domain (residues 185-279, coloured red). Although the inserted domain has the same secondary structural elements as the parent domain, it has different topology and a different fold. Insert resembles the fold observed in human thioredoxin.*

I followed the definition of protein domains in the Structural Classification Of Proteins (SCOP) database (version 1.61) (Murzin *et al.*, 1995). Although there are several available schemes of protein structure classification, I chose SCOP because it is a manually curated classification of protein structures based on their structural and evolutionary relationship. In SCOP, a protein domain is considered as a unit of evolution if it occurs independently or in combination with other domains.

SCOP represents a hierarchical classification scheme with four principal levels: family, superfamily, fold and class. Domains clustered into families are evolutionarily related and can be detected at the sequence level. Domains grouped into superfamilies can have low sequence identity but their structural and functional features suggest a common evolutionary

origin. Superfamilies with similar topology are grouped under a fold. Folds are assigned to classes based on their secondary structure. For my analysis, I considered the fold and superfamily levels of SCOP hierarchy and the five major classes (all-α, all-β, α/β, α+β and 'small proteins'). All-α and all-β classes include proteins with abundant α-helices or β-sheets, respectively. The α/β class is distinguished mainly by parallel beta sheets (β-α-β units), whereas the α+β class contains proteins with predominantly anti-parallel beta sheets (segregated α and β regions). Small proteins are distinguished by their size rather than other features.

Data for this analysis was obtained from the Protein Data Bank (PDB) (Berman *et al.*, 2002). To overcome the redundancy inherent in PDB, I chose a pre-computed list of non-redundant protein chains provided by PDB_Select (April 2002 release obtained from *ftp://ftp.embl-heidelberg.de/pub/databases/protein_extras/pdb_select*) (Hobohm and Sander, 1994). I used the set of proteins that had pair-wise sequence identities less than 90% and designated this set as PDB_90. Out of the 6182 chains in PDB_90, only 5883 chains were assigned SCOP domain definitions, extracted from the SCOP parseable file *dir.cla.scop.txt_1.61*. Table 24 shows the distribution of SCOP folds, superfamilies, families and domains in each class for chains present in PDB_90.

Table 24. SCOP (1.61 release) classification statistics for chains in PDB_90 (April 2002 release)

Class	Number of Folds	Number of superfamilies	Number of families	Number of proteins	Number of species	Number of domains
All alpha Proteins	147	244	379	719	996	1291
All beta Proteins	109	200	328	784	1475	1981
Alpha and Beta Proteins (a/b)	112	183	434	917	1365	1545
Alpha and Beta Proteins (a+b)	204	287	442	864	1194	1419
Multi-domain proteins	32	32	44	77	124	127
Membrane and cell surface proteins	10	16	28	42	58	120
Small proteins	57	82	123	324	393	698
Coiled coil proteins	4	33	33	48	57	150
Low resolution protein structures	4	4	4	6	6	9
Peptides	40	41	41	59	70	103
Designed proteins	14	14	14	18	18	27
Total	733	1136	1870	3858	5756	7470

It is self-evident that insertions can only be found in multi-domain proteins, where one domain (insert) is contained within another domain (parent). Parent and insert domains can belong to the same or different SCOP superfamilies. Likewise, a combination of two domains can be viewed as a combination of superfamily combinations. I obtained a total of

140 proteins that conformed to this definition. When I considered the 140 pairs of parent-insert superfamily combinations, I observed that several pairs were identical. Whenever there was also the same topological relationship between the parent and insert domains, I retained only one example of a pair of superfamily combinations. This procedure left 40 unique parent-insert superfamily combinations. Variations on the simple scheme 'one insert within one parent' were present; they are shown in Figure 68.

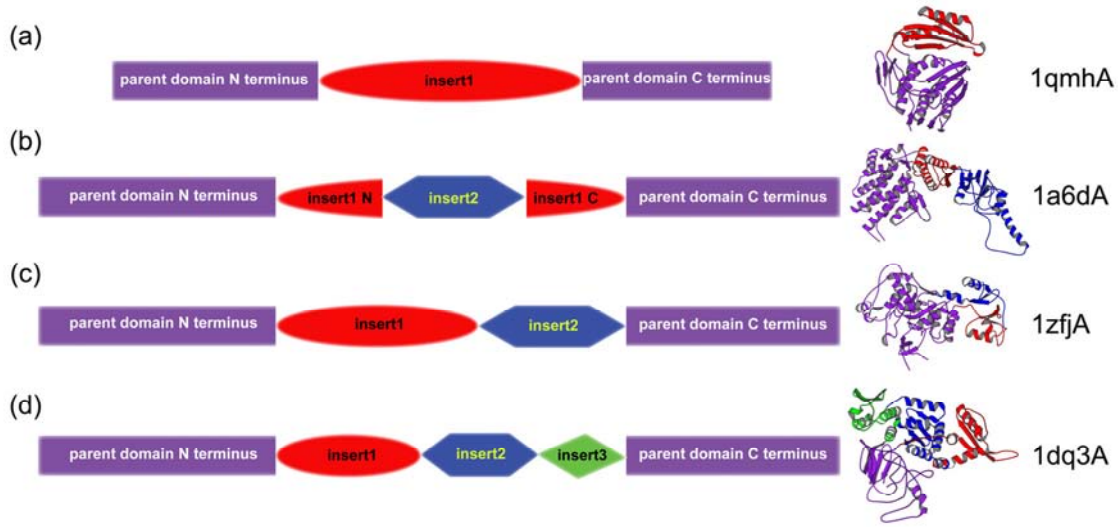

Figure 68. Schematic representation of types of domain insertions observed in protein structures. (a) Single insertion (e.g., 1qmhA). (b) Nested insertion (e.g., 1a6dA). 'insert1 N' and 'insert1 C' represent the N- and C-terminus of insert, respectively. (c) Two-domain insertion (e.g., 1zfjA). (d) Three-domain insertion (e.g., 1dq3A).

For all cases of identified domain insertions, I checked for artefacts arising from missing coordinates. This was necessary because SCOP domain definitions are based on atomic coordinates provided in PDB. To ascertain consistency, I compared atomic coordinates (ATOM records) *versus* sequences (SEQRES records) that were obtained from the ASTRAL compendium (Chandonia *et al.*, 2002). In the majority of cases, sequences were completely covered by coordinates, but in other cases, there were parts of sequences with missing coordinates. However, in none of the latter cases did the absent coordinates obscure the position of inserts.

I then calculated unique superfamily combinations for all multi-domain proteins and found 450 unique superfamily combinations for 5883 single or multi-domain proteins in SCOP. Thus, domain insertions constitute 9% (40/450) of all unique superfamily occurrences.

A.2 Types of domain insertions

Domain insertions can be categorized as either single or multiple depending on the number of inserts (Figure 68). In single insertions, one domain is inserted into another domain, and both domains can belong to the same or different superfamilies. For example, in Figure 68a, the *Escherichia coli* enzyme RNA 3'-terminal phosphate cyclase (PDB: 1qmhA, Palm *et al.*, 2000) has two domains, a small insert and a larger parent that belong to different superfamilies. Close to 90% (36/40) of observed insertions are single insertions. In multiple insertions, more than one domain, either of the same or different superfamily, is inserted into the parent domain. I observed three types of multiple insertions (i) Nested insertions: In *Thermoplasma acidophilum* thermosome (PDB: 1a6dA, Ditzel *et al.*, 1998), the archael chaperonin, the apical domain is inserted into the intermediate domain, which is in turn inserted into an ATPase domain (ii) Two-domain insertions: The type II inosine monophosphate dehydrogenase from *Streptococcus pyogenes* (PDB: 1zfjA, Zhang *et al.*, 1999) contains two tandem cystathionine-β-synthase domains inserted into the catalytic TIM-barrel domain. The second example is the *Saccharomyces cerevisiae* PI-*Sce*I intein (PDB: 1ef0A, Poland *et al.*, 2000), a homing endonuclease with protein splicing activity, which has the duplicated endonuclease domain inserted into the Hint domain (iii) Three-domain insertions: In PI-*Pfu*I, an intein-encoded homing endonuclease from the archaebacteria *Pyrococcus furiosus* (PDB: 1dq3A, Ichiyanagi *et al.*, 2000), the Hint domain has three tandem inserts, two intein endonuclease domains with αββαββαα structural motifs, and one Stirrup domain.

Previous work on intron-encoded homing endonucleases, from the dodecapeptide family, showed that for their folding, dimerisation and catalysis, they should form a dimer that has two copies of the LAGLIDADG motif (one copy per subunit of a dimer), or alternatively they could be monomeric if a monomer has both copies of the motif (Jurica and Stoddard, 1999). I found that in PI-*Sce*I (case [ii] above) and PI-*Pfu*I (case [iii] above), two monomeric domains were tandemly inserted into one parent domain. The previous observation that motifs are only functional as a dimer suggests that during the course of evolution, there was a simultaneous insertion of two monomeric domains into the parent domain, rather than an insertion of one monomeric domain followed by its duplication.

In this analysis, I treated multiple insertions as several separate parent-insert combinations, resulting in the total of 45 such combinations within 40 protein chains. There were 41 unique parent-insert superfamily combinations. Upon examination of relationships among proteins containing insertions, levels of SCOP hierarchy, and superfamily participation of parent and inserted domains, I identified several biologically meaningful patterns. These findings are discussed below.

A.3 Nature and characteristics of domain insertions: Class level

As mentioned before, I considered five SCOP classes, leading to a maximum of 25 (5*5) pair-wise combinations. From the data, I observed only 15 combinations when investigating class participation of parent-insert pairs. The combination of α/β-parent-α+β-insert was predominant, while 50% of all parents belonged to α/β class and 40% of all inserts belonged to α+β class. Domains from α/β class were parent domains, which were two and four fold more often than domains from all-β and all-α class respectively. Domains from the class of small proteins were seen only as inserts. This bias could be explained, at least to a certain extent, by taking into consideration the size and function of parents and inserts, which is discussed in the next section.

A.3.1 Size and function of domains involved in insertions

Figure 69a shows the domain length distribution for proteins from PDB_90 set across the five SCOP classes. The average domain length was longest for α/β class followed by the all-β, α+β, and all-α class. When I calculated distribution of average domain lengths for 41 parent domains, I observed the same trend (Figure 69b). However, the average length of parent domains was noticeably larger than the average length of domains from PDB_90 set; this was true for each SCOP class (compare Figure 69a and Figure 69b). Thus, combining the fact that α/β parent domains are the most abundant with the fact that α/β domains are the longest on average, I arrived at the explanation that longer domains more readily accept insertions during evolution. As for the inserted domains, α+β and all-α class were equal and major contributors to the number of domains. Therefore, the trend observed for parents is not applicable for inserts.

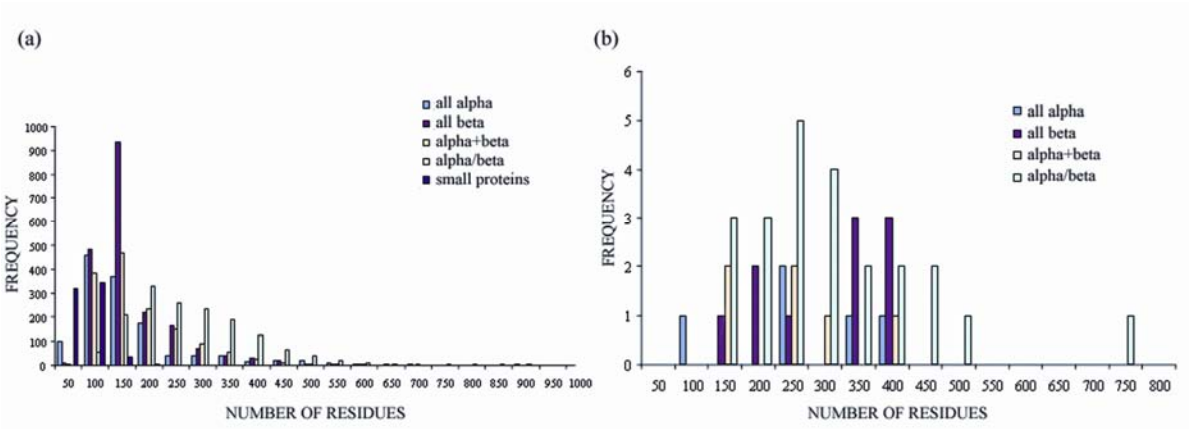

Figure 69. (a) Domain length distribution for all domains in the non-redundant set of proteins (PDB_90). (b) Domain length distribution for parent domains.

In most cases, inserted domains were shorter than parent domains. This is despite the fact that inserted domains could belong to SCOP classes with the longest average domain length (Figure 70a). Parents comprised 50-80% of protein length, while inserts comprised 20-50%. Close to 80% of inserts were shorter than 175 residues, which is the average length of a protein domain calculated from crystal structures (Gerstein, 1997). More than 60% of inserts were shorter than 130 residues. This observation is consistent with the heuristic logic that smaller domains are less likely to disturb the structure and folding of parent domains; it could explain short lengths of inserted domains. This explanation does not contradict an important experiment by Doi and colleagues (Doi *et al.,* 1997). They were able to show that when random sequences of 120-130 amino acid residues were inserted into a surface loop region of *Escherichia coli* RNase HI, about 10% of the clones retained >1% of the wild-type RNase HI activity (Doi *et al.,* 1997).

The high proportion of α/β class domains, as parents, can be correlated with their biochemical function. Previous work showed that more than a half of PDB families are enzymes and close to one half of all enzyme families contain multi-domain proteins. Multi-domain enzymes often consist of a catalytic domain and a nucleotide binding domain (Hegyi and Gerstein, 1999). It is therefore possible to predict that domain insertions are likely to occur in enzymes. Indeed, in the dataset, 39 out of 40 parent-insert pairs conform to this prediction. The remaining non-enzymatic protein is the bluetongue virus capsid protein vp-7, which has the central domain from all-β class inserted into the multi-helical parent domain. A genome-scale analysis of the structural features of proteins revealed that proteins

with α/β fold are frequently involved in fusion events (Hua *et al.*, 2002). α/β folds are also known to be disproportionately associated with enzymatic function (Hegyi and Gerstein, 1999), which lends further credence to the prominent role of α/β folds in accepting insertions.

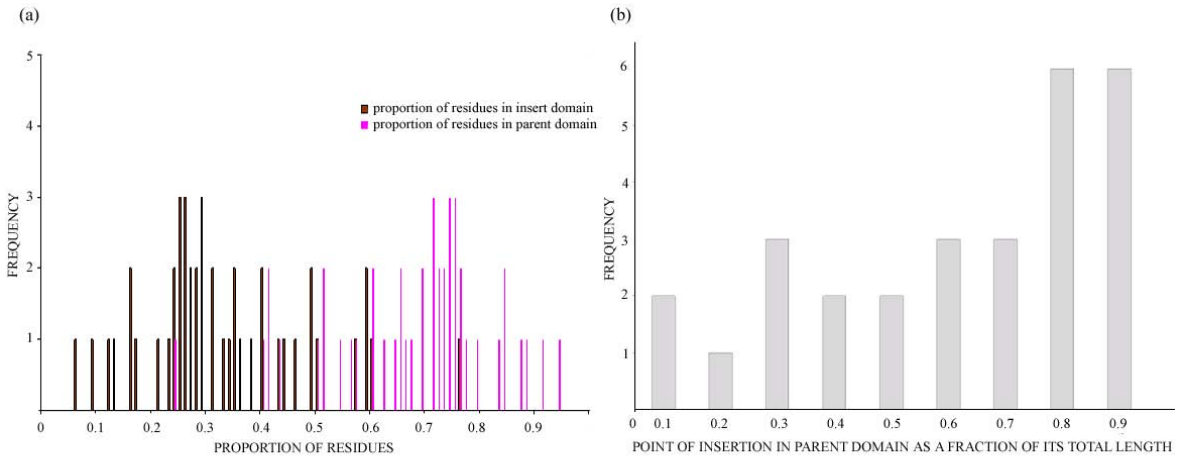

Figure 70. (a) Proportion of residues in parent and insert domains in parent-insert combinations. (b) Point of insertion in parent domain. Insert position is given as a fraction of total length of parent domain.

A.4 Nature and characteristics of domain insertions: Fold and superfamily level

Out of 57 folds in the class of small proteins, two domains with one fold (Rubredoxin fold) were found as inserts; both inserted domains belong to the same superfamily. Within the α+β class, the 18 inserted domains (from 15 superfamilies) spanned 11 folds; there are 204 different folds in the α+β class (Table 25). The trend was the same for the other SCOP classes, where folds of inserted domains constituted minor fractions of all known folds. In contrast to the inserts, all parent domains had different folds. Thus, I observed another distinction between parents and inserts at the fold level.

Similarly, parent superfamilies were found to be more versatile than insert superfamilies (most insert superfamilies combine with only one parent superfamily). There are merely 3 out of 45 insert superfamilies that combine with two different parent superfamilies. These

insert superfamilies are NAD(P)-binding Rossmann superfamily, FAD/NAD(P)-binding superfamily and C-terminal domain of FAD-linked reductases superfamily.

Table 25. Distribution of inserted and parent domains at the SCOP class and fold level. The number of domains and the number of folds they come from is given for inserted and parent domains across the five different classes in the SCOP hierarchy. Percentage gives the number of folds contributing to insertions over total number of folds under the class.

SCOP Class	Total number of folds	Inserted domains			Parent domains		
		Number of domains	Number of folds	Percentage of folds	Number of domains	Number of folds	Percentage of folds
All-α	147	6	5	3.4	5	5	3.4
All-β	109	9	9	8.3	11	11	9.2
α/β	112	10	6	5.4	23	23	20.6
α+β	204	18	11	5.4	6	6	3
Small proteins	57	2	1	1.8	0	0	0

While many parent superfamilies conservatively combine with one insert superfamily, there are conspicuous exceptions. There are three parent superfamilies each combining two different insert superfamilies. The three parent superfamilies in question are Zn-dependent exopepetidases superfamily, nucleotidyl transferase superfamily, and nucleotide-binding domain superfamily. Moreover, there are two parent superfamilies each combining with three different insert superfamilies. The two parent superfamilies are P-loop containing NTP hydrolases superfamily, and FAD/NAD(P)-binding domain superfamily.

Two further observations at the superfamily level are worth mentioning. Firstly, all parents and inserts belong to different superfamilies. There is only one exception: in *Escherichia coli* enzyme glutathione reductase (PDB: 1gesB), the parent and insert belong to the same superfamily of FAD/NAD(P)-binding domains. Secondly, superfamilies that are popular in the parent or insert context also appear to be popular in the sequential domain combination context (Apic *et al.*, 2001). They were found combining with more than one superfamily in the sequential domain order. One exception to this correlation is the superfamily of C-terminal domains of FAD-linked reductases; this superfamily is popular in the insert context, but does not tandemly combine with other superfamilies.

A.5 Point of insertion

I did not find any bias in the distribution of insertion points within 41 unique parent-insert combinations. However, a significant bias in the location of the insertion point was observed when I considered a subset of 28 parent-insert combinations, where either the parent or insert superfamily also participated in sequential combination with other superfamilies. As shown in Figure 70b, for the 28 cases in question, the insertion point occurred in the last third part of the parent domain sequence (confidence level 98%). Spatially, all 41 insertions were observed in loop regions of the 3D structure of parent domains.

Though it may not be feasible to provide a definitive explanation for the observation of bias towards C-terminus for insertion in the parent domain, an event in the N-terminus or the middle of the domain are likely to disrupt the gene structure and pose a problem during transcription or translation.

Also insertions in the C-terminus indicate most of the insertions seen in the database are not *strictly* insertions but normal sequential combinations with the second domain starting before the end of the first domain. This stem from the fact, C-terminus bias in insertion is found only in cases of parent-insert combinations, where either the parent or insert also occur in sequential combinations with other superfamilies. Further research on the domain insertions involving the core structure of the parent and insert domains can throw more light on this view.

A.6 Proximity of N- and C-termini in inserts

I wanted to determine how the insertion context affects the distance between N- and C-terminus of an inserted domain. The distance between termini was defined as the distance between C-alpha atoms of the first and the last residue of the domain. I first calculated distances for domains that do not participate in insertions. In order to do this, I considered 1000 domains, each representative of one SCOP superfamily. I obtained sequences and coordinates for the domains from the ASTRAL compendium (Chandonia *et al.*, 2002). Only 687 domain sequences were completely covered by coordinates. Using AEROSPACI scores (Chandonia *et al.*, 2002), I was able to find 60 substitutes for the 313 representative domains that were not entirely covered by coordinates. Altogether, I obtained complete coordinate

information for 747 domains (687 + 60). Because I confined the analysis to five major SCOP classes, I calculated distances between termini for the 711 domains, which belong to the five classes being investigated. The average distance for representative domains was 25 Å.

Calculation of distances between the termini of inserted domains was less straightforward. Domain boundaries reported in SCOP are human defined. Therefore, I compared SCOP domain boundaries for 41 inserted domains against the domain boundaries reported in CATH database (Orengo *et al.*, 2002). In contrast to SCOP, CATH structural classification of proteins has been produced automatically. However, only 28 out of 41 inserted domains were available in CATH, whereas the other 13 have either differences in domain classification or the corresponding proteins were absent from CATH classification. For 28 inserted domains, boundaries were identical between SCOP and CATH. The average distance between domain termini of inserted domains was 8 Å (confidence level 99%), which is two-thirds shorter than the distance between termini in normal domains.

There are two superfamilies that occur in both parent and insert context. This example allowed me to compare distances between termini for a parent and an insert from the same superfamily. In case of FAD/NAD(P)-binding domain superfamily, the distances were 30 Å and 5 Å for parent and an insert, respectively. These figures were 11 Å and 8 Å for NAD-binding Rossmann domain superfamily. Thus, this analysis shows that the ends of inserted domains are significantly closer than ends of parent domains or domains not participating in insertions. However one must be cautious in interpreting the results as the N and C termini distances for the parent domain is not calculated for the core structure.

It is interesting to speculate how the distance between domain termini can affect stability and conformational flexibility of a protein domain. While insertion context might generally reduce conformational freedom of the domain, it can simultaneously contribute to the stability of the domain, which would in turn affect its function. One can also imagine how the close proximity of domain termini can restore protein conformational flexibility by mimicking an inter-domain link observed in sequentially ordered domains.

A.7 Conclusions

Utilising an evolutionary basis of domain classification, I described the nature and characteristics of domain insertions in protein structures. Domain insertions represent an unusual but abundant case of multi-domain proteins. This analysis gave several novel insights into the nature and characteristics of domain insertions.

(1) Close to 9% multi-domain proteins contain insertions.
(2) The majority of insertions are the single domain insertions. Also found there were two-domain, three-domain, and nested insertions in PDB.
(3) α/β class has a higher propensity to accept insertions. This could be correlated to the size and function of proteins within the class.
(4) Parent domains were found to be longer than the inserted domains in most cases.
(5) When fold and superfamily combinations were considered for parents and inserts, the former was found to be more versatile than the latter, in that the parent domains combined with more partners.
(6) The point of insertion is biased towards the C-terminus of parents whenever the parent domain belongs to the superfamily that sequentially combines with other superfamilies.
(7) Inserted domains have juxtaposed termini compared to parent domains.

Perhaps, domains are more viable in the insert context when their termini are close in space; small size can further contribute to their viability.

These results clearly indicate that despite the structural and functional constraints inherent in the process of domain insertion, this process is an effective way of creating multi-domain proteins. This description of the many features of domain insertions could be used in protein engineering for producing novel multi-functional fusion proteins. Betton and co-workers (Betton *et al.,* 1997) created hybrid proteins by inserting a penicillin-hydrolysing enzyme TEM beta-lactamase (Bla) into the maltodextrin-binding protein (MalE); they used the permissive insertion sites identified before (Duplay *et al.,* 1987). Two insertions resulted in the functional hybrids, one insertion occurred in the first quarter of the MalE protein, while the other occurred in the last quarter. The parent protein (MalE) belongs to the α/β class, and the authors experimentally showed the 5 Å distance between the termini of the inserted

domain (Bla). Thus, there is recent experimental data that nicely fit into the picture of insertions found in natural multi-domain proteins.

APPENDIX B: PROTEIN EVOLUTION

B.1 Introduction

Divergence in structure and function of proteins is due to an evolutionary process driven by functional and environmental constraints. These constraints bring about changes in the protein sequence through mutations, insertions and deletions with the preservation of residues important for the structure and function of the protein (Chothia and Lesk, 1986). However, not all the sequence modifications are incorporated or maintained since some changes may be deleterious to the structure or function of the protein. Hence, the structural *'core'* (Chothia and Lesk, 1986) tends to be well conserved during evolution. When proteins evolve, the constraints on the protein structure are relaxed or rather replaced by new constraints and the sequence and structure can change more radically. These changes are generally slow processes and leave a trail of *homologs*. Homologs are proteins evolved from a common ancestor and their evolutionary relationship is evident from similarities in sequence, structure and function. Homologous proteins have been studied for a long time to understand their evolutionary relationships and to assign function or structure to new protein sequences. For homolog searches in the sequence databases, one needs an alignment algorithm, residue similarity matrix, scoring scheme and knowledge about scoring thresholds to identify true relationships.

Among the available pairwise alignment algorithms, one of the most sensitive is the Smith-Waterman algorithm (Smith and Waterman, 1981) adopted in the SSEARCH program (Pearson, 1991). Although this algorithm is more sensitive and rigorous, it is computationally expensive in comparison to FASTA (Pearson and Lipman, 1988) and BLAST (Altschul *et al.*, 1990). The speed and convenience of BLAST made it the most popular program, although it compromises sensitivity. FASTA ranks between these two programs and can be run in two modes: either at greater speed (ktup = 2) or greater accuracy (ktup =1). Pearson (Pearson, 1991, 1995) did a comparison of these three methods and showed that the Smith-Waterman algorithm worked slightly better than FASTA, which was in turn much more effective than BLAST.

Although pairwise comparison methods are a common way to find sequence homologs, they have difficulty in detecting remote homologs when sequence identity falls below 30% (Brenner *et al.*, 1998). Alternate methods like Profile Hidden Markov Models (Eddy, 1996; Krogh *et al.*, 1994), psi-BLAST (Altschul *et al.*, 1997) and Intermediate Sequence Search (Park *et al.*, 1997) reduce this limitation and increase sensitivity.

Intermediate Sequence Search (ISS) is a search technique, wherein two related sequences which cannot be detected directly by pairwise sequence comparison methods are matched using an intermediate sequence sharing close homology with the two distantly related sequences. This concept has been extended to include multiple intermediate sequences (MISS) between two distant sequences (Salamov *et al.*, 1999). The disadvantage with ISS is that the errors caused in the intermediate are likely to propagate as it is not dependent on multiple sequence alignment. Errors caused by ISS when comparing multi-domain protein sequences, can be avoided by splitting query sequence to individual domains. Figure 71 gives an overall idea on how different methods are exploring the sequence space (Lindahl and Elofsson, 2000).

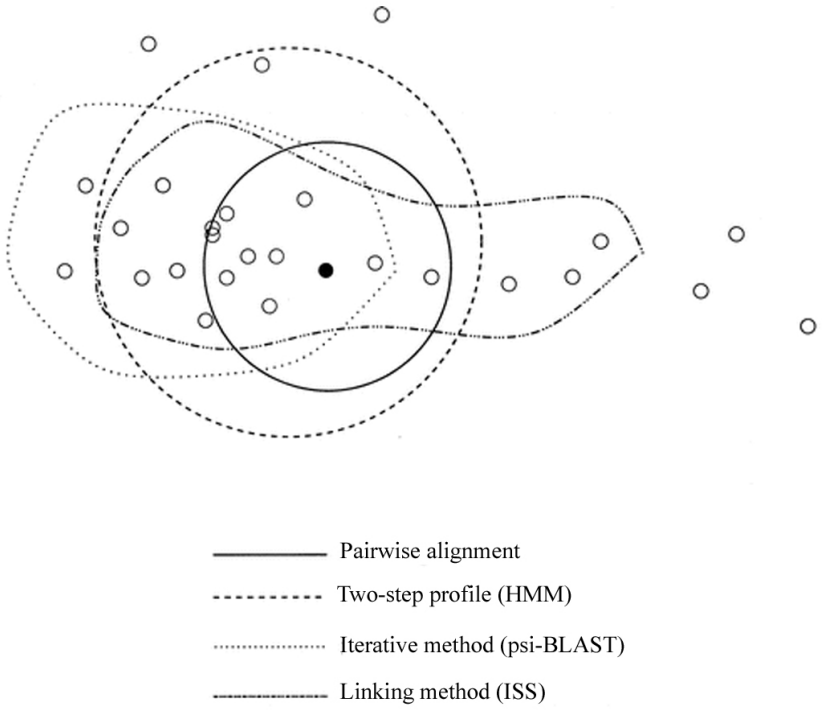

Figure 71. Schematic diagram showing performance of different sequence comparison methods. The filled circle represents the query sequence used in the database search and the open circles represent family members. The distance between two circles represents some arbitrary distance.

A comparison of these recent methods with pairwise sequence comparison methods, performed by searching remote homologs in a Structural Classification Of Proteins (SCOP, Murzin *et al.,* 1995) sequence database having less than 40% identity, show that ISS performs one and half times better than FASTA. In sequences with less than 30% identity, a HMM-based SAM-T98 and psi-BLAST detected three times more relationships than pairwise sequence comparison methods (Park *et al.,* 1998). Sauder *et al.* compared the quality of alignments produced by BLAST, psi-BLAST, ISS and ClustalW (Thompson *et al.,* 1994) with structural alignments. ISS produced longer alignments than psi-BLAST with nearly comparable per-residue alignment quality. At 10-15% identity, BLAST correctly aligned 28%, psi-BLAST 40% and ISS 46% of residues to the structural alignment (Sauder *et al.,* 2000).

All these results show that ISS performs as well as psi-BLAST in identifying distant homologs. However it is not yet clear how ISS is able to detect remote relationships. Moreover, I was interested to determine whether intermediates identified by ISS can provide any knowledge about protein evolution. This study tries to find answers to these questions.

To aid this objective, I also used structure comparisons to understand relationships between proteins. The degree of fitness between structures is usually calculated by a scoring scheme. The common way to represent the structural fitness is Root Mean Square Deviation (RMSD) for all residues of the two protein structures. The RMSD gives a measure of the average level of deviations over the superposed atoms.

$$\sqrt{\sum_{i=1}^{n} \frac{D_i^2}{N}}$$

Where, *D* refers to deviation of the atoms and *N* refers to the number of atoms matched.

There are different structural alignment methods adopting the aforementioned algorithms. Amongst the common implementations are DALI (Holm and Sander, 1993), Combinatorial Extension (CE) (Shindyalov and Bourne, 1998), and Protein Informatics System for Modelling (PrISM) (Yang and Honig, 2000). Here, I used PrISM to compare the structures.

Protein evolution may occur in two ways: divergent or convergent evolution. When a protein structure diverges to form a new fold or function, it results in divergent evolution

(e.g., P-loops). However if two evolutionarily independent folds converge to represent similar structure or function it becomes convergent evolution (e.g., serine proteases). Proteins evolved through a divergent mechanism are likely to have a trail of homologs and can be detected using sequence and structure comparisons. Here, I attempt to study this using two well known protein families – *Cytochrome c* and *P-loops* and answer the following questions.

(1) Is it possible to understand the evolutionary pattern of any protein family or superfamily based solely on its structure and sequence divergence?
(2) Whether understanding this will help us in assigning hierarchies for a protein in the existing classification of protein structures?

B.2 Datasets

I used SCOP database for this study (please refer to Appendix A for details of SCOP). The *All-α* protein class contains a fold level called *cytochrome c*, which in turn is composed of a single superfamily named *cytochrome c*. This superfamily has four families. The *Di-haem cytochrome c peroxidase* family has only synthetic protein structures and, therefore, only domains from the other families (39 sequences) were used in this analysis.

P-loop domains are found in the class *α/β* and fold/superfamily *P-loop containing nucleotide triphosphate hydrolases* (this fold has only one superfamily). The superfamily has domains composed of parallel beta sheets of varied sizes connected by helices. For example, the *Nucleoside and nucleotide kinases* family has 5 strands with architecture type 23145 and *Nitrogenase iron-protein like group* family has 7 strands with architecture type 3241567. The superfamily is composed of 14 families. I used all the domains (85 sequences, excluding domains involving multiple chains) from these 14 families for this analysis.

From these datasets, I then found sequence homologs and structure homologs that can be detected by the above described methods.

B.3 Intermediate sequence search

I collected homologs for each of the domains in the two superfamily datasets using FASTA 3.3 (with BLOSUM 62 matrix, ktup = 1) by searching against the pdb90d_1.53 database. The pdb90d_1.53 database is derived from sequences of SCOP domains (version 1.53) sharing 90% or less sequence identity.

Domains (query and target), with scores better than the threshold value 0.01, are referred as 'direct hits'. For domains that cannot be detected directly, I used the ISS procedure described above to link the query and target.

A comparison of ISS hits with psi-BLAST shows that psi-BLAST can detect all the remote homologs identified by ISS in P-loops superfamily and only about half of them in cytochrome c superfamily. The advantage ISS has in some cases might be due to the match score it gains by producing longer alignments around conserved regions of the protein. However, both the methods fail to detect remote homologs from P-loops superfamily than found from cytochrome c superfamily. This might be due to the extensive divergence of sequences in P-loops superfamily (they are quoted to have some converged domains (Bossemeyer, 1994) and differences in sequence length (average length of P-loops is ≈ 230 amino acids, twice the size of cytochrome c).

Intermediate searches based on structural information could find new remote homologs that ISS could not detect. This is expected because it is known that different sequences can have similar folds. Therefore, by comparing structures it is more likely to detect remote homologs. I suggest that by using intermediate structural search, even more distant relationships can be detected.

Then I used the alignments obtained from the query-intermediate and target-intermediate to generate a "progressive alignment" (i.e., a multiple sequence alignment generated by progressively aligning pairwise alignments using *ClustalW* alignments and structure information) of query-intermediate-target or query-intermediate-intermediate-target.

These progressive alignments show that the intermediates can improve the quality of alignments between query and target. An example of this alignment is shown in Figure 72.

The figure shows the improvement in alignment between query-target (SCOP Ids: *d1a56__* - *d1c75a_*) produced by FASTA (Figure 72a) and the progressive alignment generated manually after introducing one (*d451c__*) and two intermediate (*d1ayg__* and *d451c__*) sequences (Figure 72b and Figure 72c). The alignment shows that there are some residues common in all the sequences and some between query-intermediate, target-intermediate and intermediate-intermediate.

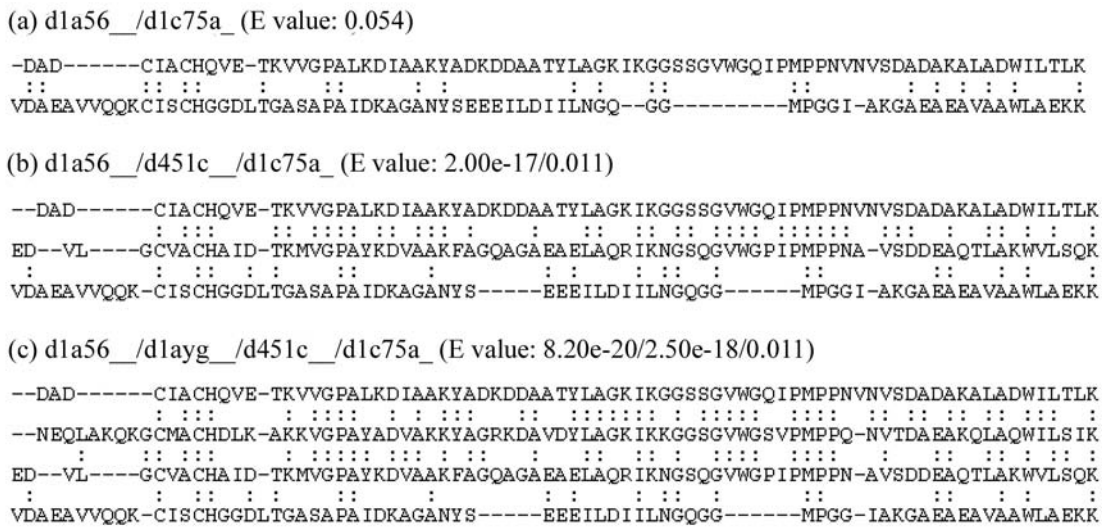

Figure 72. Comparison of alignments of two distant proteins with and without intermediates. (a) Alignment of the two domain produced by FASTA 3.3. (b) The progressive alignment generated by including one intermediate. (c) The progressive alignment generated by including two intermediates.

Likewise, I selected closely clustered domains from each of the four SCOP protein groups (*mitochondrial cytochrome*, *cytochrome* c_2, *cytochrome* c_{551} and *cytochrome* c_6) to make a progressive alignment. These groups were used due to the fact that they represent most of the members of the superfamily. From the progressive alignment made for each of the protein groups, I derived a consensus (Figure 73). This consensus was then used to derive an overall consensus shown in Figure 74. The figure shows that there are 10 invariable residues in the consensus and it agrees with the consensus derived by Ptitsyn by aligning 164 sequences from the cytochrome c superfamily (Ptitsyn, 1998). His alignments were generated using the PileUP program and manually edited taking functional residues into consideration.

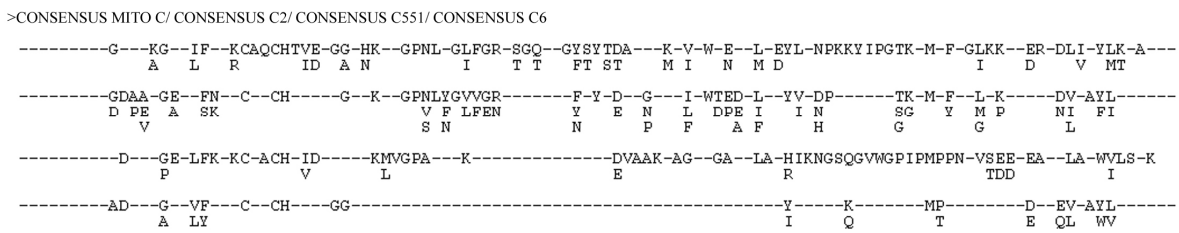

Figure 73. Consensus sequences derived for the four SCOP protein group in monodomain cytochrome c family

Figure 74. Consensus of consensus for sequences in monodomain cytochrome c family

The conserved residues were involved in heme binding and needed for functional role of the protein. The other conserved residues do not have any functional role and are found to be key residues needed to maintain structural fold of cytochromes. The key residues reported here agree well with the results found in the literature (Ptitsyn, 1998). Figure 74 shows the key residues identified by Ptitsyn. The differences include two additional residues conserved at position 3 (aliphatic residue) and position 10 (aliphatic residue), the presence of a proline at position 1 and a phenyalanine instead of an isoleucine at position 8. These discrepancies might be due to number of sequences compared and the kind of alignment generated. Ptitsyn used 164 sequences whereas here only 19 sequences were used. Although comparatively very few sequences were used, the result seems to be almost the same. This is a promising result opening opportunities in extending the procedure to other superfamilies. However, an attempt on P-loops failed primarily due to the fact that the superfamily is much more diverged and only very few sequences form distinct clusters.

B.4 Structural homologs

I did an all-against-all structural comparison of the domains using PrISM. Then I used the alignment from PrISM as input to another program called MSARMS (Hubbard, 1994) that measures the distance in Angstrom between the matched residues in the superposition. These RMSD values from PrISM and MSARMS programs were used for this study.

B.5 Clustering

With these homologs and their relationship (given as *E-value* for sequences and *RMSD* for structures), I represented proteins as clusters in two-dimensional space. This was done using the procedure given in Figure 75 using sequence/structure distance matrices (or similarity matrices).

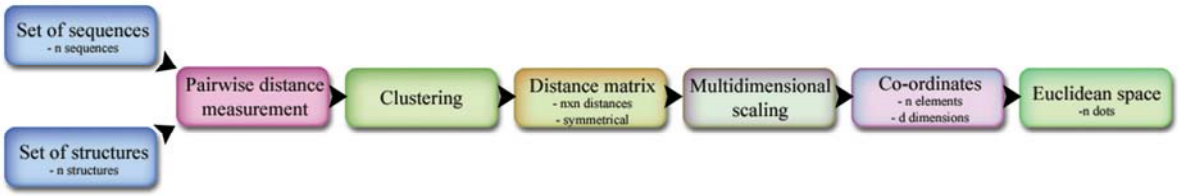

Figure 75. Flow chart describing steps used in clustering and visualisation of data.

I did initial clustering based on the sequence based distance matrix using single and complete linkage methods with a threshold E-value of 0.001 and 0.05 respectively. Then I merged the resulting sets of clusters based on the RMSD values using the Unweighted Pair Group Method using Arithmetic average approach. A threshold value of 4.00Å was used for the P-loops superfamily and a threshold of 2.00Å was used for the cytochrome c superfamily. I also applied the complete linkage approach to merge the initial set of clusters using a threshold value of 6.00Å for both superfamilies.

To find co-ordinates of the data set in 2D space, I used Principal Co-ordinate Analysis (PCoA). For a problem of *N* objects, there could be *N*(N-1)* distances and displayed in *(N-1)* dimensional space. This *(N-1)* dimensional space was reduced to 2D/3D space and plotted.

A manual plotting of the data gave a cluster map for both cytochrome c (Figure 76) and P-loops superfamilies (Figure 77). Figure 78 shows the demarcation of clusters into family and protein levels based on the SCOP classification for cytochrome c. Similarly, Figure 79 shows the demarcation of family levels in P-loops. The protein levels were not marked in P-loops to avoid the complexity in the figure.

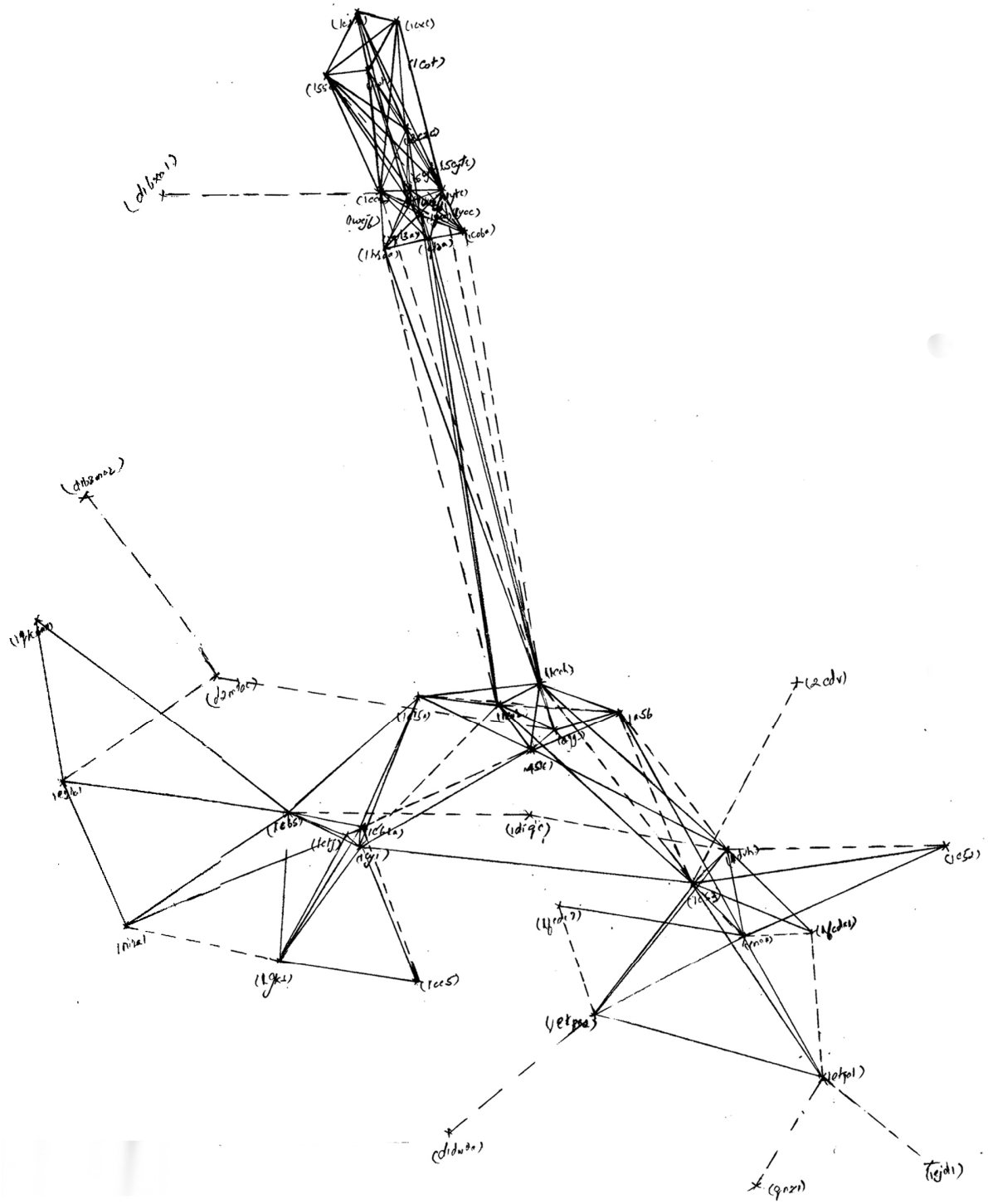

Figure 76. Cluster map of cytochrome c superfamily

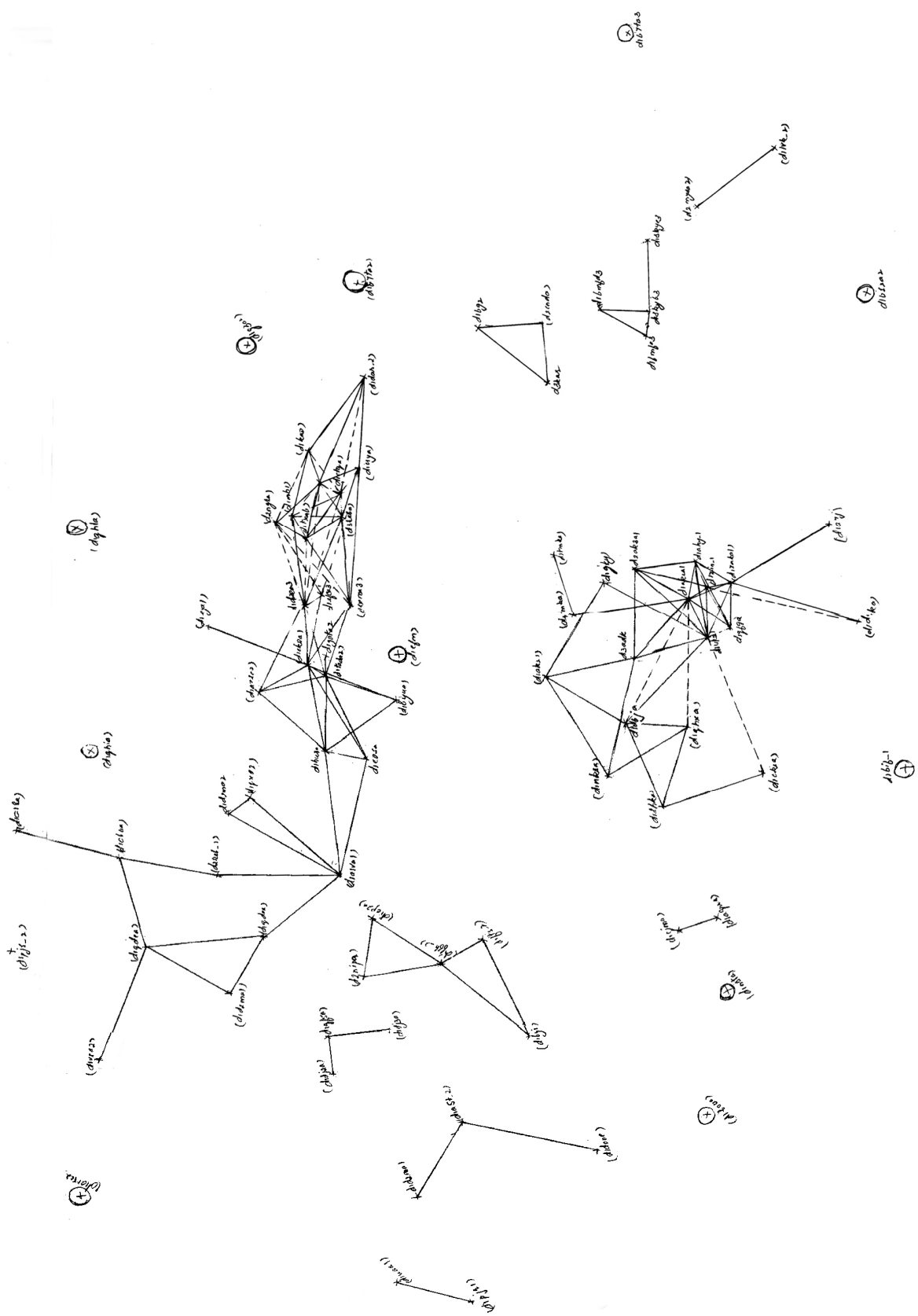

Figure 77. Cluster map of P-loops superfamily

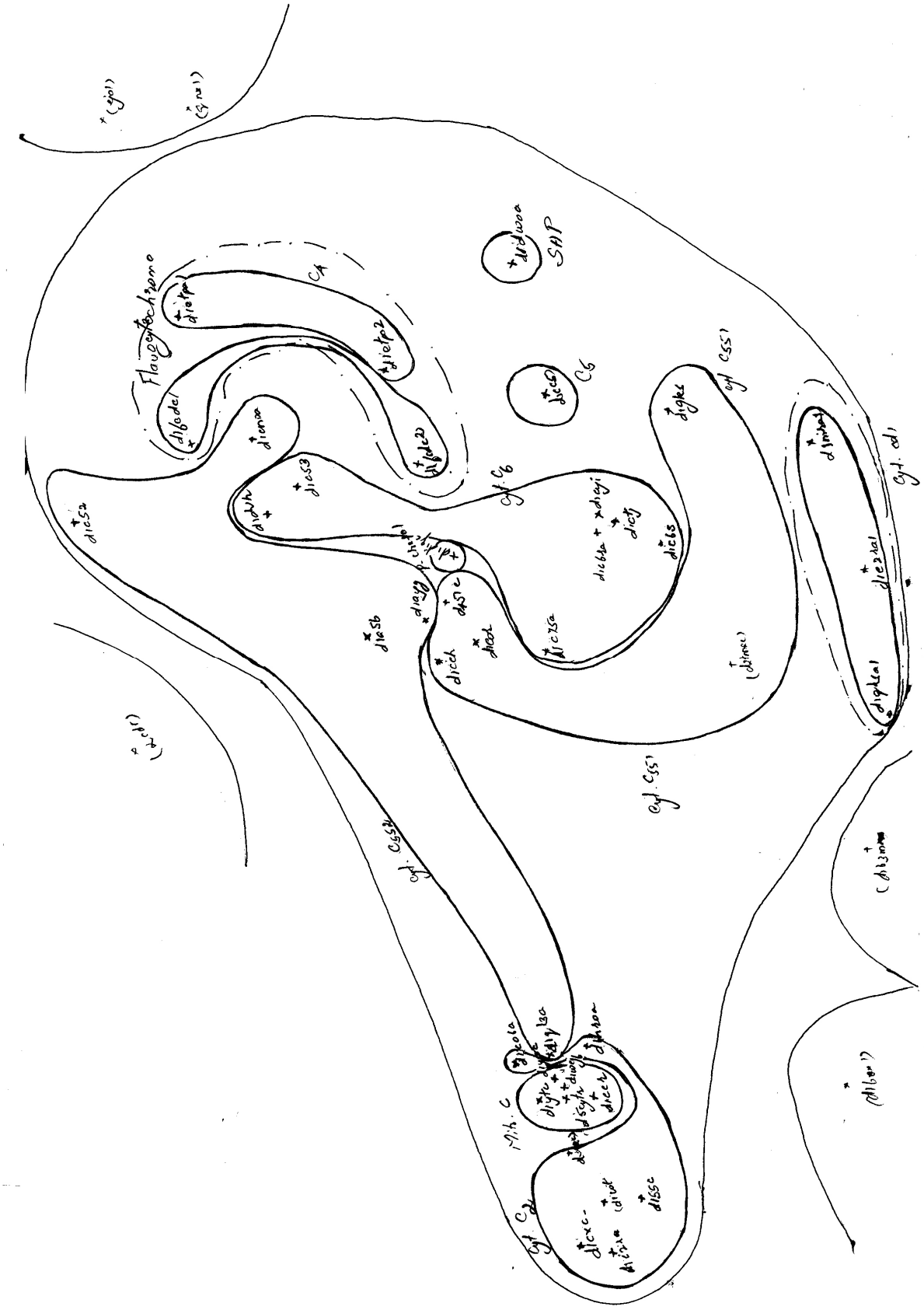

Figure 78. Cluster map of cytochrome c superfamily with demarcation of SCOP superfamily, family and protein levels

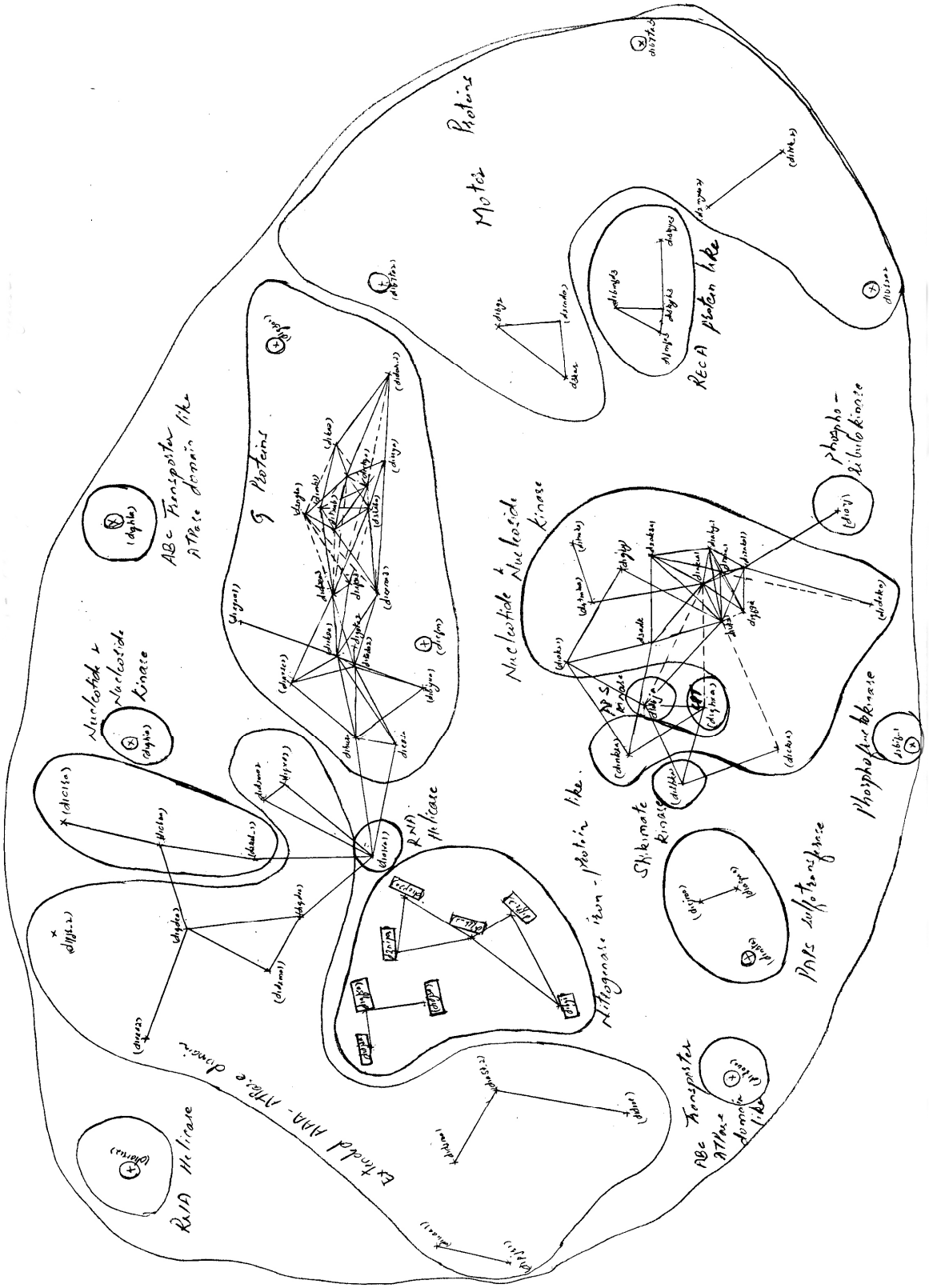

Figure 79. Cluster map of P-loops superfamily with demarcation of SCOP superfamily, family levels

The maps (Figure 76 and Figure 77) show domain relationships either by solid lines or dashed lines. The solid lines indicate domains having strong relationship between them (E-value < 0.4 and RMSD < 4 Å). Also, the length of the solid line represents real Euclidean distance in the cluster map. The dashed lines show there is a relationship between the connected domains. However, the position of domains in the map is not true. This is due to the non-availability of a relationship between the connected domains and its neighbors. Also, the length of the broken line does not represent real Euclidean space in the map.

The cytochrome maps (Figure 76 and Figure 78) show that two SCOP protein groups, *mitochondrial cytochrome c* and *cytochrome* c_2, were well separated from other protein groups. The domains forming the *cytochrome* c_{552} cluster show that they have diverged more than any other SCOP protein group. Also, it can be seen that most of the domains from the *cytochrome* c_6 and *cytochrome* c_{551} SCOP protein groups form closer clusters while some of them get away from this cluster and act as outliers.

P-loops cluster maps (Figure 77 and Figure 79) show that the domains have diverged more when compared to the cytochrome c domains. The maps show a number of domains represented as singletons or as small groups not connected to each other. As stated earlier, absence of a line between domains means no relationship can be identified among them (with score below the threshold limit), although some of the singletons belong to SCOP family. Only members of two families (*Nucleoside and nucleotide kinase* and *G-proteins*) were found to be grouped together on the map. This may be due to more environmental constraints and less active site requirements on P-loop superfamily or may be due to a convergence phenomena as seen in phosphate binding proteins (Bossemeyer, 1994).

These cluster maps are a useful tool to aid in understanding of the relationship between protein members of a family:

(1) It gives an overall picture of the divergence of a protein superfamily.
(2) It shows the relationships between SCOP families.
(3) The method could be used as an initial automated classification procedure of protein structures. A new protein structure can be used as a query to find its sequence or structure homologs. Then based on the sequence and structural relationship (E-value and

RMSD), the protein can be added in the cluster map. Such a map will give a good idea to which of the superfamily or family the new protein belongs. Then with detailed knowledge, the protein can be allocated in a specific family (manual curation). The clustering approach can be exploited to assign function to an unknown protein (Sternberg, 2001), but it cannot be trusted fully as a similar structure does not always represent the same function.

(4) It gives a clear picture about any particular SCOP family and allows the identification of any outliers in it. In the P-loops cluster map (Figure 79), there are two clusters one with domains *d1d2ja__, d1qf5a__ and d1dj3a_* and another with *d2nipa_, d1cp2a_, d1ffh__, d1byi__ and d1fts__* (boxed). But all of these domains are placed in the same family in SCOP. On discussion with Alexey Murzin (the primary curator of SCOP database), he recalled he considered that it might be better to keep these two clusters in two separate groups, say as, two different sub-families/families. He only kept them together due to limitations in the current SCOP classification system.

Likewise the domain *d1qhia_,* classified in the *Nucleotide and nucleoside kinase* family in SCOP, are positioned separately from the main cluster. The outlier was later cross-checked with structural analysis (Morea, 2001). The analysis also agreed that the domain is distinct from its family members. The probable reason for the isolated cluster of *d1qhia__* is that it is a chimeric protein and does not exist naturally i.e. it does not have sequence or structure homology with other *Nucleotide and nucleoside kinase* proteins even though it retains the same function. It was for this reason and since the domain satisfied minimal the P-loop topology, that Alexey Murzin classified the domain under the same family.

Thus, cluster maps might help us to be aware of outliers in a particular superfamily/family classification before starting any kind of detailed analysis on it.

Because of these advantages of the cluster maps, I automated the clustering process to extend the study later for other families. A comparison between manual and automated clustering procedures shows that the automated method performed equally well with the manual method (Figure 80 and Figure 81). Also, the automated methods provide similar results with another automated clustering procedure based on the MCL algorithm (Enright *et al.*, 2002).

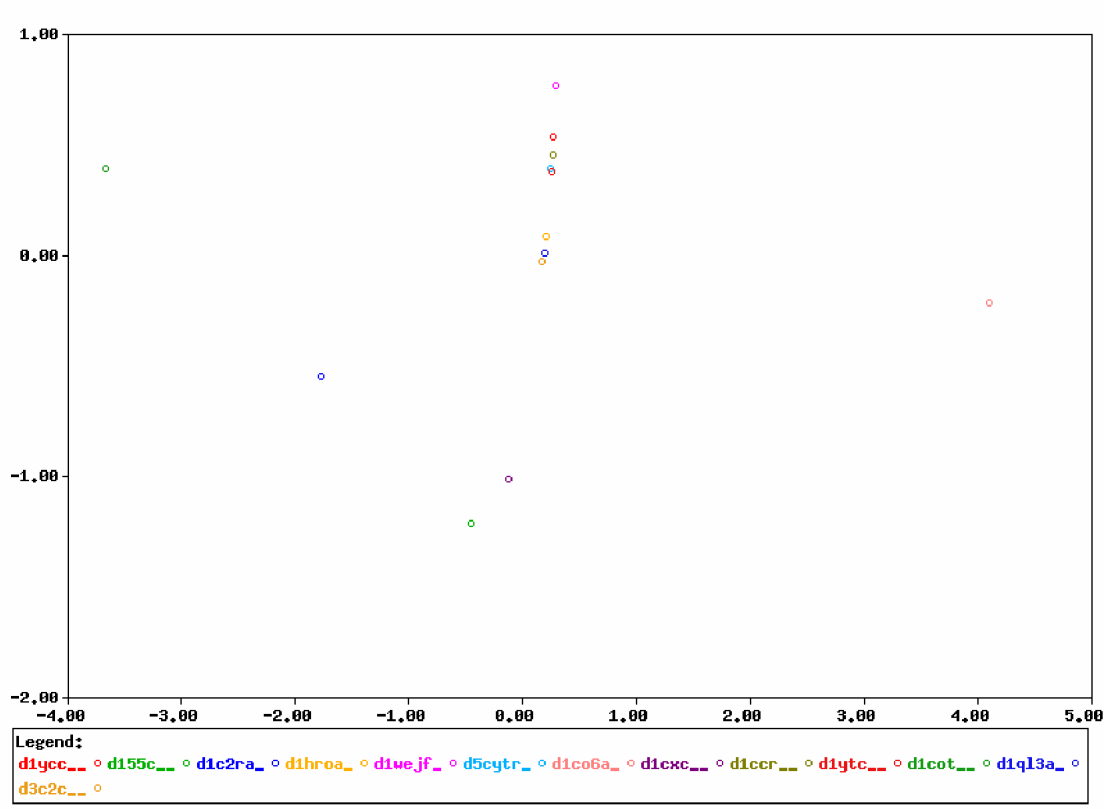

Figure 80. A cluster produced by the automated method for cytochrome c superfamily

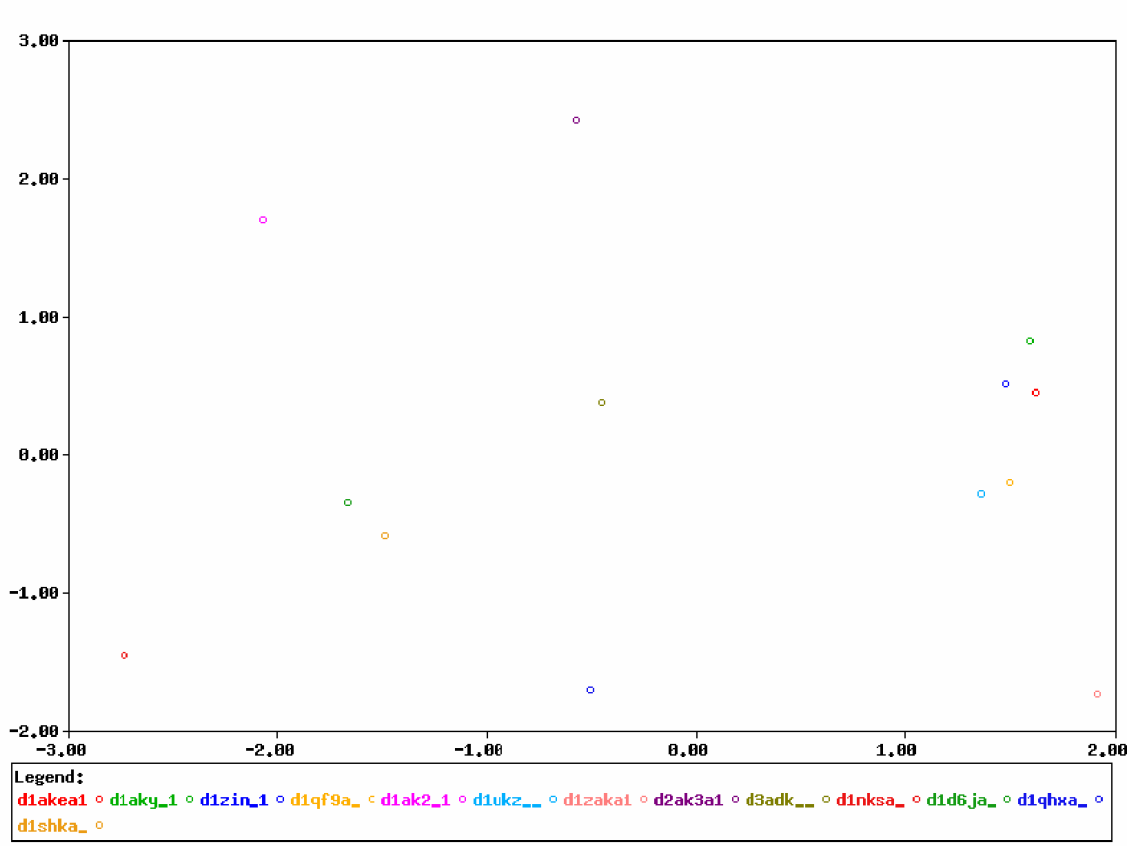

Figure 81. A cluster produced by automated method for P-loops superfamily

In both manual and automated processes, clustering was done using sequence and structural relationships, but it is possible to be done with sequence information alone. However, this will give only the number of clusters that can be formed from the superfamily and members in each cluster. A two dimensional representation of data is difficult with sequence information alone due to the fact that the data needs to undergo significant normalization procedures before it can be used to find co-ordinates.

B.6 Orthology and paralogy

The sequence and structural information, used above to generate cluster maps, can also form the basis for detecting orthologous relationships within protein families in the study of protein evolution. Such a group of ortholog domains was found in P-loops superfamily. The group comprises adenylate kinases from *Escherichia coli*, *Bacillus stermathermophilus* and *Saccharomyces cerevisiae*. Using species as a time scale, it can be said that adenylate kinase of *Escherichia coli* and *Bacillus stermathermophilus* appeared earlier than yeast protein. However, it does not mean that yeast protein evolved from *Escherichia coli* or *Bacillus* and it would be extremely difficult in assessing the proper time scale for these proteins based on sequence and structure information alone.

All the three adenylate kinases clustered close to each other on the map. So, from tightly clustering domains, it can be presumed that they are possibly to be orthologous to each other.

The TOPS (Westhead *et al.,* 1999) diagrams of these three proteins (Figure 82) shows that *Escherichia coli* and yeast adenylate kinases are identical whereas in *Bacillus*, there is an extra β strand and its orientation is reversed. Interestingly, this part of the protein is not under SCOP domain definition, which means that there is no functional or structural role for this part of the protein. Since this part does not have structural or functional constraints, it is more likely to be subject to mutations and may be influenced by environmental factors of *Bacillus* compared with yeast or *Escherichia coli*. From this, I conclude that the evolution of adenylate kinase would have more likely started from a common ancestor and given rise to *Escherichia coli* and or *Bacillus* and later to yeast protein. Later, *Bacillus* adenylate kinase would have acquired some changes in its protein.

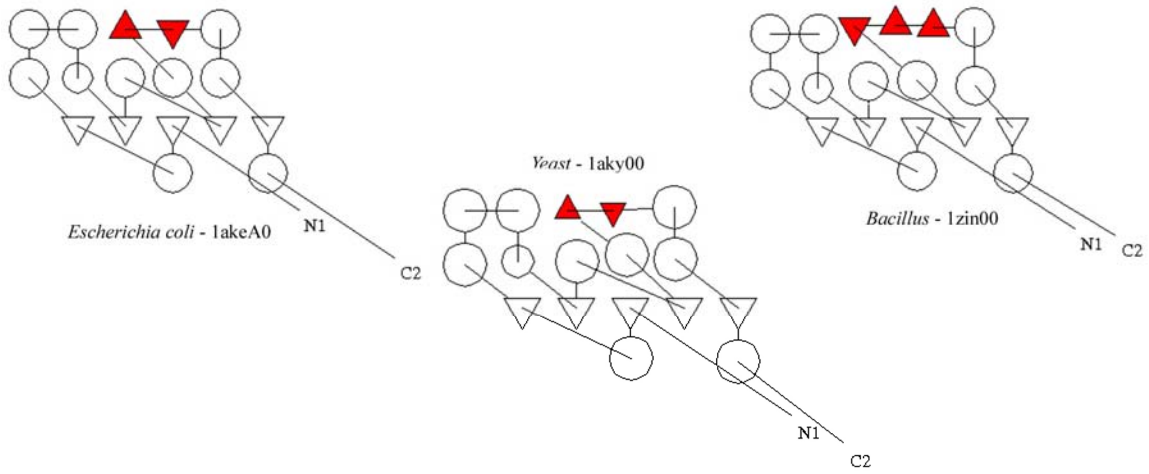

Figure 82. Topology diagram for adenylate kinase

Likewise, from the cytochrome map, two SCOP protein groups form distinct clusters from the rest of the cytochrome members. The overall topology of the cytochrome superfamily members were analyzed using TOPS (Figure 83). Generally, cytochrome c fold has 5 helices. However, some members of *cytochrome* c_{551} group have 6 helices and *cytochrome* c_2 group has 5 helices and 2 β strands except *d3c2c__*, which has only 5 helices. The topology of *cytochrome* c_{552} group (5 helices) remains the same, although its sequence has diverged greatly. However, the domains of this group (*cytochrome* c_{552}) forms close cluster with domains of different cytochrome c protein groups than among itself. It might be one of the typical cases, where orthology/homology cannot be resolved based on sequence identity because an extensive sequence divergence has occurred. However, it can also be argued that *cytochrome* c_{552} proteins were actually formed from convergence of different cytochrome c proteins. But this is highly unlikely to occur given the clear picture of overall divergence of cytochrome proteins and absence of any convergence reports in the cytochrome c fold.

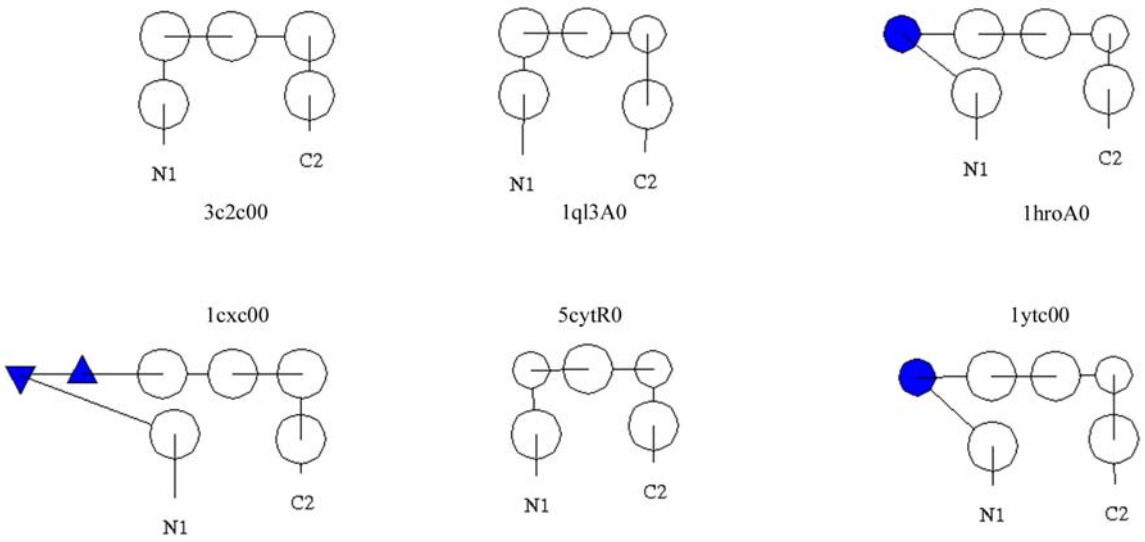

Figure 83. Topology diagram for cytochrome c proteins

Mitochondrial cytochrome c was seen later in the time-scale when compared to bacterial cytochrome c. Given the endosymbiotic hypothesis, it is likely that any bacterial cytochrome c would have given rise to *mitochondrial cytochrome*. Here, it can be seen that *cytochrome* c_2 clustered closely with *mitochondrial cytochrome* (Figure 78). So it is likely that *cytochrome* c_2 would have been the ancestral protein for *mitochondrial cytochrome*. This was confirmed with expertise knowledge of Alexey Murzin. The topology study of these two SCOP protein groups also confirmed this. The general topology of *cytochrome* c_2 and *mitochondrial cytochrome* are 5 helices + 2 β strands and 5 or 6 helices respectively. However, some of the domains of *cytochrome* c_2 (e.g., *d3c2c__*), clustering near to *mitochondrial cytochrome* lack the two β strands, confirming that the earlier forms of *cytochrome* c_2 with β strands, later lost the β strands and have given rise to *mitochondrial cytochrome*.

Thus, cluster maps made with sequence and structural homology is useful in understanding the ancestry of proteins.

B.7 Conclusions

Protein evolution, driven by structural and functional constraints, may leave a trail of homologs. Homologs are identified using sequence comparison methods like BLAST, FASTA, psi-BLAST and ISS. A comparison of ISS with psi-BLAST was made in two

protein superfamilies: cytochrome c and P-loops. The result showed that psi-BLAST detected all the remote homologs identified by ISS in P-loops and only half in cytochrome c superfamily. Although, I cannot generalize using these limited results, it can be said that ISS performs better in some cases than psi-BLAST. The advantage ISS has in some cases might be due to the match score it gains by producing longer alignments around conserved regions of the protein. Intermediate search conducted using structural information revealed that more remote homologs that could not be identified with sequence information alone. So structures might be useful in intermediate search when sequence information is inadequate in detection. From the progressive alignments generated using most of the domains in four SCOP protein groups (*mitochondrial cytochrome*, *cytochrome* c_2, *cytochrome* c_{551} and *cytochrome* c_6), an overall consensus was generated. The highly conserved residues found in the overall consensus are in tandem with the key structural and functional residues needed for the cytochrome c fold (Ptitsyn, 1998). Thus ISS alignments might be useful in understanding highly conserved residues in a protein fold.

Along with sequence information, I used structural comparisons by PrISM to produce a manual cluster map. The cluster map showed a useful representation of the general evolutionary relationships within P-loops and cytochromes. These might be helpful in depicting the relationship between SCOP families, assigning hierarchies to a new protein structure in the existing structural classification and understanding the likely ancestor of a protein. For example, in cytochrome c superfamily, it was shown that the *cytochrome* c_2 protein is likely to be an ancestor for *mitochondrial cytochrome*. The manual process has been automated and can now be used as a tool in exploring evolutionary relationships of any protein family.

APPENDIX C

C.1 Eponine transcription termination parameters

The parameters used to create Eponine transcription termination model –

```
<? xml version="1.0" ?>

<app xmlns=http://www.sanger.ac.uk/Users/td2/specs/epoapps/0/2
jclass="eponine.TrainingCore">

        <bean name="dataSource" jclass="eponine.datasource.XMLDataSource">
                <string name="fileName" value="Datasets/trainingdata.xml " />
        </bean>

        <bean name="basisSource" jclass="eponine.model.MultiplexedBasisSource">
                <int name="reweightFrequency" value="15" />
                <double name="reweightPseudocounts" value="10.0" />

                <child jclass="eponine.model.NewBasisSource">
                    <boolean name="maximize" value="false" />
                    <double name="stringency" value="0.55" />
                    <double name="stringencyVariance" value="0.03" />
                    <int name="minLength" value="4" />
                    <int name="maxLength" value="8" />
                    <double name="minDistWidth" value="2.5" />
                    <double name="maxDistWidth" value="200.0" />
                    <boolean name="reversible" value="false" />
                    <string name="name" value="nbs1_narrow" />
                    <int name="minPos" value="-190" />
                    <int name="maxPos" value="1990" />
                </child>

                <child jclass="eponine.model.SampleWMBasisSource">
                    <double name="nullModelWeighting" value="7.0" />
                    <double name="nullModelPerMarginalColumn" value="1.0" />
                    <int name="sampleCounts" value="203" />
                    <double name="nullModelWeightingN" value="9.0" />
                    <int name="sampleCountsN" value="120" />
                    <string name="name" value="samplewm2" />
                </child>

                <child jclass="eponine.model.DropColumnBasisSource" />

                <child jclass="eponine.model.DistributionBasisSource">
                    <double name="distChangeWidth" value="3.0" />
                    <double name="distChangeGamma" value="3.0" />
                    <double name="distChangeScale" value="25.0" />
                    <double name="distChangeBias" value="0.06" />
                    <!-- double name="shapeChangeProbability" value="0.05" / -->
                    <double name="flipEnvelopeProbability" value="0.00" />
```

```
                <string name="name" value="distwidth" />
            </child>

            <child jclass="eponine.model.PositionBasisSource">
                <double name="shiftWidth" value="4" />
            </child>

            <child jclass="eponine.model.CrossWMBasisSource" />

            <child jclass="eponine.model.AppendColumnBasisSource" />

            <child jclass="eponine.model.FlipMaxBasisSource" />
        </bean>

        <bean name="trainer" jclass="stats.glm.VRVMTrainer">
            <int name="numThreads" value="4" />
            <int name="maxCycles" value="11000" />
            <!--int name="cleaningCycles" value="0" /-->
            <int name="maxWorkingSet" value="28" />
            <int name="minWorkingSet" value="25" />
            <int name="initialWorkingSet" value="50" />
            <double name="initialAlpha" value="1.0" />
            <boolean name="unityHack" value="true" />
            <double name="unityHackThreshold" value="1.0" />
            <boolean name="resetAlphaHack" value="true" />
            <boolean name="insertUnity" value="true" />
        </bean>

        <bean name="retrainer" jclass="stats.glm.VRVMTrainer">
            <int name="maxCycles" value="100" />
            <!--int name="cleaningCycles" value="0" /-->
            <double name="initialAlpha" value="1.0" />
            <boolean name="unityHack" value="true" />
            <double name="unityHackThreshold" value="1.0" />
        </bean>

        <string name="fileName" value="Models/terminationmodel.xml" />
        <int name="checkpointFrequency" value="500" />
</app>
```

C.2 GAZE gene structure models

The configuration file explaining the gene model with translation features for predicting genes using GenePred –

```
<? xml version="1.0" encoding="US-ASCII" ?>

    <gaze>
        <declarations>
            <feature id="tss" st_off="0" en_off="1" />
            <feature id="tis" st_off="0" en_off="3"/>
            <feature id="5ss" st_off="1" en_off="1" />
            <feature id="3ss" st_off="1" en_off="1" />
```

```
        <feature id="tts" st_off="3" en_off="0"/>
        <feature id="polyA" st_off="1" en_off="1"/>

        <feature id="tss_rev" st_off="1" en_off="0" />
        <feature id="tis_rev" st_off="3" en_off="0" />
        <feature id="5ss_rev" st_off="1" en_off="1" />
        <feature id="3ss_rev" st_off="1" en_off="1" />
        <feature id="tts_rev" st_off="0" en_off="3" />
        <feature id="polyA_rev" st_off="1" en_off="1"/>

        <!--lengthfunction id="intron_pen" />
        <lengthfunction id="intergene_pen" />
        <lengthfunction id="inital_exon_pen" />
        <lengthfunction id="internal_exon_pen" />
        <lengthfunction id="terminal_exon_pen" />
        <lengthfunction id="single_exon_gene_pen" /-->
    </declarations>

    <gff2gaze>
        <!-- Features -->
        <gfffeat feature="TSS" strand="+" source="Eponine">
            <feat id="tss"/>
        </gfffeat>

        <gfffeat feature="TSS" strand="-" source="Eponine">
            <feat id="tss_rev"/>
        </gfffeat>

        <gfffeat feature="TIS" strand="+" source="Eponine">
            <feat id="tis"/>
        </gfffeat>

        <gfffeat feature="TIS" strand="-" source="Eponine">
            <feat id="tis_rev"/>
        </gfffeat>

        <gfffeat feature="5SS" strand="+" source="Eponine">
            <feat id="5ss"/>
        </gfffeat>

        <gfffeat feature="5SS" strand="-" source="Eponine">
            <feat id="5ss_rev"/>
        </gfffeat>

        <gfffeat feature="3SS" strand="+" source="Eponine">
            <feat id="3ss"/>
        </gfffeat>

        <gfffeat feature="3SS" strand="-" source="Eponine">
            <feat id="3ss_rev"/>
        </gfffeat>

        <gfffeat feature="TTS" strand="+" source="Eponine">
            <feat id="tts"/>
        </gfffeat>
```

```
    <gfffeat feature="TTS" strand="-" source="Eponine">
        <feat id="tts_rev"/>
    </gfffeat>

    <gfffeat feature="POLYA" strand="+" source="Eponine">
        <feat id="polyA"/>
    </gfffeat>

    <gfffeat feature="POLYA" strand="-" source="Eponine">
        <feat id="polyA_rev"/>
    </gfffeat>
</gff2gaze>

<dna2gaze>
    <!--dnafeat pattern="tataaa">
        <feat id="tss" />
    </dnafeat>

    <dnafeat pattern="atg" score="0.001">
        <feat id="tis" />
    </dnafeat>

    <dnafeat pattern="taa" score="0.001">
        <feat id="tts" />
    </dnafeat>

    <dnafeat pattern="tag" score="0.001">
        <feat id="tts" />
    </dnafeat>

    <dnafeat pattern="tga" score="0.001">
        <feat id="tts" />
    </dnafeat>

    <dnafeat pattern="aataaa" score="0.001">
        <feat id="polyA" />
    </dnafeat>

    <dnafeat pattern="tttata">
        <feat id="tss_rev" />
    </dnafeat>

    <dnafeat pattern="cat" score="0.001">
        <feat id="tis_rev" />
    </dnafeat>

    <dnafeat pattern="tta" score="0.001">
        <feat id="tts_rev" />
    </dnafeat>

    <dnafeat pattern="cta" score="0.001">
        <feat id="tts_rev" />
    </dnafeat>
```

```
    <dnafeat pattern="tca" score="0.001">
        <feat id="tts_rev" />
    </dnafeat>

    <dnafeat pattern="tttatt" score="0.001">
        <feat id="polyA_rev" />
    </dnafeat-->

    <!--takedna id="5ss_1" st_off="0" en_off="1"/>
    <takedna id="3ss_1" st_off="1" en_off="-1"/>
    <takedna id="5ss_2" st_off="-1" en_off="1"/>
    <takedna id="3ss_2" st_off="1" en_off="0"/>
    <takedna id="5ss_1_rev" st_off="1" en_off="0"/>
    <takedna id="3ss_1_rev" st_off="-1" en_off="1"/>
    <takedna id="5ss_2_rev" st_off="1" en_off="-1"/>
    <takedna id="3ss_2_rev" st_off="0" en_off="1"/-->
</dna2gaze>

<model>
    <target id="END">
        <source id="BEGIN" out_feat="No_genes"/>
        <source id="polyA" out_feat="GEN_DNA" />
        <source id="tss_rev" out_feat="GEN_DNA"/>
    </target>

    <!--Forward strand gene-->

    <target id="tss">
        <source id="BEGIN" out_feat="GEN_DNA"/>
        <source id="polyA" mindis="1" out_feat="intergenic"/>
        <source id="tss_rev" mindis="1" out_feat="intergenic"/>
    </target>

    <target id="tis">
        <source id="tss" mindis="1" out_feat="5UTR" out_str="+"/>
    </target>

    <target id="5ss">
        <!--killfeat id="tts"/-->
        <source id="tis" out_feat="inital_exon" mindis="3" maxdis= "10000" out_str="+" />
        <source id="3ss" out_feat="internal_exon"  mindis="6" maxdis= "10000" out_str="+"
    />
    </target>

    <target id="3ss">
        <source id="5ss" out_feat="intron"  mindis="6" out_str="+"/>
    </target>

    <target id="tts">
        <!--killfeat id="tts" /-->
        <!--source id="tis" out_feat="single_exon_gene" mindis="60"out_str="+"/-->
        <source id="3ss" out_feat="terminal_exon"  mindis="3" out_str="+"/>
    </target>

    <target id="polyA">
```

```
            <source id="tts" out_feat="3UTR" mindis="1" out_str="+"/>
        </target>

        <!--Reverse strand gene-->

        <target id="polyA_rev">
            <source id="BEGIN" out_feat="GEN_DNA"/>
            <source id="polyA" out_feat="intergenic" mindis="1"/>
            <source id="tss_rev" out_feat="intergenic" mindis="1"/>
        </target>

        <target id="tts_rev">
            <source id="polyA_rev" out_feat="3UTR" mindis="1" out_str="-"/>
        </target>

        <target id="3ss_rev">
            <!--killfeat id="tts_rev"/-->
            <source id="tts_rev" out_feat="terminal_exon" mindis="3" maxdis= "10000"
        out_str="-"/>
            <source id="5ss_rev" out_feat="internal_exon" mindis="6" maxdis= "10000"
        out_str="-"/>
        </target>

        <target id="5ss_rev">
            <source id="3ss_rev" out_feat="intron" mindis="6" out_str="-"/>
        </target>

        <target id="tis_rev">
            <!--killfeat id="tts_rev" phase="0"/-->
            <!--source id="tts_rev" out_feat="single_exon_gene" mindis="60" out_str="-"/-->
            <source id="5ss_rev" out_feat="initial_exon" mindis="3" out_str="-"/>
        </target>

        <target id="tss_rev">
            <source id="tis_rev" out_feat="5UTR" mindis="1" out_str="-"/>
        </target>
    </model>

    <lengthfunctions>
        <!-- lengthfunc id="intron_pen" file="./tables/intron_penalty"/>
        <lengthfunc id="initial_exon_pen" file="./tables/exon_penalty.initial"/>
        <lengthfunc id="terminal_exon_pen" file="./tables/exon_penalty.terminal"/>
        <lengthfunc id="internal_exon_pen" file="./tables/exon_penalty.internal"/-->

        <!--lengthfunc id="single_exon_gene_pen">
            <point x="500" y ="0.001"/>
            <point x="20000" y="0.2"/>
        </lengthfunc-->

        <!--lengthfunc id="intergene_pen">
            <point x="200000" y ="0.01"/>
            <point x="200001" y="0.01"/>
        </lengthfunc -->
    </lengthfunctions>
</gaze>
```

The configuration file explaining the gene model without translation features for predicting genes using GenePred –

```
<?xml version="1.0" encoding="US-ASCII"?>

    <gaze>
        <declarations>
            <feature id="tss" st_off="0" en_off="1" />
            <!--feature id="tis" st_off="0" en_off="3"/-->
            <feature id="5ss" st_off="1" en_off="1" />
            <feature id="3ss" st_off="1" en_off="1" />
            <!--feature id="tts" st_off="3" en_off="0"/-->
            <feature id="polyA" st_off="1" en_off="1"/>

            <feature id="tss_rev" st_off="1" en_off="0" />
            <!--feature id="tis_rev" st_off="3" en_off="0" />-->
            <feature id="5ss_rev" st_off="1" en_off="1" />
            <feature id="3ss_rev" st_off="1" en_off="1" />
            <!--feature id="tts_rev" st_off="0" en_off="3" />-->
            <feature id="polyA_rev" st_off="1" en_off="1"/>

            <!--lengthfunction id="intron_pen" />
            <lengthfunction id="intergene_pen" />
            <lengthfunction id="inital_exon_pen" />
            <lengthfunction id="internal_exon_pen" />
            <lengthfunction id="terminal_exon_pen" />
            <lengthfunction id="single_exon_gene_pen" />-->
        </declarations>

        <gff2gaze>
            <!-- Features -->
            <gfffeat feature="TSS" strand="+" source="Eponine">
                <feat id="tss"/>
            </gfffeat>

            <gfffeat feature="TSS" strand="-" source="Eponine">
                <feat id="tss_rev"/>
            </gfffeat>

            <!--gfffeat feature="TIS" strand="+" source="Eponine">
                <feat id="tis"/>
            </gfffeat>

            <gfffeat feature="TIS" strand="-" source="Eponine">
                <feat id="tis_rev"/-->
            </gfffeat>

            <gfffeat feature="5SS" strand="+" source="Eponine">
                <feat id="5ss"/>
            </gfffeat>

            <gfffeat feature="5SS" strand="-" source="Eponine">
                <feat id="5ss_rev"/>
```

```
        </gfffeat>

        <gfffeat feature="3SS" strand="+" source="Eponine">
            <feat id="3ss"/>
        </gfffeat>

        <gfffeat feature="3SS" strand="-" source="Eponine">
            <feat id="3ss_rev"/>
        </gfffeat>

        <!--gfffeat feature="TTS" strand="+" source="Eponine">
            <feat id="tts"/>
        </gfffeat>

        <gfffeat feature="TTS" strand="-" source="Eponine">
            <feat id="tts_rev"/-->
        </gfffeat>

        <gfffeat feature="POLYA" strand="+" source="Eponine">
            <feat id="polyA"/>
        </gfffeat>

        <gfffeat feature="POLYA" strand="-" source="Eponine">
            <feat id="polyA_rev"/>
        </gfffeat>
    </gff2gaze>

    <dna2gaze>
        <!--dnafeat pattern="tataaa">
            <feat id="tss" />
        </dnafeat>

        <dnafeat pattern="atg" score="0.001">
            <feat id="tis" />
        </dnafeat>

        <dnafeat pattern="taa" score="0.001">
            <feat id="tts" />
        </dnafeat>

        <dnafeat pattern="tag" score="0.001">
            <feat id="tts" />
        </dnafeat>

        <dnafeat pattern="tga" score="0.001">
            <feat id="tts" />
        </dnafeat>

        <dnafeat pattern="aataaa" score="0.001">
            <feat id="polyA" />
        </dnafeat>

        <dnafeat pattern="tttata">
            <feat id="tss_rev" />
        </dnafeat>
```

```
    <dnafeat pattern="cat" score="0.001">
        <feat id="tis_rev" />
    </dnafeat>

    <dnafeat pattern="tta" score="0.001">
        <feat id="tts_rev" />
    </dnafeat>

    <dnafeat pattern="cta" score="0.001">
        <feat id="tts_rev" />
    </dnafeat>

    <dnafeat pattern="tca" score="0.001">
        <feat id="tts_rev" />
    </dnafeat>

    <dnafeat pattern="tttatt" score="0.001">
        <feat id="polyA_rev" />
    </dnafeat-->

    <!--takedna id="5ss_1" st_off="0" en_off="1"/>
    <takedna id="3ss_1" st_off="1" en_off="-1"/>
    <takedna id="5ss_2" st_off="-1" en_off="1"/>
    <takedna id="3ss_2" st_off="1" en_off="0"/>
    <takedna id="5ss_1_rev" st_off="1" en_off="0"/>
    <takedna id="3ss_1_rev" st_off="-1" en_off="1"/>
    <takedna id="5ss_2_rev" st_off="1" en_off="-1"/>
    <takedna id="3ss_2_rev" st_off="0" en_off="1"/-->
</dna2gaze>

<model>
    <target id="END">
        <source id="BEGIN" out_feat="No_genes"/>
        <source id="polyA" out_feat="GEN_DNA" />
        <source id="tss_rev" out_feat="GEN_DNA"/>
    </target>

    <!--Forward strand gene-->
    <target id="tss">
        <source id="BEGIN" out_feat="GEN_DNA"/>
        <source id="polyA" mindis="1" out_feat="intergenic"/>
        <source id="tss_rev" mindis="1" out_feat="intergenic"/>
    </target>

    <!--target id="tis">
        <source id="tss" mindis="1" out_feat="5UTR" out_str="+"/>
    </target-->

    <target id="5ss">
        <!--killfeat id="tts"/-->
        <!--source id="tis" out_feat="inital_exon" mindis="3" maxdis= "10000" out_str="+"
    /-->
        <source id="tss" mindis="1" out_feat="initial_exon" out_str="+"/>
```

```
            <source id="3ss" out_feat="internal_exon"  mindis="6" maxdis= "10000"
        out_str="+" />
        </target>

        <target id="3ss">
            <source id="5ss" out_feat="intron"  mindis="6" out_str="+"/>
        </target>

        <!--target id="tts"-->
            <!--killfeat id="tts" /-->
            <!--source id="tis" out_feat="single_exon_gene" mindis="60" out_str="+"/-->
            <!--source id="3ss" out_feat="terminal_exon"  mindis="3" out_str="+"/>
        </target-->

        <target id="polyA">
            <!--source id="tts" out_feat="3UTR" mindis="1" out_str="+"/-->
            <source id="3ss" out_feat="terminal_exon"  mindis="3" out_str="+"/>
        </target>

        <!--Reverse strand gene-->

        <target id="polyA_rev">
            <source id="BEGIN" out_feat="GEN_DNA"/>
            <source id="polyA" out_feat="intergenic" mindis="1"/>
            <source id="tss_rev" out_feat="intergenic" mindis="1"/>
        </target>

        <!--target id="tts_rev">
            <source id="polyA_rev" out_feat="3UTR" mindis="1" out_str="-"/>
        </target-->

        <target id="3ss_rev">
            <!--killfeat id="tts_rev" phase="0"/-->
            <source id="polyA_rev" out_feat="terminal_exon" mindis="3" maxdis= "10000"
        out_str="-"/>
            <source id="5ss_rev" out_feat="internal_exon" mindis="6" maxdis= "10000"
        out_str="-"/>
        </target>

        <target id="5ss_rev">
            <source id="3ss_rev" out_feat="intron" mindis="6" out_str="-"/>
        </target>

        <!--target id="tis_rev"-->
            <!--killfeat id="tts_rev" phase="0"/-->
            <!--source id="tts_rev" out_feat="single_exon_gene" mindis="60" out_str="-"/-->
            <!--source id="5ss_rev" out_feat="initial_exon" mindis="3" out_str="-"/>
        </target-->

        <target id="tss_rev">
            <source id="5ss_rev" out_feat="initial_exon" mindis="1" out_str="-"/>
        </target>
    </model>
    <lengthfunctions>
        <!-- lengthfunc id="intron_pen" file="./tables/intron_penalty"/>
```

```
        <lengthfunc id="initial_exon_pen" file="./tables/exon_penalty.initial"/>
        <lengthfunc id="terminal_exon_pen" file="./tables/exon_penalty.terminal"/>
        <lengthfunc id="internal_exon_pen" file="./tables/exon_penalty.internal"/-->

        <!--lengthfunc id="single_exon_gene_pen">
            <point x="500" y ="0.001"/>
            <point x="20000" y="0.2"/>
        </lengthfunc-->

        <!--lengthfunc id="intergene_pen">
            <point x="200000" y ="0.01"/>
            <point x="200001" y="0.01"/>
        </lengthfunc -->
    </lengthfunctions>
</gaze>
```