Exploring the Role of Pfam Families in Protein Function

Senior Honors Thesis

submitted by

Daniel John Meyer,

In partial fulfillment of the requirements
for the degree of

Bachelor of Science

in

Computer Science

TUFTS UNIVERSITY

May 2018

ADVISOR: Lenore Cowen

Acknowledgments

Thank you to my advisor, Lenore Cowen, for guiding me through the crazy universe of proteins. Thank you to Donna Slonim for your support and insights. Thank you to the Tufts BCB group for all of your help and feedback. Thank you to my friends and family for your continued support. Without any of you, none of this would be possible.

Daniel John Meyer

TUFTS UNIVERSITY

May 2018

Exploring the Role of Pfam Families in Protein Function

Daniel John Meyer

ADVIS0R: Lenore Cowen

It is widely accepted that the structure of proteins determine their function. Many computational inference methods that act on proteins or sets of proteins rely on assigned functional labels from a popular ontology, often the Gene Ontology (GO). On the other hand, there is rich, easily obtained structural motif information that is captured for proteins using libraries of Hidden Markov Models (HMMs), such as Pfam. Because structure relates to function, and many of these HMM-based-motifs are actually signatures of 3-dimensional substructures within the overall protein fold, it would follow that annotation of these HMM domains in a protein structure should aid in inference of its correct GO functional labels. However, creating a useful Pfam to GO mapping remains a difficult endeavor, first, because it is certainly not a one-to-one mapping. Second, different Pfam-derived domains within a protein structure, either individually, or as a set, might yield different amounts of specificity in regards to the set of possible GO labels that are appropriate. Estimating the amount of specificity that a single, or set of, Pfam-derived domains gives, in regards to GO labeling, is confounded by the unequal representation and/or the lack of coverage of annotation in both domains across the protein universe.

We revisit issues of sequence patterns, diversity, and representation in the light of all the new data in current sequence databases. We have developed a suite of parsers and an Object-Relational Mapping using Python and SQLAlchemy to

represent selected information of proteins and families from the UniProt and Pfam databases respectively, while making it easy to access and reason about information stored in the graphical structures of the GO and Evidence Code Ontology (ECO). We use this framework to compare dcGO (Fang and Gough, 2013) and GODM (Alborzi et al., 2017), which are designed to optimize different tradeoffs for coverage versus false-positives.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

DNA and protein sequencing technologies have been improving since their inception, resulting in an exponential increase in sequence information being stored in databases. Over the same time frame, the rate at which scientists are able to experimentally validate the functions of proteins or the genes has remained fairly linear, despite some automation of experiments. The difference between these two rates has resulted in a disparity between sequence information and the number of proven functions of proteins, leaving biological databases filled with protein sequences that have unknown function. Given this large amount of sequence information and a smaller, but still substantial, amount of protein structural information, many efforts have been made to try to deduce the functions of proteins of unknown function given the information available.

One of these methods involves looking at evolutionarily conserved regions of a protein's sequence or structure for information that could provide insight into the functions of the protein. Proteins are often thought of as molecular machines, so we can think of deconstructing a protein into its parts according to the secondary or super-secondary parts of the protein. If we break down every protein of known function into such a bag of parts, we can try to look at correlation between the functions of proteins and the parts that they contain. As an analogy to this, suppose

that we had a bag of Lego blocks that represent secondary and super-secondary structures of proteins, and we were given this bag in a box with the name of the protein that could be built out of it and the functional terms associated with the protein that could be built out of these blocks. Now, if we look only at parts that are evolutionarily conserved, we limit our view to be a bit more focused, and hope that the fact that these parts are conserved are indicative of the parts either having influence over the overall function of the protein or its structural integrity.

1.1.1 Pfam

The Pfam database[F+15] allows us to do exactly this by defining 16,712 Hidden Markov Models (HMMs) constructed using the HMMER program [JEP10]. These HMMs are trained on the multiple sequence alignments of some proteins (only using sequences from a reference proteome in the UniProt database[F+15][Con16]) to known conserved regions of sequence and/or corresponding structure. Pfam calls these HMMs families and stores the decomposition of all protein sequences in the UniProt database[Con16].

Since Pfam allows us to decompose protein sequences into occurrences of Pfam families, we can get our bag of Legos for all 115,316,915 proteins in UniProt (557,275 of which have a known function), in comparison to the 139,717 proteins for which we have known structures found in the RCSB Protein Data Bank[KXdlC+06]. So, only about 0.12% of proteins with known sequence also have a solved structure, and about 0.48% of proteins with known sequence have at least one known function. Luckily, Pfam allows us to gain structural information about any protein for which we have a known sequence. Not all Pfam families are not constructed from structural motifs; however, those that are not should still be considered as indicative of structural motifs. The the amino acid sequence of a protein, referred to as its primary structure, has a direct mapping to how it will fold in 3-dimensional space. There are exceptions to this, where two proteins could have similar sequence,

yet dissimilar structure (and therefore function), but these cases are few[KK08]. Thus, for most Pfam families, especially those with rich functional information or high correlation to protein function, we assume that can talk about Pfam families and 3-dimensional folds or domains interchangeably. By this method, Pfam classifies proteins into families in a flat rather than hierarchical manner, in contrast to the popular SCOP[MBHC95] and CATH[PTS$^+$05] databases of structural domains. Pfam looks for anything that is determined to be a family, and does not worry about assigning types and subtypes to families (although organization into clans was introduced to Pfam in 2006[F$^+$06]). Furthermore, both SCOP and CATH deal with structural information alone, and thus are designed to classify proteins whose 3D structures have been fully determined, whereas Pfam's approach relies only on sequence information of a protein.

1.1.2 The Gene Ontology (GO)

When trying to predict protein function computationally it is necessary to encode protein function in a meaningful way. One sensible and widely accepted manner in which protein function is organized is using the Gene Ontology (GO). The GO is an ontology, meaning that it is a collection of meaningful terms related to each other in the structure of a directed acyclic graph (DAG). In language, nouns form an ontology, as everything can be considered a thing, an apple is a thing, and a macintosh is an apple and a computer. The GO contains three main root terms that define namespaces for the all of the terms in the DAG: Molecular Function, Biological Process, and Cellular Component. Each of these namespaces contains information that can provide different types of information about a given gene or protein. Terms in the Molecular Function namespace describe how a protein acts on a biochemical level. These might be of particular use to a biochemist trying to identify proteins in a metabolic pathway where many of the compounds are known, but the protein responsible for a particular step in the pathway may be unknown. Terms in the Biological Process namespace describe the role that a protein may play in a metabolic pathway, and are useful for predicting the function of a protein by

having an idea of what pathway it might affect. The Cellular Component namespace contains terms that provide known locations of a protein in the cell, and providing context that may help to infer the function of a protein or at least to help in trying to experimentally derive a protein's function.

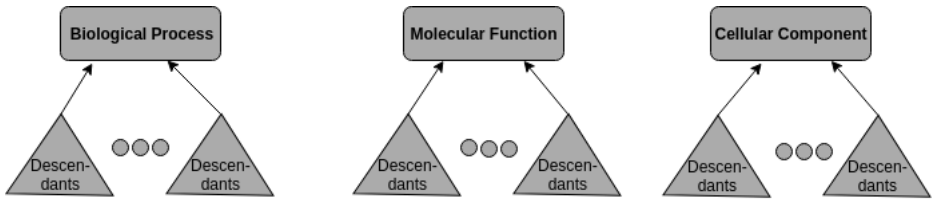


Figure 1.1: Directed Acyclic Graph (DAG) structure of the GO

Some Pfam families have known associations to function, and these are determined by a large effort being made by EMBL-EBI (The European Bioinformatics Institute) to generate manual annotations of domains in the InterPro database[CBB+03], which has a subset of entries that matches to Pfam. InterPro draws entries from many different databases of protein family classification based on conserved regions or domains. Unfortunately, at this point in time, the annotations of Pfam families from InterPro2GO are relatively sparse; however, we believe that there is something to be learned from annotated proteins that contain occurrences of Pfam families.

Our research focuses on understanding the relationship between protein families and the functions of proteins contained within each family, in the hope that we can construct rules that read: if a protein contains an instance of this family, then we are confident that its set of correct functional labels must come from a restricted set, which can be correlated to that Pfam family.

In order to analyze the intricate relationships between proteins, families, and GO terms, we mapped out the relationships between each of these entities and their various properties that should be considered when analyzing how they relate to each other. We then developed an Object-Relational Mapping (ORM) in Python, using the SQLAlchemy package, to create objects in code that could act as virtual

analogs for these entities that could be analyzed with ease, while also maintaining constraints for any entries stored in a corresponding MySQL database.

Our Object-Relational Mapping and the results found through our work open the opportunity to try to answer a large number of questions in a powerful way that abstracts the data and relationships of the entities involved into well-defined classes in code. This abstraction allows users to quickly write queries to the database, written in Python, that can help them to understand the relationships between these entities, and in our case, can help us to understand the relationship between Pfam families and GO terms. We hope that this model will provide a powerful tool for investigating the functions of proteins based on synthesizing information from the decomposition of proteins into Pfam families.

1.2 Past Work

The dcGO database was an early attempt to try to consider relating GO terms and protein domains/families, including SCOP domains and Pfam families. In particular, dcGO combines full-length protein-level annotations with the domain composition of the proteins to statistically infer the domains, or supra-domains (combination of evolutionarily-associated domains) associated with different GO terms[FG13]. To make this inference, dcGO creates a matrix with each row representing a GO term and each column representing a domain or supra-domain. Each entry in this matrix represents the number of UniProt proteins that contain the domain and are labeled with that GO term. Using this matrix, dcGO then uses Fisher's exact test to infer the correspondence between rows and columns, thus between GO terms and domains. The mapping created by dcGO allowed for the creation of a protein-function prediction tool, that could predict functions for proteins with no tested functional labels, but where the domain composition of the protein is known. This idea opens the door to labeling many proteins with previously unknown function, and further allows any protein with a known sequence or structure to be assigned functional labels if it matches any known domains with corresponding GO terms. For Pfam

specifically, this is possible by matching sequences to the Hidden Markov Models that define each family, and by mapping structures to Pfam families through the SIFTS mapping[VDJ+13]. There are two flavors of the dcGO data set, one that favors quality, only including annotations that comes from single-domain proteins of known function, and another that is focused on quantity, or rather coverage of the number of domains that could be annotated, and in turn the number of proteins that could be annotated[FG13].

Albborzi et al.[ADR17] were also interested in tying GO terms to protein domains, inspired by a desire to uncover many more annotations than were being found by existing tools that do this, in particular, dcGO[FG13]. The result of their research is a content-based filtering approach called GODM. Content-based filtering is a type of recommendation system, which tries to predict a list of items that correspond to a customer (in the analogy of online shopping). In particular, content-based filtering achieves this by identifying common attributes among the items that a consumer has previously purchased. Extending this to recommending Pfam families that could be associated with GO terms, the method would look at existing relationships containing a given GO term and the proteins annotated with that GO term, and would then examine the Pfam-domain composition of those proteins to make recommendations of which Pfam domains a GO term seemed to be correlated with. To get a confidence score in the annotations generated by this method, GODM verified each GO-Pfam association against individual GO-protein associations from several databases, including SIFTS, Swissprot, Swissprot-IEA, TrEMBL, and TrEMBL-IEA.

1.3 Motivation

Fully understanding relationships between Pfam conserved regions, member proteins with various functional annotations, and the protein sequences within a Pfam family could lead to insights that could open the doors to assigning latent functional information to proteins that were previously unannotated. This information could

then, even further, be bootstrapped to existing methods of protein function prediction, such as Protein-Protein-Interaction network methods, to potentially improve their performance by adding information where there was previously none.

By improving methods of automatic inference of these domains, we are able to generate functional labels for proteins that would normally not have priority to be studied in depth, and as a result, better automatic annotations could mean easier treatment of rare diseases as precision medicine gains traction.

Chapter 2

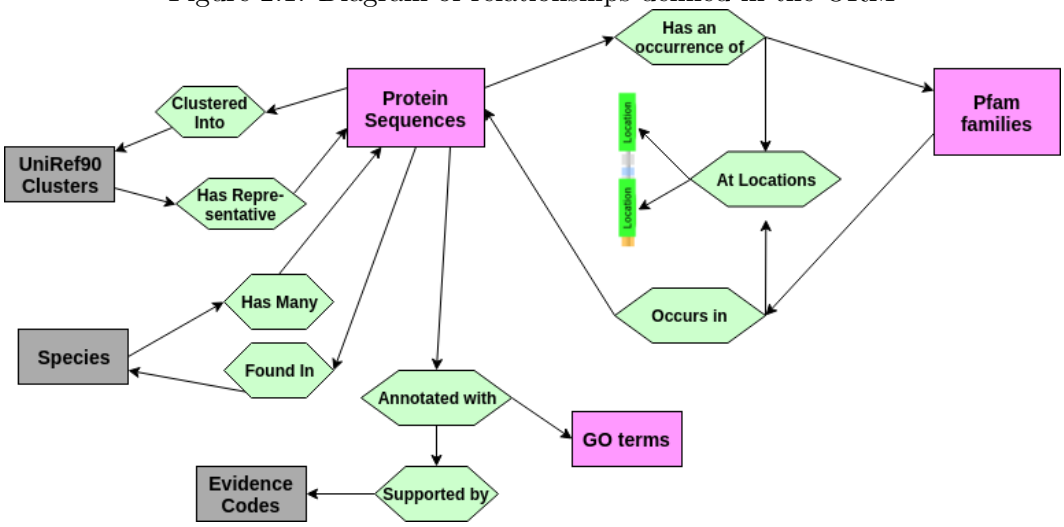
Methods

2.1 Defining Relationships

While trying to relate Pfam families, proteins, and GO terms, we found it difficult to keep track of the many critical relationships between each of the entities we were examining. To solve this problem, we first develop a way of thinking about the different entities and relationships of our problem space. We break down the problem space by distinguishing three main entities: proteins, Pfam families, and Gene Ontology labels (depicted in pink in figure 2.1). Then we can define relationships between the entities.

For any given protein p, p can be decomposed into a set of Pfam families, $p.families$. For a given family F, there is a set of proteins that contains at least one instance of this family, which we call $F.proteins$. Since a family can occur at multiple locations within a protein sequence, we can denote the list of occurrences of a family F in a protein p as $assoc(F, p).locs$, giving us the locations of each occurrence of F in p. Any given protein p also has associated with it GO terms, $p.GOterms$, which cumulatively represent the function of the protein p. Among these GO labels, there are relationships defined according to a rooted ontology (or rather a directed acyclic graph or DAG). Suppose p is labeled with only 1 GO term, φ_3, then if we go to the entry for φ_3 in the DAG of the GO, we see that φ_3 has directed edges to φ_1 and φ_2 with the label "is a" meaning that if p "is a" φ_3, then p is also a φ_1 and a

Figure 2.1: Diagram of relationships defined in the ORM



φ_2. So given that a protein is labeled with a set of GO terms $p.GOterms$, we also want to consider that the protein is associated with all ancestors of every term in $p.GOterms$. Thus we define $annot_p$ as the set of all terms which we associate with a protein p, which we restrict to a cut version of the tree, containing GO terms only of level 3 or below (>= 3 edges away from a root node):

$$annot_p = \left(set(p.GOterms) \cup \left[\bigcup_{\varphi \in p.GOterms} set(ancestors(\varphi)) \right] \right) - set(terms < level3) \tag{2.1}$$

2.2 The Database and Object-Relational Mapping (ORM)

To protect against human error and ease the process of writing code that works with data abiding to these various relationships, we decided to use a relational database with a corresponding Object-Relational model that could be used from within Python.

An Object-Relational Mapping (ORM) is more than just an API that allows us to interact with the database, as it defines classes in Python while also defining how they are mapped to underlying tables in the database and ensuring that the

relationships between those tables are maintained. So our ORM defines relationships and behaviors of these various entities and the result is a cleaner way of thinking of problems and a model that can be asked questions in code that is not too far from how you may ask them in English.

Although many databases containing all this data already exist, there were numerous drawbacks to each of them. The web APIs that they provide often limit the number of queries we were able to make at a time, setting up these databases locally was usually nontrivial, and even if we did manage to set up a database locally, the schemas would take some time to decipher and we would need to write our own ORM in Python that follows the relationships and behaviors of every entity in whatever database we chose to use.

After considering these factors we decided to create our own MySQL database using data files provided by Pfam, UniProt, and Gene Ontology respectively, accompanied by an ORM written in Python using SQLAlchemy. This allowed us to make a relatively simple model that was well suited to our task and gave us full control to make any changes to the database schema and to ensure that we had the most up-to-date datasets when gathering our results. Using our own dataset further resulted in the creation of two parsers written in Python for Gene Ontology and for Gene Ontology Annotation files, that we plan to make publicly available after completing this work.

To design this ORM and corresponding database, we started by figuring out what the different entities were and what relationships they had with each other. The model contains the following classes: `Protein`, `Family`, `Species`, `GO`, `Cluster`, `Comment`, `ProteinFamily`, `Location`, `Protein2GO`, `ECO`, and `Protein2Species`. Each of these classes defined in the ORM has a corresponding table in the database: `protein`, `family`, `go`, `cluster`, `comment`, `protein_family`, `location`, `protein_go`, `eco`, and `protein_species` respectively.

The `Protein2Family` and `Protein2GO` classes not only serves as join tables between proteins and families and between proteins and GO terms, but, more precisely, they represent the association between a protein and a family and the associa-

tion between a protein and a GO term respectfully. Instances of the `Protein2Family` class are contained within the lists `Protein.families` and `Family.proteins`, and instances of the `Protein2GO` class are stored in the lists `Protein.GOterms` and `GO.proteins` (N.B. This differs from the other many-to-many relationships defined by `Protein2Species`, as `Protein.species` contains instances of `Species`, and `Species.proteins` contains instances of `Protein`).

This is an important point and was also necessary to define the relationships between proteins, families, and the location of the conserved regions associated with a given family within a given protein sequence within our ORM, as one conserved region can appear multiple times in different locations within the same protein sequence, and our model needed to reflect this.

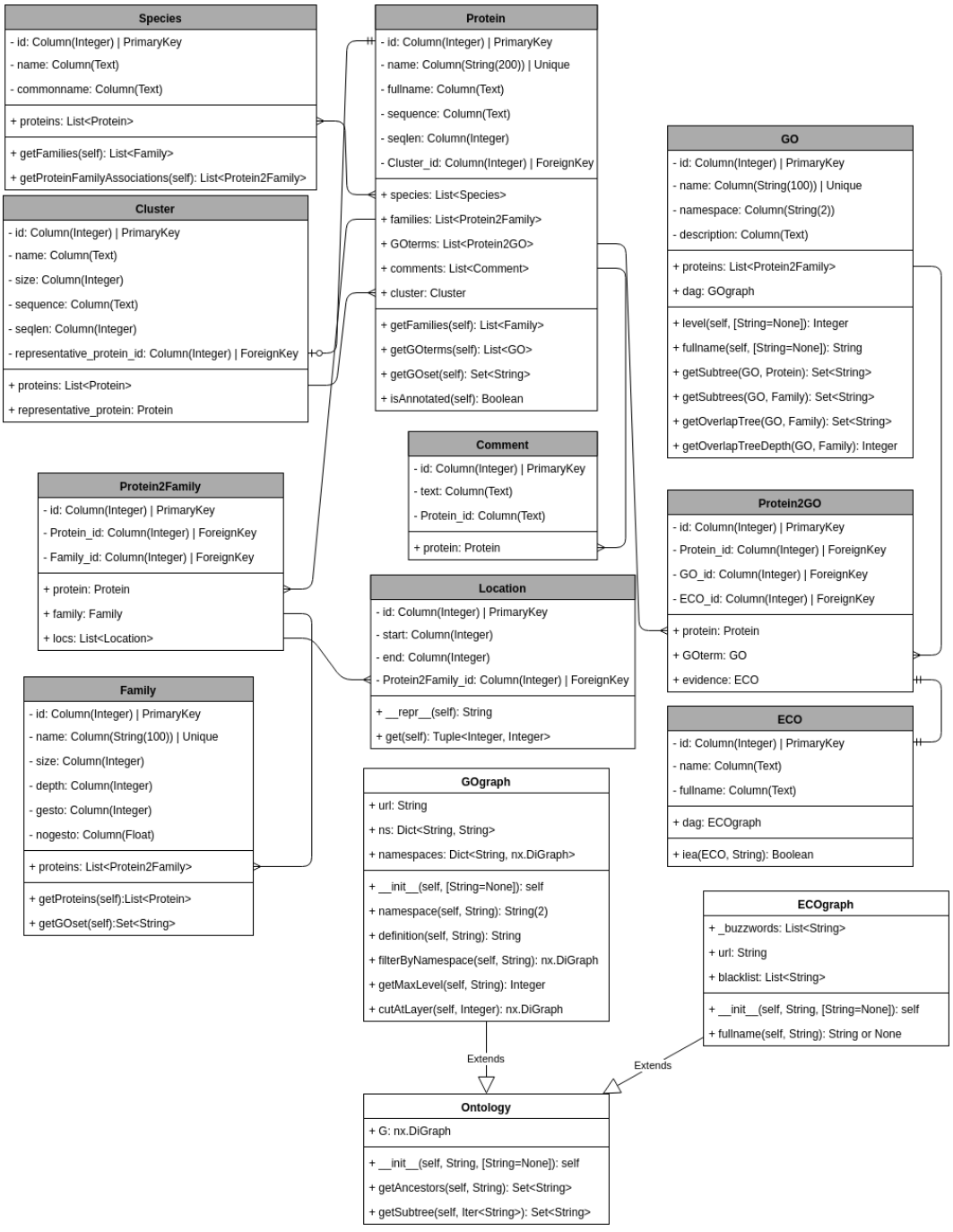


Figure 2.2: Design of ORM classes and relationships between entities

To really grasp just how useful and expressive an ORM can be, let us walk through an example of a function. Suppose that we want to find the names of all proteins found in humans that have at least one conserved region that appears exactly twice in its sequence, then we can write in code:

```
from schema import Session, Protein, Species, Protein2Family


def doubles():
  '''
     Generates names of all proteins found in humans
     that have at least one Pfam motif that appears
     exactly twice in its sequence
  '''

  s = Session()
  human = s.query(Species).filter_by(name='Homo
sapiens').first()
     Gets the species object with name "Homo sapiens" '''

  for protein in human.proteins:
    '''
       For each protein sequence, look for associated
       families that have exactly two occurrences
       in the sequence.
    '''
    for family_association in protein.families:
      if family_association.locs.count() is 2:
        yield protein.name
        '''
           Found a protein that meets criteria,
           continue to next protein sequence
        '''
        break
  s.close()

double_family_proteins = list(doubles())
''' Use the function to generate our list of protein names '''
```

Figure 2.3: Example code written using the ORM

Walking through this function, we create a new session, which is needed to interact with the database, then we find the entry in the species table that represents a

human, then iterate through each protein object in the set of all proteins found within humans, and for each of those proteins we then iterate over all Pfam families that it is a member of and add the protein to a list if any conserved region associated with a family appears in the protein sequence in exactly two locations.

Notice again that we say `for family_association in protein.families` instead of `for family in protein.families` since each element of the list `protein.families` is an instance of the class `Protein2Family`, not of Family, and will thus have different properties. However, we do say `for protein in human.proteins`, since `human.proteins` is of the type `Species.proteins`, which is a list of objects of the class `Protein`.

Since entities in the ORM are defined as Python classes, we are also able to include properties of the data that existed outside of the relational database as well as methods associated with each of these entities that can help to write more simplified and reusable code. The most powerful example of how we were able to take advantage of this was by creating a class-level object representing the DAG of GO as a graph using the networkx package. This allowed us to have rapid access to the graph structure of GO, while still having a bridge between each GO term and any other entities in the database. This was particularly feasible because of the relatively small size of Gene Ontology, and in fact the model pulls the most up-to-date data file from Gene Ontology's website each time it is run, and the impact on runtimes is small enough that the tradeoff is worthwhile.

2.3 Measurements

There is much variety among Pfam families and the associated set of proteins that contain at least one occurrence of the Pfam family, which we call member proteins of the Pfam family. Some properties of Pfam families include diversity of member proteins, the sizes of each occurrence, the number of occurrences per protein, similarity in the decomposition of member proteins, functional similarity between members, and quality of annotations of members.

We used our Object-Relational Mapping to analyze these properties of Pfam families and the sets of proteins that contain them in order to develop a definition of what we consider to be a functionally informative family. The first of these properties that we consider include the sequence diversity of the member proteins for a given family and the spread of GO terms associated with these proteins. We define the spread of GO terms as the dissimilarity of GO terms associated with a given set of proteins, with a tight spread giving strong correlation of proteins in the set to a specific subset of GO terms.

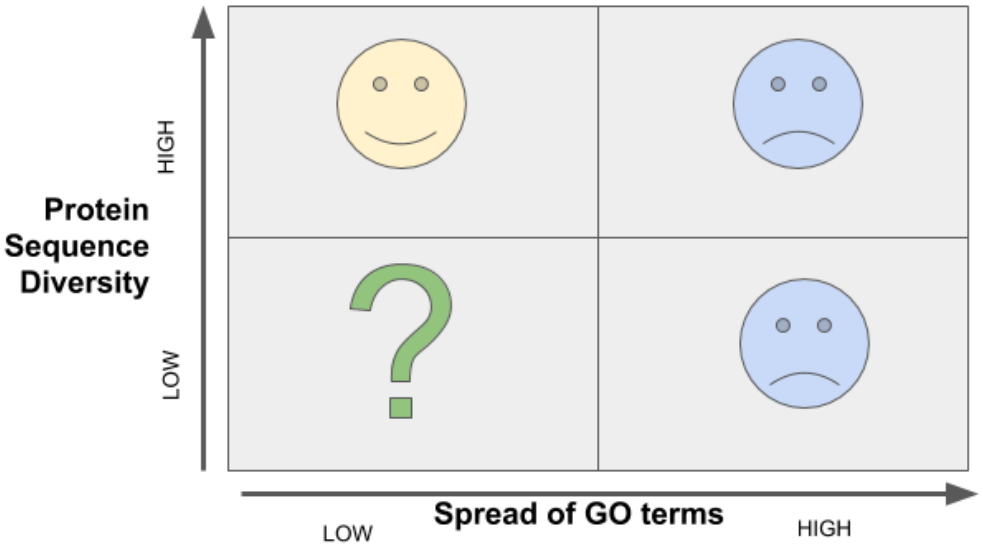


Figure 2.4: Classification of Informative Families

We consider families that fall into the top left quadrant of figure 2.4 to be functionally informative. For these families, the GO terms associated to each member protein all fall within a close measure of similarity to each other and the set of member proteins is adequately diverse enough that we can infer that the family appears in a diverse enough set proteins that the cause of the correlation between the set of proteins and function seems to be the due to the conserved region(s), detected by Pfam, that all of these proteins have in common. If the spread of GO terms is too high, we have a low correlation between a protein containing a family and having a particular GO term, thus we cannot consider any families with high spread of

Figure 2.5: Classification of Homologous Informative Families

GO terms to be functionally informative. For Pfam families where the member proteins have low diversity, which we call homologous, and have a low spread of GO terms, we find that we need to investigate further. There are two cases of Pfam families that can be considered homologous: the first is Pfam families where a single occurrence takes up most of the protein sequence for every member protein of the Pfam family (left half of figure 2.5), and the second is Pfam families where the Pfam family does not cover most of every member protein's sequence, but the sequences are all homologous (right half of figure 2.5). This second case is inconclusive as to whether or not we should consider a family informative to function, as we have no evidence that if another sequence matched to this family, but was otherwise diverse in sequence from all other member proteins, that it would have the same functional labels as the other member proteins.

So, if a Pfam family has a tight spread of associated GO terms and a homologous set of member proteins, we only consider it to be informative if there is a high percentage of every member protein's sequence covered by a single occurrence of the Pfam family.

2.3.1 Measuring Diversity of Pfam Family Members

In order to measure sequence diversity, we sought to cluster homologous protein sequences within a given family, and then count the number of clusters of homologous

sequences that we found. Our approach to this involved removing regions of the proteins that align well within the region(s) that determine their membership in the given family, and then comparing sequence composition after removing this Pfam-family-specific homology. Seeking to limit the number of redundant sequences as much as possible, we use the UniRef90[SWH+15] data set from UniProt to reduce the Swiss-Prot proteins associated with each family to only contain one representative sequence per 90% identity UniRef90 cluster, ensuring that no two sequences within the set for a given family will be 90% or more identical.

First, we perform a multiple sequence alignment of all representative protein sequences in a given family, using MAFFT[KS13], then we remove all columns from the alignment where all residues in the column are identical and all residues in the column fall within the bounds of an occurrence of the Pfam conserved region. We then stitch these cut-up sequences back together and calculate the pairwise similarity, using blastp[A+97] between every two proteins in the family, creating a similarity matrix representing homology of the family. In order to keep runtimes down, for families with more than 50 genes we took a random sampling of 50 proteins to run this procedure on. By creating a sequence similarity matrix, we could then imagine a Pfam family as an undirected weighted graph of sequence similarity between member proteins. This graphical view allowed us to visualize clusters of similar sequences within a given family by removing edges below a certain weight (i.e. $< 80\%$ similarity) and counting the number of connected components. If the number of connected components in the graph after removing all edges is still 1, then we can say that all proteins are likely within the same group of homologous sequences (note that this does not mean that each protein in the family is $\geq 80\%$ similar to every other protein).

We then wanted establish a normalized measure of diversity that accounted for not only this ratio of connected components to total number of nodes in the graph, but also that rewarded more for large and diverse families, than for small and diverse families. For a given family, f_i, with c_i connected components and n_i proteins in its sequence similarity matrix (note that this will be capped at $m = 50$),

we define

$$Diversity(f_i) = \frac{c_i}{n_i} \log_m c_i \tag{2.2}$$

Note that we run our measure of diversity only on member proteins that are contained in the Swiss-Prot dataset, excluding any sequences in the TrEMBL dataset. The result of this is that our diversity score also penalizes against Pfam families for which few of the proteins have known function and verified sequences, and also penalizes against Pfam families for which the sequences in Swiss-Prot are homologous, even if there are matches in TrEMBL that would provide evidence of the family being diverse. The rationale for this was that this would also help us in ensuring that when we have a family that scores well on our diversity metric and has an annotation with high coverage, we could consider that protein to be functionally informative.

2.3.2 Measuring the Spread of GO terms among Pfam Family Members

If we want to measure the correlation of a Pfam family F to a given GO term φ, we would divide the number of proteins in the family labeled with that term by the number of proteins in the family, giving us the proportion of terms in a family labeled with term φ.

$$P(p \text{ labeled with } g | p \in F) = P(p \leftarrow \varphi | p \in F) = \sum_{i \in |F|} y_i \qquad y_i = \begin{cases} 1 \text{ if } p_i \text{ labeled with } \varphi \\ 0 \text{ otherwise} \end{cases} \tag{2.3}$$

Since GO is an ontology, we know that if a protein p is labeled with term φ (ensured by methods in our ORM), then, by the definition of an ontology, it is understood that p can be correctly labeled with every ancestor term of φ. Also, in most cases, we have evidence that a protein p has multiple GO annotations,

$\varphi_1, \cdots, \varphi_n$, and would then also be annotated with each term in the set of the ancestors of $\varphi_1, \cdots, \varphi_n$. We call the annotation set of p, $annot_p$, the set containing $\varphi_1, \cdots, \varphi_n$ and ancestors of $\varphi_1, \cdots, \varphi_n$.

If we extend the idea of correlation between a single family-GO term pair to the correlation of a set of GO terms to a set of genes or proteins, as we do when we consider all members of a Pfam family, then we would want to calculate the probabilities

$$P(p \leftarrow \varphi_i | p \in F) \forall \varphi_i \in \bigcup_{p_i \in F} annot_{p_i} \tag{2.4}$$

If we only consider all of these terms φ_i where the probability of an arbitrary protein in F being labeled with φ_i is 1, we can calculate this set as:

$$overlap_F = \bigcap_{p_i \in F} annot_{p_i} \tag{2.5}$$

This overlap set gives us a prediction of terms that could be correlated to a given family; however, we also wanted to measure how representative this prediction was of the family.

To measure the annotation diversity across a given Pfam family, we used a generalized form of the Term Overlap (TO) score presented by Mistry and Pavlidis [MP08].

We call this new metric as Gene Set Term Overlap (GeSTO), and its derivation from TO is quite trivial.

TO is defined by Mistry and Pavlidis as:

$$sim_{TO}(p_1, p_2) = |annot_{p_1} \cap annot_{p_2}| \tag{2.6}$$

Asking the question: how many functional annotations do genes p_1 and p_2 share?

Our generalized metric asks the question: how many functional annotations do genes $p_1, \cdots, p_n$ have in common? This is equivalent to calculating the size of the overlap set of these genes. When dealing with a set of genes that may not all be

adequately annotated, we further ensure that if the annotation set of a given gene is empty, we ignore it in the computation of our GeSTO score, as not to build up more Pfam families with a GeSTO score of 0.

We define GeSTO as

$$sim_{TO}(p_1, \ldots, p_n) = \left| \bigcap_{i=1}^{n} annot_{p_i} || annot_{p_i} | > 0 \right| \tag{2.7}$$

The normalized counterpart to GeSTO, NoGeSTO, which derives from the Normalized Term Overlap score defined by Mistry and Paylidis, would then be

$$sim_{NTO} = \frac{sim_{TO}(p_1, \ldots, p_n)}{\min_i(|annot_{p_i}| > 0)} \tag{2.8}$$

NoGeSTO is then the size of the overlap set in proportion to the most sparsely annotated annotation set of any given protein in the family. In other words, NoGeSTO is measuring how well the overlap set represents the most sparsely annotated protein. To supplement this, we also measure overlap coverage, to measure how well the overlap set represents the entire family, or gene set.

We define overlap coverage as:

$$coverage(overlap_F) = \frac{|overlap_F|}{|\bigcup_{p_i \in F} annot_{p_i}|} \tag{2.9}$$

So we can now score the assignment of a set of GO terms to a given family by how well it can represent any protein in the family and additionally, how well it covers the functions of all other proteins in the family.

To further ensure that we are obtaining only high-quality annotations, we add a constraint on how we define $annot_p$. We build $annot_p$ by ignoring any found labeling of p with function φ where there is no manual evidence that p should have label φ, but rather, the evidence code found alongside the association of protein p with label φ conveys that this association was determined by computational inference, rather than by experimental evidence or manually-curated inference.

UniProt provides evidence codes alongside each association of a GO term

with a protein, and each of these evidence codes are part of an ontology called the Evidence Code Ontology, which has the same directed acyclic graph structure as GO. We leverage this structure when determining whether or not a protein-term association is automatic by searching for term names that contain one of a few key words that may infer that the association was determined computationally. These 'blacklist' words include 'auto', 'high', 'throughput', 'computational', and 'not_recorded'. Though crude, this method enables us to be selective about which annotations we include within the annotation set for a given protein, ensuring that the overlap set that we calculate for each family is as meaningful as possible.

Chapter 3

Results and Analysis

3.1 Results and Analysis

First, we break down sequences by diversity. Based on the distribution of the diversity score (see figure 3.1), we visually classify all Pfam families with a score greater than or equal to 0.5 to be diverse and less than or equal to 0.2 to be homologous. Any Pfam families where the sequences of all proteins that contain the family are 80% more composed of one or more occurrences of that Pfam family (these Pfam families are assigned a diversity score of 0 for purposes of classification).

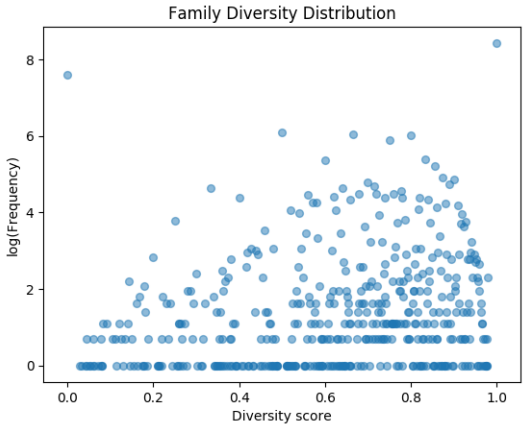


Figure 3.1: Distribution of Diversity Scores

We then break down all 16,712 Pfam families according to 4 categories: Diverse, Homologous Informative, Homologous Uninformative, and Other (as seen in figure

3.2). We found that 44.3% of Pfam families (Homologous Low Sequence Coverage + Others) were deemed uninformative to function by this measure of diversity alone, leaving 9,314 informative families.

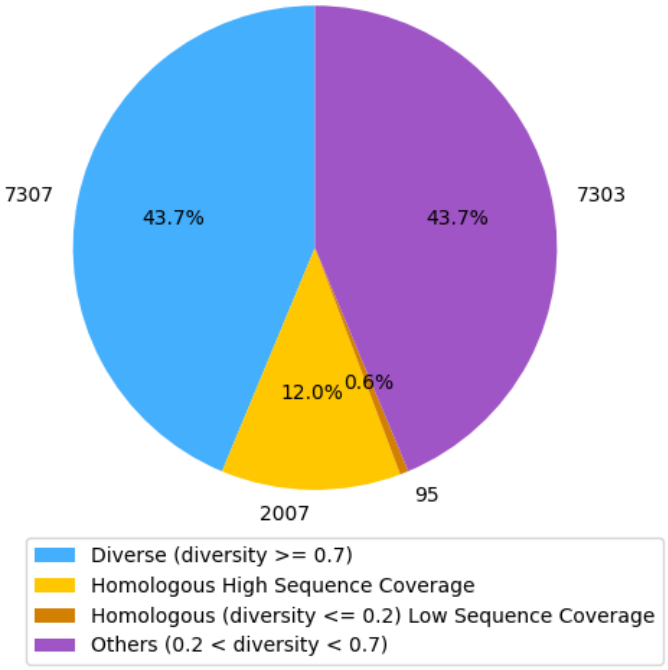


Figure 3.2: Categorization of Pfam Families by Sequence Diversity

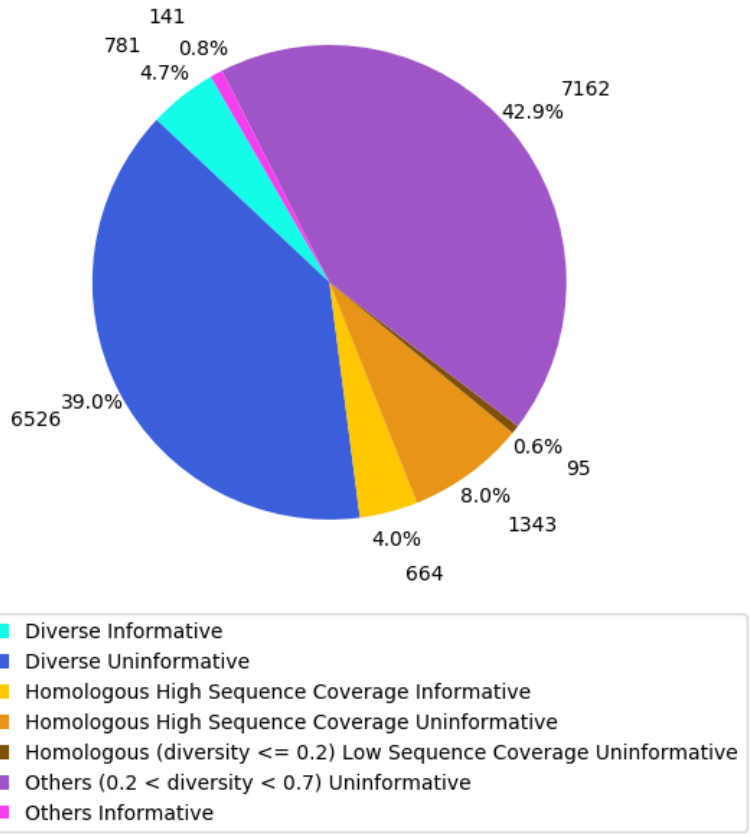


Figure 3.3: Final categorization of Pfam families using overlap annotations (our method). Note that we consider the family "Diverse" in the legend if the family scores >= 0.7 by our diversity measure (See equation 2.2), "Homologous" if the family scores <= 0.2, and "Other" if our diversity measure scores between 0.2 and 0.7.

Next, we measure the spread of GO terms that we want to assign to a given Pfam family. To do this, we need to consider different methods of assigning GO terms to Pfam families. We compare the annotations defined manually in InterPro2GO against those derived by dcGO, GODM, and by our measure of taking the overlap set of member proteins of a Pfam family (being the intersection of all GO terms and their ancestor trees associated with each member protein). We can think of the two measurements that we defined as measuring precision and accuracy. The NoGeSTO score that we define can be thought of as how precisely can we predict the function of a member protein given the annotation set of this particular Pfam

family. The coverage score that we define can be thought of as accuracy, measuring the ratio between possible functions of member proteins and common functions between member proteins. By a combination intuition and trial and error, we set thresholds for these values both at 0.7 or higher and obtain the results seen in figure 3.4 for families considered to be diverse informative and in figure 3.5 for families considered to be homologous informative.

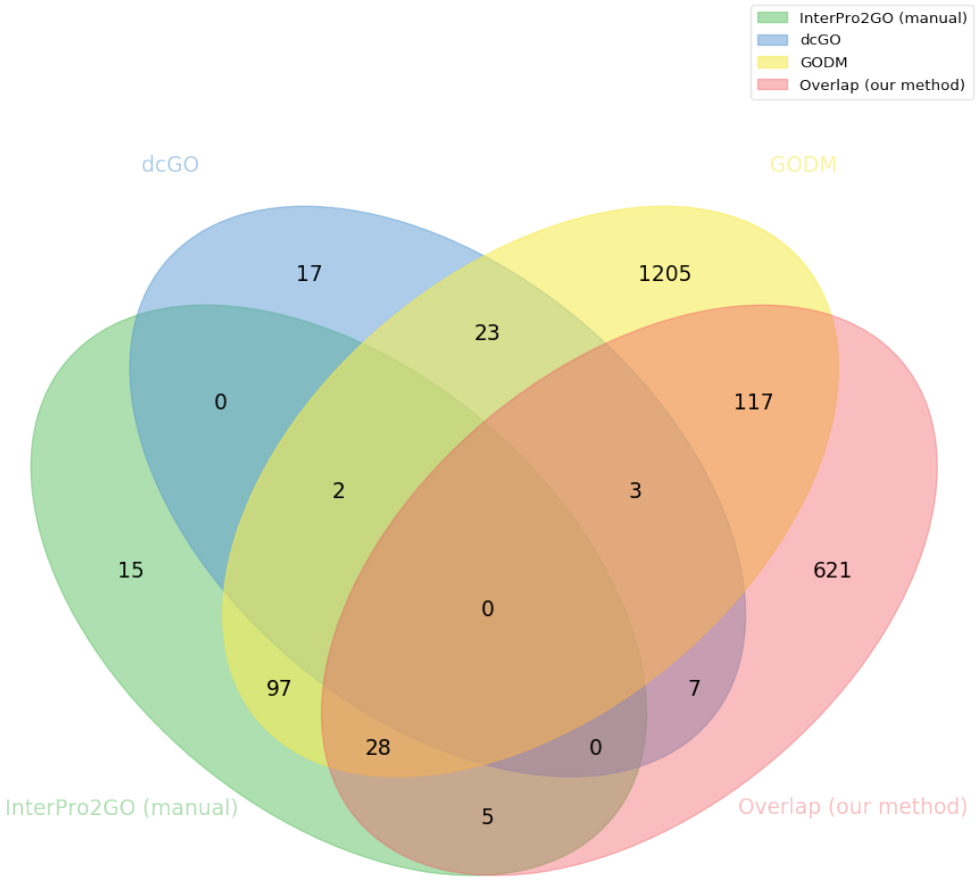


Figure 3.4: Venn diagram showing how the 781 diverse informative families found by our method (light blue in figure 3.3) compared to diverse informative families found by other Pfam-family GO annotations.

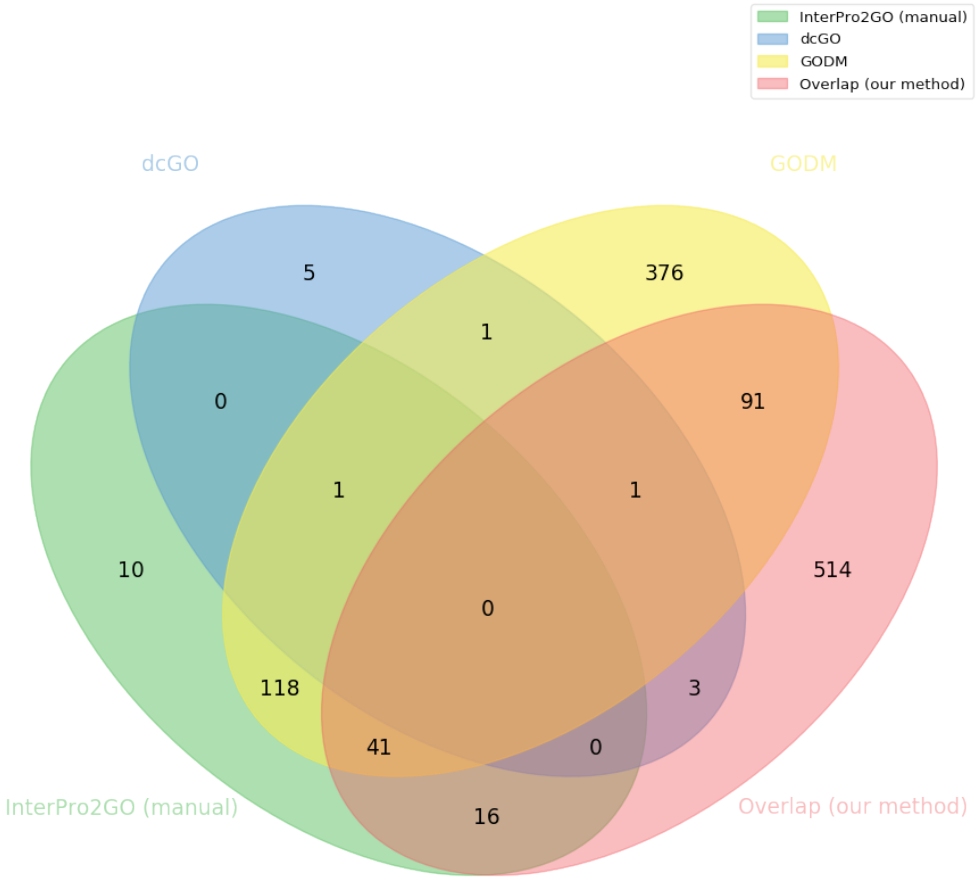


Figure 3.5: Venn diagram showing how the 664 homologous informative families found by our method (light orange in figure 3.3) compared to homologous informative families found by other Pfam-family GO annotations.

We notice that there is not much overlap between every of these methods, but it appears that our method of taking the overlap set of GO terms associated with member proteins of families detects more of the same informative families as the manual annotation set than any other metric we compare to, and additionally, that GODM finds a respectable number of informative families that are not detected by our method. The disconnect between these methods lead us to believe that taking an approach of using multiple methods of determining Pfam-family-level annotation

sets and choosing the best annotation set from these various methods will yield the best results when trying to use Pfam family level annotations for predicting protein function.

Chapter 4

Conclusion and Future Work

We have presented a framework in which users can write readable Python code, using our Object-Relational Mapping, that preserves the complex relationships between the various entities that need to be considered when relating Pfam families to protein function. Further, we have shown how this framework can be used to make meaningful insights about these entities, as well as to retrieve any of these properties and their relationships efficiently from the database in a manner that could be easily incorporated into a protein function prediction implementation written in Python, or that could be used to generate input for implementations not written in Python.

We also provide an analysis of various annotation mappings, showing how well they each represent the sets of proteins that each Pfam family contains. We found that across these different Pfam-family to GO term mappings, the sets of informative families found by each has little overlap. This opens up a new set of questions about properties of these mappings and properties of the families that each finds exclusively. For the time being, we suggest that integration of multiple mappings to generate an aggregate mapping for informative families would be the best solution to finding the most Pfam-family-level annotations.

There are many paths that could extend from this research. The first of which would be to test the performance of existing methods of protein function prediction after incorporating these different family-level annotation sets both blindly and for families above different thresholds of the measurements discussed. One could also

extend the ORM to make queries of the databases without needing to reconstruct them locally. This would involve the use of the UniProt SPARQL Endpoint, which allows for SQL-like queries to be made to the publicly available instance of the UniProt database, and the SIFTS subset of the PDBe REST API to retrieve Pfam entries. By retaining the model that we define, while removing the need for a separate database, this framework becomes far more usable, and ensures that the most up-to-date information is being retrieved.

Another path could be to look more closely at proteins in the TrEMBL dataset that match to Pfam families. This could help to increase the number of diverse families detected, and thus give us more families for which we could annotate. Different HMM-derived families could also be incorporated, and structural mappings to families through the SIFTS database could also provide more evidence of correlation between domain and function.

Bibliography

[A+97] SF Altschul et al. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[ADR17] Seyed Ziaeddin Alborzi, Marie-Dominique Devignes, and David Ritchie. Associating gene ontology terms with pfam protein domains. In *Lecture Notes in Computer Science*, pages 127–138. Bioinformatics and Biomedical Engineering, Springer, April 2017.

[CBB+03] Evelyn Camon, Daniel Barrell, Catherine Brooksbank, Michele Magrane, and Rolf Apweiler. The gene ontology annotation (goa) projectapplication of go in swiss-prot, trembl and interpro. *Comparative and Functional Genomics*, 4(1):71–74, 2003.

[Con16] UniProt Consortium. Uniprot: the universal protein knowledgebase. *Nucleic acids research*, 45(D1):D158–D169, 2016.

[F+06] Robert D. Finn et al. Pfam: clans, web tools and services. *Nucleic Acids Research*, 34:D247–D251, 2006.

[F+15] Robert D. Finn et al. The pfam protein families database: towards a more sustainable future. *Nucleic Acids Research*, 44:D279–D285, 2015.

[FG13] Hai Fang and Julian Gough. dcgo: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Research*, 41:D536–D544, 2013.

[JEP10] L Steven Johnson, Sean R Eddy, and Elon Portugaly. Hidden markov model speed heuristic and iterative hmm search procedure. *BMC Bioinformatics*, 11:431), 2010.

[KK08] Mickey Kosloff and Rachel Kolodny. Sequence-similar, structure-dissimilar protein pairs in the pdb. *Proteins: Structure, Function, and Bioinformatics*, 71(2):891–902, 2008.

[KS13] Kazutaka Katoh and Daron M Standley. Mafft multiple sequence alignment software version 7: improvements in performance and usability. *Molecular biology and evolution*, 30(4):772–780, 2013.

[KXdlC+06] Andrei Kouranov, Lei Xie, Joanna de la Cruz, Li Chen, John Westbrook, Philip E Bourne, and Helen M Berman. The rcsb pdb information portal for structural genomics. *Nucleic acids research*, 34(suppl_1):D302–D305, 2006.

[MBHC95] Alexey G Murzin, Steven E Brenner, Tim Hubbard, and Cyrus Chothia. Scop: a structural classification of proteins database for the investigation of sequences and structures. *Journal of molecular biology*, 247(4):536–540, 1995.

[MP08] Meeta Mistry and Paul Pavlidis. Gene ontology term overlap as a measure of gene functional similarity. *BMC Bioinformatics*, 9:327, 2008.

[PTS+05] Frances Pearl, Annabel Todd, Ian Sillitoe, Mark Dibley, Oliver Redfern, Tony Lewis, Christopher Bennett, Russell Marsden, Alistair Grant, David Lee, et al. The cath domain structure database and related resources gene3d and dhs provide comprehensive domain family information for genome analysis. *Nucleic acids research*, 33(suppl_1):D247–D251, 2005.

[SWH+15] Baris E. Suzek, Yuqi Wang, Hongzhan Huang, Peter B. McGarvey, Cathy H. Wu, and the UniProt Consortium. Uniref clusters: a comprehensive and scalable alternative for improving sequence similarity searches. *Bioinformatics*, 31(6):926–932, 2015.

[VDJ+13] Sameer Velankar, Jos M. Dana, Julius Jacobsen, Glen van Ginkel, Paul J. Gane, Jie Luo, Thomas J. Oldfield, Claire ODonovan, Maria-Jesus Martin, and Gerard J. Kleywegt. Sifts: Structure integration with function, taxonomy and sequences resource. *Nucleic Acids Research*, 41(D1):D483–D489, 2013.