# Data-driven Methods and Models for Predicting Protein Structure using Dynamic Fragments and Rotamers

Steven J. Rysavy

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington

2014

Reading Committee:

Valerie Daggett, Chair

James Brinkley

Ira Kalet

Program Authorized to Offer Degree:
Biomedical Informatics and Medical Education

University of Washington

# Abstract

Data-driven Methods and Models for Predicting Protein Loop Structure using Dynamic Fragments and Rotamers

Steven J. Rysavy

Chair of the Supervisory Committee:
Professor Valerie Daggett
Bioengineering

Proteins play critical roles in cellular processes. A protein's conformation directly relates to its biological function and, consequently, determination of such structure can provide great insight into a protein's function. Using a computational technique called molecular dynamics (MD), we are able to simulate and observe protein dynamics at a much higher temporal and spatial resolution than allowed by experimental methods. Dynameomics is a research endeavor that uses MD to produce thousands of protein simulations, resulting in hundreds of terabytes of data. Using novel visual analytics techniques, we have mined the Dynameomics data warehouse for data on protein backbone segments and side-chain behavior, called fragments and rotamers, respectively. Knowledge derived from these dynamic fragments and rotamers was used to improve the quality of protein loop structure predictions. We have created novel data models to store, analyze and compare fragments and side-chain rotamers, then developed methods to predict loop structures

with information inferred from these data models. Protein loop regions predicted from these fragments and rotamers produce biologically relevant structures that improve upon current protein loop prediction methods. In conjunction with the fragment and rotamer research, we produced a novel visual analytics framework called DIVE, a Data Intensive Visualization Engine. This software has been instrumental in advancing our bioinformatics research, but it is a general-purpose framework applicable to a wide range of big data problems.

## TABLE OF CONTENTS

## LIST OF FIGURES

# LIST OF TABLES

# ACKNOWLEDGEMENTS

I would like to thank Dr. Valerie Daggett for the mentorship I received throughout my graduate studies. I would also like to thank Drs. Jim Brinkley, Ira Kalet, and Jim Pfaendtner for serving on my committee and providing advice throughout the dissertation process. Thanks goes to the many members of the Daggett lab past and present, including Drs. David Beck, Noah Benson, Gene Hopping, Amanda Jonsson, Michelle McCully, Erik Merkley, Dustin Schaeffer, Tom Schmidlin, Alex Scouras, Andrew Simms, Rudesh Toofanny, and Clare Towse as well as Denny Bromley, Jonathan Cheng, Peter Law, Sarah Nowakowski, and Robert Su. I would also like to thank my cohort including Drs. Wynona Black, Daniel Capurro, Walter Curioso, and Rupa Patel as well as Melissa Clarkson. I am also grateful for the financial support provided by the National Library of Medicine and the National Institute of Health and administrative support provided by the Division of Biomedical and Health Informatics and the Department of Bioengineering.

# DEDICATION

I would like to thank my wife, Erin Cipolla, for the unending support and encouragement as well as the constant reminders to get away from the computer and enjoy life.  I would also like to thank my dog Kai for bringing me a ball to play with during the tough times and my dog Benny for always keeping his tail wagging.

# Chapter 1

# PROTEIN STRUCTURE PREDICTION

Proteins play a critical role in nearly every cellular process, including catalyzing chemical reactions, transporting molecules, responding to environmental signals, and providing structural support (Fetrow, 1995; Leszczynski & Rose, 1986; Smock & Gierasch, 2009; Wu & Dean, 1996). A protein's structure directly relates to its biological function and, consequently, determination of a protein's structure can provide great insight into a protein's purpose (Branden & Tooze, 1991). Current experimental methods for determining protein structures have several drawbacks. They cannot determine structures for all proteins, often omitting certain regions within a protein due to indeterminate data, and introduce experimental artifacts (Eicken et al., 2002; Wagner, Hyberts, & Havel, 1992). Furthermore, these methods are expensive and time-consuming, resulting in low overall sampling for certain aspects of the data such as the conformations of side-chains (rotamers) and flexible loops. Experimental methods are also extremely limited in extracting dynamic structural information, which can be critical to understanding a protein's function (Glazer, Radmer, & Altman, 2009; Karplus & Kuriyan, 2005).

Computational structure prediction is an evolving methodology that addresses short-comings of experimental methods (Kryshtafovych et al., 2013). Fragment-based structure prediction is one such computational approach that uses small peptides derived from known protein structures to model backbone conformations of unknown structures (Baeten et al., 2008; Choi & Deane, 2010; Verschueren et al., 2011). Backbone prediction methods are typically combined with rotamer libraries, which provide side-chain conformations and propensities, to refine the overall structure prediction (Dunbrack & Cohen, 1997; Lovell, Word, Richardson, &

Richardson, 2000; Shapovalov & Dunbrack Jr., 2011). Both of these methods rely heavily on pre-determined structures, typically a subset of the best X-ray crystallography structures in the Protein Data Bank (PDB) (Frances C. Bernstein et al., 1977). However, as mentioned, PDB structures can be incomplete, incorrect, poorly sampled, and bereft of dynamic information. Additional structural information is essential to improve state-of-the-art prediction methods and new approaches are needed to access this information.

I propose that molecular dynamic (MD) protein simulations provide a diverse sampling of biologically relevant protein structures that can improve the quality of structure predictions, and I use Dynameomics (Beck et al., 2008; Day, Beck, Armen, & Daggett, 2003) to test this hypothesis. The Dynameomics project has produced a repository of MD simulations that includes representatives for 95% of all known autonomous protein folds (Day et al., 2003; Schaeffer, Jonsson, Simms, & Daggett, 2011). The Dynameomics data warehouse (DDW) (Simms & Daggett, 2012; Simms, Toofanny, Kehl, Benson, & Daggett, 2008; van der Kamp et al., 2010) contains 104 times as many protein structures as the next largest protein structure repository, the PDB. The data in the DDW, especially high-resolution dynamic structure information and thorough sampling of rotameric states, are informative for protein structure prediction.

Using the DDW for protein structure prediction poses a number of significant informatics challenges. First, current visualization and analysis tools are inadequate for interpreting the complexity and size of data in the DDW. Second, no data representations exist for analyzing Dynameomics fragments and rotamers in conjunction with the Dynameomics data model and other structural repositories. Third, current fragment-based retrieval and prediction methods do not scale to either the size or distributed architecture of the DDW. Fourth, new methods are required to evaluate whether the knowledge contained in the fragment and rotamer representations is sufficient for structure prediction. Here I present the novel contributions I have made to address these challenges and I summarize the scientific findings produced from these contributions.

## *1.1 Visual Analytics*

Chapter 2 describes the general visual analytics framework we developed, named the Data Intensive Visualization Engine (DIVE) (Bromley, Rysavy, Su, Toofanny, et al., 2013; Rysavy, Bromley, & Daggett, 2014). This framework contains general methods for programmatic visualization and analysis of the DDW to improve scientific investigation. One novel method highlighted in this chapter is streaming access to the DDW. Large amounts of data can now be dynamically visualized and directly manipulated from the DDW for structure prediction. A second important method, the automatic conversion of object-oriented (OO) structures to the DIVE graph structure, is based on a set of rules to transform an OO system into a frame-like system. This transformation allows for a more expressive representation of the DDW, fragment and rotamer data and simplifies the interaction for 'non-programmer' scientists.

Chapter 3 further demonstrates the capabilities of the DIVE framework by showing specific applications and case studies. These applications demonstrate the breadth of DIVE's visual analytics abilities, including the investigation of single nucleotide polymorphisms, biomedical ontologies, and baseball statistics. The Protein Dashboard, a visual analytics interface to the DDW, is a specific DIVE application that enables scientific research of our protein simulations. Overall this chapter demonstrates the broad range of informatics challenges that DIVE addresses.

### *1.2 Software Engineering*

In Chapter 4 we present an application programming interface, the Dynameomics API, which is a software package containing methods and models for protein structure research. The Dynameomics API provides unified access to the DDW, the PDB, fragment repository, and rotamer repository. It also exposes protein structure data models which enable predictions of both protein backbone and side-chain conformations. Additional libraries are also included for general protein structure analysis.

### *1.3 Protein Fragments*

Chapter 5 describes the fragment data model and how it address the DDW scale and integration challenges with respect to backbone structure prediction (Rysavy, Beck, & Daggett, 2014). This model evolved from an internal coordinate (IC) representation for fragment comparison. The fragment data model reuses coordinate data already present in the DDW, searches fragments several orders of magnitude faster than the traditional approach, and, through the IC methods, provides accurate fragment structural comparisons. The model also integrates with existing Dynameomics analyses, is incorporated into the Dynameomics API, and is amenable to new fragment-based analysis. Now, for the first time, we can use the DDW for backbone structure prediction.

To assess the applicability of these new models we also created new visualization and analysis methods for evaluating the terabyte-scale results. For the fragment data, we used machine learning methods to understand the fragment conformational space and explored this space using DIVE. The computational analysis was performed using methods contained within the

Dynameomics API. We can now compare the conformational content of disparate structure repositories and identify areas of interest to the structure prediction community.

### *1.4 Amino Acid Side Chain Rotamers*

Chapter 6 covers the rotamer data model which supports side-chain conformation prediction through the DDW. Unlike previous Dynameomics rotamer representations (Scouras & Daggett, 2011), this organization takes into account backbone variability to provide more accurate rotamer conformations. Here we present a histogram-based approach to compare backbone-dependent and backbone-independent rotamers (Rysavy, Towse, & Daggett, 2014). The comparison technique enables researchers to identify important variations in rotamer probabilities.

The techniques implemented to generate rotamer data use sparse matrices, linear running-time circular statistics, and distributed computing to overcome limitations of previous methods. Since the model is integrated into the DDW, we can generate rotamer statistics on any simulation group specified in the Dynameomics dimensional model. All side-chain data were processed with the Dynameomics API and the resulting rotamer data can be accessed through Dynameomics API methods described in Chapter 4. DIVE was also used to visually analyze and investigate the rotamer data. We now have dynamic backbone-dependent rotamer libraries for structure prediction.

### *1.5 Continuing Research*

We developed novel data models and methods to provide a complete framework for structure prediction based on the DDW. However, the models and methods are designed to apply to more generalized problems. In Chapter 7 we summarize ancillary and continuing research ventures using the models and methods presented in earlier chapters. This research includes investigation of cross-linker distances, evaluation of the protein fold space, identification of transition states, characterization of pentapeptide structures, and comparison of D-amino acid rotamer libraries. Overall these methods and data models improve protein structure prediction, which is critical to understanding protein function, and provide a framework for future researchers.

## *1.6 Conclusions*

Through these contributions we have produced results relevant to the protein structure prediction problem, specifically loop structure prediction. The fragment research has identified regions of protein conformational space where Dynameomics fragments are important for improving structure prediction. We have shown how predicting loop regions with a consensus set of Dynameomics and PDB fragments can improve prediction results over using PDB fragments alone. We have also provided experimentally-validated cases where Dynameomics fragments predict loop structures better than X-ray crystallography fragments. The rotamer research has shown in which regions of backbone-conformational space it is important to use backbone-dependent rotamer probabilities over backbone-independent rotamer probabilities for native state structures. These results show that the data models and methods advance backbone structure prediction and improve our understanding of preferred side-chain conformations. The revised backbone-independent library has also been published in Chimera (Pettersen et al., 2004), a popular protein visualization and editing software package.

Additionally, we provide two significant software packages. The Dynameomics API is a general-purpose interface for modeling and transforming protein structures. DIVE is a general purpose visual analytics tool built for informatics research.  Both of these software packages contain novel contributions for informatics research.

Chapter 2

# DIVE: A GRAPH-BASED VISUAL ANALYTICS FRAMEWORK FOR BIG DATA

## *2.1 Summary*

The need for data-centric scientific tools is growing; domains such as biology, chemistry, and physics are increasingly adopting computational approaches. So, scientists must deal with the challenges of big data. To address these challenges, researchers built a visual-analytics platform named DIVE (Data Intensive Visualization Engine). DIVE is a data-agnostic, ontologically expressive software framework that can stream large datasets at interactive speeds. In particular, DIVE makes novel contributions to structured-data-model manipulation and high-throughput streaming of large, structured datasets.

## *2.2 Contributions*

This chapter is published in the journal *IEEE Computer Graphics and Applications* (Rysavy, Bromley, et al., 2014). As joint First Authors, Dennis Bromley and I performed the majority of the research in this chapter. My specific contributions presented here include the object parser, the pass-through SQL, the integration of the Dynameomics API, and the shared design of the DIVE framework.

## *2.3 Introduction*

Bioinformatics research depends increasingly on high-performance computation and large-scale data storage. Also, datasets are often complex, heterogeneous, or incomplete. These two aspects make bioinformatics appropriate for visual analytics (VA). Many powerful scientific toolsets are available, including software libraries such as SciPy (Jones, Oliphant, & Peterson, 2001) specialized visualization tools such as Chimera (Pettersen et al., 2004) and scientific workflow tools such as Taverna (Wolstencroft et al., 2013), Galaxy (Goecks, Nekrutenko, & Taylor, 2010), and the Visualization Toolkit (VTK) (Schroeder, Martin, & Lorensen, 1996). Some of them can handle large datasets. Others—typically, those originally designed for small, local datasets—haven't been updated to handle recent advances in data generation and acquisition.

To help fill this technological gap, we developed DIVE (Data Intensive Visualization Engine), which makes big-data VA approaches accessible to scientific researchers (see Figure 2.1). DIVE employs an interactive data pipeline that's extensible and adaptable. It encourages multiprocessor, parallelized operations and high-throughput, structured data streaming. DIVE can act as an object-oriented database by joining multiple disparate data sources. And, although we present bioinformatics applications here, DIVE can handle data from many domains.

## *2.4 The DIVE Architecture*

DIVE is an API whose primary component is the data pipeline, which can stream, transform, and visualize datasets at interactive speeds. The pipeline can be extended with plug-ins; each plug-in can operate independently on the data stream.

Data exploration is supported through command-line interfaces, GUIs, and APIs. Figure 2.2 shows an example DIVE application.  All these interfaces support scripting interaction. DIVE also supports typed events, letting users trigger targeted analyses from a point-and-click interface. Programmatically, DIVE inherits much functionality from the .NET environment, as we discuss later.

Finally, DIVE is domain independent and data-agnostic. The pipeline accepts data from any domain, provided an appropriate input parser is implemented. Currently supported data formats include SQL, XML, comma- and tab-delimited files, and several other standard file formats (see Figure 2.3).

### 2.4.1 *Data Representation*

Ontologies are gaining popularity as a powerful way to organize data. An ontology is a semantically and syntactically formal structure for organizing information (Horrocks, 2008). As organized datasets' size and complexity have grown, so has the need for formal semantics and syntax. In particular, the need for such formalisms is driven by the desire to handle these large, complex datasets programmatically. Ontologies enforce a strict formalism that guarantees that structured information is both meaningful and extensible. Once this is established, such information can be clearly reasoned with, built on, and discussed. An ontology can be represented as a graph in which nodes represent specific concepts and edges represent specific relationships.

We developed DIVE's core data representation with ontologies in mind. The fundamental data unit in DIVE is the datanode. Datanodes somewhat resemble traditional object instances from object-oriented (OO) languages such as C++, Java, or C#. They're typed, contain strongly typed properties and methods, and can exist in an inheritance hierarchy.

However, datanodes extend that traditional model. They can exist in an ontological network or graph; that is, multiple relationships beyond simple type inheritance can exist between datanodes. DIVE implements these relationships with dataedges, which link datanodes. Dataedges themselves are implemented by datanode objects and consequently might contain properties, methods, and inheritance hierarchies. Because of this basic flexibility, DIVE can represent arbitrary, typed relationships between objects, objects and relationships, and relationships and relationships.

Datanodes are also dynamic; every method and property can be altered at runtime, adding much flexibility to the system. (The DIVE pipeline contains various data integrity mechanisms to prevent unwanted side effects, as we discuss later.) The inheritance model is also dynamic; as a result, objects can gain and lose type qualification and other inheritance aspects at runtime. This allows runtime classification schemes such as clustering to be integrated into the object model. Finally, datanodes provide virtual properties. These properties are accessed identically to fixed properties but store and recover their values through arbitrary code instead of storing data on the datanode object. Virtual properties can extend the original software architecture's functionality, allowing data manipulation, as we describe later.

Dataedges implement multiple inheritance models. Besides the traditional is-a relationship in OO languages, ontological relationships such as contains, part-of, and bounded-by can be expressed. Each of these relationships can support varying levels of inheritance (see Table 2.1):

- With OO inheritance, which is identical to OO languages such as C++ and Java, subclasses inherit the parent's type, properties, and methods.
- With type inheritance, subclasses inherit only the type.
- With property inheritance, subclasses inherit only the properties and methods.

Like OO language objects, property-inheritance subclasses can override superclass methods and properties with arbitrary transformations. Similarly, type-inheritance subclasses can be cast to superclass types. Because DIVE supports not only multiple inheritance but also multiple kinds of inheritance, we implement casting by traversing the dataedge ontology. Owing to the coupling of the underlying data structure and ontological representation, every datanode and dataedge is implicitly part of a system-wide graph. This means we can use graph-theoretical methods to analyze both the data structures and ontologies represented in DIVE. This approach has already proved useful in structural biology (Bromley, Anderson, & Daggett, 2013).

Because all data are represented by datanodes and dataedges, DIVE analysis modules are presented with a syntactically homogenous dataset. Owing to this data-type independence, any modules can be connected so long as the analyzed datanodes have the expected properties, methods, or types, as we describe later. A module needn't concern itself with the data's origin or access syntax. So, DIVE supports code and tool reuse.

Data-type handling is a challenge in modular architectures. For example, Taverna uses typing in the style of MIME (Multipurpose Internet Mail Extensions). The VTK uses strongly typed classes. Python-based tools, such as Biopython (Cock et al., 2009) and SciPy, often use Python's dynamic typing.

For DIVE, the datanode and dataedge ontological network is a useful blend of these approaches. The dynamic typing of individual datanodes and dataedges lets us build arbitrary type networks from raw data sources. (See the Gene Ontology (Ashburner et al., 2000) taxonomy example described in the DIVE application note (Bromley, Rysavy, Su, Toofany, et al., 2013).) The underlying strong typing of the actual data (doubles, strings, objects, and so on) facilitates parallel processing, optimized script compilation, and fast, non-interpreted handling for operations such as filtering and plotting. Furthermore, the fact that the datanodes and dataedges themselves are strongly typed objects facilitates programmatic manipulation of the dataflow itself.

Although each typing approach has its strengths, DIVE's approach lends itself to fast, agile data exploration and fast, agile updating of DIVE tools. The datanode objects' homogeneity also simplifies the basic pipeline and module development. The tool updating is a particularly useful feature in an academic laboratory where multiple research foci, a varied spectrum of technical expertise, and high turnover are all common.

### 2.4.2 *Data Import*

Data must be imported into DIVE before they are accessible to the DIVE pipeline. In many cases, DIVE's built-in functionality handles this import. In the case of tabular data or SQL data tables, DIVE constructs one datanode per row, and each datanode has one property per column. DIVE also supports obtaining data from Web services such as the Protein Data Bank (Frances C. Bernstein et al., 1977). Once DIVE obtains the data, simple mechanisms establish relationships between datanodes. Later, we describe a more sophisticated way to acquire structured data that uses native object parsing.

### 2.4.3 *The DIVE Pipeline*

DIVE's pipeline is comparable to Taverna, Pipeline Pilot (http://accelrys.com/products/pipeline-pilot), Cytoscape (Shannon et al., 2003), Galaxy, and, most similarly, the VTK. Although all these platforms are extendable, two factors led us to develop DIVE. This first was platform considerations, which we discuss later. The second was our focus on agile data exploration instead of remote, service-based workflows. Fortunately, all these platforms have made interoperability a priority. So, we can leverage Cytoscape's graph capabilities or the VTK's visualization capabilities while maintaining DIVE's benefits by connecting their respective pipelines.

In the DIVE pipeline, plug-ins create, consume, or transform data. These plug-ins are simply compiled software libraries whose objects inherit from a published interface. The DIVE kernel automatically provides subsequent plug-in connectivity, pipeline instantiation, scripting, user interfaces, and many other aspects of plug-in functionality. Plug-ins move data through pins much like an integrated circuit: data originate at an upstream source pin and are consumed by one or more downstream sink pins. Plug-ins can also move data by broadcasting and receiving events. Users can save pipeline topologies and state to a file and share them.

When DIVE sends a datanode object through a branching, multilevel transform pipeline, it must maintain the datanode's correct property value at every pipeline stage. Otherwise, a simple plug-in that scaled incoming values would scale all data, everywhere in the pipeline. The naive option is to copy all datanodes at every pipeline stage, but this is extremely CPU- and memory-intensive and dramatically worsens the user experience.

To address this problem, DIVE uses read and write contexts. Essentially, this creates a version history of each transformed value. We key the history on each pipeline stage such that each plug-in reads only the appropriate values and not, for instance, downstream values or values from another pipeline branch. This approach maintains data integrity in a branching transform pipeline. It's also parallelizable. In addition, it keeps an accurate account of the property value at every stage in the pipeline, with a minimum of memory use. Finally, it's fast and efficient because the upstream graph traversal is linear and each value lookup occurs in constant time.

### 2.4.4 *Software Engineering Considerations*

We designed DIVE to provide a dynamic, scalable VA architecture. Although such an architecture doesn't require a specific platform, we built DIVE on the Microsoft Windows platform and .NET framework because of several significant built-in capabilities. These capabilities include the dynamic-language runtime, expression trees, and Language-Integrated Query (LINQ). .NET also provides coding features such as reflection, serialization, threading, and parallelism. Extensive documentation and details of these capabilities are at www.microsoft.com/net.

Many of these capabilities directly affect DIVE's functionality and user experience. Support for dynamic languages allows flexible scripting and customization that would be difficult in less expressive platforms. These components are crucial for both the data model we described earlier and the scripting capabilities we describe later. Furthermore, LINQ is useful in a scripted data-exploration environment. Expression trees and reflection provide the underlying object linkages for the DIVE object parser (which we also describe later), and DIVE streaming heavily uses the .NET framework's threading libraries. Finally, since .NET supports 64-bit computations and simple parallelism, DIVE is able to transparently scale with processor capabilities.

.NET also supports not only Microsoft-specific languages such as C#, Visual Basic, and F# but also more general languages such as Python and C++. This lets us author DIVE plug-ins in many languages. In addition, we can use these languages to develop command-line, GUI, and programmatic tools that embed and drive the DIVE kernel (as our case study shows later). .NET's wide user base also provides multiple external libraries with which to jump-start our development efforts, including molecular visualizers, clustering and analysis packages, charting tools, and mapping software. In particular, one such library is the VTK, wrapped by the ActiViz .NET API (see www.kitware.com/opensource/avdownload.php).

Finally, for our Dynameomics project (see Appendix A), we store data in a Microsoft SQL Server data warehouse. So, it made sense to adopt a software platform with deep support for these data services.

### *2.5 Object Parsing*

Module-management systems such as the Java-based OSGi (*OSGi Service Platform, Release 3,* 2003) support module life-cycle management and service discovery. However, module authors often must be aware of the module-management system when creating a module. We aimed to make .NET assemblies written without a priori knowledge of DIVE accessible to the ontological data representation. We also didn't require the life-cycle services of such module-management systems. So, we developed the DIVE object parser.

The parser automatically generates datanodes and dataedges from any .NET object or assembly (see Figure 2.4). Using reflection and expression trees, it consumes .NET object instances and translates them into propertied datanodes and dataedges. Usage patterns typically involve standard object creation by library-aware code, followed by automated object parsing and injection into the DIVE pipeline.

Generic rules define the mapping between the .NET object hierarchy and DIVE data structures. Generally, complex objects such as classes are parsed into datanodes, whereas built-in .NET system objects, primitive fields, primitive properties, and methods with primitive return types are translated into properties on those datanodes. Interfaces, virtual classes, and abstract classes are all translated into datanodes. The .NET inheritance and member relationships are interpreted as OO and property inheritance dataedges, respectively; these dataedges then connect the datanode hierarchy.

Using this approach, the object parser recursively produces an ontological representation of the entire .NET instance hierarchy in DIVE. Additional rules handle other program constructs. For example, the parser translates static members into a single datanode. Multiple object instances with the same static member all map to a single, static datanode instance in the DIVE data structure. Public objects and members are always parsed, whereas private members, static objects, and interfaces are parsed at the user's discretion.

Throughout this process, no data values are copied to datanodes or dataedges. Instead, dynamically created virtual properties link all datanode properties to their respective .NET members. So, any changes to the runtime .NET object instances are reflected in their DIVE representations. Similarly, any changes to datanode or dataedge properties propagate back to their .NET object instance counterparts. This lets DIVE interactively operate on any runtime .NET object structure.

With object parsing, users can import and use any .NET object without special handling. Furthermore, as we discussed before, the.NET application's architect doesn't need to be aware of DIVE to exploit its VA capabilities. For example, assume we have a nonvisual code library that dynamically simulates moving bodies in space (this example is available with the DIVE program download at www.dynameomics.org/dive). A DIVE plug-in, acting as a thin wrapper, can automatically import the simulation library and add runtime visualizations and interactive analyses. As the simulation progresses, the datanodes will automatically reflect the changing property values of the underlying .NET instances. Through a DIVE interface, the user could change a body’s mass. This change would propagate back to the runtime instance and immediately appear in the visualization. This general approach is applicable to many specialized libraries, taking advantage of their efficient data models. We describe an example of this later.

### *2.6 Scripting*

To let users rapidly interact with the DIVE pipeline, plug-ins, data structures, and data, DIVE supports two basic types of scripting: *plug-in scripting* and *µscripting* (microscripting). In the DIVE core framework, C# is the primary scripting language. Externally, DIVE can host components written in any .NET language and, conversely, can be hosted by any .NET environment. Here we focus on C# scripting.

Both scripting types are controlled in the same way. The user script is incorporated into a larger, complete piece of code, which is compiled during runtime using full optimization. Finally, through reflection, the compiled code is loaded back into memory as a part of the runtime environment. Although this approach requires time to compile each script, the small initial penalty is typically outweighed by the resulting optimized, compiled code. Both scripting types, particularly µscripting, can work on a per-datanode basis; optimized compilation helps create a fast, efficient user experience.

Plug-in scripting is simpler and more powerful than µscripting and is the most similar to existing analysis tools’ scripting capabilities. Through this interface, the user script can access the entire .NET runtime, the DIVE kernel, and the specific plug-in.

We developed µscripting to give complete programmatic control to power users and simple, intuitive control to casual users. Essentially, µscripting is an extension of plug-in scripting in which DIVE writes most of the code. The user needs to write only the right-hand side of a C# lambda function. Here’s a schematic of this function:

```
func(datanode dn) => ???;
```

The right-hand side is inserted into the function and compiled at runtime. The client can provide any expression that evaluates to an appropriate return value. Table 2.2 shows μscripting examples.

## *2.7 Data Streaming*

DIVE supports the following two SQL data-streaming approaches.

### 2.7.1 *Interactive SQL*

This approach (see Figure 2.5a) handles the immediate analysis of large, nonlocal datasets; it’s for impromptu, user-defined dynamic SQL queries.

### 2.7.2 *Pass-through SQL*

This approach (see Figure 2.5b) handles interactive analysis of datasets larger than the client’s local memory; it’s for streaming complex object models across a preset dimension.

Pass-through SQL accelerates the translation of SQL data into OO structures by shifting the location of values from the objects themselves to a *backing store*, an in-memory data structure. A backing store is essentially a collection of tables of instance data; each table contains many instance values for a single object type. Internally, object fields and properties have pointers to locations in backing-store tables instead of local, fixed values. A backing-store collection comprises all the tables for the object instances occurring at the same point, or frame, in the streaming dimension.

Once this approach creates a backing store, it generates copies of the backing-store structure with a unique identifier for each new frame. It then inserts instance values for new frames into the corresponding backing-store copy. This reduces the loading of instance data to a table-to-table copy, bypassing the parsing normally required to insert data into an OO structure. This approach also removes the overhead of allocating and de-allocating expensive objects by reusing the same object structures for each frame in the streaming dimension.

Pass-through SQL enables streaming through a buffered set of backing stores representing frames over the streaming dimension. A set is initially populated client-side for frames on either side of the frame of interest. Buffer regions are defined for each end of this set. Frames in the set are immediately accessible to the client. When the buffer regions' thresholds are traversed during streaming, a background thread is spawned to load a new set of backing stores around the current frame. If the client requests a frame outside the loaded set, a new set is loaded around the requested frame. Loaded backing stores no longer in the streaming set are deleted from memory to conserve the client's memory.

## *2.8 Case Study*

A major research focus in the Daggett laboratory at the University of Washington is the study of protein structure and dynamics through molecular dynamics (MD) simulations using the Dynameomics data warehouse (see Appendix A). The Dynameomics project contains much more simulation data than what typical, domain-specific tools can handle. Analysis of this dataset was the impetus for creating DIVE.

One of the first tools built on the DIVE platform was the Protein Dashboard, which provides interactive 2D and 3D visualizations of the Dynameomics dataset. These visualizations include interactive explorations of bulk data, molecular-visualization tools, and integration with external tools such as Chimera.

A tool implemented independently of DIVE and the Protein Dashboard is the Dynameomics API. Written in C#, it establishes an object hierarchy, provides high-throughput streaming of simulations from the Dynameomics data warehouse, contains domain-specific semantics and data structures, and provides multiple domain-specific analyses. However, it's designed for computational efficiency and doesn't specify any data visualizations or user interfaces.

We wanted to use the Dynameomics API's sophisticated data handling and streaming while keeping the Protein Dashboard's interactive visualization and analysis, without reimplementing DIVE's API.  Through the object parser, DIVE can integrate and use the Dynameomics API structures without changing its own API. This process creates strongly typed objects, including Structure, Residue, Atom, and Contact as datanodes, with each datanode containing properties defined by the Dynameomics API. Semantic and syntactic relationships specified in the API are similarly translated into dataedges. Once processed, these datanodes and dataedges are available to the DIVE pipeline, indistinguishable from any other datanodes or dataedges. Figure 2.6 diagrams this dataflow.

With the Dynameomics data and semantics available to the DIVE pipeline, we can apply a VA approach to the Dynameomics data. As before, we can use the Protein Dashboard to interact with and visualize the data. However, because the data flows through the Dynameomics API, wrapped by DIVE datanodes and dataedges, we can load multiple protein structures from different sources, including the Protein Data Bank (Frances C. Bernstein et al., 1977), align the structures, and analyze them in different ways.

Furthermore, because the Protein Dashboard has access to additional data from the Dynameomics API, its own utility increases. For instance, it’s useful to color protein structures on the basis of biophysical properties such as solvent-accessible surface area or deviation from a baseline structure. By streaming the data through the pipeline, we can watch these properties (many of which were accessed through the data’s inheritance hierarchy) change over time.

## *2.9 Discussion*

By necessity, most data analysis tools such as DIVE have some functional overlap; basic visualization and data analysis routines are simply required for functionality. However, several DIVE features are both novel and useful, particularly in a big-data, interactive setting. Here we discuss these features, their benefits, and how we see them integrating with existing technologies.

### 2.9.1 *Ontological Data Structure*

Besides simply representing the conceptual structure of the user’s dataset, DIVE’s graph-based data representation can effectively organize data. For example, using DIVE’s object model, we merged two ontology definitions from disparate sources.  These two ontologies, represented as DIVE datanodes and dataedges, were merged through property inheritance. This allowed the second ontology to inherit definitions from the first, resulting in a new ontology compatible with both data sources but amenable to new analysis approaches.

Besides these structural benefits, the datanodes are software objects that can update both their values and structures at runtime. Furthermore, the datanodes' ontological context can also update at runtime. So, DIVE can explore dynamic data sources and handle the impromptu user interactions commonly required for visual analysis.

### 2.9.2 *Object Parsing*

As the case study showed, the ability to parse a .NET object or assembly distinct from the DIVE framework circumvents the need to add DIVE-specific code to existing programs. In addition, this lets us augment those programs with DIVE capabilities such as graphical interaction and manipulation. For the Dynameomics API, we integrated the underlying data structures and the streaming functionality into the Protein Dashboard without modifying the existing API code base. This let us use the same code base in the DIVE framework and in SQL Common Language Runtime implementations and other non-DIVE utilities.

### 2.9.3 *Streaming Structured Data*

The most obvious benefit of DIVE is big-data accessibility through data streaming. Interactive SQL's flexibility effectively provides a visualization front-end for the Dynameomics SQL warehouse. However, for datasets not immediately described by the underlying database schema or other data source, a more advanced method for streaming complex data structures is desirable.

We developed pass-through SQL to make hundreds of terabytes of structured data immediately accessible to users. These data are streamed into datanodes and can be accessed either directly or indirectly through the associated ontology (for example, through property inheritance).

Furthermore, these data are preemptively loaded via background threads into backing stores; these backing stores are populated using efficient bulk transfer techniques and predictively cache data for user consumption. Finally, when the object parser is used with pass-through SQL, methods as well as data are parsed. So, the datanodes can access native .NET functionality in addition to the streaming data.

Preexisting programs also can benefit from DIVE’s streaming capabilities. For example, Chimera can open a network socket to DIVE’s streaming module. This lets Chimera stream MD data directly from the Dynameomics data warehouse.

## *2.10 Conclusions*

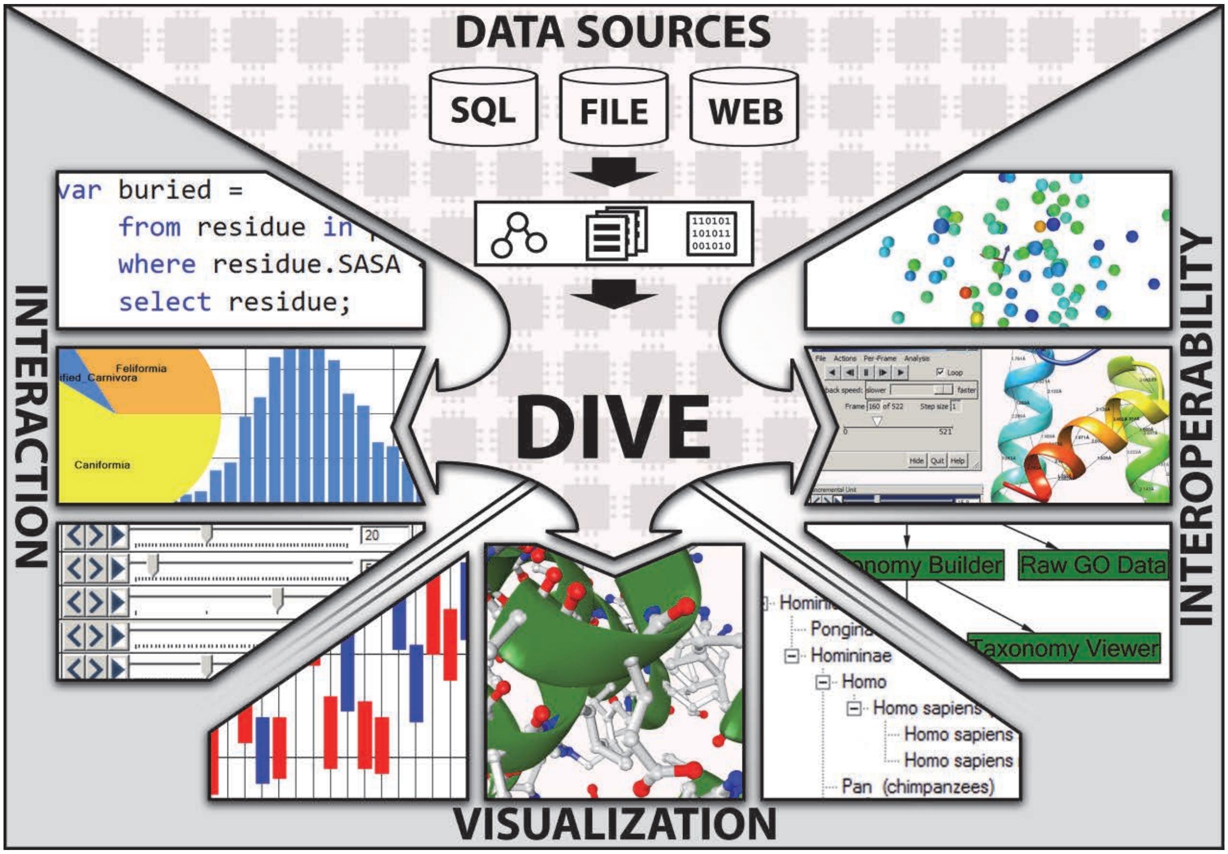
Large-scale data analysis will remain a pillar of scientific investigation; the challenge facing investigators is how best to leverage modern computational power. DIVE and other VA tools are providing insights into this challenge. Although it’s unlikely that any general tool will ever supplant domain-specific tools, the concepts highlighted here—accessibility, extensibility, simplicity of representation, integration, and reusability—will remain important.

**Table 2.1 DIVE Inheritance Models**

| Inheritance model | Inherits | | | Example or description |
|---|---|---|---|---|
| | Type | Properties | Methods | |
| Object-oriented (OO) inheritance | Yes | Yes | Yes | *Protein* **is a** *Molecule*. |
| Type inheritance | Yes | No | No | Used with property inheritance to implement OO inheritance. |
| Property inheritance | No | Yes | Yes | *Molecule* **contains** *Atom*. |

## Table 2.2 Examples of µScripting

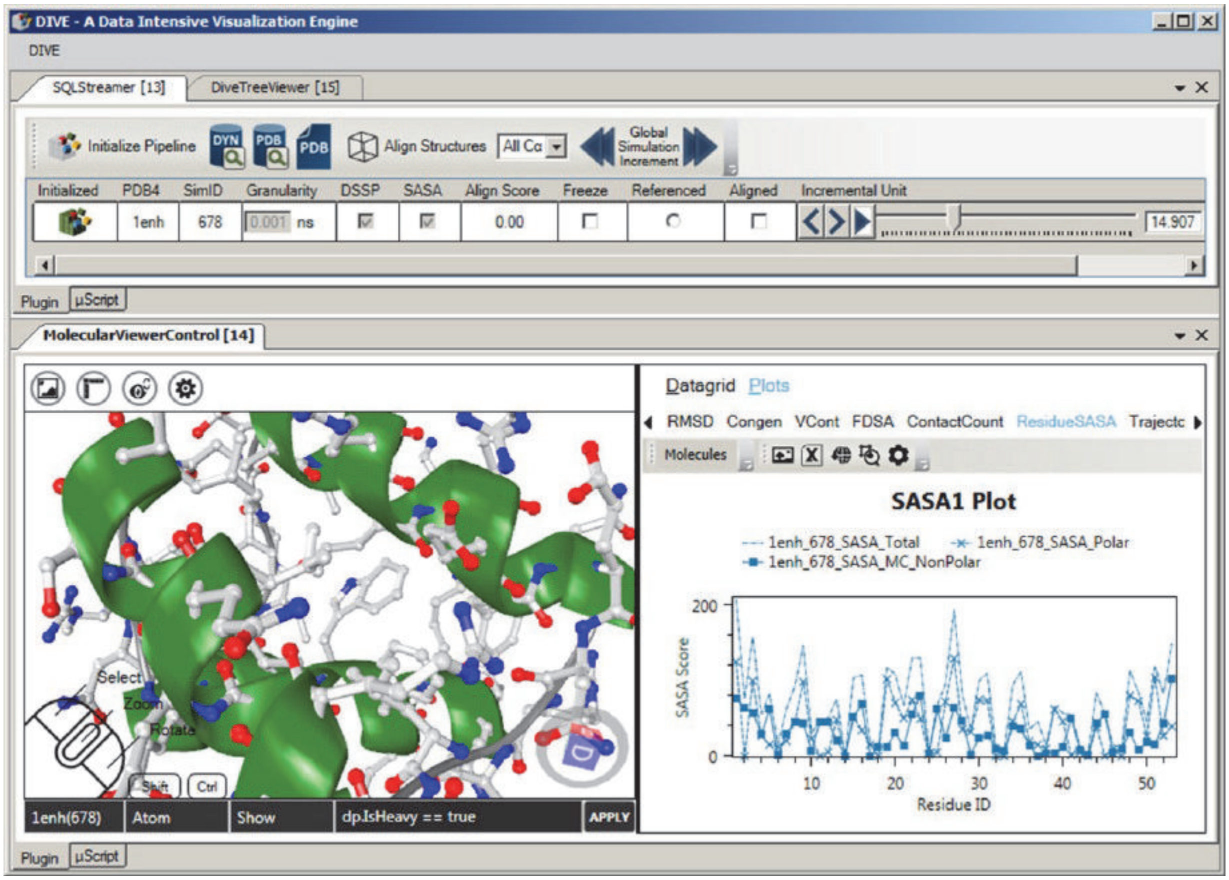
| Argument | Return type | Code | Comments |
|---|---|---|---|
| datanode dn | double | 3 | This is the simplest case of scripted numeric input. |
| | | dn.X | This is a simple per-datanode µscript. |
| | | Math.Abs(dn.X) | The µscript is given access to the full .NET library. |
| | int | dn.X > 0 ? 1 : -1; | Simple syntax can be powerful. |
| void | bool | {<br>int hour = DateTime.Now.Hour;<br>return hour < 12;<br>} | Any .NET code is allowed, including complex, multistatement functions. |
| datanode[] | Dynamic set | from dn in dns<br>group dn by Math.Round(dn.X, 2) into g<br>select new<br>{<br>bin = g.Key, population = g.Count()<br>}; | This creates a histogram based on the datanode objects' "X" property. |
| | | from dn in dns<br>where dn.X > Math.PI<br>&& dn.is_Superclass<br>&& dn.Func() == true<br>select dn; | This filters a subset of datanodes on the basis of properties, methods, and inherited type. |
| | | from dn1 in dnSet1<br>join dn2 in dnSet2 on dn1.X equals dn2.X<br>select new {X = dn1.X, Y = dn2.Y} | DIVE can act as an object-oriented database by joining multiple potentially disparate datasets. |



```
var buried =
    from residue in p
    where residue.SASA <
    select residue;
```

**Figure 2.1 An overview of DIVE (Data Intensive Visualization Engine), with screenshots.**

Users can access and structure data in various ways, including interactive and real-time data streaming. DIVE allows various types of interoperability, including interoperability with existing software libraries, interoperability with existing software tools, and interoperability among DIVE plug-ins. Interactive DIVE visualizations have included a 2D chart of baseball statistics, a 3D rendering of a protein molecule, and a taxonomy from the Gene Ontology. Interaction scenarios include scripted data manipulation, GUI interaction via charts and graphs, and event-driven data loading.



| Initialized | PDB4 | SimID | Granularity | DSSP | SASA | Align Score | Freeze | Referenced | Aligned | Incremental Unit |
|---|---|---|---|---|---|---|---|---|---|---|
|  | 1enh | 678 | 0.001 ns | ☑ | ☑ | 0.00 | □ | ○ | □ | 14.907 |

1enh(678) Atom Show dp.IsHeavy == true APPLY

**Figure 2.2 The DIVE GUI with the Protein Dashboard pipeline loaded.**

At the top is a data loader with which users can load and interact with protein structures and molecular-dynamics trajectories (see Appendix A) from different sources. On the lower left is an interactive 3D rendering of a protein molecule, rendered using a cartoon representation for the protein backbone and a ball-and-stick representation for a subset of atoms selected through the scripting window at the bottom. On the lower right is one of many linked interactive charts that stream synchronized data from the Dynameomics database.

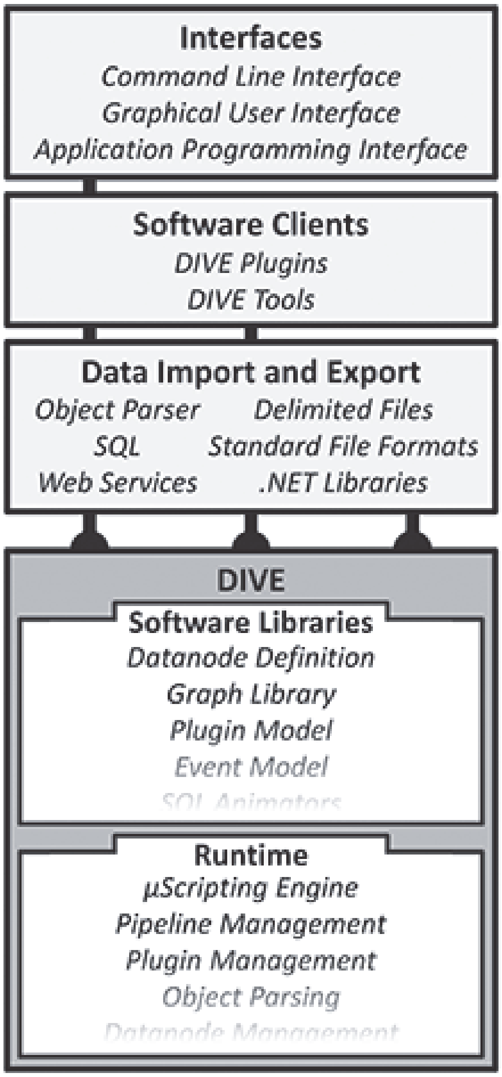


**Figure 2.3 The DIVE architecture.**

The DIVE kernel acts as both a software library and runtime environment. In both cases, DIVE can import and export data and functionality from a variety of sources. Pipeline plug-ins use DIVE primarily as a software library, exploiting DIVE's data-handling capabilities. DIVE tools are applications that instantiate and launch a DIVE pipeline for a specific analysis task. DIVE supports multiple types of interfaces.

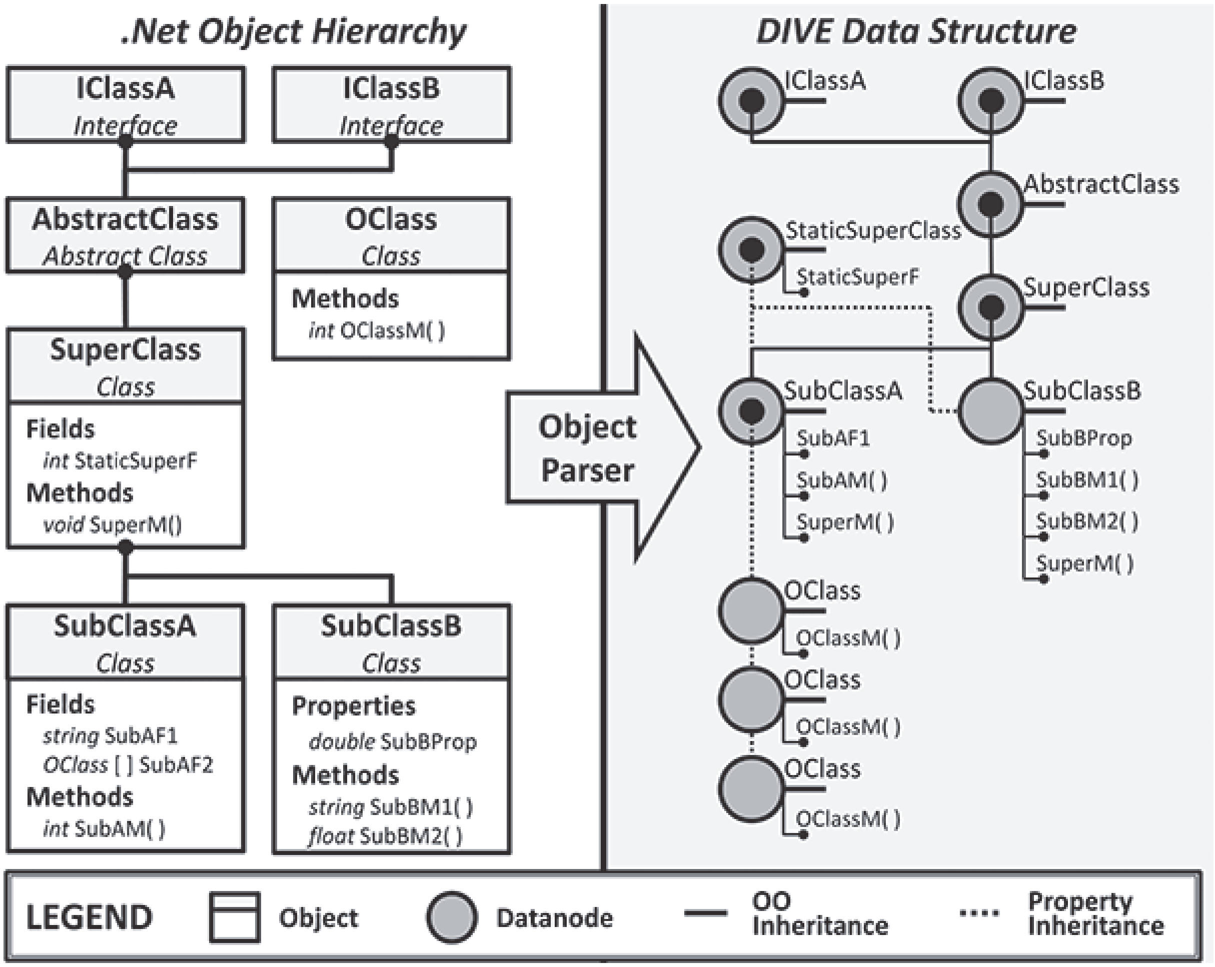
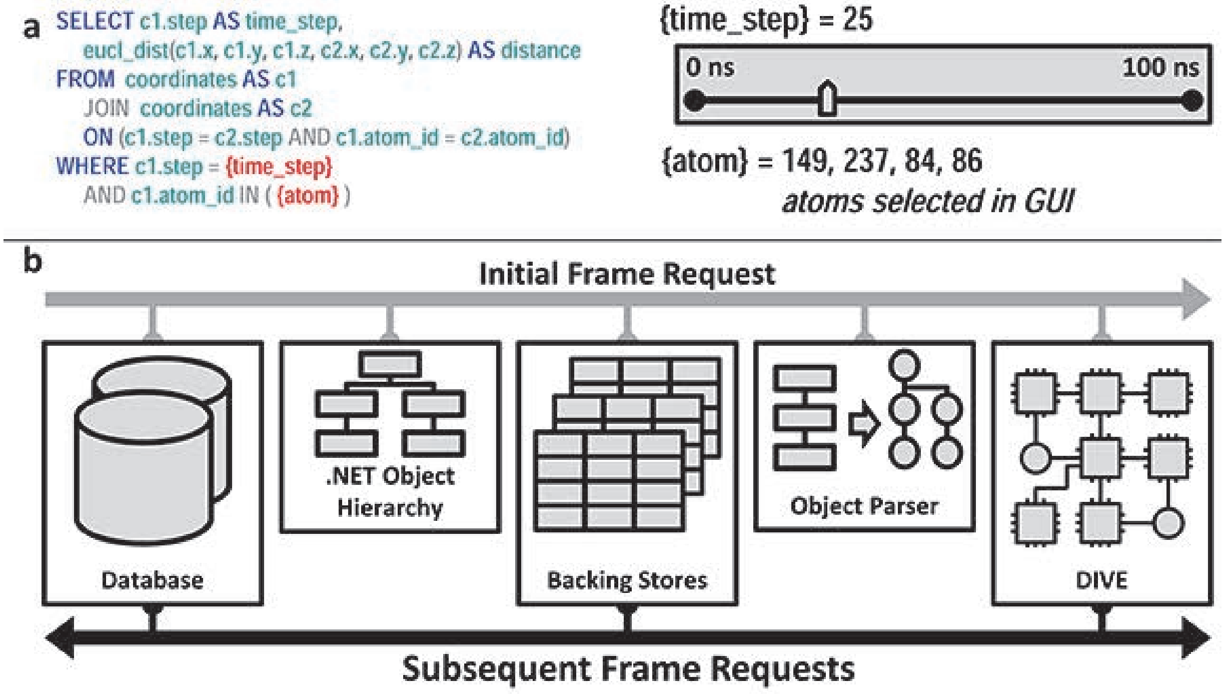


**Figure 2.4 A mapping of a datanode ontology from a third-party .NET assembly.**

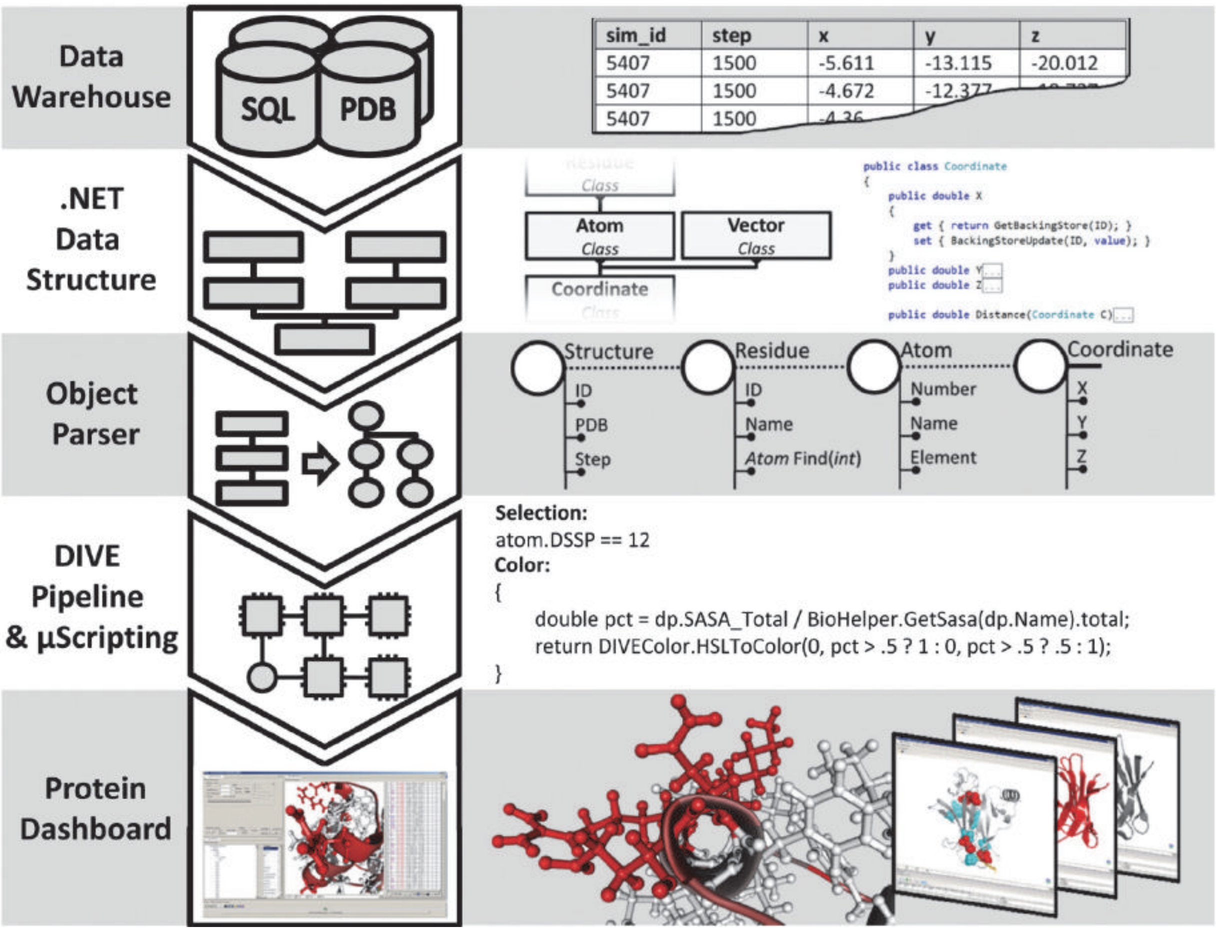
On the left, a generic .NET class hierarchy contains interfaces; class inheritance; and member fields, properties, and methods. On the right, the automatically generated ontology replicates the strongly typed objects and relationships from the .NET assembly. Instance-specific data are maintained on the subclass datanode object (that is, data aren’t stored in superclass datanodes). The original .NET object’s fields, properties, and methods are accessible through the datanodes by virtual properties.



```
SELECT c1.step AS time_step,
    eucl_dist(c1.x, c1.y, c1.z, c2.x, c2.y, c2.z) AS distance
FROM coordinates AS c1
    JOIN coordinates AS c2
    ON (c1.step = c2.step AND c1.atom_id = c2.atom_id)
WHERE c1.step = {time_step}
    AND c1.atom_id IN ( {atom} )
```

**Figure 2.5 SQL streaming in DIVE.**

(a) Interactive SQL. On the left is an SQL template with tags for time_step and atom. This approach replaces the tags with input from GUI elements, and the final query calculates the distances between all user-selected atoms at the specified time. (b) Pass-through SQL. On the initial frame request, this approach constructs a datanode hierarchy around the .NET objects and then creates backing stores. On all subsequent frame requests, DIVE buffers SQL data directly into the backing stores using multiple threads. This approach then propagates large amounts of complex data through DIVE at interactive speeds by bypassing object-oriented parsing.



| sim_id | step | x | y | z |
|---|---|---|---|---|
| 5407 | 1500 | -5.611 | -13.115 | -20.012 |
| 5407 | 1500 | -4.672 | -12.377 | |
| 5407 | 1500 | -4.36 | | |

```
public class Coordinate
{
    public double X
    {
        get { return GetBackingStore(ID); }
        set { BackingStoreUpdate(ID, value); }
    }
    public double Y...
    public double Z...

    public double Distance(Coordinate C)...
```

```
Selection:
atom.DSSP == 12
Color:
{
    double pct = dp.SASA_Total / BioHelper.GetSasa(dp.Name).total;
    return DIVEColor.HSLToColor(0, pct > .5 ? 1 : 0, pct > .5 ? .5 : 1);
}
```

**Figure 2.6 The Protein Dashboard case study.**

First, data are parsed in from the Dynameomics SQL warehouse or the Protein Data Bank (PDB), populating the Dynameomics API's backing stores. DIVE then parses these data structures and creates corresponding datanodes and dataedges available to the DIVE pipeline. The molecular-visualizer plug-in uses a µscript to select the atoms to display and their color. Finally, the user interacts with the data in the Protein Dashboard. In this example, residues in helical structures (the Selection µscript) are red if at least 50 percent of their maximum surface area is exposed to solvent (the Color µscript). With the Protein Dashboard, the user can access multiple interactive simulations simultaneously.

Chapter 3

# DIVE: A DATA INTENSIVE VISUALIZATION ENGINE

## *3.1 Summary*

Modern scientific investigation is generating increasingly larger datasets, yet analyzing these data with current tools is challenging. DIVE is a software framework intended to facilitate big data analysis and reduce the time to scientific insight. Here we present features of the framework and demonstrate DIVE's application to the Dynameomics project, looking specifically at two proteins.

## *3.2 Contributions*

This chapter is published in the journal *Bioinformatics* (Bromley, Rysavy, Su, Toofanny, et al., 2013). As joint First Authors, Dennis Bromley and I performed the majority of the research in this chapter. My specific contributions presented here include the shared design of the DIVE framework and components as specified in Chapter 2, much of the research and development behind the Protein Dashboard, and the application of the DIVE framework for biophysical analysis of the human Cu-Zn superoxide dismutase 1 (SOD1) mutations.

## *3.3 Introduction*

The advent of massive networked computing resources has enabled virtually unlimited data collection, storage and analysis from low-cost genome sequencing, high-precision molecular dynamics simulations, and high-definition imaging data for radiology, to name just a few examples. This explosion of 'big data' is changing traditional scientific methods; instead of relying on experiments to output relatively small, targeted datasets, data mining techniques are being used to analyze data stores with the intent of learning from the data patterns themselves. Unfortunately, data analysis and integration in large data storage environments is challenging even for experienced scientists. Furthermore, most existing domain-specific tools designed for complex, heterogeneous datasets are not equipped to visually analyze big data.

DIVE is a software framework designed for exploring large, heterogeneous, high-dimensional datasets using a visual analytics approach (Figure 3.1). Visual analytics is a big data exploration methodology emphasizing the iterative process between human intuition, computational analyses, and visualization. DIVE's visual analytics approach integrates with traditional methods, creating an environment that supports data exploration and discovery.

## *3.4 System and Implementation*

DIVE provides a rich, ontologically expressive data representation and a flexible, modular streaming-data architecture, or pipeline (Figure 3.2). It is accessible through an application programming interface (API), command line interface or graphical user interface. Applications built on the DIVE framework inherit features such as a serialization infrastructure, ubiquitous scripting, integrated multithreading and parallelization, object-oriented data manipulation, and multiple modules for data analysis and visualization. DIVE can also interoperate with existing analysis tools to supplement its capabilities, such as the Visualization Toolkit (Schroeder et al., 1996), Cytoscape (Shannon et al., 2003) and Bing maps (http://bing.com) by either exporting data into known formats or by integrating with published software libraries. Furthermore, DIVE can import compiled software libraries and automatically build native ontological data representations, reducing the need to write DIVE-specific software. From a data perspective, DIVE supports the joining of multiple heterogeneous data sources, creating an object-oriented database capable of showing inter-domain relationships. And while DIVE currently focuses on bioinformatics, DIVE itself is data-agnostic; data from any domain may enter the DIVE pipeline.

A core feature of DIVE's framework is the flexible, graph-based data representation. DIVE data are stored as nodes in a strongly-typed, ontological network defined by the data. These data can be a simple set of numbers or a complex object hierarchy with inheritance and well-defined relationships. Data flow through the system explicitly as a set of data points passed down the DIVE pipeline or implicitly as information transferred and transformed through the data relationships (Figure 3.3e). A thorough description of the novel technical contributions of DIVE is provided elsewhere (Rysavy, Bromley, et al., 2014).

### *3.5 Results*

The impetus for DIVE was data mining the Dynameomics dataset (van der Kamp et al., 2010). Dynameomics is a large, data-intensive project that contains atomistic molecular dynamics (MD) simulations of the native state and unfolding pathways of representatives of essentially all protein folds (van der Kamp et al., 2010). These protein simulations and associated biophysical analyses are stored in a mixed data warehouse (Simms & Daggett, 2012) and file system environment distributed over multiple servers containing hundreds of terabytes of data and over $10^4$ times as many structures as the Protein Data Bank (PDB) (Bernstein et al., 1977), representing the largest collection of protein structures and protein simulations in the world.

In the domain of structural biology, Dynameomics exemplifies the challenges of big data. Here we present DIVE applications involving two proteins where specialized modules built on the DIVE framework are used to accelerate biophysical analysis. The first protein is the transcription factor p53, mutations in which are implicated in cancer. The second protein is human Cu-Zn superoxide dismutase 1 (SOD1), mutations in which are associated with amyotrophic lateral sclerosis (Rakhit and Chakrabartty, 2006).

The Y220C mutation of p53 is responsible for destabilizing the core domain (Joerger et al., 2006), leading to approximately 75,000 new cancer cases annually (Boeckler et al., 2008). We have used the DIVE framework to analyze the structural and functional effects of the Y220C mutation through a module called ContactWalker (Bromley et al., 2013), which identifies amino acids' interatomic contacts disrupted significantly as a result of mutation. The contact pathways between disrupted residues are identified using DIVE's underlying graph-based data representation.

Figure 3.4a shows the most disrupted contacts in the vicinity of the Y220C mutation. Specific residues, contacts and simulations were identified for more focused analysis. Interesting interatomic contact data are isolated and then specific MD time points and structures are selected for further investigation. For example, see the contact data mapped onto a structure containing a stabilizing ligand, which docks closely to many of the disrupted residues, suggesting a correlation between the mutation-associated effects and the observed stabilizing effects of the ligand (Figure 3.4a).

As another example of the use of DIVE, we have over 300 simulations of 106 disease-associated mutants of SOD1 (Schmidlin et al., 2009). Through extensive studies of A4V mutant SOD1 simulations, Schmidlin et al. (2009) previously noted the instability of two β-strands in the SOD1 Greek Key β-barrel structure. However, that analysis took several years to complete and such manual interrogation of simulations does not scale to allow us to search for general features linked to disease across hundreds of simulations. Using DIVE, we were able to further explore the formation and persistence of the contacts and packing interactions in this region across multiple simulations of mutant proteins. DIVE facilitates isolation of specific contacts, rapid plotting of selected data, easy visualization of the relevant structures, and geographic locations of specific mutations while providing intuitive navigation from one view to another (Figure 3.1 and Figure 3.4).

The top panel of Figure 3.4b maps secondary structure for different variants as an example of DIVE's charting tools. This chart is quickly generated, contains results for over 300 SOD1 mutant simulations, is customizable, and links to the protein structure property data (in this case the change in the structure over time) with a single mouse click (Figure 3.4b). These data are in turn linked to protein structure modules, allowing interactive visualization of over 60,000 structures from each of the 300 simulations, all streamed from the SQL data warehouse (Figure 3.4b). With DIVE, we simplified the transition between high-level protein views and atomic level details, facilitating rapid analysis of large amounts of data. DIVE can also show the context of the detailed results on other levels, such as worldwide disease incidence (Figure 3.1).

DIVE's utility is not limited to protein simulations. To demonstrate its versatility, usability and data-agnostic nature, we applied it to additional domains. Brief details of these applications are provided in the Case Studies Section. One example shows an interaction with the Gene Ontology (Ashburner at al., 2000) and another example explores professional baseball statistics.

## *3.6 Case Studies*

### 3.6.1 *Protein Dashboard*

The protein dashboard (Figure 3.3) is a data exploration application that uses the DIVE framework and Dynameomics Application Programming Interface (Rysavy *et al.*, DIVE – A

Graph-Based Visual Analytics Framework for Big Data, 2013, *submitted for publication*) to visually and interactively present the Dynameomics data. Through the DIVE object model, the protein dashboard organizes these data and renders them in multiple, linked modules at once; interaction with one module can update connected modules. For example, an ontological relationship exists between interatomic contacts and protein residues: a contact connects two atoms and therefore, through the protein dashboard's structural hierarchy, two residues. In Figure 3.3, the protein dashboard depicts a pair of aligned 3D protein structures and 2D contact map. Selecting a contact in the contact map will highlight the associated residues and metadata in the 3D structure. Furthermore, these structured data and relationships persist while streaming from the data warehouse and they are also available to DIVE's scripting engine. In this way, the protein dashboard uses the DIVE framework to bring an additional level of navigable order to the Dynameomics data warehouse. The protein dashboard and associated documentation are included in the DIVE software download.

### 3.6.2 *Gene Ontology*

DIVE is a versatile framework that can be useful for a variety of scientific domains. We created a DIVE pipeline to explore an area of bioinformatics not typically used for research in the Daggett lab, specifically the Gene Ontology (GO) database (Ashburner et al., 2000). We wrote a simple script to generate an interactive taxonomy of species contained in the GO database (Figure 3.5). A more complex example that selects sections of the taxonomy in order to chart groups of interest is shown in Figure 3.6. This illustrates DIVE's ability to operate in other areas of the bioinformatics domain as well as shows DIVE's support for ontologies. This pipeline took approximately one hour to build by a person unfamiliar with the GO database.

### 3.6.3 *Professional Baseball Statistics*

To illustrate the domain-independence of DIVE, we analyzed approximately 200 years of baseball statistics (http://seanlahman.com/). These data were contained in several comma-separated value (CSV) files (Figure 3.7). Because DIVE is data-independent, we were able to load, explore, chart, and interact with the baseball data using the same tools and techniques that were used to explore protein structure data. In addition to the general-purpose analysis and visualization plugins, we also used the general-purpose analysis functionality in the DIVE kernel to perform edge-detection filtering on each player's year-to-year earned-run-average (ERA). By sorting the players on their average year-to-year ERA differences (left-to-right, Figure 3.8), we were able to identify those pitchers with the most-consistent and least-consistent pitching histories. Furthermore, by using DIVE to integrate free-form data sources such as web searching (Figure 3.9), data anomalies were easily explained. For example, Whitey Ford, one of the most consistent baseball pitchers, missed the 1951 season due to military service.

## *3.7 Conclusions*

Overall, DIVE provides an interactive data-exploration framework that expands upon conventional analysis paradigms and self-contained tools. We provided analytic examples in the protein simulation domain, but the DIVE framework is not limited to this field. DIVE can adapt to existing data representations, consume non-DIVE software libraries, and import data from an array of sources. As research becomes more data-driven and reliant on data mining and visualization, big data visual analytics solutions should provide a new perspective for scientific investigation.

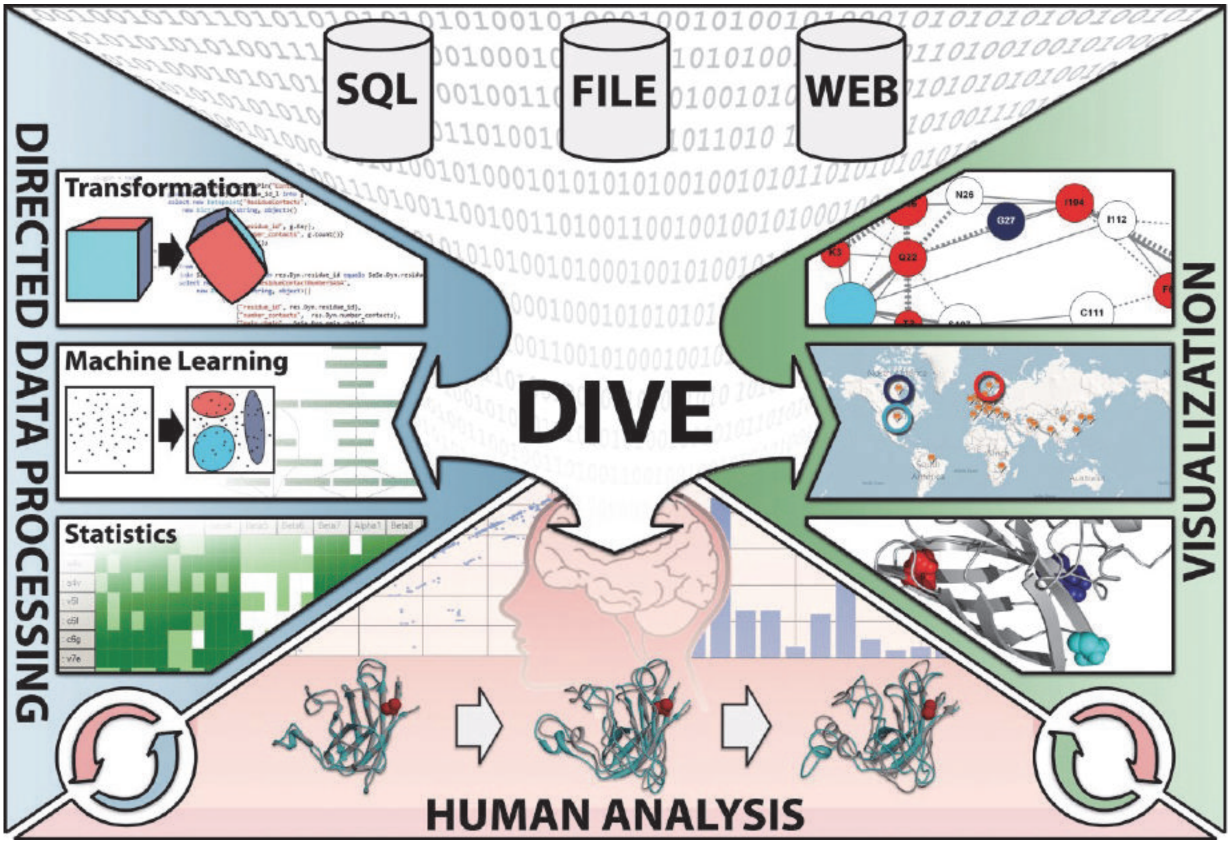


**Figure 3.1 Schematic of the data flow within DIVE.**

Data enter DIVE from a variety of sources and are processed, analyzed and visualized by specific DIVE modules. The user interacts with these modules, iteratively refining the investigation as scientific insights develop. Analyses, visualizations, and data organization can all be controlled by the user, and changes are saved for future work. Visualization and analysis options include charting and graphing, specialized representations, clustering and filtering, arbitrary script interactions, and domain-specific analyses such as interatomic contact occupancy. The DIVE framework can be extended with custom functionality.

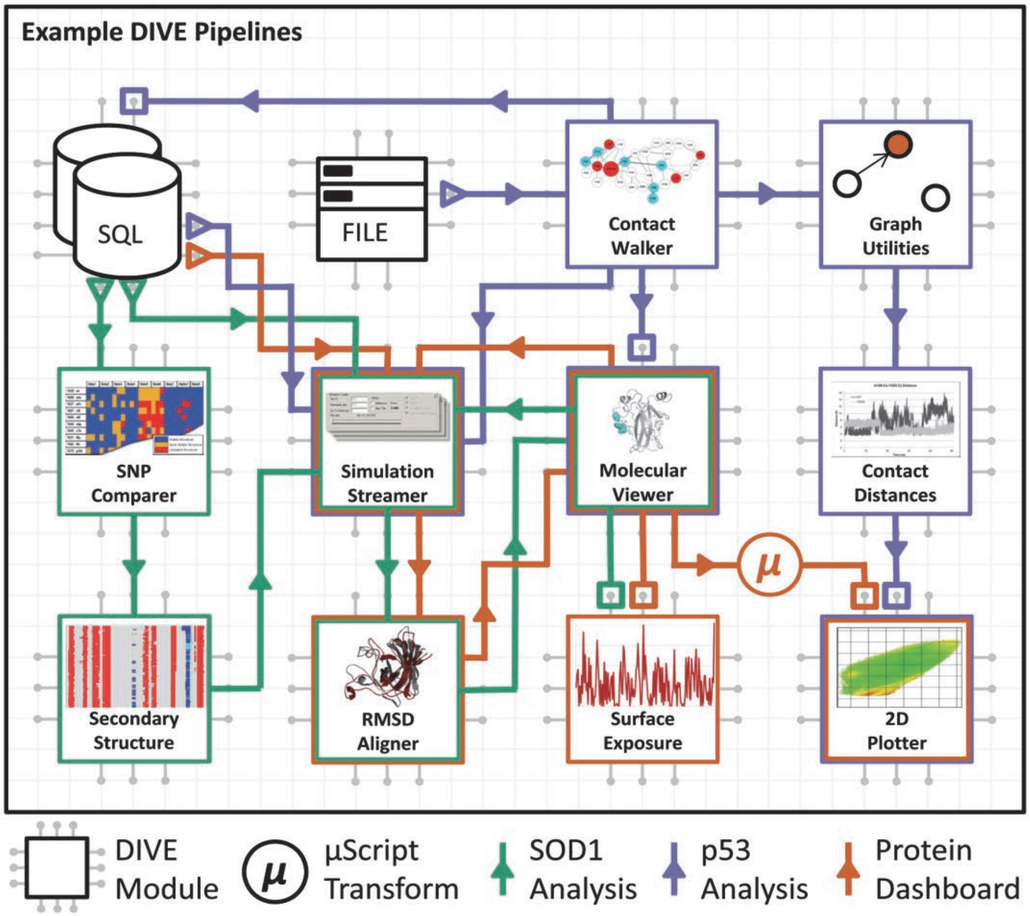
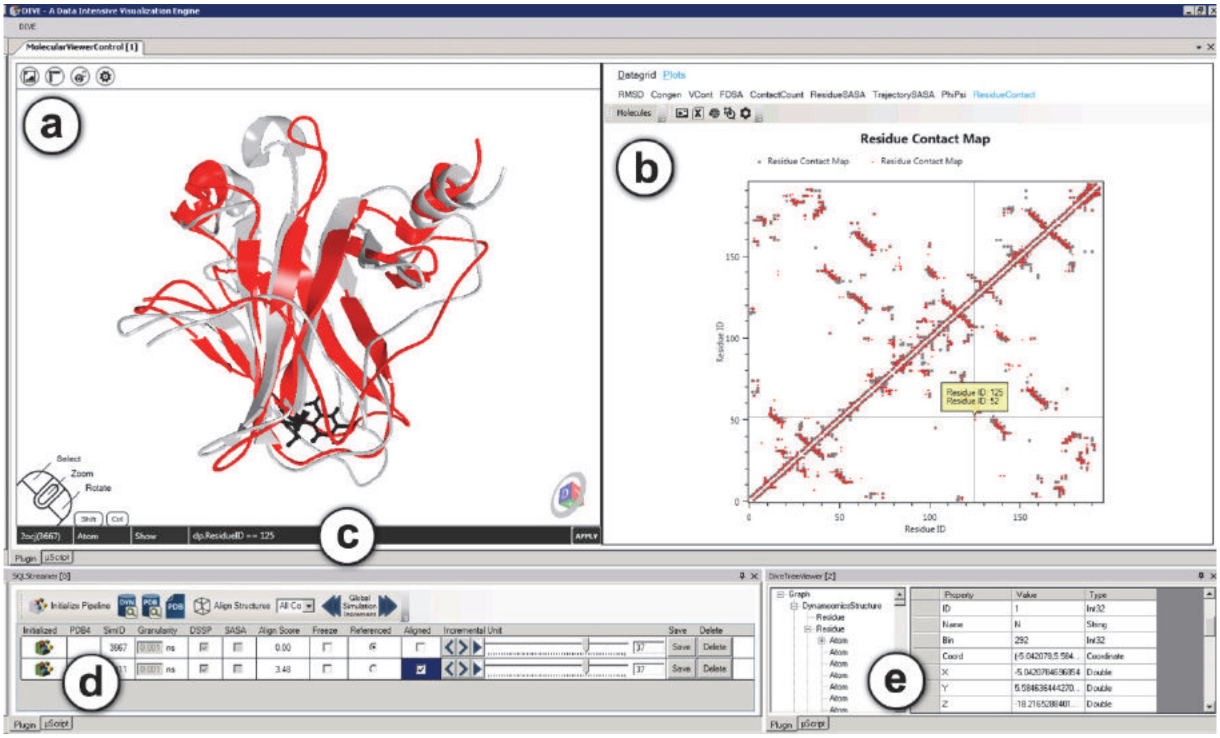


**Figure 3.2 Conceptual representation of DIVE modules and visual analytic processes.**

Each module has multiple pins that send output or receive input using data points, creating individual data pipelines. Data points can be transformed with μScripting as they flow through the pipelines. Three separate processes are portrayed: SOD1 analysis (green), p53 analysis (purple), and the protein dashboard (orange). In the SOD1 analysis, data points flow to the SNP comparer module for secondary structure assessment. Simulations of interest are then routed to another module to display the trend of secondary structure over time. Data points for these same simulations are then streamed directly from the data warehouse, aligned on Cα root-mean-square deviation, and visually analyzed in a molecular viewer. In the p53 analysis, ContactWalker reads data from the file system and calculates occupancy differences between the wild type and mutant simulations. These data are then used to create a contact graph. This graph is searched and contact pathways between significantly disrupted residues are identified. Additionally, the occupancy data are mapped onto a protein structure and visualized. From here, further analysis can involve analyzing specific contact distances or viewing the full trajectory in the Protein Dashboard.



| 2ocj(3667) | Atom | Show | dp.ResidueID == 125 | APPLY |
|---|---|---|---|---|

| Initialized | PDB4 | SimID | Granularity | DSSP | SASA | Align Score | Freeze | Referenced | Aligned | Incremental Unit | Save | Delete |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  | 3667 | 0.001 ns | ☑ | □ | 0.00 | □ | ● | □ | 37 | Save | Delete |
|  |  | 11 | 0.001 ns | ☑ | □ | 3.48 | □ | ○ | ☑ | 37 | Save | Delete |

| Property | Value | Type |
|---|---|---|
| ID | 1 | Int32 |
| Name | N | String |
| Bin | 292 | Int32 |
| Coord | (-5.042078,5.584... | Coordinate |
| X | -5.04207846968854 | Double |
| Y | 5.584636444270... | Double |
| Z | -18.2165288401... | Double |

**Figure 3.3 Screenshot of the Protein Dashboard.**

(a) 3D molecular visualization module depicting two aligned structures of p53 with the mutated residue highlighted in black. (b) Interactive residue-residue contact maps of the two p53 structures. The crossbar indicates a contact of the mutated residue. (c) µScript specifying the explicit display of the atoms contained in the mutated residue. (d) Streaming module used to access Dynameomics simulations. This figure depicts two separate simulations of p53 simultaneously streaming from the data warehouse. (e) Interactive view of the simulation hierarchies and associated strongly typed fields and methods.

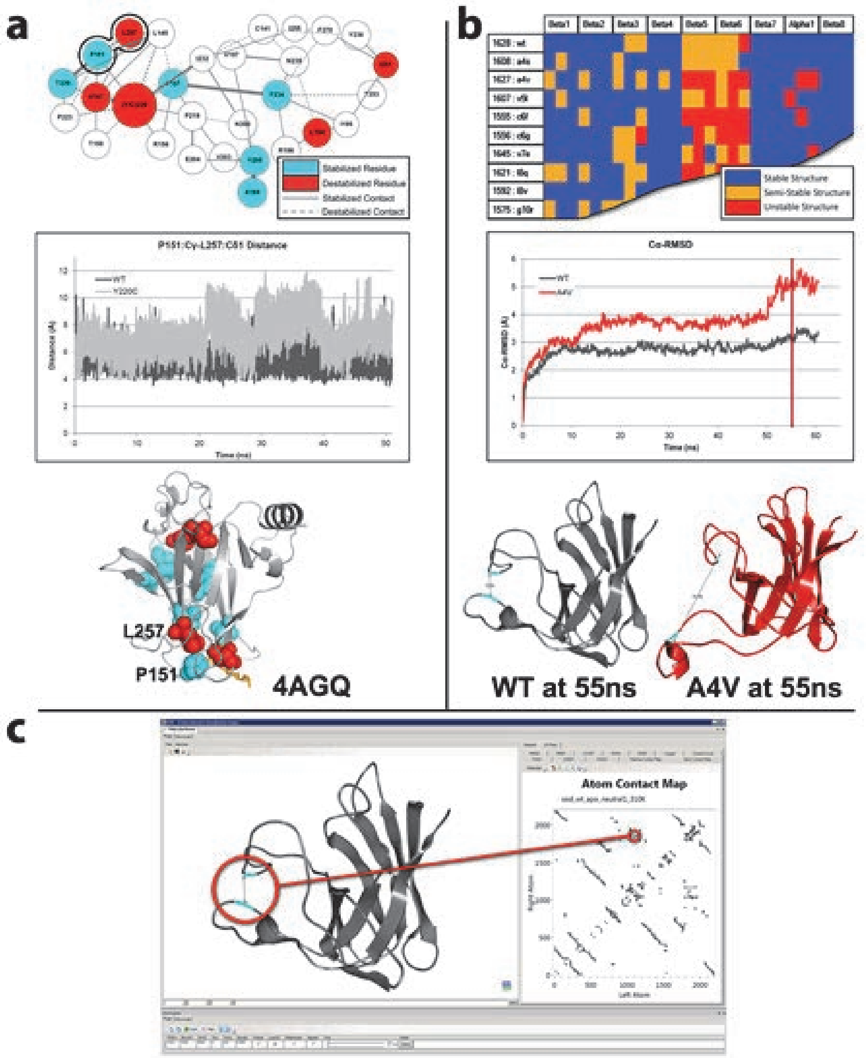
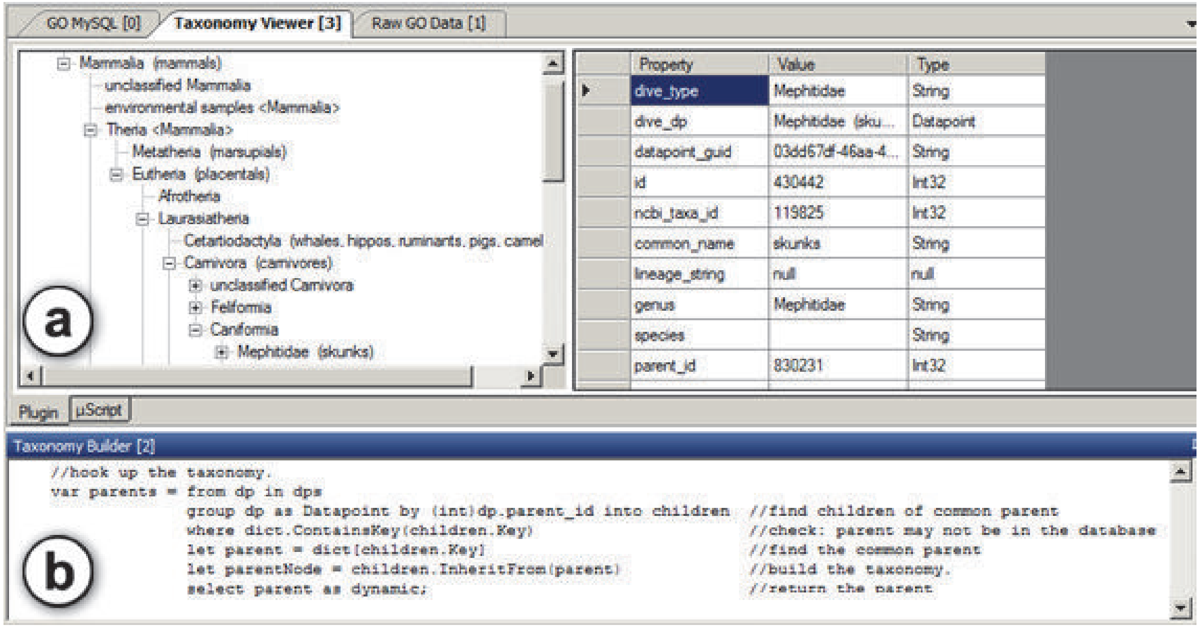


**Figure 3.4 Interactive visualizations in DIVE.**

(a) p53 analysis visualizations. Top, Contact-Walker summary of contact differences between wild-type and Y220C simulations. The colored residues have contacts with a ≥ 50% occupancy change. Middle, distances between P151 and L257, outlined in black in the map above. Bottom, p53 with ligand (gold) (PDB code 4AGQ) in close proximity to disrupted colored residues. (b) SOD1 analysis visualizations. Top, aggregated secondary structural data from mutant simulations. Middle, plot of the Cα RMS deviation of the wild type and A4V mutant simulations. Bottom, MD structures. (c) Protein dashboard application showing a viewer and interactive contact map.



```
⊟ Mammalia (mammals)
    unclassified Mammalia
    environmental samples <Mammalia>
  ⊟ Theria <Mammalia>
      Metatheria (marsupials)
    ⊟ Eutheria (placentals)
        Afrotheria
      ⊟ Laurasiatheria
          Cetartiodactyla (whales, hippos, ruminants, pigs, camel
        ⊟ Carnivora (carnivores)
          ⊞ unclassified Carnivora
          ⊞ Feliformia
          ⊟ Caniformia
            ⊞ Mephitidae (skunks)
```

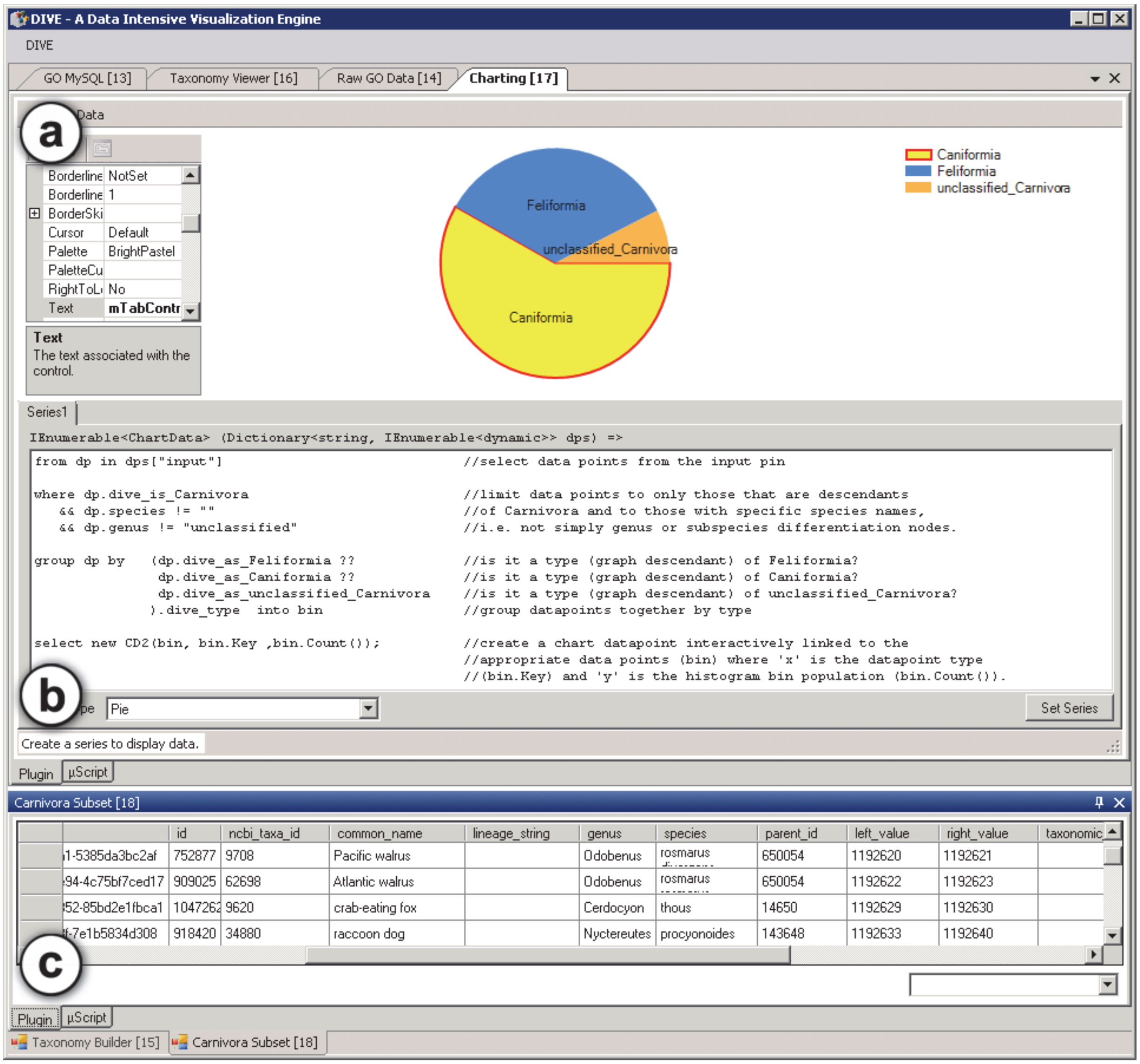
| Property | Value | Type |
|---|---|---|
| dive_type | Mephitidae | String |
| dive_dp | Mephitidae (sku... | Datapoint |
| datapoint_guid | 03dd67df-46aa-4... | String |
| id | 430442 | Int32 |
| ncbi_taxa_id | 119825 | Int32 |
| common_name | skunks | String |
| lineage_string | null | null |
| genus | Mephitidae | String |
| species |  | String |
| parent_id | 830231 | Int32 |

Taxonomy Builder [2]

```
//hook up the taxonomy.
var parents = from dp in dps
              group dp as Datapoint by (int)dp.parent_id into children  //find children of common parent
              where dict.ContainsKey(children.Key)                      //check: parent may not be in the database
              let parent = dict[children.Key]                           //find the common parent
              let parentNode = children.InheritFrom(parent)             //build the taxonomy.
              select parent as dynamic;                                 //return the parent
```

**Figure 3.5 Screenshot of DIVE displaying the Gene Ontology database.**

Two generic DIVE plugins were used to create this view. (a) Interactive view of the Gene Ontology species taxonomy. (b) Script (written in C#) to create taxonomy from the Gene Ontology species table.



| | |
|---|---|
| Borderline | NotSet |
| Borderline | 1 |
| ⊞ BorderSki | |
| Cursor | Default |
| Palette | BrightPastel |
| PaletteCu | |
| RightToL | No |
| Text | **mTabContr** |

**Text**
The text associated with the control.

```
IEnumerable<ChartData> (Dictionary<string, IEnumerable<dynamic>> dps) =>
```

```
from dp in dps["input"]                             //select data points from the input pin

where dp.dive_is_Carnivora                          //limit data points to only those that are descendants
   && dp.species != ""                              //of Carnivora and to those with specific species names,
   && dp.genus != "unclassified"                    //i.e. not simply genus or subspecies differentiation nodes.

group dp by   (dp.dive_as_Feliformia ??             //is it a type (graph descendant) of Feliformia?
               dp.dive_as_Caniformia ??             //is it a type (graph descendant) of Caniformia?
               dp.dive_as_unclassified_Carnivora    //is it a type (graph descendant) of unclassified_Carnivora?
              ).dive_type  into bin                 //group datapoints together by type

select new CD2(bin, bin.Key ,bin.Count());          //create a chart datapoint interactively linked to the
                                                    //appropriate data points (bin) where 'x' is the datapoint type
                                                    //(bin.Key) and 'y' is the histogram bin population (bin.Count()).
```

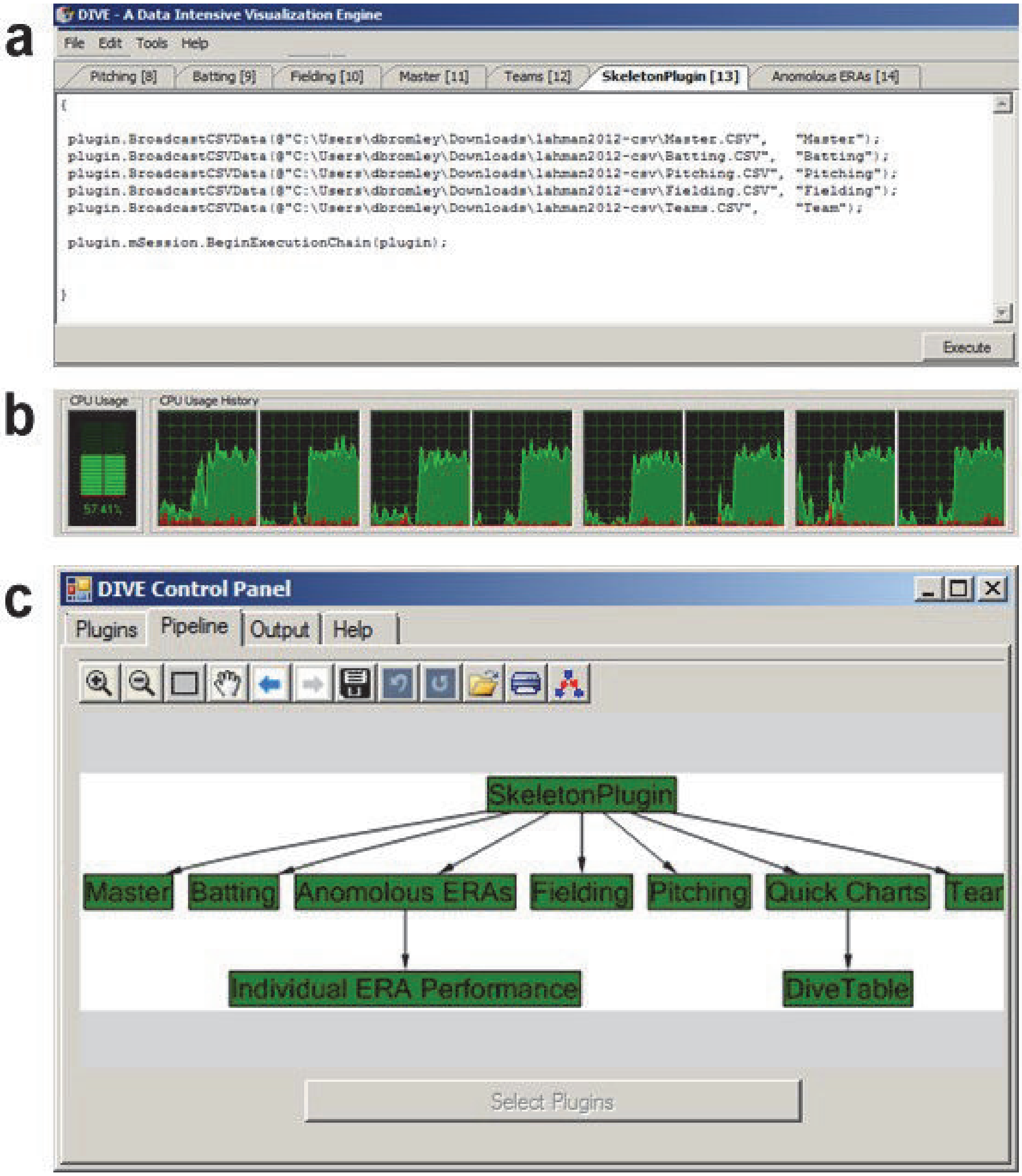
pe Pie

Create a series to display data.

| | id | ncbi_taxa_id | common_name | lineage_string | genus | species | parent_id | left_value | right_value | taxonomic |
|---|---|---|---|---|---|---|---|---|---|---|
| 1-5385da3bc2af | 752877 | 9708 | Pacific walrus | | Odobenus | rosmarus | 650054 | 1192620 | 1192621 | |
| 94-4c75bf7ced17 | 909025 | 62698 | Atlantic walrus | | Odobenus | rosmarus | 650054 | 1192622 | 1192623 | |
| 52-85bd2e1fbca1 | 1047262 | 9620 | crab-eating fox | | Cerdocyon | thous | 14650 | 1192629 | 1192630 | |
| f-7e1b5834d308 | 918420 | 34880 | raccoon dog | | Nyctereutes | procyonoides | 143648 | 1192633 | 1192640 | |

**Figure 3.6 Screenshot of DIVE showing the Gene Ontology species taxonomy.**

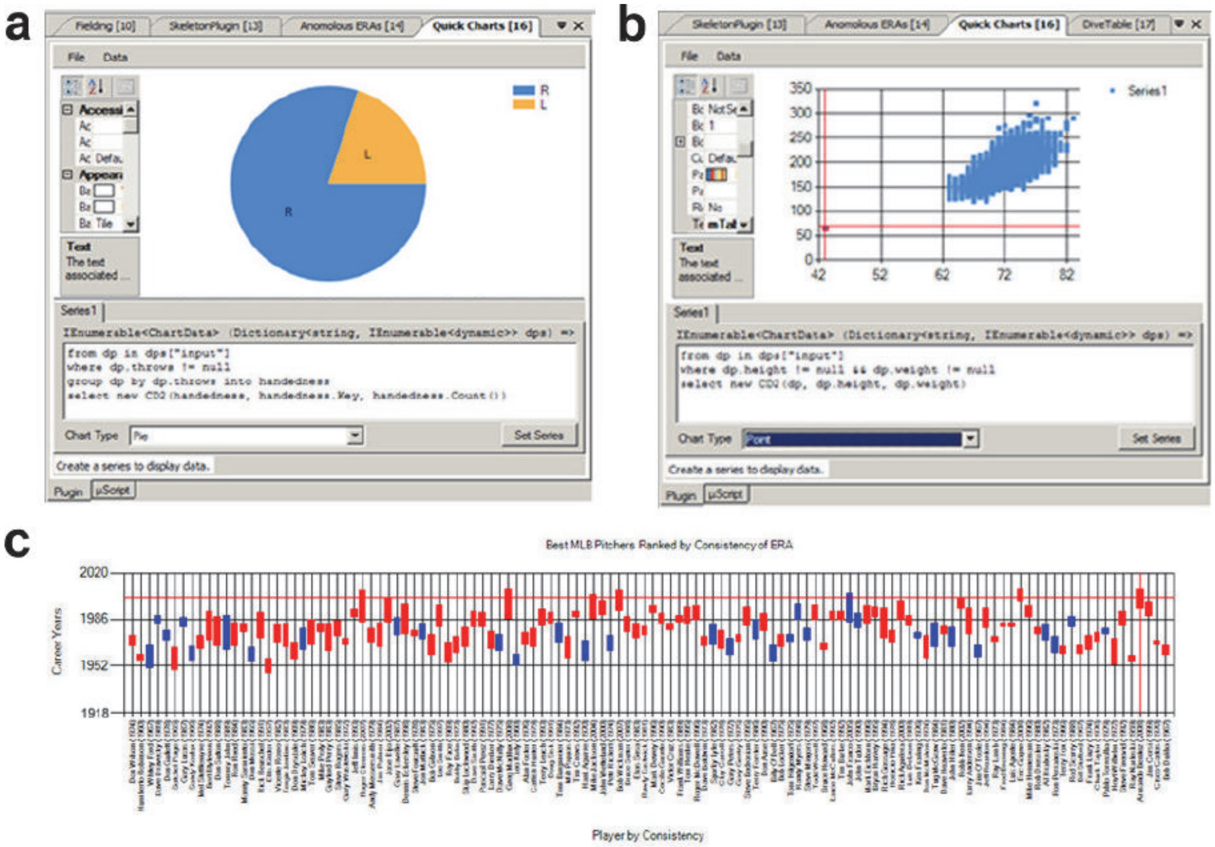
This demonstration is included in the downloadable DIVE software package. (a) Chart showing various types of carnivores. (b) Script to create histogram of carnivore subspecies from the species taxonomy. (c) Interactive view of raw data included in the taxonomy. This specific table was created by selecting the ‘Caniformia’ group in (a).



```
{

 plugin.BroadcastCSVData(@"C:\Users\dbromley\Downloads\lahman2012-csv\Master.CSV",   "Master");
 plugin.BroadcastCSVData(@"C:\Users\dbromley\Downloads\lahman2012-csv\Batting.CSV",  "Batting");
 plugin.BroadcastCSVData(@"C:\Users\dbromley\Downloads\lahman2012-csv\Pitching.CSV", "Pitching");
 plugin.BroadcastCSVData(@"C:\Users\dbromley\Downloads\lahman2012-csv\Fielding.CSV", "Fielding");
 plugin.BroadcastCSVData(@"C:\Users\dbromley\Downloads\lahman2012-csv\Teams.CSV",    "Team");

 plugin.mSession.BeginExecutionChain(plugin);


}
```

**Figure 3.7 Reusable DIVE components used to analyze professional baseball statistics.**

(a) Data loading requires one line per data file and one line to begin the load process. (b) Data loading is automatically parallelized across local processors. (c) Example plugin pipeline used for analyzing the professional baseball statistics.



```
IEnumerable<ChartData> (Dictionary<string, IEnumerable<dynamic>> dps) =>
from dp in dps["input"]
where dp.throws != null
group dp by dp.throws into handedness
select new CD2(handedness, handedness.Key, handedness.Count())
```

Chart Type: Pie

Set Series

Create a series to display data.

```
IEnumerable<ChartData> (Dictionary<string, IEnumerable<dynamic>> dps) =>
from dp in dps["input"]
where dp.height != null && dp.weight != null
select new CD2(dp, dp.height, dp.weight)
```

Chart Type: Point

Set Series

Create a series to display data.

**Figure 3.8 Reusable DIVE charting plugins used for data exploration.**

(a) Pie chart showing distribution of right-hand and left-hand pitchers. (b) Scatter plot of players' height and weight. Outlier identifies Edward Carl Gaedel, a major league player with dwarfism who played in 1951. (c) A more sophisticated DIVE chart illustrating relationships among career timespan, pitching handedness, and earned run average (ERA) consistency.

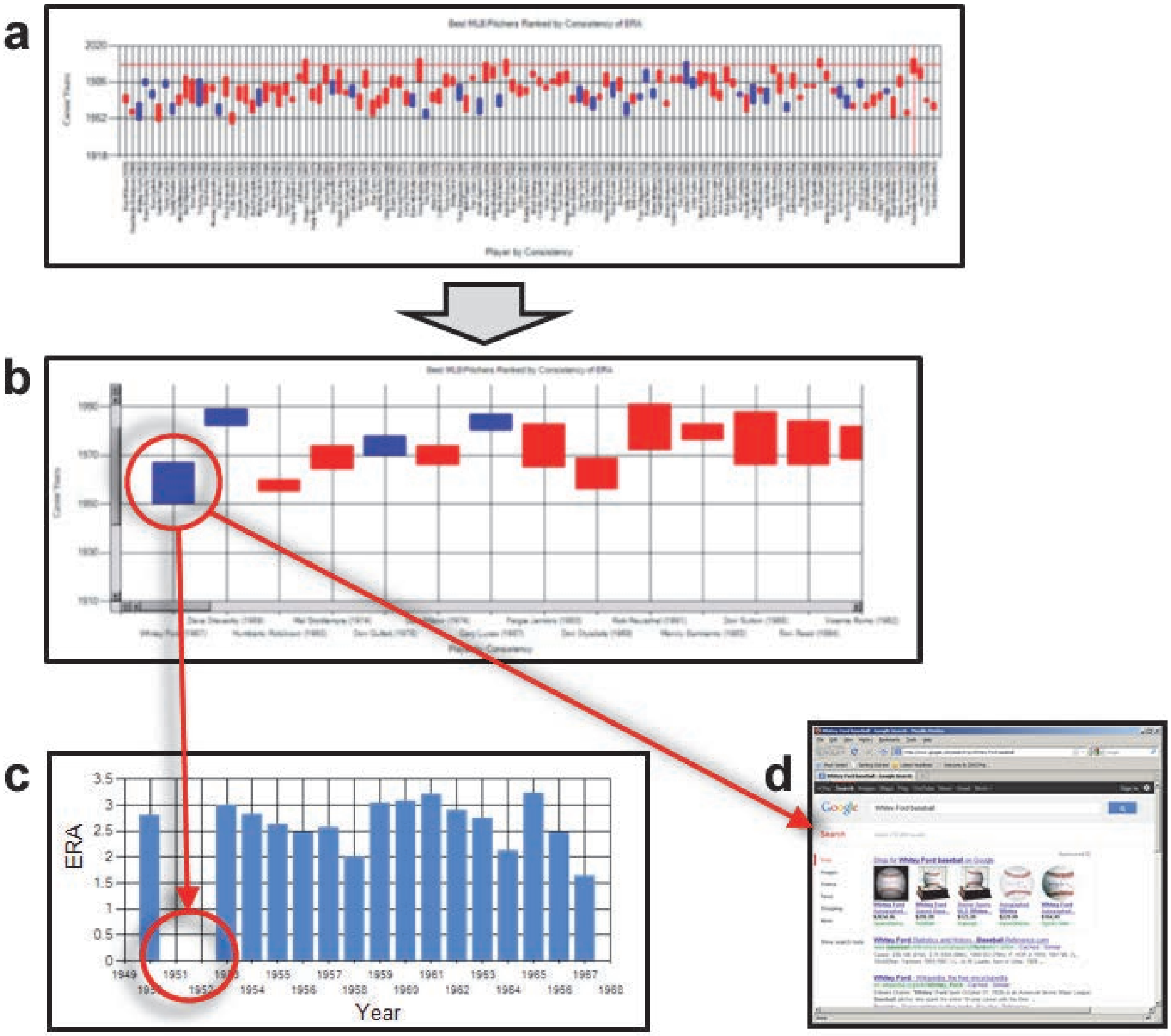
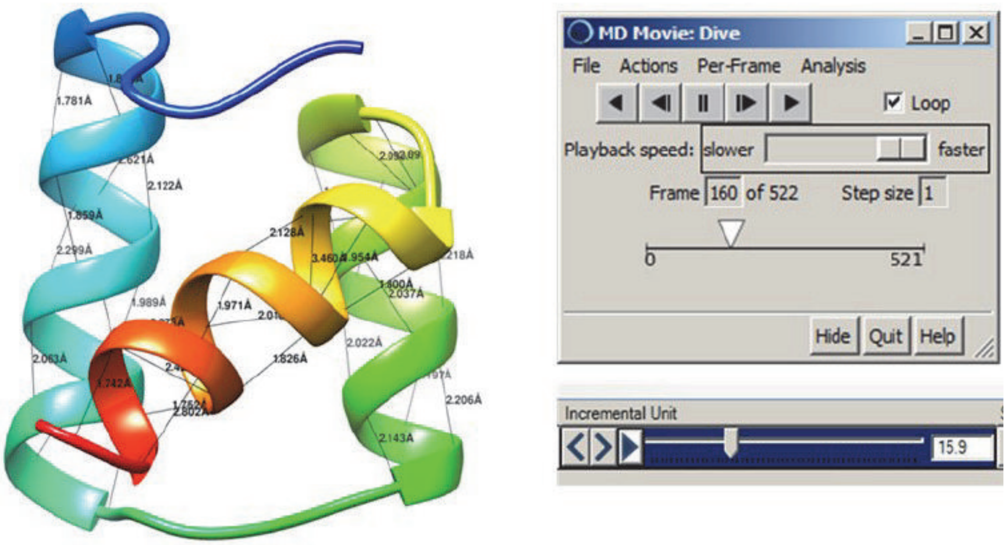


**Figure 3.9 Conceptual representation of DIVE interactions among various plugins.**

This investigation used various data sources to identify a player's absence due to military service. (a) DIVE chart illustrating relationships among career timespan, pitching handedness, and ERA consistency. (b) Close-up of chart shown in (a). DIVE supports interactive zooming of chart data. (c) Bar chart linked to chart in (b) through DIVE pipeline. Double-clicking on player data in (b) displays yearly ERA and launches a web search with the player's name. (d) Screenshot of web search automatically launched by DIVE.



☑ Loop

Frame 160 of 522 Step size 1

Incremental Unit 15.9

**Figure 3.10 DIVE using the Chimera plugin (Pettersen et al., 2004).**

Chimera is one of the major protein visualizers in use today. DIVE is able to incorporate tools like Chimera into a pipeline for scientific investigation.

Chapter 4

# THE DYNAMEOMICS API: AN APPLICATION PROGRAMMING INTERFACE FOR MOLECULAR DYNAMICS SIMULATIONS

## *4.1 Summary*

Data intensive scientific investigations often require the development of complex structures and relationships to interpret the large volumes of scientific data. These structures and relationships can be characterized in a variety of resources, including databases, standardized file formats, and ontologies. Application programming interfaces (APIs) provide specifics of how software components interact with these various resources as well as methods to operate on and manage these data. However, a significant development effort is required to design an efficient and useable API. Development of an API for protein simulation data presents a variety of challenges, including modeling multi-dimensional simulations and experimental data from multiple sources, addressing problems inherent in large data projects, and developing methods appropriate for researchers. Here we present the Dynameomics API, a software library developed to interface with the Dynameomics data warehouse and the Protein Data Bank, and to provide a variety of methods to access large volumes of protein related data.

## *4.2 Introduction*

Many scientific disciplines are generating exceptional amounts of data due to advances in computer storage, increased processing power, and the greater availability of supercomputing resources. These vastly enlarged volumes of data bring a multitude of informatics challenges, including data organization, accessibility, management, mining and analysis. Unfortunately, no one solution generally addresses all of these issues; it is often necessary to approach these challenges with a number of techniques and tools.

Databases, in many cases, offer an excellent mechanism to deal with the complexity of the data organization and management. For example, molecular dynamic (MD) simulation data have been organized into a data warehouse using a dimensional model for the Dynameomics project (van der Kamp et al., 2010). This data warehouse contains well-defined schemas representing protein simulations that define semantic structures, provide constraints to reduce the occurrences of erroneous data, and enforce policies to ensure new data is correctly inserted (Simms & Daggett, 2012; Simms et al., 2008). However, issues of data mining and analysis are still challenging, especially for non-expert scientists. Additionally, it can be difficult to undertake research endeavors that require multiple data sources, such as a data warehouse and a web service.

Application programming interfaces (APIs) can facilitate data mining and analysis and further address some of the informatics issues inherent in data intensive scientific investigations. An API can be comprised of standards, documentation or software libraries, either individually or in combination with one another. In all cases, these APIs programmatically specify how software components interact with one another. Specifically in the context of data-intensive scientific research, APIs can provide mechanisms to efficiently interact with these data resources as well as provide methods to operate on these data.

We have developed an API, called the Dynameomics API, to enable scientific research on protein data contained in the Dynameomics data warehouse (DDW) and Protein Data Bank (PDB) (Frances C. Bernstein et al., 1977). The Dynameomics API contains an object-oriented (OO) hierarchy that maps to specific facts (metrics or facts about the entities) and dimensions (discrete values used for classification of facts) of both the DDW and PDB. Protein-related methods are also contained in the core software library of the Dynameomics API to handle many aspects of protein-related research. Additionally, the API is designed for reading, writing, and updating the information whereas the DDW is generally for read-only access.

The Dynameomics API is central to several novel data access and analysis methods described in detail in Chapters 2, 5, and 6, covering simulation streaming, object parsing, protein fragments and rotamers. Additional research efforts that benefit from the Dynameomics API are outlined in Chapter 7. The following sections describe the Dynameomics API object hierarchy, methods and implementation. Supplemental technical information to the topics in Chapters 2 and 6, specifically simulation streaming and rotamer analysis, are also provided in detail here.

### *4.3 An Object Oriented Design for MD Simulations and Experimental Structures*

OO software is intended to characterize concepts using data fields to represent properties or attributes, methods to represent procedures, and inheritance and membership to represent relationships. In the case of MD, we use OO design to model molecular simulations and their associated analyses. Here we describe the major components of the Dynameomics API OO design and implementation, including the molecular structure representation, integration of the Molecular Mechanics Parameter markup Language (MMPL), data input and output, repository navigation, structural repositories, and data streaming optimizations.

The Dynameomics API contains an OO model and methods for representing and interacting with protein data. The model defines a unified semantic context for interpreting, analyzing, and modifying protein simulations and structures from either Dynameomics or the

PDB. The object hierarchy is generalized to represent both the simulation and structure dimensions and facts from the DDW and the experimental model and structure dimensions and facts from the PDB (Figure 4.1). This generalization uses a frame dimension to represent both the simulation dimension and experimental model dimension. We are able to make this generalization because both the simulation and experimental model dimensions represent a collection of instances of structures. Similarly, fragments, described in detail in Chapter 5, conform to this generalization via abstract interfaces with the frame dimension representing a collection of fragment search results.

### 4.3.1 *Molecular Structure Representation*

Figure 4.1 depicts the OO model the Dynameomics API uses to represent molecular simulations and structures. The OO elements portrayed in the figure have been simplified for presentation here, but convey the necessary overview of the interfaces and classes in the actual codebase. There are three distinct top-level branches to the hierarchy highlighted by the dotted line: Dynameomics, PDB, and fragments. The light grey boxes represent interfaces, which define the common properties and methods of the distinct classes at that level of the hierarchy. Classes defining specific object implementations for each hierarchy are shown in green. Memberships are depicted by the arrows with the associated text showing cardinality of the relationship.

An *IFrame* is an OO interface that defines properties and methods that are shared between the simulation dimension, experimental model dimension, and fragment search dimension, such as the number of frames, the starting frame, and functions to navigate to specific frames within the dimension. This interface is implemented by a *Simulation* class for Dynameomics simulations, a *PDBModel* class for PDB records, and a *FragmentSearch* class for the fragment results. These classes contain simulation, PDB model, or fragment specific properties and methods, such as the simulation ID or time step resolution for a simulation, a model to frame mapping for a PDB record, or the fragment search result ordering.

Similarly, an *IStructure* is an interface that defines general properties and methods common between Dynameomics and PDB structures. This interface defines properties such as the number of residues within a structure and the chain identifier. Common methods between the different types of structures include enumerating the member residues or adding individual atoms. The *IStructure* interface is implemented through separate classes for the Dynameomics and PDB structures, with the Dynameomics or PDB specific properties and methods defined in the classes themselves. Individual fragments are not considered complete protein structures, so they are not represented at this level of the hierarchy.

The next level of the hierarchy is defined by the *IPeptide* interface. Both Dynameomics and the PDB use the same class to implement the *IPeptide* interface and all levels of the hierarchy below the peptide interface. For Dynameomics and PDB structures, there is no significant distinction between the *IPeptide* and *IStructure* interfaces. On the other hand, the fragment hierarchy contains a unique class implementing the *IPeptide* interface which models short peptide structures for predictions and protein model building. *Fragments* are defined as small sections of structures and therefore are conceptually significantly different objects. A *Structure* can contain many *Fragments*, which is shown through the relationship between the *Fragments* class and *IStructure* interface.

The remaining interfaces and classes contain the same property and method definitions for each of the Dynameomics, PDB and fragment hierarchies. These represent the fundamental components of a protein structure, including the amino acid (residue), atoms and contacts. *Residues*, which usually correspond to amino acids but can represent other small molecules, contain a specific collection of atoms. A sample Unified Modeling Language (UML) diagram of the *Residue* and *Rotamer* classes is provided in Figure 4.2. *Atoms* are the smallest object analyzed in both Dynameomics and the PDB and each contains a *Coordinate* object, from which most analyses are calculated. We provide an *IVector* interface to the *Coordinate* object to simplify mathematical operations, which are common when dealing with protein structures. Finally, the *Contact* object represents an atomic contact between a pair of atoms.

Although the implementations of these remaining classes are the same for all hierarchies, the available data for the instantiated objects may differ between the data sources. Therefore, the way in which the information is populated into the instantiated objects may differ. For example, a *Residue* property specifies the φ and ψ angles of the peptide's backbone structure. The DDW stores these pre-calculated values in a relational table, so these properties are populated from the database for a Dynameomics residue. A PDB record does not contain pre-calculated φ and ψ angles, so these properties are calculated at runtime by Dynameomics API methods.

Interfaces are a powerful tool in this hierarchy because it allows objects in the different hierarchies to be used in the same way. Code can be written for a general structure interface, and work for both Dynameomics structures and PDB structures. Cross-hierarchy comparisons are also much more easily implemented, which is essential for analyzing data from multiple data sources.

### 4.3.2 *MMPL Integration*

MMPL is a markup language for describing molecular structure. The components for generating MMPL code are integrated into the DDW and used by our in-house simulation software, *ilmm* (Beck, Alonso, & Daggett, 2000). The Dynameomics API is designed to also use MMPL in conjunction with a spatial hashing function (Toofanny, Simms, Beck, & Daggett, 2011) to automatically generate the connectivity of protein structures. Using the shared object hierarchy described above and custom MS SQL stored procedures, the Dynameomics API is able to use MMPL for structures sourced from either the DDW or from the PDB. Fragments can also implement the MMPL functionality due to the shared peptide interface.

### 4.3.3 *Data Input, Output and Persistence*

As previously stated, the Dynameomics API is designed to read, write and transform data stored within its instantiated objects. Several parsers are integrated into the Dynameomics API to facilitate the population of the object hierarchies by reading data from either Dynameomics databases, the PDB online repository, or fragment libraries. These parser rules are based on both the syntax and semantics of the source data to appropriately populate the object hierarchies in the Dynameomics API. The parsers handle an array of source formats, including SQL databases, web services, and flat files.

Unlike the DDW and the PDB repository, data stored in the Dynameomics API is not inherently persistent. Objects are instantiated at runtime from the class definitions described above, but data in these objects do not remain in memory after the program is terminated. In order to persistently store object data, the Dynameomics API contains functionality to write to the standardized PDB file format (F. C Bernstein et al., 1977). This format is capable of recording much of the object state information and the majority of the semantic structure from each of the hierarchies.

The type of data in each of the instantiated objects will often differ from instance to instance due to the shared interfaces between the contrasting protein hierarchies. To address this issue, the PDB writer methods support customizable outputs at runtime with delegate. A delegate is a reference to a method encapsulated inside an object, which can be passed through a reference, much like a function pointer in C++. This allows the user to programmatically specify what data is written into the PDB file output. Figure 4.3 shows sample code for the peptide's PDB writer method and associated call to write only heavy atoms (N, O, C, Cα), sorted by atom number.

The Dynameomics API contains specialized output methods specifically designed to work with the DDW. These output methods include writing directly to structural repositories, as described below, or generating customized analysis data, such as the results for topics presented in Chapter 7.

### 4.3.4 *Repository Navigation*

The Dynameomics API contains libraries to navigate the DDW, the PDB online repository, the fragment repositories, and the rotamer repositories. The Dynameomics navigation library provides a number of search functions to identify the location of simulation data, simulation metadata, and associated properties. Simulations can be retrieved individually or in groups using simulation identifiers, PDB codes, structure identifiers, or keyword searches. Fragments and rotamers are retrieved using their respective identification systems, described below. The PDB navigation library searches the PDB repository with standardized PDB codes only.

Both of the navigation libraries expose the necessary metadata to initialize protein object hierarchies. The Dynameomics navigation library directs the Dynameomics API parsers to the correct SQL server and database locations within the DDW. This is automated for simulations, simulation properties, fragments and rotamers alike. The PDB navigation library directs the Dynameomics API either to a local copy of the repository to minimize internet traffic, or to the online PDB web service if necessary.

### 4.3.5 *Backing Stores*

Backing stores are a novel concept implemented in the Dynameomics API that replace the conventional location of object data with an in-memory database. Although the implementation adds a layer of complexity to certain objects within the Dynameomics API, it enables several useful technologies. First, it allows SQL-like queries to operate across objects which simplifies some commonly used protein analysis techniques. Second, it is fundamental to *Pass-through SQL*, which we describe in detail in Chapter 5. Here we provide a brief overview of the backing store implementation.

A backing store is a collection of tables that stores instance data for a set of related objects. Each table contains instance data for a homogeneous set of objects, and the table relations are defined by the object relations. The table structure, including unique row identifiers and column definitions, are specified for an object within that object's class definition at compile time. A row of data within a table maps to a single instance of an object. Data in the tables can be accessed for individual objects via conventional OO properties such as accessors, or for many objects simultaneously by querying the table data directly.

Technically the use of a backing store redirects the object property's pointer from a variable stored directly in memory to a specific cell within a backing store table. This cell is identified by a unique column (the property name), row (the object instance identifier), table (the object type), and store (the backing store identifier for the collection of objects). Sample code for the pointer redirection of the X coordinate on an atom object is provided in Figure 4.4.

### *4.4 Analysis Libraries*

The Dynameomics API supports a number of methods to analyze and transform protein structures. These include a math library, root mean square deviation (RMSD) library, φ-ψ library, secondary structure library, steric library, and sequence library. The math library handles the majority of geometric transformations and angle calculations. The RMSD is a specialized library to deal with protein structure alignment and makes heavy use of delegates for flexible method calls, much like the PDB writing method shown in Figure 4.3. The φ-ψ library contains methods to handle backbone angle analysis, while the secondary structure library translates between different secondary structure definitions such as DSSP (Wolfgang Kabsch & Sander, 1983) and φ-ψ. Finally, the sequence library provides methods to compare and score structural sequences using BLOSUM (Henikoff & Henikoff, 1992) and PAM (Dayhoff & Schwartz, 1978; Schwartz & Dayhoff, 1978) matrices. The math, RMSD and steric libraries make heavy use of the vector interface of atoms to perform functions like geometric transformations and angle calculations.

### *4.5 Structural Libraries*

The Dynameomics API generates and interfaces with two collections of structural repositories that are integrated into the DDW: the fragment libraries (Chapter 5) and the rotamer libraries (Chapter 6). Since fragment and rotamer libraries can be generated for different sets of protein structures, a repository is defined as a collection of these individual libraries. For each of these repository types, the Dynameomics API contains methods to automatically generate database schemas, populate the repositories using the API object hierarchies, search the resulting repositories, and incorporate the final data into other protein instances.

The schema for both repository types is automatically generated by the Dynameomics API. In the case of a fragment library, the schema is customized based on the protein structure data source, the range of fragment lengths in residues, and the server that will ultimately store the data. All table definitions, relationships, constraints, indexes, stored procedures and functions are generated and installed to a SQL server instance through a series of API calls.

SQL stored procedures and functions control the population of the data into the individual libraries, any necessary updates, and all search and retrieval operations. Access to these methods are available programmatically through the Dynameomics API and a subset is also exposed through Microsoft SQL server and web services. Using the Dynameomics navigation library previously mentioned, the Dynameomics API automatically distributes the operations across the servers within the DDW. Software engineering details of the rotamer library generation are provided below.

### 4.5.1 *Rotamer Libraries*

Rotamer libraries are generated by collecting statistics of similar side-chain structures. These rotamers are divided into bins based on their dihedral angle measurements. Each dihedral angle is dependent on the previous angle, so the bins are nested within one another. Each additional bin causes a combinatorial increase in the number of bins necessary to complete the calculation. Due to the large number of rotamers in Dynameomics, this calculation can become very memory intensive. Previous efforts have produced backbone-independent rotamer libraries (Scouras & Daggett, 2011) but, due to the combinatorial increase in dihedral bins required for backbone-dependent libraries, memory limitations caused the generation of backbone-dependent rotamer libraries to fail. The Dynameomics API implements new methods to support the generation of backbone-dependent rotamer libraries.

To overcome memory allocation issues previously experienced, we implemented a nested, sparse hashing function to store the metrics for each rotamer bin as needed. Each dihedral bin implements a hashed data structure instead of a pre-allocated array, meaning it is necessary to only allocate memory for bins with data. This data structure is accessible in constant time and, when fully populated, requires marginally more allocated memory than a conventional multi-dimensional array. However, due to the sparse nature of the rotamer library, the nested hash structure uses significantly less overall memory.

As mentioned above, rotamer library generation is distributed across the DDW but collection of the side-chain metrics is still a lengthy and server-intensive process. A standard deviation value is necessary for each rotamer calculation, but conventional standard deviation calculations require two passes over the data. In order to avoid collecting the side-chain metrics twice, we used an alternative standard deviation calculation described below.

The summary statistics consist of the average angle, the standard variance, the total count for each χ bin, and the probability of rotamer occurrence. A directional statistics approach (Gaile & Burt, 1980) was needed to properly assess angles by accumulating the *sin* and *cos* averages independently using the Equations 4.1, 4.2 and 4.3,

$$\bar{s} = \frac{1}{N}\sum_{i=1}^{N} \sin x_i \qquad 4.1$$

$$\bar{c} = \frac{1}{N}\sum\nolimits_{i=1}^{N} \cos x_i \tag{4.2}$$

$$\bar{\theta} = \left\{ \begin{array}{cc} \arctan\left(\frac{\bar{s}}{\bar{c}}\right) & \bar{s} > 0, \bar{c} > 0 \\ \arctan\left(\frac{\bar{s}}{\bar{c}}\right) + 180° & \bar{c} < 0 \\ \arctan\left(\frac{\bar{s}}{\bar{c}}\right) + 360° & \bar{s} < 0, \bar{c} > 0 \end{array} \right\} \tag{4.3}$$

where $x_i$ is the rotamer sample and $N$ is the number of samples. All angle metrics were converted to the range (0°, 360°] to properly use the equations. The standard deviation was calculated using Equations 4.4 and 4.5:

$$\bar{R} = \sqrt{\bar{s}^2 + \bar{c}^2} \tag{4.4}$$

$$\bar{V} = \sqrt{-2\ln(\bar{R})} \tag{4.5}$$

where $R$ is the resultant vector and $V$ is the standard deviation (Gaile & Burt, 1980). Due to the large number of samples, 64-bit primitives were used for calculations in conjunction with iterative equation updates to minimize floating point errors.

### *4.6 Implementation Details*

The Dynameomics API is intended to be used in a variety of programmatic interfaces. The assembly can be used in conjunction with command line interfaces, graphical user interfaces, standalone processes, and web services. In order to work closely with the DDW, the software was developed on the Windows Platform (Microsoft Corporation, 2007c) using the Microsoft .NET framework (Microsoft Corporation, 2007a) using mostly the C# and SQL programming languages. A major benefit of this platform is that the Dynameomics API can be deployed directly to Microsoft SQL server instances (Microsoft Corporation, 2007b) for improved computational performance at the point of data using Microsoft Common Language Runtime (CLR), reducing the necessity for data transfer over a network and exploiting the more powerful server hardware. Beyond the tight integration with the DDW, the use of the Microsoft .NET platform also allows the compiled Dynameomics API assembly to be used with a variety of additional languages (Gough & Gough, 2001).

### *4.7 Conclusions*

We have presented details of how MD simulations and experimental structures are modeled within the Dynameomics API. The API's methods operate seamlessly on both the DDW and the PDB, and its functionality extends to many types of analyses and applications. One example of this extension is the integration of the Dynameomics API into our Data Intensive Visualization Engine (DIVE), described in detail in Chapters 2 and 3. More generalized extensions of the Dynameomics API are presented in Chapter 7.

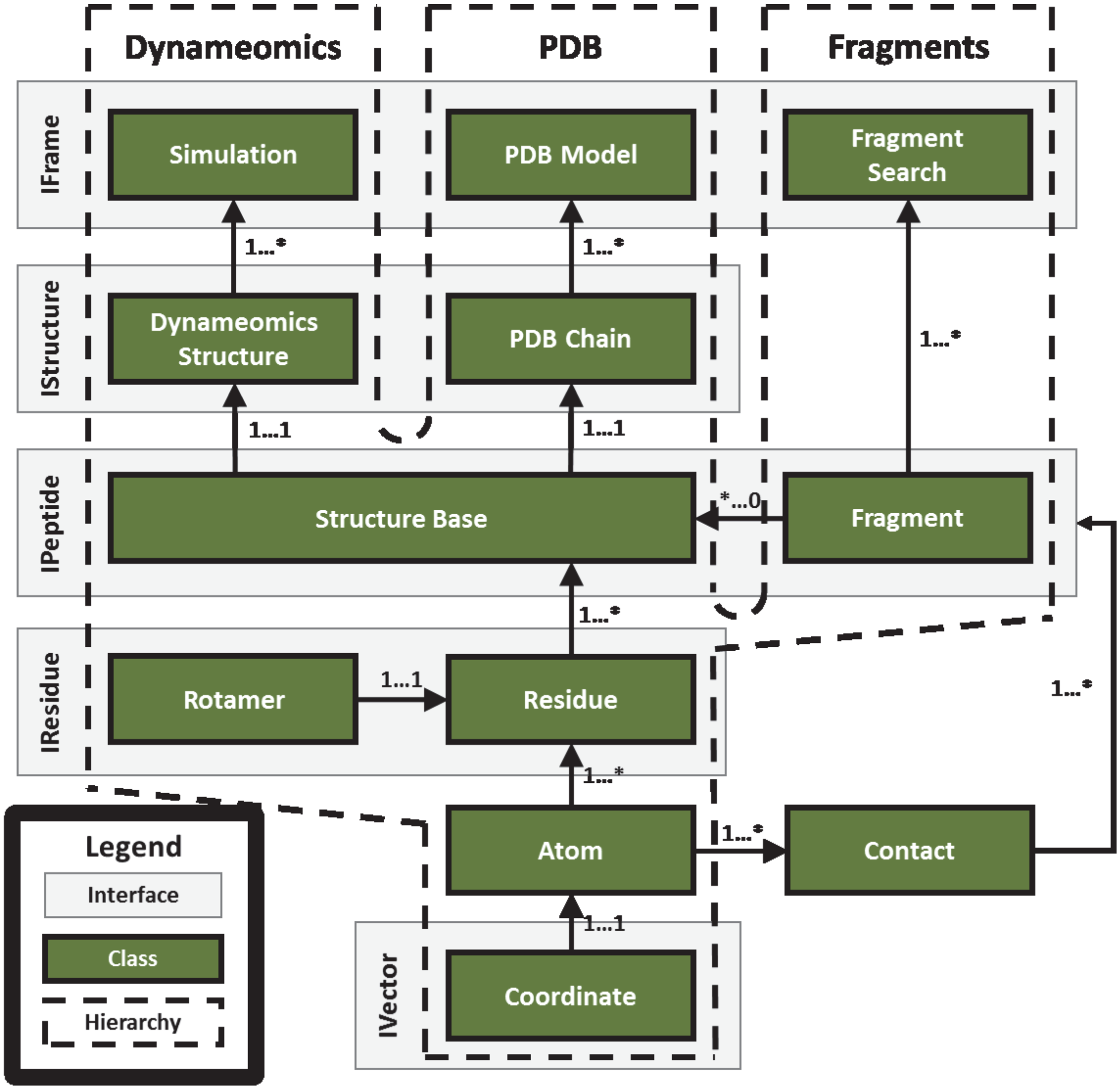


**Figure 4.1 Visual description of the Dynameomics API's object hierarchies.**

This graphic summarizes the object hierarchies within the Dynameomics API for modeling Dynameomics, PDB and Fragment protein simulations, records, and peptides. Interfaces are shown in light grey and classes (including abstract classes) are shown in green. The hierarchies are outlined with the dotted lines, which merge near the bottom as the hierarchies converge. Membership relationships and cardinalities are shown with arrows. This view represents a simplified version of the Dynameomics API to improve clarity.

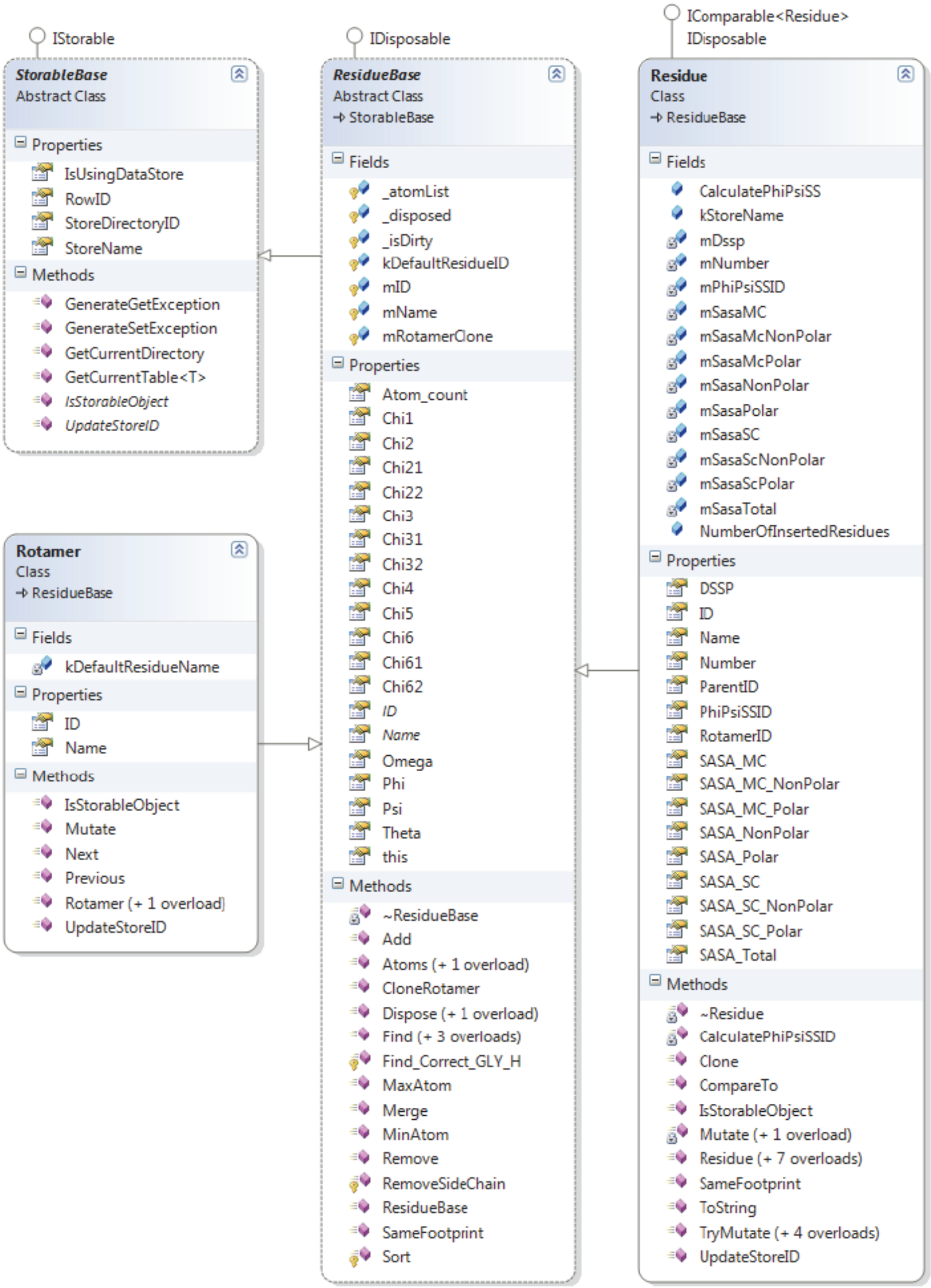
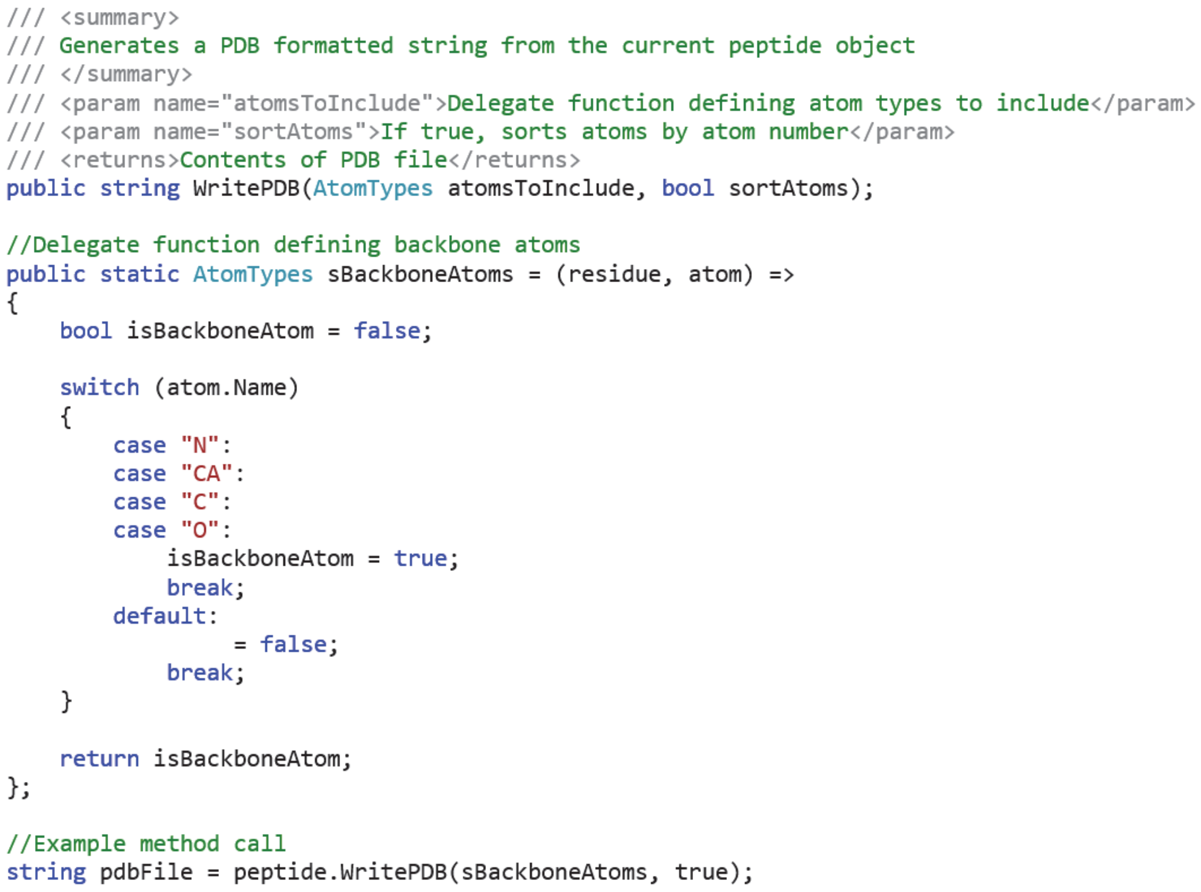


**Figure 4.2 UML diagram of residue and rotamer classes and abstract classes.**

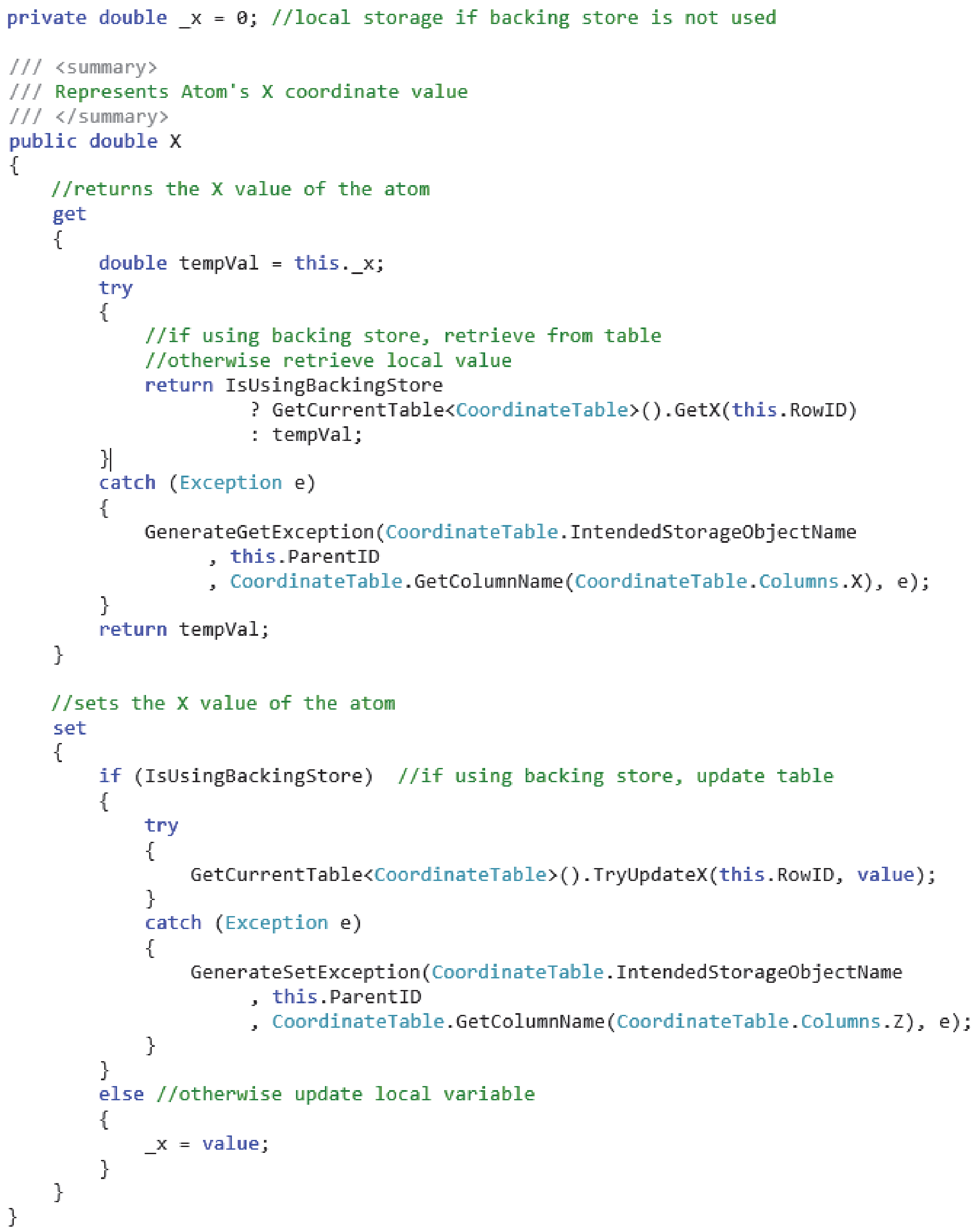
The UML diagram shows the fields, properties, methods and constructor definitions for each of the classes. The *ResidueBase* class is an abstract class that contains definitions shared between the *Rotamer* and *Residue* classes. The *StorableBase* class is an abstract class providing backing store functionality.



```
/// <summary>
/// Generates a PDB formatted string from the current peptide object
/// </summary>
/// <param name="atomsToInclude">Delegate function defining atom types to include</param>
/// <param name="sortAtoms">If true, sorts atoms by atom number</param>
/// <returns>Contents of PDB file</returns>
public string WritePDB(AtomTypes atomsToInclude, bool sortAtoms);

//Delegate function defining backbone atoms
public static AtomTypes sBackboneAtoms = (residue, atom) =>
{
    bool isBackboneAtom = false;

    switch (atom.Name)
    {
        case "N":
        case "CA":
        case "C":
        case "O":
            isBackboneAtom = true;
            break;
        default:
                 = false;
            break;
    }

    return isBackboneAtom;
};

//Example method call
string pdbFile = peptide.WritePDB(sBackboneAtoms, true);
```

**Figure 4.3 Sample code for PDB writer method call and delegate.**

This is sample code for writing a peptide structure to a PDB file format using the Dynameomics API methods and delegates. The delegate here specifies that only backbone atoms (N, O, C, Cα) be included in the PDB output.



```
private double _x = 0; //local storage if backing store is not used

/// <summary>
/// Represents Atom's X coordinate value
/// </summary>
public double X
{
    //returns the X value of the atom
    get
    {
        double tempVal = this._x;
        try
        {
            //if using backing store, retrieve from table
            //otherwise retrieve local value
            return IsUsingBackingStore
                     ? GetCurrentTable<CoordinateTable>().GetX(this.RowID)
                     : tempVal;
        }|
        catch (Exception e)
        {
            GenerateGetException(CoordinateTable.IntendedStorageObjectName
                 , this.ParentID
                 , CoordinateTable.GetColumnName(CoordinateTable.Columns.X), e);
        }
        return tempVal;
    }

    //sets the X value of the atom
    set
    {
        if (IsUsingBackingStore)  //if using backing store, update table
        {
            try
            {
                GetCurrentTable<CoordinateTable>().TryUpdateX(this.RowID, value);
            }
            catch (Exception e)
            {
                GenerateSetException(CoordinateTable.IntendedStorageObjectName
                     , this.ParentID
                     , CoordinateTable.GetColumnName(CoordinateTable.Columns.Z), e);
            }
        }
        else //otherwise update local variable
        {
            _x = value;
        }
    }
}
```

**Figure 4.4 Sample code for backing store implementation.**

This figure shows the methods necessary to add a backing store reference to the X coordinate of the atom class.  Most of the functional code for the backing store is located inside the try blocks for both the set and get methods.  The remaining code is predominantly for error handling.

Chapter 5

# DYNAMEOMICS: DATA-DRIVEN METHODS AND MODELS FOR UTILIZING LARGE-SCALE PROTEIN STRUCTURE REPOSITORIES FOR IMPROVING FRAGMENT-BASED LOOP PREDICTION

## *5.1 Summary*

Protein function is intimately linked to protein structure and dynamics yet experimental methods currently cannot determine structures for all proteins and often omit certain regions within a protein due to indeterminate data. Detailed experimental characterization of protein dynamics is even more challenging. We propose that atomistic molecular dynamics simulations provide a diverse sampling of biologically relevant structures that can improve structural modeling and structure prediction. Here we make use of the Dynameomics data warehouse containing simulations of representatives of essentially all known protein folds. We developed novel computational methods to efficiently identify, rank and retrieve small peptide structures, or fragments. We also created a novel data model to analyze and compare large repositories of structural data, such as contained within the Protein Data Bank and the Dynameomics data warehouse. Our evaluation compares these structural repositories for improving loop predictions and analyzes the utility of our methods and models. We find that the inclusion of Dynameomics structures in fragment-based methods improves the quality of loop predictions without being dependent on sequence homology. We provide cases where Dynameomics fragments provide better predictions for NMR loop structures than fragments from crystal structures. Online access to these fragment libraries is available at http://www.dynameomics.org.

## *5.2 Introduction*

Proteins play a critical role in nearly every cellular process. Protein structure and dynamics are critical to biological function. Loops constructing secondary structure segments are frequently essential to mediating biological function by forming the active sites and epitope binding sites of proteins. Specifically, the conformation and dynamics of loops are crucial in molecular recognition, protein-protein interaction, and ligand binding mechanisms (Fetrow, 1995; Leszczynski & Rose, 1986; Wu & Dean, 1996).

The intrinsic flexibility and mobility of loops makes these structures difficult to determine experimentally because they often adopt a multitude of conformations. Crystal structures commonly omit loop regions and, even though these structures are considered the gold standard (Spronk, Nabuurs, Krieger, Vriend, & Vuister, 2004). they can contain indeterminate experimental data (Eicken et al., 2002) and artifacts introduced from the crystallization process (Wagner et al., 1992). NMR spectroscopy can provide structural ensembles of loops in some cases, but this is generally only performed on smaller protein targets due to the method's size limitations. Additional structural information is needed to accurately model loops as they exist in their natural, solvated environments.

We propose that atomistic molecular dynamics (MD) simulations provide a diverse sampling of biologically relevant protein structures that can improve the quality of structural modeling in general and loop predictions in particular. To this end, we have developed novel computational methods to efficiently identify, rank and retrieve peptide fragments from structural databases using internal coordinates (IC). We also created novel data models to analyze and compare large repositories of structural data, namely the Protein Data Bank (PDB) (Berman et al., 2000) and the Dynameomics data warehouse (van der Kamp et al., 2010). Our evaluation analyzes and compares these structural repositories, identifies our contributions to improving loop predictions, and shows the utility of our methods and models.

Computational methods are an established mechanism for supplementing missing or poorly modeled loop regions in experimental data. In particular, database-driven methods are commonly used to predict loop regions in proteins using small peptide backbones, or fragments (Verschueren et al., 2011). These methods, considered fragment-based methods, produce excellent results (Baeten et al., 2008; Choi & Deane, 2010; Vanhee et al., 2011) but are often dependent on sequence homology, which underperforms when similar sequences are not available, or cluster representatives, which potentially eliminates specific loop conformations. Furthermore, most methods rely on relevant related structures being present in the PDB. However, PDB structures may be biased due to experimental methods (Jacobson, Friesner, Xiang, & Honig, 2002; Søndergaard, Garrett, Carstensen, Pollastri, & Nielsen, 2009) or be missing fragment conformations altogether. Fragment-based methods that use sequence similarity to bolster their performance also depend on matching sequences in the PDB, but matches are unlikely for longer loop regions. Many of these methods operate on relatively small repositories of structural data, such as subsets of the PDB, artificially limiting conformational diversity.

MD simulations can provide a plethora of structures to supplement information collected from the PDB (Beauchamp, Lin, Das, & Pande, 2012; Cino, Choy, & Karttunen, 2012). Structure variability in MD arises from dynamic motion instead of sequence variability. The Dynameomics project (Beck et al., 2008) in particular is well suited to structure prediction as it is a repository of MD simulations that includes representative structures from nearly all known protein fold families (Day et al., 2003; Schaeffer et al., 2011) and contains $10^4$ times as many protein structures than the entire PDB.  Current fragment-based methods are unable to scale to this amount of data, nor are they able to access the diverse metadata in the Dynameomics data warehouse.

Our findings show that Dynameomics fragments improve conventional loop predictions that use fragments from crystal structures without being dependent on sequence homology.  While Dynameomics fragments provide similar predictions of crystal loop targets as crystal fragments, overall predictions are improved when both types of fragments are employed in a consensus set. We also show that Dynameomics fragments can provide better predictions for dynamic, solvated loop structures.

Here we present a variety of results demonstrating how the Dynameomics fragments supplement the structural coverage of the PDB fragments.  We highlight where there are improvements in predicting loop structures using the Dynameomics fragments and present several instances where Dynameomics fragments provide better loop predictions than fragments sourced from crystal structures.  In the Methods Section, we present the general architecture of our fragment data model, called a library, and describe the associated search method to operate on large protein structure repositories.  This search method integrates into the distributed data warehouse architecture of the Dynameomics project (Simms et al., 2008; van der Kamp et al., 2010) to efficiently retrieve relevant fragment structures and is central to our data model.  We also elucidate a computationally efficient method to compare the structural coverage of the individual fragment libraries. Online access to these fragment data and models is available at http://www.dynameomics.org.

## *5.3 Results*

We assessed the utility of the Dynameomics fragment libraries in providing structures for improved loop prediction directly from the Dynameomics data warehouse by evaluating numerous fragment libraries generated using our methods and data models. These libraries are labeled and described in Table 5.1. First, we provide a comparison of the structural coverage of each of the fragment libraries. Second, we show how a consensus between PDB and Dynameomics fragment libraries can inform prediction algorithms to improve the overall results for crystal structure predictions. Finally, we present examples where fragments from the Dynameomics library better predict loops in NMR structures than fragments from the PDB library.

Our results are based on an internal coordinate (IC) method we developed for efficiently searching, ranking and comparing fragment structures. When comparing fragment structures against one another or a gap, the peptides were aligned using only the end, or anchor, residues of the fragment or gap. Detailed information on all aspects of these methods and evaluations are provided in the Methods Section.

### 5.3.1 *Comparison of Fragment Libraries*

The Dynameomics project contains native MD simulations of 807 fold representatives which, as determined during the development of our consensus domain dictionary,(Schaeffer et al., 2011) represent 95% of all known autonomous protein folds. In addition, we have multiple thermal unfolding simulations for each of the 807 proteins. By capturing the dynamics of these proteins using simulations, fragments generated from the relatively small number of 807 proteins can represent nearly all fragment structures in the PDB. To confirm, we first compared the library of Dynameomics starting structures ($DYN_{start}$, which is comprised of 576 crystal structures and

231 NMR structures) to the library generated from a broad subset of crystal structures in the PDB ($PDB_{xtal}$), detailed in Table 5.1. We represented each of the fragment libraries as a histogram and calculated statistics for the intersections and unions of the overlapping histograms as described in the Methods Section.

The fragments generated from the 807 static starting structures cover a small portion of the overall fragment conformational space of the 23,144 proteins in the $PDB_{xtal}$ fragment library (Figure 5.1). Aside from very short peptides comprised of a few residues, the $DYN_{start}$ set captured little of the structural diversity of fragments in the $PDB_{xtal}$ library. The $DYN_{298}$ set contains the structures for the 807 proteins simulated over time at 298K. The coverage of the $DYN_{298}$ fragment library was much more comprehensive due to the incorporation of native state dynamics. As shown in Figure 5.1, the coverage for the $DYN_{298}$ and $PDB_{xtal}$ libraries was very similar. This confirms that the simulations of Dynameomics domain representatives provided excellent coverage of PDB structures despite using only 1/30th of the number of unique experimental structures.

The next analysis we performed was to compare fragment structural coverage between the $PDB_{xtal}$, $PDB_{nmr}$, and $DYN_{all}$ fragment libraries. The $DYN_{all}$ library contains the $DYN_{298}$ set as well as structures from high temperature (498K) unfolding simulations of the 807 proteins, referred to as the $DYN_{498}$ set. As shown in Figure 5.2, most of the conformational space was shared between the three libraries for all residue lengths. The majority was comprised of the unique $DYN_{all}$ conformations and conformations shared between the $PDB_{nmr}$ and the $DYN_{all}$ fragment libraries. On average, 12.8% of the known conformational space was represented only in the

$\mathrm{DYN}_{\mathrm{all}}$ and $\mathrm{PDB}_{\mathrm{nmr}}$ fragment libraries, meaning these experimentally derived conformations are modeled by Dynameomics but are not contained within the crystal structures in the PDB.

### 5.3.2 *Evaluation of MD Fragments for Improving Prediction*

To compare the quality of loop structure predictions between different fragment sources, we made structure predictions using each of the $\mathrm{PDB}_{\mathrm{xtal}}$, $\mathrm{DYN}_{298}$ and $\mathrm{DYN}_{498}$ libraries. Predictions were scored by heavy-atom (N, O, C, Cα) RMSD and all predictions using homologous structures were removed using a sequence similarity metric, as described in the Methods Section. We used a standard loop test set provided by Choi and Deane that consists of 510 target loop regions.(Choi & Deane, 2010) These loop regions are located between two secondary structure regions and range in length from 4 to 20 residues. There are 30 sample loops for each loop length. Table 5.2 shows the average heavy-atom RMSD and standard deviation of the best predictions for each loop length. The best prediction is the fragment with the lowest heavy-atom RMSD in the search result set generated for each target.

We performed this test individually on the $\mathrm{PDB}_{\mathrm{xtal}}$, $\mathrm{DYN}_{298}$, and $\mathrm{DYN}_{498}$ fragment libraries, as well as a consensus test between all three fragment libraries. The $\mathrm{PDB}_{\mathrm{xtal}}$ fragment library on average had the lowest RMSD scores for the best prediction metrics. However, this average was not representative of each individual predicted loop structure. A substantial number of the 510 loop structures were best predicted with one of the two Dynameomics-based fragment libraries. Figure 5.3 shows how many of the 30 best-performing predictions originated from each repository. Over the entire set of 510 loop structures, 47% originated from Dynameomics fragment libraries and 53% originated from the $\mathrm{PDB}_{\mathrm{xtal}}$ fragment library. The average improvement in RMSD from the Dynameomics fragment libraries is listed in the bottom-right of

Figure 5.3. Loops with a length of 20 residues had the highest proportion of best predictions originating from Dynameomics, with 73% originating from Dynameomics fragment libraries (27% from $DYN_{298}$ and 47% from $DYN_{498}$) and 27% originating from the $PDB_{xtal}$ fragment library. Predictions for 20 residue loops had an average improved RMSD of 2.53Å over their counterparts originating from the $PDB_{xtal}$ library.

Searching across all three repositories simultaneously results in a consensus set. The $PDB_{xtal}$, $DYN_{298}$, and $DYN_{498}$ repositories first provide a result set for each target loop using the IC method, then these result sets are combined to provide a best overall prediction. The average heavy-atom RMSD and standard deviation is shown in the last two columns of Table 5.2. Due to the contributions of the $DYN_{298}$, and $DYN_{498}$ libraries, the consensus set outperforms all other libraries for every loop length. This improvement is quantified in Figure 5.3.

### 5.3.3 *Case Studies of Improved Prediction*

To investigate the quality of our prediction methods for solvated structures, we identified loops in NMR ensembles with a large number of experimental Nuclear Overhauser Effect (NOE) crosspeaks so we could quantitatively evaluate our predictions. Here we present several instances of NMR loop structures that were best modeled by fragments in the $DYN_{298}$ fragment library. Two of these loop structures and associated predictions are shown in Figure 5.4. The NMR structure is in grey, the fragment sourced from the $PDB_{xtal}$ library is in red, and the fragment sourced from the $DYN_{298}$ library is in blue. A loop structure of 12 residues is shown in Figure 5.4a. Here the $PDB_{xtal}$ fragment had an RMSD of 3.4 Å and NOE fulfillment of 61% while the $DYN_{298}$ fragment had an RMSD of 1.9 Å and NOE fulfillment of 79%. Similarly, in Figure 5.4b depicting a loop structure of 15 residues, the $PDB_{xtal}$ fragment had an RMSD of 4.5 Å and NOE fulfillment of 66% while the $DYN_{298}$ fragment had an RMSD of 1.3 Å and NOE fulfillment of 91%. The $DYN_{298}$ predicted fragment fulfils the same percentage of NOEs as the NMR derived structure. Notably, for NOEs that were not satisfied, the $PDB_{xtal}$ structures have much larger average violations. In both cases, the $DYN_{298}$ prediction outperforms the $PDB_{xtal}$ prediction in both heavy-atom RMSD and NOE satisfaction.

## *5.4 Discussion*

We have evaluated the utility of Dynameomics structural fragments in predicting loop regions of proteins. Our primary goal was to develop methods and data models to efficiently access and analyze the large numbers of fragments in both the MD and experimental structure repositories. We desired to identify the underlying differences in fragment conformations between the Dynameomics and PDB repositories and in what contexts the Dynameomics fragments are applicable to loop prediction.

From Figure 5.1 it follows that the collection of native state Dynameomics structures contain fragments with conformational coverage similar to that of the crystal derived fragments from the PDB. It is important to note that the fragments generated only from the Dynameomics starting structures have extremely poor coverage as compared to the $PDB_{xtal}$ fragment library. Although these structures are representative of most of the known protein folds, it is only through MD that the resulting fragments exhibit a wide range of relevant conformations. Since the sequence variability in Dynameomics fragments is low relative to the $PDB_{xtal}$ fragments, sequence similarity is not needed in order to make accurate structure predictions with Dynameomics fragments.

We established that Dynameomics contains fragment conformations that are also found in solvated NMR structures, but are not commonly identified through X-ray crystallography methods. This is apparent from the overlapping $\text{PDB}_{\text{nmr}}$ and $\text{DYN}_{\text{all}}$ regions colored in magenta on Figure 5.2, representing multidimensional histogram bin overlaps as described in the Methods Section. After investigating the locality of these regions in conformational space, we found that they occur in the outer layers and extended tail of the multidimensional histograms. These bins may contain more transient and flexible conformations, which is characteristic of solvated loops. This implies that Dynameomics structures capture flexible conformations, which are observed in NMR structures, that X-ray crystallography methods cannot currently capture.

Predictions from the consensus set of $\text{PDB}_{\text{xtal}}$ and Dynameomics fragment libraries are better than predictions from either individual library, as shown in Figure 5.3 and Table 5.2. Dynameomics fragments can therefore provide additional structure information for fragment-based methods and improve loop structure predictions. One interesting observation is that all results have overall high standard deviations. This is due to the spread between good predictions and bad predictions; predictions are either extremely similar to the actual loop or fall into a broad distribution of high-scoring predictions, causing the high deviation from the average RMSD.

Finally, we show that Dynameomics outperforms predictions from the $\text{PDB}_{\text{xtal}}$ for both NMR structures in Figure 5.4. The $\text{DYN}_{298}$ predictions deliver lower RMSD scores and higher NOE satisfaction rates as compared to the $\text{PDB}_{\text{xtal}}$ predictions. Furthermore, the average violation for unsatisfied NOEs is much higher for the $\text{PDB}_{\text{xtal}}$ predictions.

The methods and data models we developed, detailed in the Methods Section, work for either MD or experimental structure repositories. There are several advantages to these methods and models that make them advantageous for big data architectures. First, internal coordinates, unlike Cartesian coordinates, do not require alignment calculations as they are already intrinsically aligned. The necessary computational power for large-scale comparisons is therefore vastly reduced. Second, internal coordinates naturally work with Structured Query Language (SQL) filters to obtain small, relevant result sets quickly and efficiently instead of performing the all-by-all matrix used in conventional methods. Third, the data model does not require explicit storage of Cartesian coordinates, but only the pre-calculated distances between atoms required for the internal coordinate method. This lends itself well to distributed architectures such as Dynameomics and is easily applied to repositories such as the PDB. Due to the flexibility of these approaches, fragment libraries can be created from any structure repository and searches can be run on one or many of the libraries simultaneously.

It is noteworthy that the Dynameomics fragment library is explicitly linked to the Dynameomics data warehouse. As such, a variety of analyses described in detail by van Der Kamp *et al* (van der Kamp et al., 2010) are also easily accessible. Of particular interest may be the solvent accessible surface area analysis, which provides additional context-specific information for generation of protein surface loop ensembles (Shehu & Kavraki, 2012). Flexibility analysis (Benson & Daggett, 2008) may prove beneficial for predicting highly flexible protein structures such as intrinsically disordered regions. Side-chain conformations for fragment backbones can be predicted using the linked Dynameomics rotamer libraries (Rysavy, Towse, et al., 2014; Scouras & Daggett, 2011). Furthermore, additional high-resolution dynamic structures in the Dynameomics data warehouse are readily accessible to the fragment data model, providing an expedient method for fine-tuning of fragment structures.

## 5.5 Methods and Materials

### 5.5.1 Protein Structure Collections and Fragment Libraries

We created six libraries of fragment structures for analysis. These libraries originated from two distinct sources of protein structures: experimentally derived structures and molecular dynamics (MD) derived structures. Two of the libraries were generated from the largest repository of experimentally derived structures, the PDB. The remaining four libraries, representing MD structures, were generated from the Dynameomics data warehouse. Descriptions and statistics for each of the resulting fragment libraries are shown in Table 5.2. Fragments of length 3 to 22 residues were generated for each library to support structure predictions of 1 to 20 residues in length.

The first library of experimental fragments, abbreviated $\mathrm{PDB}_{\mathrm{xtal}}$, was extracted from the majority of crystal PDB structures using a PISCES (Guoli Wang & Dunbrack Jr., 2003) query to filter out low-quality structures. The query specified a sequence identity of 95% or less, a resolution better than 2.7Å, and an R-factor of 0.3, resulting in approximately 23.1 x $10^3$ protein structures. The second set, abbreviated $\mathrm{PDB}_{\mathrm{nmr}}$, was extracted from the majority of NMR spectroscopy derived PDB structures using similar constraints of sequence identity of 95% or less, a minimum length of 40 residues, and deposits had to include experimental data. All models for each NMR structure were included. This query resulted in approximately 100.9 x $10^3$ NMR structures.

The Dynameomics project, which was created to represent the dynamic ensembles of a vast diversity of structural folds in proteins (Day et al., 2003), was used as the source repository of MD structures (Simms & Daggett, 2012; Simms et al., 2008). For this specific analysis we used the

Dynameomics v2009 Release Set (Schaeffer et al., 2011) which contains structural representatives of 95% of the known autonomous protein folds. This set contains 807 distinct protein targets (http://www.dynameomics.org). These proteins were simulated using *il*mm (*in lucem* molecular mechanics) (Beck & Daggett, 2004), which employs the Levitt *et al.* force field (Levitt, Hirshberg, Sharon, & Daggett, 1995) and uses explicit water molecules (Beck, Alonso, & Daggett, 2003; Levitt, Hirshberg, Sharon, Laidig, & Daggett, 1997) in the simulation. All Dynameomics target structures were simulated a minimum of one time at 298K and twice at 498K, with each simulation running for at least 51 ns. The atomic coordinates of these structures were recorded at 1 ps granularity and stored in the Dynameomics data warehouse (van der Kamp et al., 2010). More details regarding the simulation protocols can be found elsewhere (Beck & Daggett, 2004; Beck et al., 2008).

We generated four fragment libraries from Dynameomics. The $\mathrm{DYN}_{298}$ library contains fragments from the native state simulations run at 298K and is used as a direct comparison to the $\mathrm{PDB}_{\mathrm{xtal}}$ library. The $\mathrm{DYN}_{498}$ library contains fragments from the unfolding simulations run at 498K and is used to capture more conformational variety. The $\mathrm{DYN}_{\mathrm{start}}$ library contains fragments from the minimized starting structures of the 807 Dynameomics targets. Finally, the $\mathrm{DYN}_{\mathrm{all}}$ is the combination of both the $\mathrm{DYN}_{298}$ and $\mathrm{DYN}_{498}$ libraries.

Since Dynameomics structures within a simulation are time-dependent, we did not generate fragments from every structure. Sampling fragments at every time point is largely redundant for the applications presented in this manuscript since little structural variation occurs within a fragment for short sampling intervals. For the native state simulations, we generated fragments from structures at 1 ns intervals to optimize conformational variability while minimizing computer resources. For the unfolding simulations, we sampled at a higher rate of 100 ps due to the increased motion of the proteins in high-temperature simulations, and the structures were retrieved from the last 15 ns of the unfolding simulations.

### 5.5.2 *Internal Coordinate Scoring*

To scale fragment search, retrieval, and matching to a large, distributed structural repository, it is advantageous to pre-process the coordinate data of protein structures and represent the fragment conformations with minimal information loss. We characterized the structure of protein backbone fragments and gaps using internal coordinates (IC). This method is derived from a Cartesian coordinate representation instead of the commonly used torsion angle representation, as suggested by Holmes and Tsai (Holmes & Tsai, 2004). Using an IC representation that we previously briefly introduced (van der Kamp et al., 2010), the inter-residue and intra-residue distances between the five heavy atoms of each terminal residue of the fragment or gap represent a unique structural identifier. More explicitly, this identifier is comprised of 45 distances between each residue's N, O, C, Cα, and Cβ atoms as shown in Figure 5.5. Specific distances are referred to using the convention *Xs-to-Xe*, where *Xs* represents the starting residue's X atom and *Xe* represents the ending residue's X atom. Starting and ending residues are determined using a backbone's N-Terminus to C-Terminus directionality.

A single-valued IC score was used to evaluate the similarity between two fragments or the fit of a fragment to a gap in a protein structure. This IC score was calculated using a root-mean-square deviation calculation (RMSD)

$$RMSD_{IC} = \sqrt{\frac{1}{D}\sum_{i=1}^{D}(x_i - y_i)^2} \tag{1}$$

where $x_i$ and $y_i$ are equivalent IC atom-atom distance pairs from the fragment and gap being compared and $D$ is the number of distance in each fragment or gap. Since the distances are relative to the end residues, no alignment is necessary before calculating the IC score between fragments or gaps.

### 5.5.3 *Fragment Database Schema*

The fragment database schema was designed to integrate into the Dynameomics data warehouse architecture while supporting efficient fragment searching and filtering using the IC distances and the IC score. A simplified schema for a Dynameomics-based fragment collection is shown in Figure 5.6. The *Fragments* table stores metadata about the source protein structure for retrieval of the original PDB file or Dynameomics simulation. The fragments themselves are defined with a combination of two table types, *Fragments_X* and *FragmentsFrame_X*, where *X* denotes the length of fragments contained in the respective table combination. The *Fragments_X* table defines a unique fragment per individual protein structure. The *FragmentsFrame_X* table defines a unique instance of each structural fragment (i.e. individual conformations of an individual fragment's dynamic motion) along with each fragment instances' IC distances. The fragment frame unique identifier is unique among all fragment lengths within a fragment repository. Additional, fragment-specific analysis can be easily added through the generic, extensible *Fragments_Analysis* table definition.

Fragments can be filtered on any combination of or all individual IC distances. This filtering is implemented by including a range of desired distance thresholds in the SQL query *WHERE* clause per individual IC distances. The IC score is calculated per result row using an inline SQL function. Every IC distance is indexed within each *FragmentsFrame_X* table to accelerate the filtering process.

### 5.5.4 *Fragment Identification and Retrieval*

Our fragment libraries were designed to provide structurally relevant fragments based on their IC profile. Anchor residues are first identified in the protein structure of interest and the corresponding IC profile of those anchor residues is calculated. A query is then generated for a fragment repository, calculating the $\mathrm{RMSD}_{\mathrm{IC}}$ for all fragments of a specified length that also contain IC distances within the specified distance threshold. The default threshold is 1.5 Å to accommodate common bond variance, but can be specified at query time. The number of results is adjustable; we specified 200 fragments for all results presented in this paper. This query runs several orders of magnitude faster than the naïve approach, as is shown in Figure 5.7.

The query provides a collection of fragments enumerated by unique fragment identifications as defined in the fragment repository schema. For Dynameomics fragments, this unique identifier corresponds to a unique tuple consisting of a simulation identifier, structure identifier, structure instance, time step, starting residue and length. In the case of a PDB fragment, the unique tuple consists of a PDB code, chain identifier, model number, starting residue and length. In both cases, this information is sufficient to retrieve a unique fragment structure from either the Dynameomics data warehouse or PDB, respectively.

The fragment retrieval operation returns the coordinates of the heavy atoms (N, O, C, Cα, and Cβ) in a polyalanine peptide representation generated from the backbone of the source structure. Alanines are used to retain the Cβ coordinate information and first chi dihedral angle for improved side-chain attachment. In the case of Glycine residues, the Cβ coordinates are estimated using the chiral-appropriate hydrogen atom attached to the Cα.

### 5.5.5 *Fragment Insertion*

Two related but distinct methods can be used for the insertion of fragment structures into the protein structure of interest as shown in Figure 5.8. The first method is intended to fill a gap in an existing structure with a relevant fragment or collection of relevant fragments. The second method was developed to extend the termini of existing protein structures. Both methods take advantage of the IC profile to search existing fragment structures in the fragment repositories.

Gap-filling fragment queries use the IC profile of the gap anchor residues. Fragment results are aligned by matching the fragment anchor residues with the gap anchor residues using a heavy atom alignment of the five heavy atoms in each end-residue (N, O, C, Cα, and Cβ). Extension fragment queries use the IC profile of the terminal three residue fragment of the N- or C-terminus. Fragments retrieved for extending a C- or N-terminus were aligned using the three end residues of the protein structure of interest and the corresponding side-chain conformations can optionally be included using the Dynameomics rotamer libraries (Scouras & Daggett, 2011).

### 5.5.6 *Fragment Evaluation*

All RMSD values presented in the results were obtained by calculating the RMSD over all backbone heavy atoms (N, O, C, Cα) contained in the internal residues of the fragment. The anchor residues were not included in these metrics. All backbone heavy atoms were checked for steric clashes with the destination structure atoms. Side-chain conformations were not incorporated in the evaluations in this manuscript as we were focused solely on backbone conformations. In all instances, predictions are never made using fragments sourced from the same structure as the target structure or any MD-generated derivative structure of the target structure.

All fragments sourced from structures with high sequence similarity to the target structure were eliminated from consideration for predictions. This evaluation was done on a per-target basis and was to ensure that no homologous structures were used in the prediction results. Structures containing 50% or greater sequence similarity to the target loop's originating structure were eliminated from the search results for that target loop. We used the .NET Bio software package to perform the alignments and calculate the sequence similarity metric (Outercurve Foundation, 2013).

Evaluations of loop structure predictions were also performed against NMR ensembles using Nuclear Overhauser Effect (NOE) data. In these instances, we only considered NOEs containing at least one atom within the backbone of the target range. NOEs between side-chain atoms in this range and atoms outside of the target range were not considered. NOEs between atom-pairs altogether outside of the target range were also not considered as these would evaluate identically between candidates.

### 5.5.7 *Representing Structural Diversity in Fragment Libraries*

In order to analyze the structural distribution of the fragment populations in each library, we needed a method to programmatically represent the conformational space. To do this, we used multidimensional histograms as a computationally efficient method of clustering. These also lend themselves well to comparison due to the common bin definitions between different histograms. Fragments were binned using a subset of the end-to-end distances, which were selected to maximize the overall fragment structure representation while keeping correlation between the selected distances to a minimum. These distances were identified using principal component analysis (PCA). The resulting histogram bins contain clusters of similarly-structured fragments.

The specific distances we used in our histograms were *Cβs-to-Cβe*, *Os-to-Ne*, and *CBs-to-CAe*. As shown in Table 5.3 through Table 5.5, these distances most often had the highest correlation with the first three principal components and the percentage of variance captured was 84.4% to 99.4% so no additional distances were needed. This finding was consistent across the majority of fragment lengths in each repository. We used a histogram bin size of 0.1 Å as this results in computationally manageable histogram sizes while maintaining a valid representation of the structural diversity in the fragments.

### 5.5.8 *Comparison of Fragment Libraries*

We compared the histogram data for pairs or triplets of fragment repositories to evaluate the coverage of fragment structures in each repository. Since each histogram bin represents a cluster of similarly structured fragments, we were able to simplify each histogram bin representative to a binary value. A bin containing two or more fragments was considered to be filled while a bin containing one or fewer representative fragments was considered empty.

The sets of binary histogram bins from different repositories were aligned to complete the comparison. Bins with the same distance thresholds were considered to be in alignment between datasets. Four outcomes were possible for each bin when comparing two repositories: both repositories contain a representative fragment, only the first repository contains a representative fragment, only the second repository contains a representative fragment, or neither repository contains a representative fragment. When comparing three repositories, eight outcomes are possible for each bin. In this way we can quickly ascertain which fragment structures exist in each of the fragment collections. This technique was used for the repository comparisons in the results section.

### 5.5.9 *Internal Coordinate Performance*

We first evaluated the ability of the IC algorithm and fragment database schema to recover specific fragment structures. We selected fragment structures of length 3, 13 and 22 each of secondary structure type α-helix, β-strand and loop from crystal structures below 2.0Å resolution. The end residues of these fragment structures were then used to query both the $\mathrm{PDB}_{\mathrm{xtal}}$ fragment library and the $\mathrm{DYN}_{298}$ library. Each of the nine fragment searches in the $\mathrm{PDB}_{\mathrm{xtal}}$ fragment database resulted in an exact match from the PDB. Similarly, the nine fragment searches in the $\mathrm{DYN}_{298}$ fragment library resulted in a match of the closest fragment match. Although the search structures do not exist in the $\mathrm{DYN}_{298}$ fragment library, the minimized starting structures of simulations generated from the respective crystal structures were identified as the top match. These recovered Dynameomics structures and search fragments are shown in Figure 5.9.

A simple benchmark was also run to assess the query time of the IC fragment search and retrieval. Searches requesting 200 results were performed for each fragment length of 3 to 22 residues. The time was calculated, from start to completion, for three iterations of each search and averaged for each fragment length. This process was repeated for results containing just fragment metadata (IC without retrieval) and results containing the full fragment backbone structures (IC with retrieval). We also estimated the running time of a naïve approach which would consist of the heavy-atom RMSD of the end residues between the target fragment and all fragment instances of the same length. As is shown in Figure 5.7, the IC search algorithm is several orders of magnitude faster than the naïve approach.

### 5.5.10 *Principal Component Analysis*

We applied principal components analysis (PCA) to define a common set of representative distances between each fragment library for the comparison of the fragment structural space. The heavy-atom-pair distances between the terminal residues of each fragment were used to inform the analysis. This process identified the atom-pair distances which best represent the overall fragment structures and have minimal correlation with one another. A summary of the PCA analysis over all fragment lengths is provided in Table 5.3 through Table 5.5. We found that 84.4% to 99.4% of the structural variance was captured in the first three principal components for all of the sampled fragment lengths from all fragment libraries.

Due to the large size of our fragment library data and the memory requirements of the PCA algorithm, we used a subset of the total fragment library populations for analysis. A random sampling of $1x10^6$ fragments were chosen from each library, at each length, for PCA analysis. We did not experience any significant difference between this size of a sample set and the larger sets we evaluated.

**Table 5.1 Fragment library abbreviations and descriptions.**

| Fragment Libraries | | | | |
|---|---|---|---|---|
| **Abbreviation** | **Number of Unique Chains** | **Number of Structures** | **Number of Fragments** | **Description** |
| **DYN$_{298}$** | 807 | 42,068 | 105,112,513 | Dynameomics simulations run at 298K. Structures sampled at 1ns intervals for entire duration of each simulation. |
| **DYN$_{498}$** | 807 | 240,814 | 603,915,700 | Dynameomics simulations run at 498K. Structures sampled at 100ps intervals for last 15ns of each simulation. |
| **DYN$_{all}$** | 807 | 282,882 | 709,028,213 | This is a virtual library comprised of the DYN$_{298}$ and DYN$_{498}$ libraries. |
| **DYN$_{start}$** | 807 | 807 | 1,996,160 | Starting structure of each Dynameomics simulation. Structures are solvated and minimized. |
| **PDB$_{xtal}$** | 23,144 | 23,144 | 99,193,499 | Crystal structures from the PDB. Structures were chosen using a PISCES query as described in the text. |
| **PDB$_{nmr}$** | 5,412 | 100,973 | 180,437,054 | NMR spectroscopy structures from the PDB. Structures were chosen using parameters described in the text. |

**Table 5.2 Average RMSD of predicted values for 510 loop targets.**

This table represents averages and standard deviations for 30 samples at each fragment length. The best fragment is the fragment with the lowest heavy-atom (N, O, C, Cα) RMSD in the IC search result set. The best average score is emphasized with an underline and italics. The consensus set, which is a combination of the best predictions from each fragment library's IC result set, outperforms any individual fragment library.

| | **PDB** | | **DYN$_{298}$** | | **DYN$_{498}$** | | **Consensus** | |
|---|---|---|---|---|---|---|---|---|
| Length | Average | Standard Deviation | Average | Standard Deviation | Average | Standard Deviation | Average | Standard Deviation |
| 4 | 0.52 | 0.42 | 0.68 | 0.37 | 0.73 | 0.38 | ***0.47*** | 0.30 |
| 5 | 0.71 | 0.62 | 1.10 | 0.58 | 1.10 | 0.56 | ***0.62*** | 0.44 |
| 6 | 0.98 | 0.74 | 1.55 | 0.74 | 1.70 | 0.53 | ***0.86*** | 0.59 |
| 7 | 1.47 | 1.19 | 2.27 | 0.93 | 2.08 | 0.69 | ***1.19*** | 0.71 |
| 8 | 1.92 | 1.35 | 2.50 | 1.09 | 2.80 | 1.06 | ***1.67*** | 1.00 |
| 9 | 2.25 | 1.68 | 3.34 | 1.38 | 2.86 | 0.71 | ***1.84*** | 1.03 |
| 10 | 3.06 | 2.20 | 4.08 | 1.59 | 4.00 | 1.88 | ***2.65*** | 1.85 |
| 11 | 2.66 | 1.54 | 3.90 | 0.94 | 3.69 | 0.84 | ***2.48*** | 1.16 |
| 12 | 3.41 | 1.95 | 3.92 | 1.19 | 3.48 | 0.74 | ***2.78*** | 1.28 |
| 13 | 4.05 | 2.74 | 5.39 | 2.26 | 4.90 | 2.26 | ***3.39*** | 1.75 |
| 14 | 5.04 | 2.22 | 5.73 | 2.59 | 4.58 | 1.09 | ***4.24*** | 1.77 |
| 15 | 4.80 | 2.75 | 6.53 | 3.01 | 5.58 | 1.97 | ***4.20*** | 2.17 |
| 16 | 5.83 | 2.81 | 6.91 | 2.61 | 6.68 | 2.59 | ***5.39*** | 2.59 |
| 17 | 6.58 | 3.24 | 6.93 | 2.52 | 6.85 | 2.49 | ***5.27*** | 2.57 |
| 18 | 5.29 | 3.23 | 5.96 | 1.53 | 6.37 | 2.32 | ***4.67*** | 1.91 |
| 19 | 6.68 | 3.09 | 7.38 | 2.20 | 7.15 | 2.25 | ***5.79*** | 2.32 |
| 20 | 7.49 | 4.01 | 8.54 | 3.04 | 7.92 | 2.72 | ***6.66*** | 3.48 |

**Table 5.3 PCA correlated distances.**

Distances that best correlate with first three principal components, per fragment library. The percentage of variance captured by the first three principal is listed on the right.

| Library | Correlated Distance 1 | Correlated Distance 2 | Correlated Distance 3 | Percentage of Variance |
|---|---|---|---|---|
| $DYN_{298}$ | CBs_to_Ce | Cs_to_Oe | CBs_to_CAe | 87.7 |
| | CBs_to_CAe | Cs_to_Oe | CAs_to_CAe | 96.4 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 97.4 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 97.5 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 97.9 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.3 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.5 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 98.6 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.7 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 98.8 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 98.9 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.0 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.0 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.1 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.1 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.2 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.2 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.2 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.3 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.3 |
| $DYN_{498}$ | CBs_to_Ce | Cs_to_CAe | CBs_to_CBe | 84.4 |
| | CBs_to_CAe | Os_to_CAe | CAs_to_CAe | 93.3 |
| | CBs_to_CBe | Os_to_CAe | CBs_to_CAe | 95.7 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 96.0 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 96.4 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 96.9 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 97.3 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 97.5 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 97.8 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.0 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.2 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.4 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.6 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.7 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.8 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.9 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 99.0 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 99.1 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 99.1 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 99.2 |

**Table 5.4 PCA correlated distances (continued).**

| Library | Correlated Distance 1 | Correlated Distance 2 | Correlated Distance 3 | Percentage of Variance |
|---|---|---|---|---|
| PDB | CBs_to_CBe | Cs_to_Oe | Ns_to_Ce | 91.0 |
| | CBs_to_Ne | Cs_to_Oe | CAs_to_CAe | 97.3 |
| | CBs_to_CAe | Os_to_Ne | CBs_to_Ne | 97.8 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 97.8 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.2 |
| | CBs_to_CBe | Os_to_Ne | Ns_to_Oe | 98.6 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.7 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 98.8 |
| | CBs_to_CBe | Os_to_Ne | Ns_to_Oe | 98.9 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.0 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.1 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.1 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.2 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.2 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.3 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.3 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.3 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.4 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.4 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.4 |
| NMR | CBs_to_CBe | Os_to_Ne | CAs_to_Ce | 90.6 |
| | CBs_to_Ne | Cs_to_Oe | CAs_to_CAe | 96.7 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 97.6 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 97.8 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.2 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 98.5 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.7 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 98.8 |
| | CBs_to_CBe | Os_to_Ne | Ns_to_Oe | 99.0 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.0 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.1 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.2 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.2 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.2 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.3 |
| | CBs_to_CBe | Cs_to_Oe | CBs_to_CAe | 99.3 |
| | CBs_to_CBe | Cs_to_Oe | CBs_to_CAe | 99.3 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.3 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.4 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.4 |

**Table 5.5 PCA correlated distances (continued).**

| Library | Correlated Distance 1 | Correlated Distance 2 | Correlated Distance 3 | Percentage of Variance |
|---|---|---|---|---|
| $\text{DYN}_{\text{Start}}$ | CBs_to_CBe | Cs_to_Oe | Ns_to_Ce | 90.5 |
| | CBs_to_Ne | Cs_to_Oe | CAs_to_CAe | 97.1 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 97.7 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 97.8 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.1 |
| | CBs_to_CBe | Os_to_Ne | Ns_to_Oe | 98.5 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_Ne | 98.6 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 98.7 |
| | CBs_to_CBe | Os_to_Ne | Ns_to_Oe | 98.9 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 98.9 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.1 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.1 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.2 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.2 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.2 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.3 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.3 |
| | CBs_to_CBe | Os_to_Ne | CBs_to_CAe | 99.3 |
| | CBs_to_CBe | Cs_to_Oe | CBs_to_CAe | 99.4 |

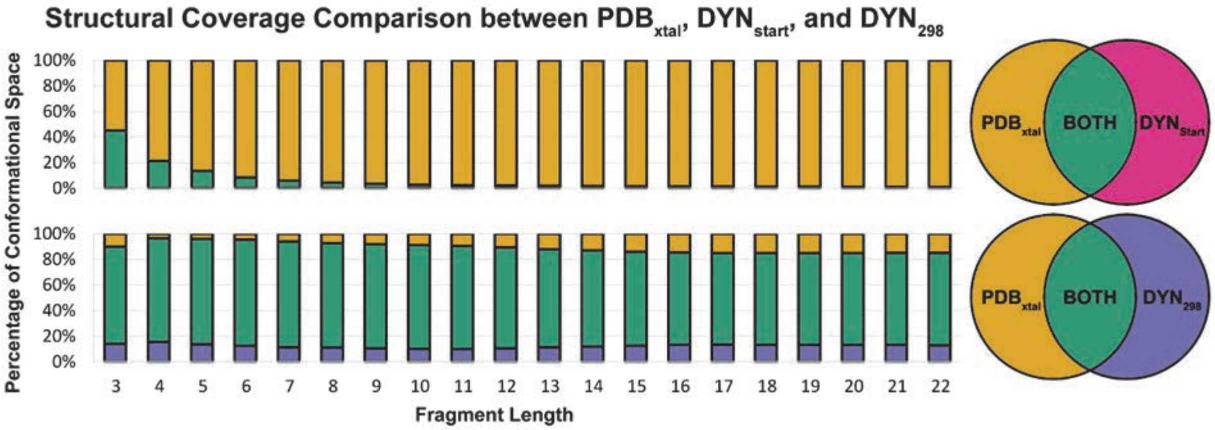


**Figure 5.1 Structural coverage comparison between PDB, DYN$_{start}$, and DYN$_{298}$**

Comparison of structural coverage between fragments generated from crystal structures and native state Dynameomics structures. The colors in the chart correspond to the Venn diagrams on the right. (a) This chart shows the percentage of conformational space shared between the DYN$_{start}$ fragment library and the PDB$_{xtal}$ fragment library. The DYN$_{start}$ library only contains a significant number of representatives at short fragment lengths. (b) This chart shows the percentage of conformational space shared between the DYN$_{298}$ fragment library and PDB$_{xtal}$ fragment library. For all fragment lengths, roughly 80% of the conformational space is shared by the two fragment libraries.

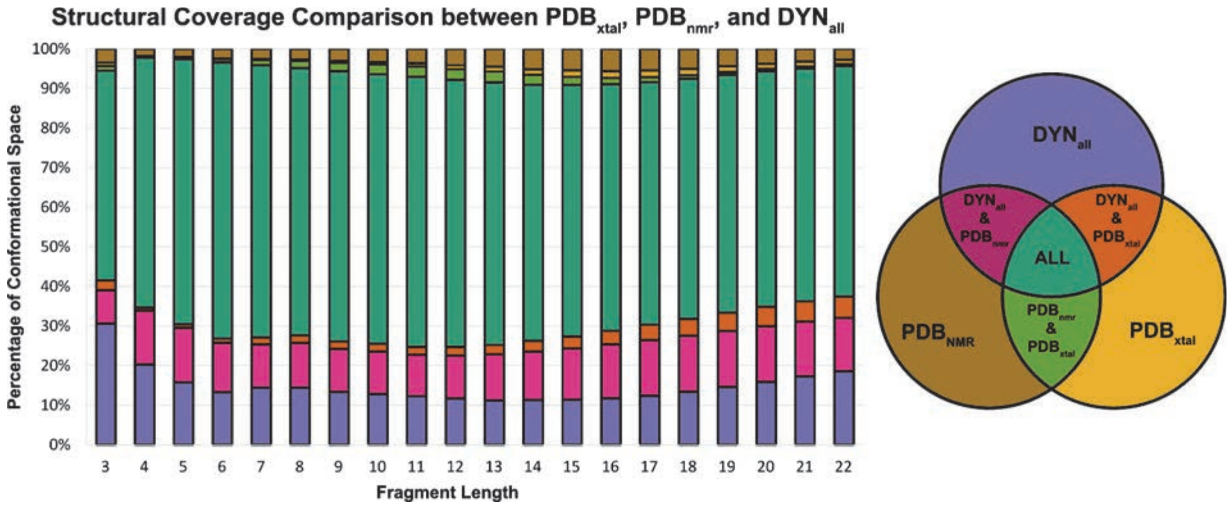
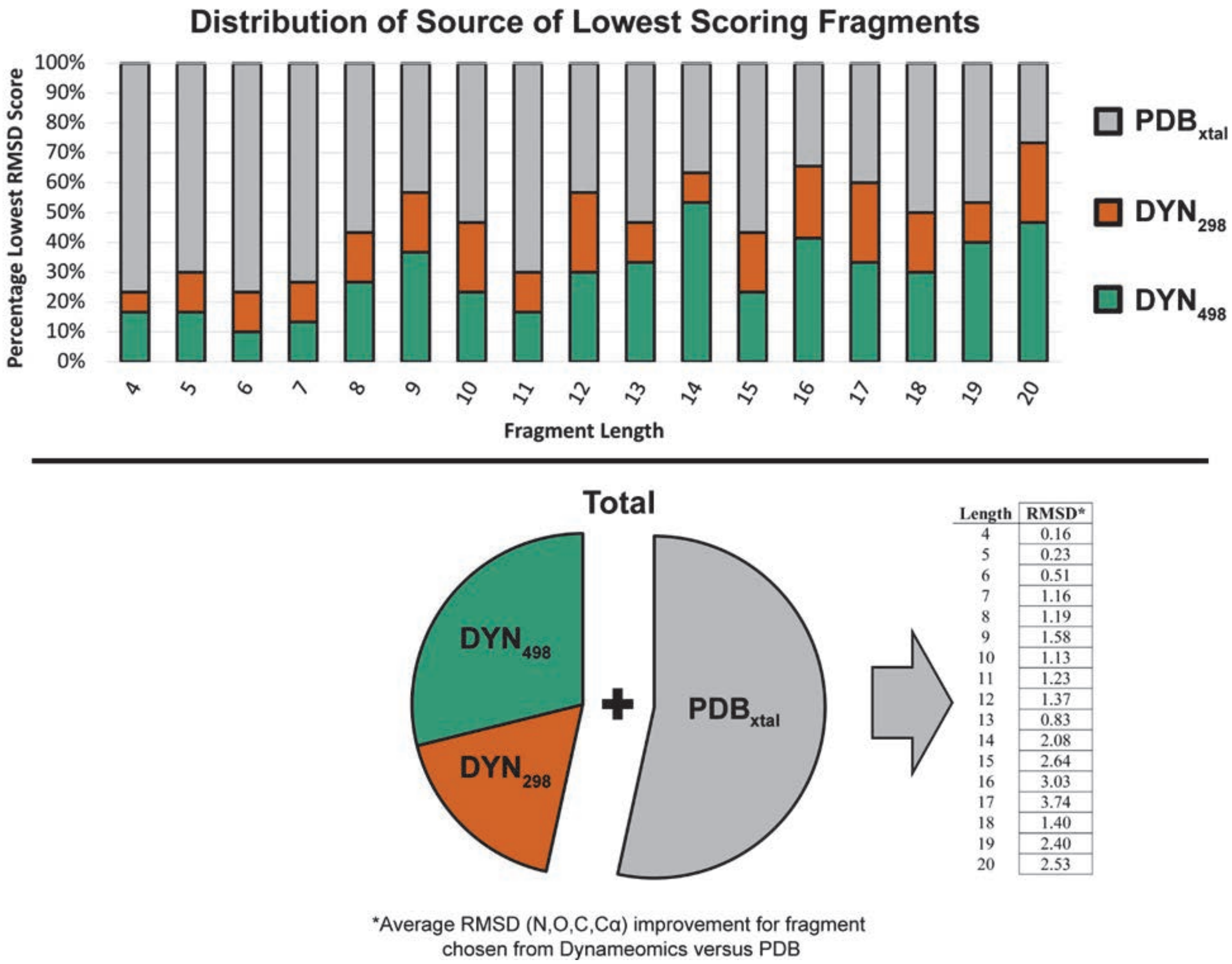


**Figure 5.2 Comparison of fragments sourced from various structure types.**

Comparison of structural coverage between fragments generated from crystal structures, NMR structures, and Dynameomics structures. We show the DYN$_{all}$ library here because fragments from both native-state and unfolding simulations provide informative conformations for loop predictions. The colors in the chart correspond to the Venn diagram on the right. The magenta area corresponds to the percentage of histogram bins that are shared between the PDB$_{nmr}$ and DYN$_{all}$ fragment libraries, but do not contain representatives from the PDB$_{xtal}$ fragment library.

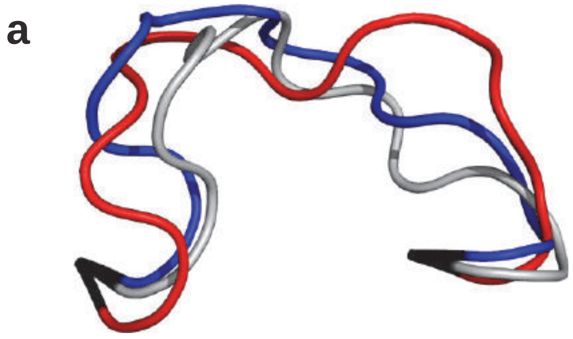


| Length | RMSD* |
|---|---|
| 4 | 0.16 |
| 5 | 0.23 |
| 6 | 0.51 |
| 7 | 1.16 |
| 8 | 1.19 |
| 9 | 1.58 |
| 10 | 1.13 |
| 11 | 1.23 |
| 12 | 1.37 |
| 13 | 0.83 |
| 14 | 2.08 |
| 15 | 2.64 |
| 16 | 3.03 |
| 17 | 3.74 |
| 18 | 1.40 |
| 19 | 2.40 |
| 20 | 2.53 |

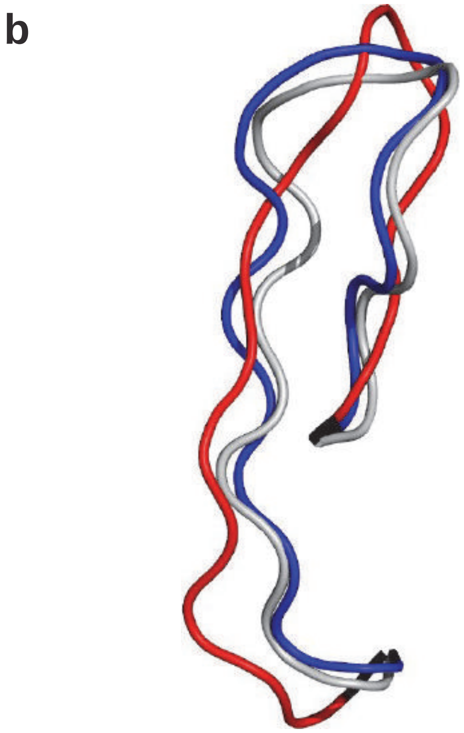
*Average RMSD (N,O,C,Cα) improvement for fragment chosen from Dynameomics versus PDB

**Figure 5.3 Distribution of Lowest Scoring Fragments.**

Distributions of the lowest scoring fragment predictions for all target loops in the 510 standard set. The top histogram shows the percentage of best predictions that are sourced from the PDB$_{xtal}$, DYN$_{298}$ and DYN$_{498}$ fragment libraries for each loop length. The total distribution for all 510 targets is shown in the pie chart. The average heavy-atom RMSD improvement per fragment length, in Å, is provided for the Dynameomics fragments on the bottom-right.



| | | | |
|---|---|---|---|
| $\text{PDB}_{\text{nmr}}$ | ⬜ | Target Structure | 1NWD, Model 1 |
| | | Residue Range | 55 to 66 |
| | | Length | 12 Residues |
| | | NOE Fulfillment | 118 of 131 (90.1%) |
| | | Average NOE Violation | 0.481 Å |
| $\text{PDB}_{\text{xtal}}$ | 🟥 | Source Structure | 3MEM |
| | | Residue Range | 81-92 |
| | | RMSD | 3.403 Å |
| | | NOE Fulfillment | 80 of 131 (61.1%) |
| | | Average NOE Violation | 3.158 Å |
| $\text{DYN}_{298}$ | 🟦 | Source Structure | 1K19 @ 40ns |
| | | Residue Range | 14-27 |
| | | RMSD | 1.873 Å |
| | | NOE Fulfillment | 103 of 131 (78.6%) |
| | | Average NOE Violation | 1.060 Å |



| | | | |
|---|---|---|---|
| $\text{PDB}_{\text{nmr}}$ | ⬜ | Target Structure | 1PS2, Model 1 |
| | | Residue Range | 30 to 44 |
| | | Length | 15 Residues |
| | | NOE Fulfillment | 119 of 131 (90.8%) |
| | | Average NOE Violation | 0.523 Å |
| $\text{PDB}_{\text{xtal}}$ | 🟥 | Source Structure | 3LD1 |
| | | Residue Range | 328-342 |
| | | RMSD | 4.474 Å |
| | | NOE Fulfillment | 86 of 131 (65.6%) |
| | | Average NOE Violation | 2.242 Å |
| $\text{DYN}_{298}$ | 🟦 | Source Structure | 1E9T @ 34ns |
| | | Residue Range | 34-48 |
| | | RMSD | 1.326 Å |
| | | NOE Fulfillment | 119 of 131 (90.8%) |
| | | Average NOE Violation | 0.651 Å |

**Figure 5.4 Predictions of loops in NMR structures.**

The NMR structures are depicted in grey, the X-ray predictions in red, the Dynameomics predictions in blue, and anchor residues in black. RMSD and NOE results do not include anchor residues in the calculations. (a) A 12 residue structure in 1NWD, not including anchor residues. (b) A 15 residue loop structure in 1PS2, not including anchor residues

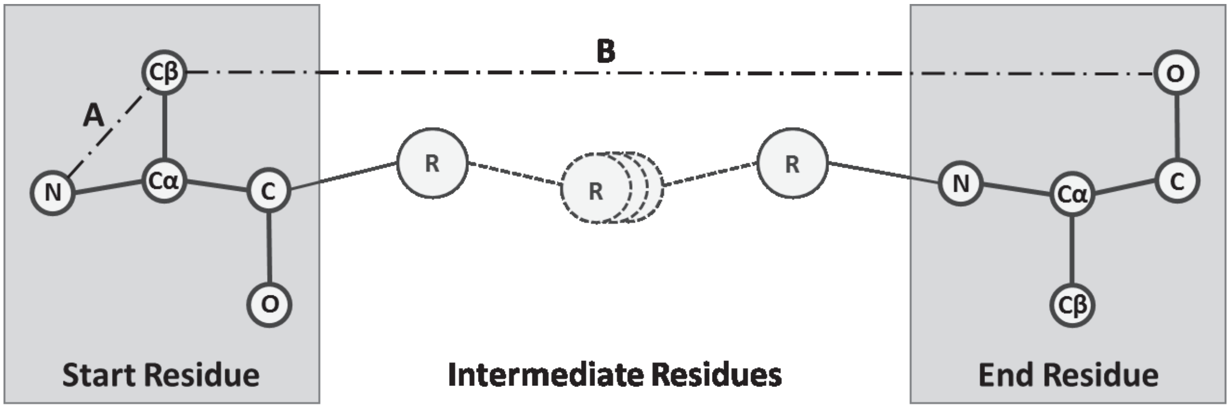


**Figure 5.5 Abstract depiction of fragment.**

Ten heavy atoms are involved in the end-to-end distance definitions. There are 20 intra-residue distances (a) and 25 inter-residue distances (b). One or more amino acids can exist as intermediate residues in the peptide structure.

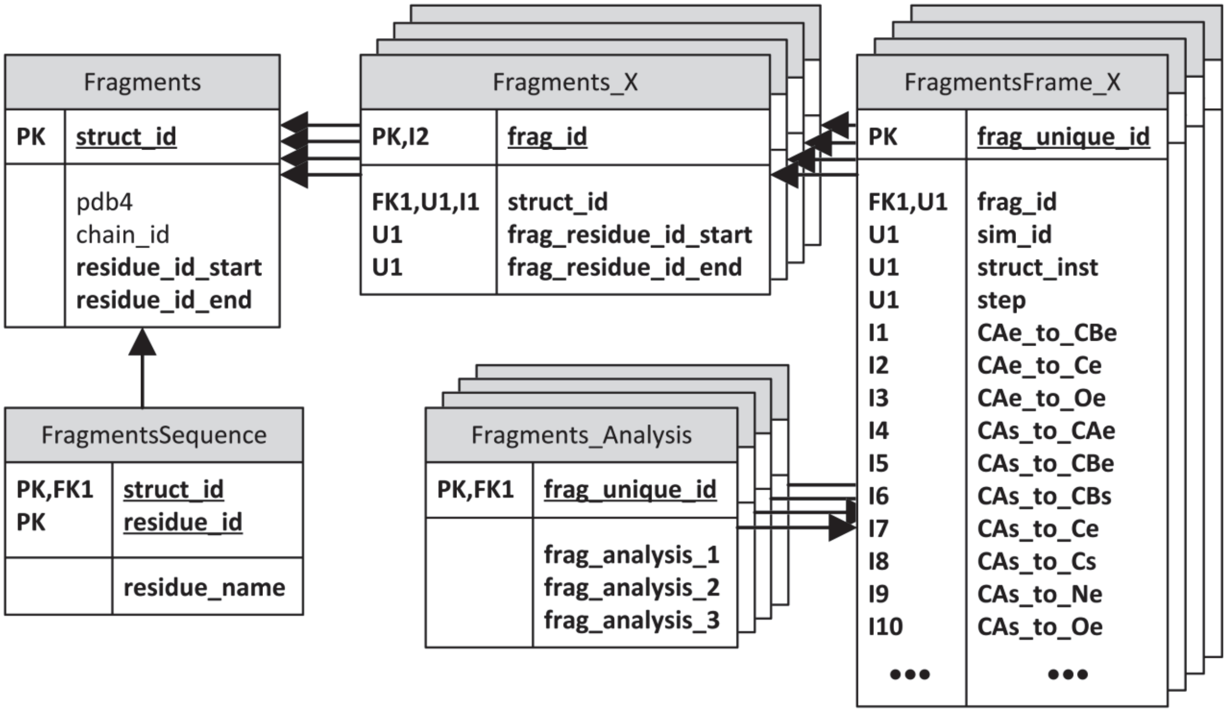


**Figure 5.6 Schema of generic fragment library database.**

Tables are represented as boxes and foreign key (FK) to primary key (PK) relationships are represented as arrows. Unique identifiers (Ux) and indexes (Ix) are also listed. The *X* term denotes fragment length.

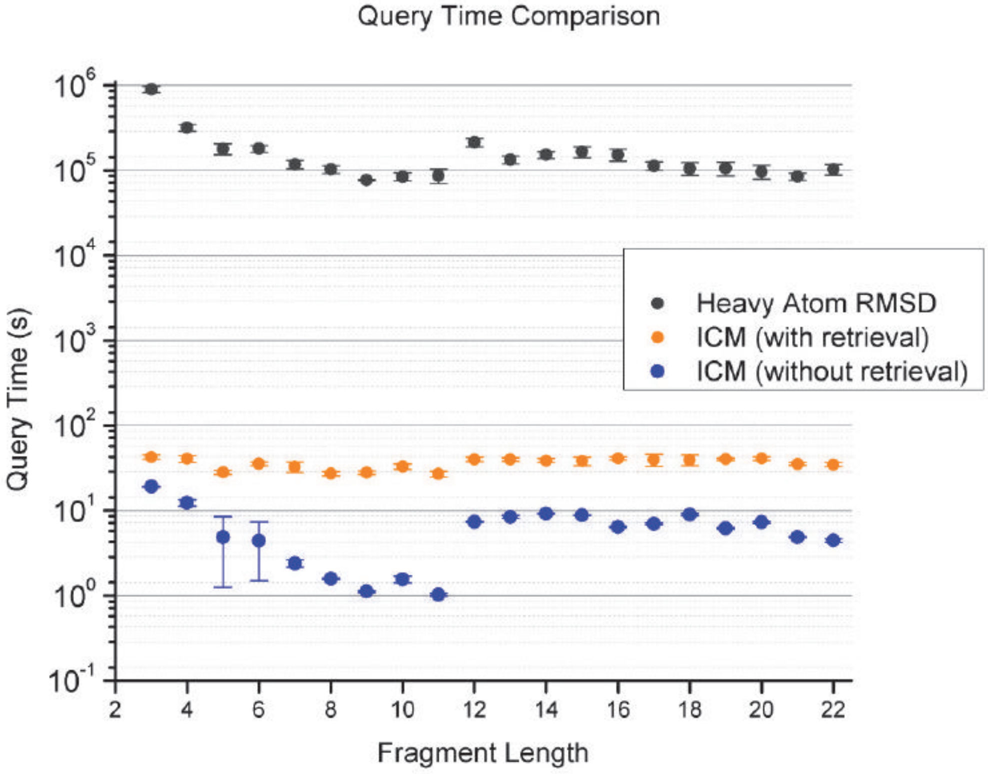


**Figure 5.7 Query execution time for fragment searches.**

Each query was run three times for each fragment length. The times shown for the naïve approach are approximations based on the running time of RMSD calculations for smaller sets of fragments.

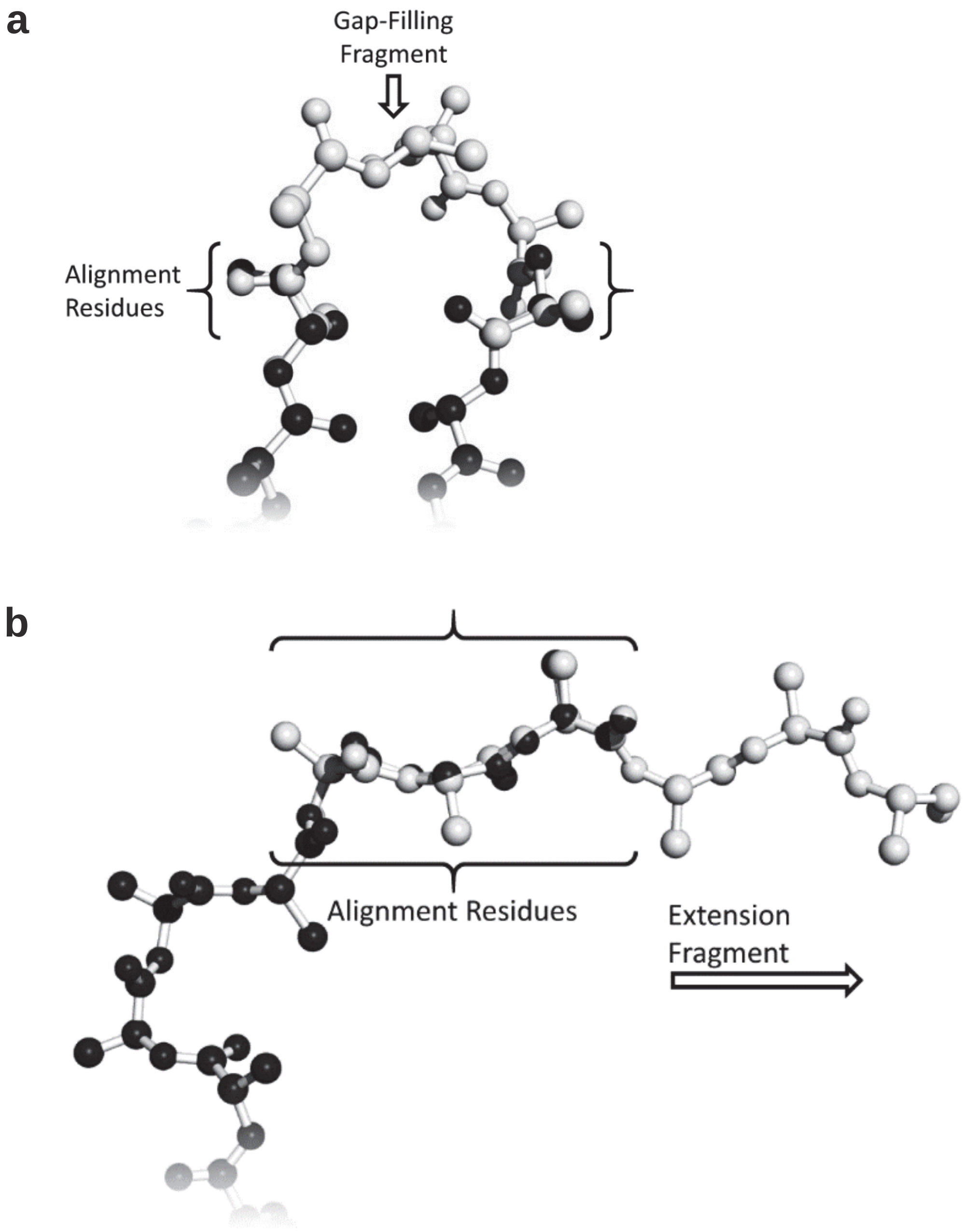


**Figure 5.8 Anchor residue alignments for fragment attachment.**

(a) Two anchor residues are used for gap-filling fragment insertions. (b) Three anchor residues are used for extension fragments.

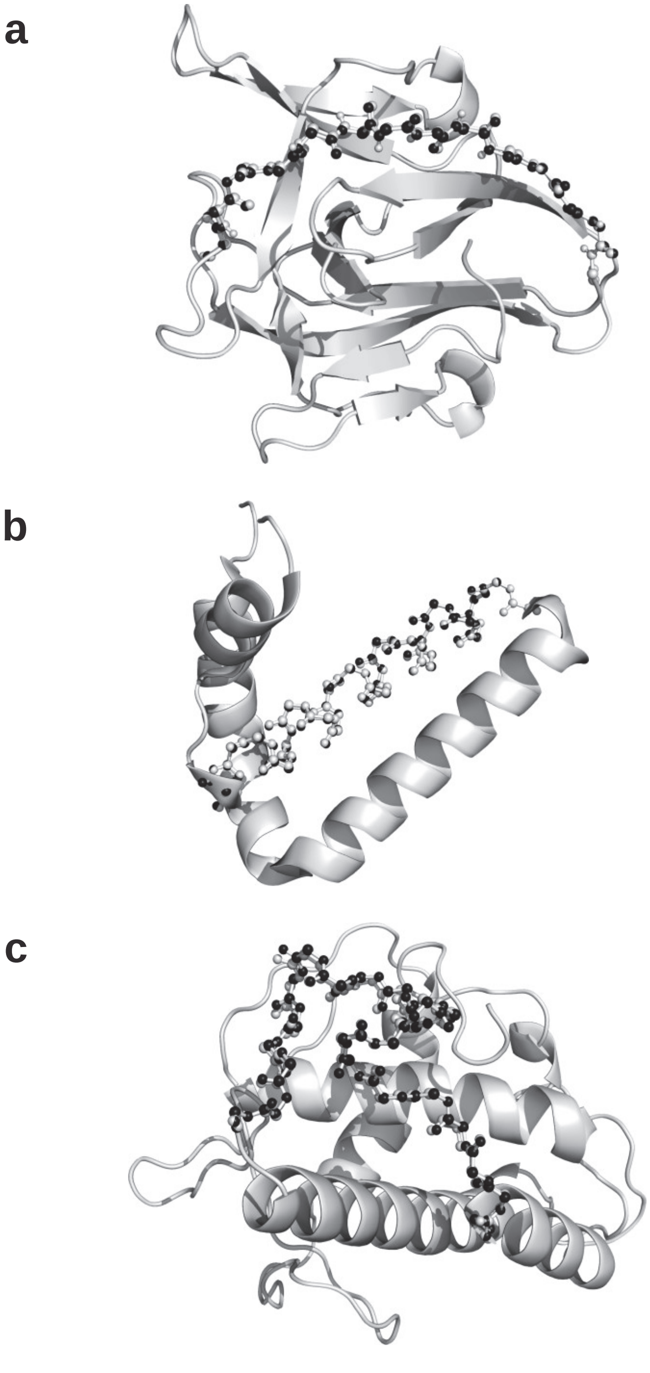


**Figure 5.9 Overlay of Dynameomics starting structure fragments.**

Depicted here is an overlay of minimized Dynameomics starting structure and X-ray crystallography fragment used for evaluation. The Dynameomics structures are shown in white and the X-ray fragments are shown in black. (a) β-strand structure of 22 residues in length. (b) α-helix structure of 22 residues in length. (c) loop structure of 22 residues in length.

Chapter 6

# DYNAMEOMICS: COMPARATIVE DATA-DRIVEN ANALYSIS OF THE CORRELATION BETWEEN ROTAMERIC STATES AND BACKBONE CONFORMATIONAL PROPENSITIES AND IMPROVED ROTAMER LIBRARIES

## *6.1 Summary*

Protein side-chain and backbone conformations and dynamics are intimately linked. Understanding the relationship between the two constituent parts is crucial to improving protein structure refinement, modeling and prediction. In particular, side-chain conformations, or rotamers, are important for protein binding and enzyme activity. Identifying rotamers and their relationship to backbone structure is key to this process; this knowledge can be captured in rotamer libraries. Backbone-dependent libraries provide statistical rotamer representatives based explicitly on backbone φ/ψ angles. However, most popular rotamer libraries are generated from subsets of the Protein Data Bank (PDB), which may contain poorly selected side-chain positions and incorrect side-chain flips, often expressly filter out dynamic side-chain samples with B-factor cutoffs, and overall do not fully represent the conformational scope of protein side-chains. This limits their utility in refining new experimental structures, understanding intrinsically disordered structure conformations, and predicting protein structures in solution at ambient temperatures. To address this limitation, we look to the physics-based approach of molecular dynamics (MD) simulations to investigate the true frequency of rotameric states. Here we present an analysis of the correlation between rotameric states and backbone conformational propensities based on MD simulation data from the Dynameomics project. We propose a backbone-dependent rotamer

library derived from Dynameomics using 4.8 x $10^9$ rotamers, sampled from at least 51,000 occurrences of each of 93,642 residues. Since our dataset has increased over 40% in size since the publication of our backbone-independent library, we provide both the backbone-dependent and updated backbone-independent libraries online at http://www.dynameomics.org.

## *6.2 Introduction*

Protein backbones have long been acknowledged for their role in forming repeating patterns of secondary structure. In contrast, side-chain orientation and dynamics are crucial for binding and enzyme activity. Together these parts are critical to improving protein structure refinement, modeling and prediction. There is a real need to predict accurate side-chain rotamers as part of structure calculation and refinement. Often the side-chain detail can be missing, incomplete, or completely absent in many PDB structures, depending on how extensive structural validation has been prior to deposition (Chang et al., 2006; Gore, Velankar, & Kleywegt, 2012). Characterizing the conformational and dynamical relationship between the two components, backbone and side-chain, is vital for our continued progression in understanding protein chemistry and structural biology. To this end, a comprehensive assessment of amino acid side-chain conformations, or rotamers, will improve the accuracy in predicting protein structure, refinement of experimentally derived structures, and the engineering of new proteins.

The orientations and sampling of the side-chain dihedral angles is not random and propensities for certain angles have been observed (Bahar & Jernigan, 1996). Rotamer libraries are used to select appropriate side-chain conformations by capitalizing on this knowledge. Furthermore, since the population of some rotameric states appear highly correlated with protein backbone conformations (Hagarman et al., 2011; Otzen & Fersht, 1995), it is increasingly important that more accurate backbone-dependent rotamer libraries are available. Currently, there are a few rotamer libraries widely used today (Lovell et al., 2000; Scouras & Daggett, 2011; Shapovalov & Dunbrack Jr., 2011). These libraries include rotamers independent of the backbone conformation, dependent on secondary structure, or dependent on the backbone conformation. The most popular backbone-dependent library available is provided by the Dunbrack laboratory, which has continually improved the coverage and quality of their library over the last two decades (Dunbrack & Cohen, 1997; Dunbrack & Karplus, 1994; Shapovalov & Dunbrack Jr., 2011). The Dunbrack rotamer library was generated by analyzing select residues, filtered for steric clashes and low B-factors, from high-quality sets of crystal structures in the Protein Data Bank (PDB) (Berman et al., 2000).

Most rotamer libraries rely on such statistical analysis of the PDB (Larriva & Rey, 2014; Lovell et al., 2000; Shapovalov & Dunbrack Jr., 2011; Xiang & Honig, 2001). Rotamer probabilities derived from the PDB are used to both replace, predict and refine side-chains in experimental structures. However, rotamer libraries generated from the PDB have their limitations (Berman, Kleywegt, Nakamura, & Markley, 2013; Davis et al., 2007; Montelione et al., 2013). Although the PDB is the largest repository of experimental structures, some rotamers have extremely low or non-existent sampling for regions of φ/ψ space. Hence, refining experimental structures with statistically derived probabilities from already solved experimental structures is flawed; since the PDB does not necessarily reflect all rotameric possibilities it may not accurately depict the conformational landscape for side-chains of new structures.

Furthermore, due to difficulties in crystallization of or gaining NMR observables from highly flexible regions, we know that the full range of rotamers is underestimated. This is especially true given that a large percentage of the PDB consists of crystal structures, which depict a single structure averaged across an ensemble of crystallized protein instances (Wagner et al., 1992). These crystal structures do not capture the highly dynamic nature commonly observed in amino acid side-chains. B-factors offer some indication of side-chain mobility but many rotamer libraries implement a B-factor cutoff. As a result of the bias towards low B-factor targets, dynamic rotamer conformations are expressly excluded. Crystal structures may also contain artifacts from experimental procedures, such as crystal contacts or general misrepresentation of the native environment due to the extremely low temperatures required by the technique. Overall, these factors may provide an unfavorable bias that is incorporated directly in many of the existing rotamer libraries, which is not ideal for modeling and predicting protein structures in solution at ambient temperatures.

Although the rate at which novel folds are being identified has slowed significantly (Skolnick, Zhou, & Brylinski, 2012), the rate of disordered protein discovery is accelerating (Dyson, 2011). This confirms that the PDB does not yet represent the true extent of protein fold space. This issue has been previously highlighted; statistical techniques exist to predict the missing data and attempts have been made to interpolate rough transitions between sampled areas. However, these estimations may miss fine-details or misrepresent the true distribution of side-chain conformations.

Molecular dynamic (MD) simulations provide a platform from which a more extensive sampling of rotameric states can be investigated that, by using a physics based approach (Levitt et al., 1995, 1997), reflects the rate of all possible rotamers for folded proteins. Here, we address the shortcomings of PDB-based rotamer libraries with a backbone-dependent library by making use of extensive atomistic MD simulations from our Dynameomics project (van der Kamp et al., 2010). We provide an analysis of our data-driven approach to generating a backbone-dependent library by showing a selection of interesting rotameric features. This work extends our previously published backbone-independent statistics of the amino acid rotameric states, also determined from our Dynameomics dataset (Scouras & Daggett, 2011). Since the raw Dynameomics data has nearly doubled in size since the previous publication, we provide an updated backbone-independent rotamer library along with the new backbone-dependent library. Both libraries are available online at http://www.dynameomics.org.

## *6.3 Results*

### 6.3.1 *Conformational Analysis: Generation of Backbone Independent and Backbone Dependent Rotamer Libraries*

Our analyses of side-chain conformations were performed on simulations from the Dynameomics 2009 Release set (Schaeffer et al., 2011; van der Kamp et al., 2010). Each of the simulation targets represents a unique fold representative, and together these targets represent 95% of the known protein fold space (Schaeffer et al., 2011). Each Dynameomics structure has undergone thorough validation, both prior to and after simulation, and has been simulated for a minimum of 51 ns at 25° C to capture native state dynamics (van der Kamp et al., 2010). Due to a 47% increase sample size since our previous rotamer publication (Scouras & Daggett, 2011), we provide both an updated backbone-independent rotamer library as well as our first backbone-dependent rotamer library. The updated backbone-independent library is consistent with our previous findings. Although the sample size has increased by 47%, the variance in the probability distribution was minimal. There was an average variance of 0.12% with a standard deviation of

0.30% for all backbone-independent rotamer representatives. The maximum change to any individual rotamer probability was 2.9%.

Although the larger number of side-chain samples had a minimal impact on the backbone-independent statistics, the Dynameomics backbone-dependent library benefited from the increased information due to the number of discretized backbone conformations. Backbone conformations were assigned to 10° x 10° bins based on their ψ/φ angles in the range [-180,180). The backbone-dependent library, on average, contained 221,668 side-chain samples for each of the 282,017 bins. The bin counts range from 1 sample to 3.1 x $10^7$ samples per bin with a median count of 130 samples. Overall, the data presented in each of the Dynameomics rotamer libraries represent 4.8x$10^9$ side-chain samples. The sampling in the Dynameomics backbone-dependent rotamer library was compared to sampling of the PDB-based Dunbrack backbone-independent library Table 6.1.

### 6.3.2 *Dependent versus Independent Probabilities*

To illustrate the difference between the backbone-dependent and backbone-independent rotamer probabilities, we present an analysis of valine in Figure 6.1. The φ/ψ plots show the independent versus dependent probabilities of each the three valine rotamers. The plots range from 0% to 100% backbone-dependent probabilities and are colored by the factor of change in comparison to the independent probability. The pink color represents a dependent probability of 0%. The blue color represents an increase in the dependent probability over the independent probability by a factor of one. Sample side-chain structures are highlighted on the right for areas of interest. The overall distribution of valine residues is shown in the upper-right hand corner with an overlay of secondary structure regions for reference.

The *trans* rotamer was the most common rotamer of valine with an overall independent probability of 55.5%. As Figure 6.1B shows, this rotamer most frequently occurred in clusters around φ angles of [-140°,-60°] and [30°,120°] and ψ angles of [-30°,-90°] and [90°,120°]. The *gauche*- rotamer occurred in the inverted regions of φ/ψ space with an independent probability of 43.1% (Figure 6.1C). Overall, the *trans* and *gauche*- rotamers commanded 98.4% of the independent probability. Although the *gauche*+ rotamer occurs in only 1.7% of the overall side-chain samples, it can be the predominant rotamer in the φ angle region of [-180°,-150°] and [150°,180°] with up to 100% dependent probability. In other words, the *gauche*+ rotamer was the only side-chain conformation sampled over a very small region of ψ/φ space where the backbone was almost completely extended (see Figure 6.1).

The right portion of A,B and C in Figure 6.1 provides specific instances of the side-chain conformations identified by number in the φ/ψ plots on the left. Each instance shows the residue of interest highlighted in red within its local context of seven residues as well as its location within its host protein.

The *gauche*+ conformation of valine has a region near [-180°,110°] which was nearly $10^3$ times more likely to occur than what the backbone-independent probability suggests. There are 32 individual bins for valine where the *gauche*+ conformation was the predominant rotamer. For these 32 bins, a total of 90,034 side-chain samples occurred in the *guache*+ conformation out of 167,099 total samples. Of note, this change in probability distribution is not observed in the Dunbrack rotamer library (Shapovalov & Dunbrack Jr., 2011).

### 6.3.3 *Secondary Structure Specific Differences*

Figure 6.2 shows the side-chain distributions and the relationship between rotameric states and secondary structure for a representative set of residues: isoleucine, serine, and threonine. The behavior of the residues varied from exhibiting consistent secondary structure probability with the backbone-independent ranking (serine) to where large departures from the backbone-independent rankings (threonine) were observed. For each residue the Ramachandran plot on the left shows the overall side-chain distributions and secondary structure regions with secondary-structure-dependent probabilities shown on the right. For simplicity, we present probabilities for only the top three rotamers from each residue. The complete secondary structure probabilities for all residues are provided in Table 6.2.

Isoleucine showed a fairly consistent ordering of the top three rotamers for all secondary structure regions. There are two exceptions for the *guache+,trans* conformation. The first exception is the αR region, which had a much lower dependent probability of 22.7% versus a 43.3% independent probability for the *gauche+,trans* conformation. The second exception is the αL region also showed a significant change from independent probability, with the *gauche+,trans* conformation decreasing to 14.4% and the *gauche-,trans* conformation increasing to 50.0%. For both the αR and near αR regions, the side-chain had an increased propensity for the *gauche-,guache-* conformation.

The dependent probabilities for Serine were consistent with the independent probabilities for all secondary structure types. However, the ratio between the probabilities shifted significantly for some regions. In particular, the αL region had an increased propensity for the *gauche-* conformation, while the *gauche+* conformation shifted from a 22.7% backbone-independent probability to a 1.9% probability.

Threonine showed a large departure from the independent rotamer probabilities for a number of secondary structures. The near αR, β, $P_{IR}$ and $P_{IIL}$ regions all shifted to a dominant *gauche-* rotameric state. In contrast, the αL and αR regions had a significantly increased probability for the *gauche+* rotamer in comparison to the backbone-independent probabilities.

## *6.4 Discussion*

We have presented an analysis of side-chain conformations and dynamics along with their correlation to backbone propensities using Dynameomics. Our primary intent was to analyze these correlations and identify aspects of side-chain dynamics that are not available through methods based on PDB crystal structures. Furthermore, we provide an extension of the backbone-independent work presented by Scouras and Daggett (Scouras & Daggett, 2011) as well as provide a new backbone-dependent rotamer library. The information presented here, along with these rotamer libraries, should further our understanding of side-chain behavior and application to protein structure problems, such as model refinement, intrinsically disordered structure assessment, and structure prediction for proteins in solution at ambient temperatures.

In agreement with others, we have demonstrated a backbone dependent preference for certain rotameric states. What we highlight here is the necessity to incorporate dynamics to determine what the more likely probability is for a selection of side-chain rotamers. Other rotamer libraries are based on data from the PDB, which can be unreliable. The PDB is limited in both the extent of what conformations side-chains can assume and by the errors that we know to exist (Davis et al., 2007).

Structure validation is a necessity, whether it pertains to corrections or refinement prior to using a PDB model in simulations or computer aided design tools. Gore *et al.* remind us that “the structures in the PDB are based on a subjective interpretation of experimental data" (Gore et al., 2012). Just recently there has been a push to improve the structural validation of both NMR and crystal structures upon deposition to the PDB (Montelione et al., 2013; Read et al., 2011). Retractions and fraudulent structures have been found in the PDB (Berman et al., 2013; Chang et al., 2006).

Our Dynameomics dataset has two strengths. First, as shown from a consensus view of protein fold space, our selected targets cover 95% of known folds. Given the slow rate at which novel folds are being discovered, our coverage is unlikely to significantly decrease over time. Second, the targets chosen were assessed for quality both before and after simulations. To maintain the validity of our dataset, certain targets have been rejected on the basis of low quality or where the simulations did not agree with experimental data and were removed from our dataset (Towse & Daggett, 2012).

Furthermore, even if the PDB was free from errors, the sample size is dramatically lower than the Dynameomics rotamer libraries (see Table 6.1). The relatively small sample size means that in using the PDB alone many bins can contain fewer samples than there are rotameric states. This estimation bias can affect the distributions of side-chain conformations across φ/ψ space. This data limitation does not present a problem for our Dynameomics dataset.

The results presented here are available online as part of our Structural Library of Intrinsic Residue Propensities (SLIRP) at http://www.dynameomics.org. Both the backbone-dependent and an update to the backbone-independent (Scouras & Daggett, 2011) rotamer libraries are available for download. The updated backbone-independent library is also included in the most recent release of Chimera (Pettersen et al., 2004), a popular molecular visualization and analysis software package.

## *6.5 Methods and Materials*

### 6.5.1 *Side-chain Dataset*

The backbone dependent rotamer library was generated from native state simulations in the Dynameomics Project (Simms & Daggett, 2012; Simms et al., 2008), which represents the diversity of structural folds in proteins (Day et al., 2003). Here we used the Dynameomics v2009 Release Set (Schaeffer et al., 2011) (http://www.dynameomics.org), which contains 807 unique protein structures representative of 95% of known protein folds. We used *il*mm (Beck & Daggett, 2004) (*in lucem* molecular mechanics) for all simulations in Dynameomics using explicit water molecules (Beck et al., 2003; Levitt et al., 1997) and the Levitt *et al.* force field (Levitt et al., 1995). Each structure was solvated and simulated for at least 51 ns at 25° C. Atom coordinates were recorded in the Dynameomics data warehouse at every 1 ps (van der Kamp et al., 2010), resulting in over 51,000 instances for each of the 93,642 residues and totaling over $4.8 \times 10^9$ samples. Additional details on the simulation protocols are available elsewhere (Beck & Daggett, 2004; Beck et al., 2008).

Rotamer populations, average dihedral angles, and standard deviations of those averages were generated in 10° bins for each of the $\varphi$ and $\psi$ backbone angles. This resulted in a grid over the backbone angles of 1296 bins, each 10° x 10° in size. The first nanosecond of simulation time was not included in the calculations to allow for equilibration of the individual simulation systems. We used the same rotamer definitions created for the Dynameomics backbone-independent library (Scouras & Daggett, 2011).

### 6.5.2 *Ramachandran Maps*

Ramachandran maps show the sample populations per backbone conformation of individual residue types in Dynameomics. We used 180 x 180, 2° x 2° bins for pairs of φ and ψ angles to emphasize the high-resolution sampling of our dataset. Secondary structure regions are defined as αR (-100° ≤ φ ≤ -30°; (-80° ≤ ψ ≤ -5°), near-αR (-175° ≤ φ ≤ -100°; (-55° ≤ ψ ≤ -5°), αL (5° ≤ φ ≤ 75°; (25° ≤ ψ ≤ 120°), β (-180° ≤ φ ≤ -50°; (80° ≤ ψ ≤ -170°), $P_{IR}$ (-180° ≤ φ ≤ -115°; (50° ≤ ψ ≤ 100°), and $P_{IIL}$(-110° ≤ φ ≤ -50°; (120° ≤ ψ ≤ 180°).

**Table 6.1 Comparison of backbone-dependent rotamer library statistics.**

Comparison of statistics for a PDB-based backbone-dependent library (Shapovalov & Dunbrack Jr., 2011) and the Dynameomics backbone dependent library. A sample is defined as one instance of a side-chain. Dynameomics has a much larger sampling of rotameric states and more thorough coverage of φ/ψ space.

| | PDB | Dynameomics |
|---|---|---|
| **Total samples** | 6.3 x $10^5$ | 4.8 x $10^9$ |
| **Average samples per residue** | 2.9 x $10^4$ | 2.4 x $10^8$ |
| **Average number of bins per residue** | 526 | 1128 |
| **Percentage of φ/ψ coverage** | 40.5% | 87.0% |
| **Average Samples per bin** | 55 | 221,668 |
| **Median Samples per bin** | 5 | 125 |

**Table 6.2 Independent and dependent rotamer probabilities per secondary structure type.**

| Residue | χ1 | χ2 | χ3 | χ4 | Ind. | Dep. α-Helix R | Dep. Near α-Helix R | Dep. α-Helix L | Dep. β-Strand | Dep. $P_{IIL}$ | Dep. $P_{IR}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ARG | g+ | g+ | g+ | g+ | 0.02% | 0.02% | 0.00% | 0.00% | 0.03% | 0.00% | 0.00% |
| ARG | g+ | g+ | g+ | t | 0.03% | 0.02% | 0.00% | 0.00% | 0.03% | 0.12% | 0.01% |
| ARG | g+ | g+ | g+ | g- | 0.00% | 0.00% | 0.00% | 0.00% | 0.01% | 0.00% | 0.00% |
| ARG | g+ | g+ | t | g+ | 0.01% | 0.00% | 0.00% | 0.00% | 0.01% | 0.06% | 0.00% |
| ARG | g+ | g+ | t | t | 0.08% | 0.01% | 0.07% | 0.00% | 0.24% | 0.18% | 0.01% |
| ARG | g+ | g+ | t | g- | 0.01% | 0.00% | 0.08% | 0.00% | 0.02% | 0.02% | 0.00% |
| ARG | g+ | g+ | g- | g+ | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| ARG | g+ | g+ | g- | t | 0.02% | 0.00% | 0.17% | 0.01% | 0.01% | 0.01% | 0.00% |
| ARG | g+ | g+ | g- | g- | 0.01% | 0.00% | 0.01% | 0.00% | 0.01% | 0.01% | 0.00% |
| ARG | g+ | t | g+ | g+ | 0.48% | 0.10% | 0.58% | 0.03% | 1.17% | 0.77% | 1.51% |
| ARG | g+ | t | g+ | t | 0.86% | 0.15% | 0.98% | 0.05% | 1.75% | 1.72% | 1.65% |
| ARG | g+ | t | g+ | g- | 0.08% | 0.01% | 0.10% | 0.00% | 0.24% | 0.12% | 0.07% |
| ARG | g+ | t | t | g+ | 0.68% | 0.24% | 1.11% | 0.01% | 1.35% | 1.15% | 0.45% |
| ARG | g+ | t | t | t | 0.64% | 0.13% | 0.72% | 0.00% | 1.86% | 1.16% | 0.76% |
| ARG | g+ | t | t | g- | 0.39% | 0.11% | 0.25% | 0.01% | 0.96% | 0.81% | 0.22% |
| ARG | g+ | t | g- | g+ | 0.08% | 0.03% | 0.05% | 0.00% | 0.20% | 0.18% | 0.01% |
| ARG | g+ | t | g- | t | 0.70% | 0.15% | 0.69% | 0.02% | 1.70% | 1.46% | 0.71% |
| ARG | g+ | t | g- | g- | 0.44% | 0.25% | 0.40% | 0.01% | 0.76% | 0.87% | 0.35% |
| ARG | g+ | g- | g+ | g+ | 0.01% | 0.00% | 0.01% | 0.00% | 0.00% | 0.01% | 0.00% |
| ARG | g+ | g- | g+ | t | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| ARG | g+ | g- | g+ | g- | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| ARG | g+ | g- | t | g+ | 0.00% | 0.00% | 0.02% | 0.00% | 0.00% | 0.00% | 0.00% |
| ARG | g+ | g- | t | t | 0.01% | 0.00% | 0.02% | 0.00% | 0.05% | 0.01% | 0.04% |
| ARG | g+ | g- | t | g- | 0.01% | 0.00% | 0.00% | 0.00% | 0.03% | 0.01% | 0.00% |
| ARG | g+ | g- | g- | g+ | 0.00% | 0.00% | 0.00% | 0.00% | 0.02% | 0.00% | 0.00% |
| ARG | g+ | g- | g- | t | 0.02% | 0.00% | 0.01% | 0.00% | 0.03% | 0.07% | 0.00% |
| ARG | g+ | g- | g- | g- | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.01% | 0.00% |
| ARG | t | g+ | g+ | g+ | 0.31% | 0.34% | 0.31% | 0.18% | 0.33% | 0.11% | 0.36% |
| ARG | t | g+ | g+ | t | 0.75% | 0.89% | 1.04% | 0.73% | 0.65% | 0.34% | 0.52% |
| ARG | t | g+ | g+ | g- | 0.05% | 0.06% | 0.03% | 0.00% | 0.06% | 0.02% | 0.09% |
| ARG | t | g+ | t | g+ | 0.58% | 0.85% | 0.64% | 0.01% | 0.38% | 0.18% | 0.64% |
| ARG | t | g+ | t | t | 1.26% | 1.87% | 0.86% | 0.36% | 0.91% | 0.44% | 0.33% |
| ARG | t | g+ | t | g- | 0.47% | 0.59% | 0.66% | 0.18% | 0.31% | 0.36% | 0.20% |
| ARG | t | g+ | g- | g+ | 0.08% | 0.14% | 0.01% | 0.00% | 0.02% | 0.00% | 0.00% |
| ARG | t | g+ | g- | t | 0.11% | 0.14% | 0.06% | 0.12% | 0.11% | 0.05% | 0.04% |
| ARG | t | g+ | g- | g- | 0.27% | 0.38% | 0.60% | 0.01% | 0.09% | 0.08% | 0.17% |
| ARG | t | t | g+ | g+ | 2.75% | 4.36% | 1.15% | 0.27% | 1.21% | 1.07% | 1.18% |
| ARG | t | t | g+ | t | 3.37% | 4.08% | 1.45% | 2.18% | 3.43% | 2.59% | 2.49% |
| ARG | t | t | g+ | g- | 0.62% | 1.06% | 0.17% | 0.05% | 0.23% | 0.13% | 0.07% |
| ARG | t | t | t | g+ | 0.71% | 0.70% | 0.72% | 0.38% | 1.05% | 0.58% | 0.68% |
| ARG | t | t | t | t | 1.92% | 1.68% | 1.34% | 0.95% | 3.43% | 1.93% | 2.36% |
| ARG | t | t | t | g- | 1.05% | 1.17% | 0.40% | 0.17% | 1.53% | 0.67% | 2.23% |
| ARG | t | t | g- | g+ | 0.45% | 0.71% | 0.25% | 0.04% | 0.21% | 0.14% | 0.13% |

| Residue | χ1 | χ2 | χ3 | χ4 | Ind. | Dep. α-Helix R | Dep. Near α-Helix R | Dep. α-Helix L | Dep. β-Strand | Dep. $P_{IIL}$ | Dep. $P_{IR}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ARG | t | t | g- | t | 2.22% | 2.65% | 2.31% | 1.16% | 2.49% | 0.98% | 2.23% |
| ARG | t | t | g- | g- | 1.17% | 1.24% | 1.29% | 0.99% | 1.44% | 0.82% | 0.84% |
| ARG | t | g- | g+ | g+ | 0.03% | 0.02% | 0.07% | 0.00% | 0.01% | 0.00% | 0.41% |
| ARG | t | g- | g+ | t | 0.03% | 0.03% | 0.02% | 0.03% | 0.04% | 0.01% | 0.10% |
| ARG | t | g- | g+ | g- | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| ARG | t | g- | t | g+ | 0.11% | 0.12% | 0.06% | 0.10% | 0.11% | 0.10% | 0.08% |
| ARG | t | g- | t | t | 0.32% | 0.28% | 0.17% | 0.05% | 0.62% | 0.36% | 0.69% |
| ARG | t | g- | t | g- | 0.05% | 0.06% | 0.08% | 0.00% | 0.07% | 0.01% | 0.11% |
| ARG | t | g- | g- | g+ | 0.09% | 0.07% | 0.02% | 0.33% | 0.17% | 0.04% | 0.11% |
| ARG | t | g- | g- | t | 0.41% | 0.61% | 0.40% | 0.09% | 0.19% | 0.18% | 0.09% |
| ARG | t | g- | g- | g- | 0.06% | 0.07% | 0.05% | 0.11% | 0.08% | 0.02% | 0.19% |
| ARG | g- | g+ | g+ | g+ | 0.17% | 0.17% | 0.39% | 0.13% | 0.11% | 0.22% | 0.06% |
| ARG | g- | g+ | g+ | t | 0.44% | 0.48% | 0.75% | 0.76% | 0.33% | 0.25% | 0.29% |
| ARG | g- | g+ | g+ | g- | 0.04% | 0.00% | 0.01% | 0.03% | 0.02% | 0.02% | 0.02% |
| ARG | g- | g+ | t | g+ | 0.10% | 0.04% | 0.23% | 0.11% | 0.18% | 0.09% | 0.47% |
| ARG | g- | g+ | t | t | 0.39% | 0.30% | 0.70% | 0.54% | 0.49% | 0.45% | 0.49% |
| ARG | g- | g+ | t | g- | 0.11% | 0.08% | 0.16% | 0.14% | 0.18% | 0.09% | 0.14% |
| ARG | g- | g+ | g- | g+ | 0.01% | 0.00% | 0.00% | 0.00% | 0.01% | 0.02% | 0.00% |
| ARG | g- | g+ | g- | t | 0.11% | 0.19% | 0.03% | 0.04% | 0.04% | 0.04% | 0.01% |
| ARG | g- | g+ | g- | g- | 0.03% | 0.03% | 0.04% | 0.01% | 0.02% | 0.05% | 0.05% |
| ARG | g- | t | g+ | g+ | 4.67% | 4.60% | 4.29% | 7.68% | 4.41% | 4.63% | 4.24% |
| ARG | g- | t | g+ | t | 11.72% | 11.58% | 11.42% | 12.40% | 10.64% | 13.00% | 13.00% |
| ARG | g- | t | g+ | g- | 0.75% | 0.54% | 0.75% | 0.82% | 0.97% | 1.12% | 1.44% |
| ARG | g- | t | t | g+ | 3.48% | 3.01% | 3.42% | 4.45% | 3.89% | 4.42% | 3.80% |
| ARG | g- | t | t | t | 11.68% | 13.08% | 9.68% | 11.80% | 9.63% | 10.09% | 11.63% |
| ARG | g- | t | t | g- | 4.00% | 3.53% | 4.12% | 4.76% | 4.46% | 5.15% | 3.25% |
| ARG | g- | t | g- | g+ | 2.17% | 2.70% | 2.16% | 1.40% | 1.28% | 1.26% | 1.94% |
| ARG | g- | t | g- | t | 10.92% | 7.28% | 13.41% | 21.19% | 13.18% | 15.14% | 15.87% |
| ARG | g- | t | g- | g- | 11.06% | 13.61% | 9.02% | 12.80% | 6.36% | 8.14% | 8.45% |
| ARG | g- | g- | g+ | g+ | 0.69% | 1.00% | 0.39% | 0.28% | 0.32% | 0.43% | 0.11% |
| ARG | g- | g- | g+ | t | 0.83% | 0.69% | 2.04% | 0.37% | 0.90% | 0.97% | 0.82% |
| ARG | g- | g- | g+ | g- | 0.13% | 0.17% | 0.13% | 0.11% | 0.09% | 0.13% | 0.03% |
| ARG | g- | g- | t | g+ | 1.30% | 1.08% | 1.57% | 1.75% | 1.45% | 1.52% | 1.34% |
| ARG | g- | g- | t | t | 3.76% | 3.41% | 5.46% | 3.38% | 3.46% | 4.58% | 2.67% |
| ARG | g- | g- | t | g- | 2.16% | 1.39% | 4.65% | 1.83% | 3.79% | 1.73% | 3.60% |
| ARG | g- | g- | g- | g+ | 0.40% | 0.54% | 0.20% | 0.05% | 0.21% | 0.32% | 0.06% |
| ARG | g- | g- | g- | t | 3.64% | 3.87% | 3.67% | 3.42% | 2.64% | 4.00% | 2.32% |
| ARG | g- | g- | g- | g- | 1.41% | 1.23% | 1.85% | 1.12% | 1.32% | 2.19% | 1.75% |
| ASN | g+ | Ng+ |  |  | 0.15% | 0.05% | 0.10% | 0.01% | 0.39% | 0.22% | 0.11% |
| ASN | g+ | Og- |  |  | 1.01% | 0.22% | 0.41% | 0.02% | 1.97% | 3.25% | 0.39% |
| ASN | g+ | Nt |  |  | 0.87% | 0.08% | 0.23% | 0.00% | 1.65% | 2.56% | 0.34% |
| ASN | g+ | Og+ |  |  | 2.38% | 0.33% | 1.14% | 0.12% | 4.00% | 8.38% | 1.32% |
| ASN | g+ | Ng- |  |  | 0.25% | 0.07% | 0.23% | 0.02% | 0.38% | 0.55% | 0.40% |
| ASN | g+ | Ot |  |  | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |

| Residue | χ1 | χ2 | χ3 | χ4 | Ind. | Dep. α-Helix R | Dep. Near α-Helix R | Dep. α-Helix L | Dep. β-Strand | Dep. $P_{IIL}$ | Dep. $P_{IR}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ASN | t | Ng+ | | | 25.68% | 16.67% | 18.33% | 46.06% | 35.35% | 26.84% | 42.35% |
| ASN | t | Og- | | | 20.39% | 16.14% | 18.43% | 28.71% | 24.46% | 24.14% | 24.29% |
| ASN | t | Nt | | | 11.18% | 15.45% | 14.84% | 6.76% | 6.97% | 4.76% | 8.96% |
| ASN | t | Og+ | | | 25.14% | 39.03% | 37.30% | 7.15% | 12.01% | 11.21% | 9.14% |
| ASN | t | Ng- | | | 1.05% | 0.95% | 1.07% | 0.95% | 1.38% | 1.02% | 1.45% |
| ASN | t | Ot | | | 0.19% | 0.17% | 0.14% | 0.19% | 0.29% | 0.08% | 0.45% |
| ASN | g- | Ng+ | | | 0.62% | 0.80% | 0.41% | 0.26% | 0.59% | 0.58% | 0.39% |
| ASN | g- | Og- | | | 5.27% | 5.26% | 3.17% | 3.29% | 4.72% | 7.42% | 5.27% |
| ASN | g- | Nt | | | 2.09% | 2.85% | 0.82% | 0.70% | 1.42% | 2.98% | 0.80% |
| ASN | g- | Og+ | | | 1.85% | 1.12% | 1.76% | 1.93% | 2.30% | 3.11% | 1.94% |
| ASN | g- | Ng- | | | 1.83% | 0.76% | 1.54% | 3.78% | 2.08% | 2.86% | 2.36% |
| ASN | g- | Ot | | | 0.05% | 0.06% | 0.07% | 0.04% | 0.04% | 0.05% | 0.04% |
| ASP | g+ | g+ | | | 1.40% | 0.16% | 0.22% | 1.16% | 3.60% | 3.69% | 1.61% |
| ASP | g+ | t | | | 0.30% | 0.07% | 0.00% | 0.00% | 0.84% | 0.62% | 0.45% |
| ASP | g+ | g- | | | 2.56% | 0.32% | 0.36% | 1.18% | 5.48% | 7.27% | 3.57% |
| ASP | t | g+ | | | 66.45% | 74.31% | 75.36% | 75.20% | 46.99% | 39.72% | 65.41% |
| ASP | t | t | | | 7.38% | 6.71% | 7.48% | 5.94% | 8.41% | 5.35% | 11.24% |
| ASP | t | g- | | | 12.81% | 10.23% | 10.67% | 11.97% | 19.14% | 22.33% | 11.95% |
| ASP | g- | g+ | | | 2.10% | 1.62% | 1.09% | 1.22% | 3.97% | 6.09% | 1.41% |
| ASP | g- | t | | | 1.48% | 1.81% | 0.67% | 0.40% | 1.78% | 4.09% | 0.33% |
| ASP | g- | g- | | | 5.51% | 4.77% | 4.15% | 2.92% | 9.78% | 10.84% | 4.02% |
| CYH | g+ | | | | 14.00% | 2.83% | 13.44% | 1.69% | 25.15% | 20.62% | 25.15% |
| CYH | t | | | | 16.97% | 23.23% | 20.05% | 13.83% | 13.39% | 10.25% | 17.16% |
| CYH | g- | | | | 69.03% | 73.94% | 66.51% | 84.48% | 61.46% | 69.13% | 57.69% |
| CYS | g+ | | | | 13.64% | 3.36% | 17.44% | 3.18% | 18.21% | 18.80% | 15.92% |
| CYS | t | | | | 30.17% | 43.79% | 29.08% | 34.61% | 28.03% | 17.34% | 39.64% |
| CYS | g- | | | | 56.19% | 52.85% | 53.48% | 62.21% | 53.76% | 63.86% | 44.43% |
| GLN | g+ | g+ | Ng+ | | 0.01% | 0.00% | 0.01% | 0.00% | 0.02% | 0.03% | 0.01% |
| GLN | g+ | g+ | Og- | | 0.01% | 0.00% | 0.00% | 0.00% | 0.01% | 0.02% | 0.00% |
| GLN | g+ | g+ | Nt | | 0.00% | 0.00% | 0.00% | 0.00% | 0.02% | 0.02% | 0.00% |
| GLN | g+ | g+ | Og+ | | 0.01% | 0.00% | 0.00% | 0.00% | 0.04% | 0.03% | 0.00% |
| GLN | g+ | g+ | Ng- | | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| GLN | g+ | g+ | Ot | | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| GLN | g+ | t | Ng+ | | 0.52% | 0.08% | 0.60% | 0.02% | 1.46% | 0.94% | 1.39% |
| GLN | g+ | t | Og- | | 0.51% | 0.10% | 0.61% | 0.02% | 1.41% | 0.98% | 1.27% |
| GLN | g+ | t | Nt | | 0.29% | 0.03% | 0.26% | 0.02% | 0.98% | 0.69% | 0.66% |
| GLN | g+ | t | Og+ | | 0.58% | 0.09% | 0.53% | 0.03% | 1.78% | 1.29% | 1.10% |
| GLN | g+ | t | Ng- | | 0.42% | 0.06% | 0.34% | 0.02% | 1.29% | 1.02% | 0.57% |
| GLN | g+ | t | Ot | | 0.08% | 0.01% | 0.08% | 0.00% | 0.28% | 0.18% | 0.16% |
| GLN | g+ | g- | Ng+ | | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| GLN | g+ | g- | Og- | | 0.01% | 0.00% | 0.00% | 0.00% | 0.04% | 0.01% | 0.01% |
| GLN | g+ | g- | Nt | | 0.01% | 0.00% | 0.00% | 0.00% | 0.02% | 0.01% | 0.00% |
| GLN | g+ | g- | Og+ | | 0.04% | 0.02% | 0.01% | 0.00% | 0.06% | 0.13% | 0.03% |
| GLN | g+ | g- | Ng- | | 0.03% | 0.01% | 0.01% | 0.00% | 0.07% | 0.07% | 0.03% |

| Residue | χ1 | χ2 | χ3 | χ4 | Ind. | Dep. α-Helix R | Dep. Near α-Helix R | Dep. α-Helix L | Dep. β-Strand | Dep. $P_{IIL}$ | Dep. $P_{IR}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| GLN | g+ | g- | Ot |  | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| GLN | t | g+ | Ng+ |  | 0.45% | 0.52% | 0.36% | 0.23% | 0.55% | 0.30% | 0.38% |
| GLN | t | g+ | Og- |  | 0.20% | 0.19% | 0.13% | 0.09% | 0.33% | 0.31% | 0.15% |
| GLN | t | g+ | Nt |  | 0.12% | 0.15% | 0.10% | 0.05% | 0.11% | 0.07% | 0.09% |
| GLN | t | g+ | Og+ |  | 0.65% | 0.86% | 0.45% | 0.26% | 0.61% | 0.26% | 0.27% |
| GLN | t | g+ | Ng- |  | 0.04% | 0.05% | 0.03% | 0.01% | 0.02% | 0.02% | 0.02% |
| GLN | t | g+ | Ot |  | 0.01% | 0.01% | 0.01% | 0.00% | 0.01% | 0.01% | 0.00% |
| GLN | t | t | Ng+ |  | 1.45% | 1.38% | 1.19% | 1.20% | 2.32% | 1.28% | 1.94% |
| GLN | t | t | Og- |  | 1.74% | 1.66% | 1.70% | 1.42% | 2.67% | 1.59% | 2.21% |
| GLN | t | t | Nt |  | 1.10% | 1.10% | 1.07% | 0.62% | 1.53% | 1.11% | 1.30% |
| GLN | t | t | Og+ |  | 2.09% | 2.05% | 1.93% | 1.11% | 3.31% | 1.68% | 2.85% |
| GLN | t | t | Ng- |  | 1.43% | 1.49% | 1.41% | 1.02% | 1.91% | 0.95% | 1.88% |
| GLN | t | t | Ot |  | 0.26% | 0.25% | 0.22% | 0.22% | 0.45% | 0.19% | 0.37% |
| GLN | t | g- | Ng+ |  | 0.01% | 0.01% | 0.01% | 0.00% | 0.01% | 0.00% | 0.01% |
| GLN | t | g- | Og- |  | 0.14% | 0.17% | 0.13% | 0.06% | 0.16% | 0.07% | 0.21% |
| GLN | t | g- | Nt |  | 0.04% | 0.04% | 0.06% | 0.02% | 0.04% | 0.03% | 0.04% |
| GLN | t | g- | Og+ |  | 0.06% | 0.04% | 0.03% | 0.04% | 0.13% | 0.08% | 0.05% |
| GLN | t | g- | Ng- |  | 0.10% | 0.11% | 0.09% | 0.07% | 0.17% | 0.07% | 0.10% |
| GLN | t | g- | Ot |  | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| GLN | g- | g+ | Ng+ |  | 0.43% | 0.48% | 0.48% | 0.47% | 0.28% | 0.40% | 0.22% |
| GLN | g- | g+ | Og- |  | 0.83% | 1.04% | 0.32% | 0.27% | 0.42% | 1.08% | 0.16% |
| GLN | g- | g+ | Nt |  | 0.34% | 0.39% | 0.19% | 0.09% | 0.21% | 0.56% | 0.15% |
| GLN | g- | g+ | Og+ |  | 0.45% | 0.39% | 0.48% | 0.63% | 0.57% | 0.52% | 0.50% |
| GLN | g- | g+ | Ng- |  | 0.01% | 0.01% | 0.01% | 0.01% | 0.02% | 0.01% | 0.01% |
| GLN | g- | g+ | Ot |  | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| GLN | g- | t | Ng+ |  | 10.84% | 10.87% | 11.33% | 11.81% | 10.14% | 11.29% | 9.94% |
| GLN | g- | t | Og- |  | 17.59% | 16.77% | 18.73% | 18.75% | 17.49% | 19.16% | 18.26% |
| GLN | g- | t | Nt |  | 10.62% | 10.96% | 10.28% | 11.12% | 9.10% | 10.43% | 9.79% |
| GLN | g- | t | Og+ |  | 20.10% | 22.67% | 17.62% | 21.51% | 15.16% | 15.54% | 18.49% |
| GLN | g- | t | Ng- |  | 15.39% | 14.85% | 15.29% | 20.08% | 15.06% | 15.71% | 16.29% |
| GLN | g- | t | Ot |  | 2.22% | 2.15% | 2.05% | 2.65% | 2.30% | 2.45% | 2.11% |
| GLN | g- | g- | Ng+ |  | 0.20% | 0.15% | 0.40% | 0.21% | 0.23% | 0.21% | 0.26% |
| GLN | g- | g- | Og- |  | 3.83% | 3.53% | 6.54% | 2.07% | 3.89% | 4.14% | 4.13% |
| GLN | g- | g- | Nt |  | 0.93% | 0.91% | 1.23% | 0.41% | 1.01% | 1.05% | 0.84% |
| GLN | g- | g- | Og+ |  | 1.26% | 1.44% | 1.04% | 1.25% | 0.77% | 1.23% | 0.53% |
| GLN | g- | g- | Ng- |  | 2.54% | 2.86% | 2.57% | 2.11% | 1.50% | 2.74% | 1.19% |
| GLN | g- | g- | Ot |  | 0.04% | 0.04% | 0.06% | 0.02% | 0.02% | 0.02% | 0.02% |
| GLU | g+ | g+ | g+ |  | 0.09% | 0.06% | 0.10% | 0.00% | 0.08% | 0.17% | 0.02% |
| GLU | g+ | g+ | t |  | 0.01% | 0.01% | 0.00% | 0.00% | 0.00% | 0.01% | 0.00% |
| GLU | g+ | g+ | g- |  | 0.03% | 0.01% | 0.11% | 0.00% | 0.07% | 0.04% | 0.00% |
| GLU | g+ | t | g+ |  | 1.31% | 0.35% | 1.41% | 0.12% | 3.22% | 2.54% | 4.01% |
| GLU | g+ | t | t |  | 0.32% | 0.06% | 0.34% | 0.02% | 1.02% | 0.65% | 0.70% |
| GLU | g+ | t | g- |  | 1.12% | 0.30% | 1.12% | 0.15% | 2.70% | 2.51% | 2.80% |
| GLU | g+ | g- | g+ |  | 0.19% | 0.12% | 0.12% | 0.00% | 0.13% | 0.58% | 0.04% |

| Residue | χ1 | χ2 | χ3 | χ4 | Ind. | Dep. α-Helix R | Dep. Near α-Helix R | Dep. α-Helix L | Dep. β-Strand | Dep. $P_{IIL}$ | Dep. $P_{IR}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| GLU | g+ | g- | t |  | 0.03% | 0.02% | 0.01% | 0.00% | 0.02% | 0.08% | 0.00% |
| GLU | g+ | g- | g- |  | 0.16% | 0.06% | 0.28% | 0.01% | 0.16% | 0.45% | 0.11% |
| GLU | t | g+ | g+ |  | 1.69% | 2.31% | 1.00% | 0.09% | 1.06% | 1.16% | 0.44% |
| GLU | t | g+ | t |  | 0.34% | 0.53% | 0.19% | 0.01% | 0.12% | 0.05% | 0.04% |
| GLU | t | g+ | g- |  | 0.35% | 0.37% | 0.19% | 0.13% | 0.43% | 0.43% | 0.14% |
| GLU | t | t | g+ |  | 5.47% | 6.50% | 4.44% | 2.64% | 6.03% | 2.92% | 5.47% |
| GLU | t | t | t |  | 1.59% | 1.85% | 1.22% | 0.62% | 1.92% | 0.96% | 1.49% |
| GLU | t | t | g- |  | 6.06% | 7.65% | 4.73% | 3.02% | 5.65% | 2.83% | 4.55% |
| GLU | t | g- | g+ |  | 0.08% | 0.08% | 0.02% | 0.07% | 0.13% | 0.10% | 0.04% |
| GLU | t | g- | t |  | 0.05% | 0.05% | 0.04% | 0.00% | 0.08% | 0.04% | 0.00% |
| GLU | t | g- | g- |  | 0.71% | 0.98% | 0.30% | 0.32% | 0.48% | 0.34% | 0.17% |
| GLU | g- | g+ | g+ |  | 1.22% | 1.06% | 1.05% | 1.82% | 0.85% | 1.96% | 0.90% |
| GLU | g- | g+ | t |  | 0.48% | 0.51% | 0.45% | 0.03% | 0.28% | 0.71% | 0.19% |
| GLU | g- | g+ | g- |  | 1.12% | 1.12% | 0.61% | 0.66% | 0.61% | 1.96% | 0.30% |
| GLU | g- | t | g+ |  | 25.45% | 22.78% | 30.66% | 35.60% | 26.51% | 28.18% | 29.13% |
| GLU | g- | t | t |  | 12.73% | 14.07% | 10.60% | 12.16% | 11.47% | 10.04% | 11.02% |
| GLU | g- | t | g- |  | 29.69% | 26.24% | 35.19% | 39.41% | 31.70% | 33.47% | 34.64% |
| GLU | g- | g- | g+ |  | 1.48% | 1.96% | 0.86% | 0.76% | 0.78% | 1.23% | 0.78% |
| GLU | g- | g- | t |  | 1.28% | 1.45% | 0.95% | 0.20% | 1.27% | 1.43% | 0.80% |
| GLU | g- | g- | g- |  | 6.97% | 9.51% | 4.03% | 2.18% | 3.22% | 5.16% | 2.18% |
| HID | g+ | Ng+ |  |  | 2.31% | 0.57% | 2.20% | 0.12% | 4.74% | 3.30% | 1.79% |
| HID | g+ | Cg- |  |  | 1.46% | 0.34% | 1.82% | 0.04% | 3.52% | 1.85% | 0.37% |
| HID | g+ | Nt |  |  | 0.04% | 0.00% | 0.00% | 0.00% | 0.03% | 0.18% | 0.01% |
| HID | g+ | Cg+ |  |  | 2.59% | 0.80% | 3.08% | 0.43% | 5.61% | 3.43% | 0.74% |
| HID | g+ | Ng- |  |  | 1.78% | 0.49% | 2.96% | 0.32% | 3.87% | 2.23% | 0.75% |
| HID | g+ | Ct |  |  | 0.17% | 0.00% | 0.00% | 0.01% | 0.23% | 0.72% | 0.02% |
| HID | t | Ng+ |  |  | 14.91% | 20.48% | 11.88% | 11.56% | 12.07% | 12.35% | 11.66% |
| HID | t | Cg- |  |  | 3.48% | 4.79% | 3.13% | 2.48% | 3.22% | 2.28% | 3.62% |
| HID | t | Nt |  |  | 0.90% | 1.11% | 0.61% | 0.85% | 0.89% | 0.45% | 1.70% |
| HID | t | Cg+ |  |  | 9.56% | 12.60% | 9.20% | 6.60% | 7.83% | 9.32% | 4.80% |
| HID | t | Ng- |  |  | 3.50% | 4.00% | 3.93% | 4.53% | 3.63% | 2.53% | 3.50% |
| HID | t | Ct |  |  | 2.88% | 3.66% | 2.83% | 2.84% | 2.86% | 1.17% | 4.25% |
| HID | g- | Ng+ |  |  | 7.93% | 8.75% | 8.04% | 5.62% | 7.84% | 7.03% | 9.46% |
| HID | g- | Cg- |  |  | 16.32% | 14.86% | 19.19% | 21.20% | 12.69% | 16.47% | 18.85% |
| HID | g- | Nt |  |  | 2.00% | 2.99% | 1.19% | 1.15% | 1.10% | 1.90% | 0.74% |
| HID | g- | Cg+ |  |  | 4.51% | 3.31% | 4.24% | 5.00% | 5.95% | 5.87% | 5.71% |
| HID | g- | Ng- |  |  | 20.04% | 14.89% | 19.85% | 31.65% | 19.26% | 23.93% | 26.37% |
| HID | g- | Ct |  |  | 5.62% | 6.35% | 5.84% | 5.59% | 4.66% | 4.98% | 5.67% |
| HIE | g+ | Ng+ |  |  | 2.62% | 1.76% | 6.11% | 0.01% | 3.85% | 1.79% | 2.01% |
| HIE | g+ | Cg- |  |  | 0.81% | 0.21% | 2.19% | 0.02% | 2.35% | 0.38% | 0.51% |
| HIE | g+ | Nt |  |  | 0.01% | 0.00% | 0.00% | 0.00% | 0.04% | 0.01% | 0.04% |
| HIE | g+ | Cg+ |  |  | 2.58% | 1.10% | 2.38% | 0.02% | 4.19% | 5.21% | 3.65% |
| HIE | g+ | Ng- |  |  | 4.68% | 5.15% | 6.54% | 0.01% | 3.50% | 4.85% | 4.32% |
| HIE | g+ | Ct |  |  | 0.31% | 0.25% | 0.35% | 0.00% | 0.35% | 0.16% | 0.13% |

| Residue | χ1 | χ2 | χ3 | χ4 | Ind. | Dep. α-Helix R | Dep. Near α-Helix R | Dep. α-Helix L | Dep. β-Strand | Dep. $P_{IIL}$ | Dep. $P_{IR}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| HIE | t | Ng+ | | | 13.17% | 16.08% | 8.79% | 11.18% | 13.62% | 9.54% | 11.11% |
| HIE | t | Cg- | | | 3.04% | 4.39% | 2.18% | 1.65% | 2.71% | 1.48% | 2.20% |
| HIE | t | Nt | | | 0.78% | 1.06% | 0.94% | 0.44% | 0.75% | 0.11% | 1.67% |
| HIE | t | Cg+ | | | 12.90% | 15.17% | 12.68% | 4.10% | 10.70% | 15.45% | 5.34% |
| HIE | t | Ng- | | | 6.73% | 5.73% | 4.09% | 3.69% | 9.45% | 10.19% | 5.57% |
| HIE | t | Ct | | | 4.20% | 2.85% | 2.40% | 5.88% | 7.09% | 4.52% | 10.00% |
| HIE | g- | Ng+ | | | 5.66% | 6.31% | 5.33% | 5.58% | 3.97% | 6.41% | 2.00% |
| HIE | g- | Cg- | | | 16.20% | 14.33% | 24.23% | 21.30% | 13.42% | 17.77% | 18.41% |
| HIE | g- | Nt | | | 1.98% | 2.82% | 2.19% | 1.35% | 1.01% | 1.26% | 1.20% |
| HIE | g- | Cg+ | | | 4.02% | 2.94% | 4.66% | 6.66% | 5.92% | 3.21% | 8.56% |
| HIE | g- | Ng- | | | 16.03% | 13.90% | 13.16% | 34.39% | 15.18% | 13.64% | 22.04% |
| HIE | g- | Ct | | | 4.29% | 5.96% | 1.77% | 3.72% | 1.90% | 4.01% | 1.26% |
| ILE | g+ | g+ | | | 0.75% | 0.55% | 0.71% | 0.01% | 0.55% | 1.51% | 0.08% |
| ILE | g+ | t | | | 43.32% | 22.67% | 64.45% | 14.42% | 44.48% | 71.16% | 51.90% |
| ILE | g+ | g- | | | 0.10% | 0.02% | 0.10% | 0.02% | 0.14% | 0.22% | 0.11% |
| ILE | t | g+ | | | 0.98% | 0.88% | 1.56% | 0.84% | 1.25% | 0.84% | 1.09% |
| ILE | t | t | | | 2.77% | 1.20% | 2.99% | 0.19% | 5.54% | 2.81% | 3.59% |
| ILE | t | g- | | | 0.05% | 0.05% | 0.07% | 0.01% | 0.07% | 0.04% | 0.03% |
| ILE | g- | g+ | | | 0.36% | 0.15% | 0.29% | 1.09% | 0.78% | 0.28% | 0.84% |
| ILE | g- | t | | | 28.55% | 39.47% | 10.93% | 50.03% | 30.00% | 11.80% | 29.70% |
| ILE | g- | g- | | | 23.12% | 35.01% | 18.90% | 33.39% | 17.20% | 11.34% | 12.67% |
| LEU | g+ | g+ | | | 0.26% | 0.02% | 0.19% | 0.01% | 0.87% | 0.45% | 0.06% |
| LEU | g+ | t | | | 0.19% | 0.01% | 0.17% | 0.00% | 0.69% | 0.23% | 0.22% |
| LEU | g+ | g- | | | 0.00% | 0.00% | 0.00% | 0.00% | 0.01% | 0.01% | 0.00% |
| LEU | t | g+ | | | 22.92% | 24.48% | 21.31% | 15.88% | 29.35% | 14.13% | 25.99% |
| LEU | t | t | | | 3.26% | 3.12% | 3.25% | 3.10% | 5.29% | 1.96% | 4.36% |
| LEU | t | g- | | | 1.19% | 1.49% | 1.64% | 0.58% | 1.12% | 0.44% | 0.82% |
| LEU | g- | g+ | | | 4.18% | 2.71% | 4.14% | 4.50% | 7.46% | 5.70% | 6.65% |
| LEU | g- | t | | | 66.49% | 66.45% | 67.31% | 75.00% | 54.11% | 75.59% | 61.27% |
| LEU | g- | g- | | | 1.51% | 1.71% | 1.99% | 0.92% | 1.11% | 1.49% | 0.63% |
| LYS | g+ | g+ | g+ | g+ | 0.01% | 0.00% | 0.00% | 0.00% | 0.03% | 0.01% | 0.00% |
| LYS | g+ | g+ | g+ | t | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | g+ | g+ | g+ | g- | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | g+ | g+ | t | g+ | 0.06% | 0.03% | 0.13% | 0.00% | 0.06% | 0.14% | 0.01% |
| LYS | g+ | g+ | t | t | 0.01% | 0.01% | 0.01% | 0.00% | 0.01% | 0.04% | 0.00% |
| LYS | g+ | g+ | t | g- | 0.08% | 0.06% | 0.06% | 0.00% | 0.10% | 0.13% | 0.00% |
| LYS | g+ | g+ | g- | g+ | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | g+ | g+ | g- | t | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | g+ | g+ | g- | g- | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | g+ | t | g+ | g+ | 0.22% | 0.03% | 0.22% | 0.01% | 0.62% | 0.39% | 0.24% |
| LYS | g+ | t | g+ | t | 0.14% | 0.02% | 0.05% | 0.00% | 0.42% | 0.31% | 0.18% |
| LYS | g+ | t | g+ | g- | 0.01% | 0.00% | 0.00% | 0.00% | 0.01% | 0.00% | 0.00% |
| LYS | g+ | t | t | g+ | 2.25% | 1.03% | 2.63% | 0.18% | 3.47% | 4.10% | 3.01% |
| LYS | g+ | t | t | t | 0.43% | 0.07% | 0.12% | 0.00% | 1.50% | 0.91% | 0.27% |

| Residue | χ1 | χ2 | χ3 | χ4 | Ind. | Dep. α-Helix R | Dep. Near α-Helix R | Dep. α-Helix L | Dep. β-Strand | Dep. $P_{IIL}$ | Dep. $P_{IR}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LYS | g+ | t | t | g- | 1.21% | 0.38% | 1.80% | 0.11% | 2.68% | 2.13% | 1.32% |
| LYS | g+ | t | g- | g+ | 0.01% | 0.00% | 0.03% | 0.00% | 0.01% | 0.01% | 0.01% |
| LYS | g+ | t | g- | t | 0.13% | 0.02% | 0.05% | 0.00% | 0.34% | 0.33% | 0.14% |
| LYS | g+ | t | g- | g- | 0.52% | 0.15% | 0.63% | 0.02% | 1.05% | 1.16% | 0.56% |
| LYS | g+ | g- | g+ | g+ | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | g+ | g- | g+ | t | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | g+ | g- | g+ | g- | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | g+ | g- | t | g+ | 0.02% | 0.00% | 0.01% | 0.00% | 0.09% | 0.02% | 0.05% |
| LYS | g+ | g- | t | t | 0.00% | 0.00% | 0.00% | 0.00% | 0.01% | 0.00% | 0.00% |
| LYS | g+ | g- | t | g- | 0.02% | 0.01% | 0.08% | 0.00% | 0.04% | 0.02% | 0.02% |
| LYS | g+ | g- | g- | g+ | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | g+ | g- | g- | t | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | g+ | g- | g- | g- | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | t | g+ | g+ | g+ | 0.22% | 0.29% | 0.27% | 0.09% | 0.14% | 0.18% | 0.09% |
| LYS | t | g+ | g+ | t | 0.19% | 0.31% | 0.07% | 0.07% | 0.10% | 0.04% | 0.11% |
| LYS | t | g+ | g+ | g- | 0.00% | 0.01% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | t | g+ | t | g+ | 2.02% | 2.54% | 1.93% | 1.15% | 1.84% | 0.90% | 2.72% |
| LYS | t | g+ | t | t | 0.29% | 0.42% | 0.17% | 0.09% | 0.22% | 0.12% | 0.29% |
| LYS | t | g+ | t | g- | 1.42% | 1.78% | 1.12% | 0.64% | 1.37% | 0.89% | 1.26% |
| LYS | t | g+ | g- | g+ | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | t | g+ | g- | t | 0.03% | 0.06% | 0.01% | 0.01% | 0.01% | 0.00% | 0.01% |
| LYS | t | g+ | g- | g- | 0.04% | 0.06% | 0.02% | 0.05% | 0.02% | 0.01% | 0.07% |
| LYS | t | t | g+ | g+ | 2.32% | 3.04% | 1.67% | 1.05% | 2.11% | 1.21% | 1.85% |
| LYS | t | t | g+ | t | 0.87% | 1.17% | 0.26% | 0.12% | 1.01% | 0.47% | 1.07% |
| LYS | t | t | g+ | g- | 0.03% | 0.03% | 0.02% | 0.02% | 0.02% | 0.02% | 0.01% |
| LYS | t | t | t | g+ | 3.52% | 3.72% | 3.12% | 1.69% | 4.85% | 2.31% | 6.09% |
| LYS | t | t | t | t | 0.77% | 0.75% | 0.37% | 0.28% | 1.39% | 0.73% | 0.87% |
| LYS | t | t | t | g- | 4.81% | 5.95% | 3.14% | 3.82% | 4.90% | 2.55% | 5.87% |
| LYS | t | t | g- | g+ | 0.02% | 0.00% | 0.00% | 0.01% | 0.06% | 0.04% | 0.01% |
| LYS | t | t | g- | t | 0.36% | 0.39% | 0.27% | 0.13% | 0.47% | 0.36% | 0.27% |
| LYS | t | t | g- | g- | 0.53% | 0.54% | 0.54% | 0.31% | 0.79% | 0.35% | 0.81% |
| LYS | t | g- | g+ | g+ | 0.01% | 0.01% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | t | g- | g+ | t | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | t | g- | g+ | g- | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | t | g- | t | g+ | 0.30% | 0.40% | 0.21% | 0.16% | 0.32% | 0.12% | 0.38% |
| LYS | t | g- | t | t | 0.07% | 0.08% | 0.05% | 0.04% | 0.07% | 0.03% | 0.06% |
| LYS | t | g- | t | g- | 0.52% | 0.70% | 0.40% | 0.23% | 0.48% | 0.13% | 0.78% |
| LYS | t | g- | g- | g+ | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | t | g- | g- | t | 0.06% | 0.08% | 0.05% | 0.00% | 0.05% | 0.02% | 0.04% |
| LYS | t | g- | g- | g- | 0.15% | 0.20% | 0.08% | 0.05% | 0.18% | 0.05% | 0.15% |
| LYS | g- | g+ | g+ | g+ | 0.11% | 0.07% | 0.11% | 0.29% | 0.12% | 0.16% | 0.08% |
| LYS | g- | g+ | g+ | t | 0.03% | 0.04% | 0.04% | 0.05% | 0.03% | 0.02% | 0.03% |
| LYS | g- | g+ | g+ | g- | 0.00% | 0.00% | 0.00% | 0.02% | 0.01% | 0.01% | 0.01% |
| LYS | g- | g+ | t | g+ | 0.61% | 0.58% | 1.07% | 0.46% | 0.66% | 0.54% | 0.59% |

| Residue | χ1 | χ2 | χ3 | χ4 | Ind. | Dep. α-Helix R | Dep. Near α-Helix R | Dep. α-Helix L | Dep. β-Strand | Dep. $P_{IIL}$ | Dep. $P_{IR}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LYS | g- | g+ | t | t | 0.06% | 0.05% | 0.06% | 0.07% | 0.06% | 0.05% | 0.06% |
| LYS | g- | g+ | t | g- | 0.39% | 0.37% | 0.48% | 0.36% | 0.44% | 0.43% | 0.45% |
| LYS | g- | g+ | g- | g+ | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | g- | g+ | g- | t | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| LYS | g- | g+ | g- | g- | 0.01% | 0.02% | 0.01% | 0.00% | 0.00% | 0.01% | 0.00% |
| LYS | g- | t | g+ | g+ | 4.79% | 4.56% | 4.27% | 4.89% | 4.94% | 5.49% | 5.01% |
| LYS | g- | t | g+ | t | 2.01% | 2.35% | 0.91% | 1.42% | 1.83% | 2.15% | 1.44% |
| LYS | g- | t | g+ | g- | 0.09% | 0.08% | 0.04% | 0.12% | 0.08% | 0.10% | 0.21% |
| LYS | g- | t | t | g+ | 14.99% | 13.34% | 16.48% | 24.88% | 14.52% | 16.08% | 15.10% |
| LYS | g- | t | t | t | 4.96% | 6.42% | 2.50% | 3.23% | 3.49% | 4.39% | 2.88% |
| LYS | g- | t | t | g- | 22.34% | 23.16% | 22.72% | 25.24% | 19.66% | 21.00% | 21.22% |
| LYS | g- | t | g- | g+ | 0.19% | 0.21% | 0.20% | 0.23% | 0.09% | 0.23% | 0.09% |
| LYS | g- | t | g- | t | 3.67% | 4.51% | 1.58% | 2.64% | 2.90% | 3.72% | 2.39% |
| LYS | g- | t | g- | g- | 5.13% | 3.62% | 5.88% | 12.35% | 5.32% | 6.37% | 5.76% |
| LYS | g- | g- | g+ | g+ | 0.26% | 0.41% | 0.18% | 0.03% | 0.09% | 0.11% | 0.07% |
| LYS | g- | g- | g+ | t | 0.09% | 0.15% | 0.05% | 0.02% | 0.03% | 0.04% | 0.02% |
| LYS | g- | g- | g+ | g- | 0.00% | 0.00% | 0.00% | 0.03% | 0.00% | 0.00% | 0.00% |
| LYS | g- | g- | t | g+ | 6.06% | 6.26% | 6.70% | 4.86% | 4.97% | 6.88% | 4.88% |
| LYS | g- | g- | t | t | 1.04% | 1.03% | 1.24% | 0.73% | 0.96% | 1.24% | 0.68% |
| LYS | g- | g- | t | g- | 7.64% | 6.31% | 13.94% | 6.52% | 7.44% | 8.85% | 9.19% |
| LYS | g- | g- | g- | g+ | 0.02% | 0.02% | 0.01% | 0.01% | 0.01% | 0.02% | 0.01% |
| LYS | g- | g- | g- | t | 0.66% | 0.82% | 0.56% | 0.21% | 0.54% | 0.65% | 0.33% |
| LYS | g- | g- | g- | g- | 1.19% | 1.25% | 1.30% | 1.02% | 0.99% | 1.20% | 0.85% |
| MET | g+ | g+ | g+ |  | 0.07% | 0.01% | 0.08% | 0.00% | 0.20% | 0.15% | 0.00% |
| MET | g+ | g+ | t |  | 0.07% | 0.01% | 0.06% | 0.00% | 0.22% | 0.12% | 0.01% |
| MET | g+ | g+ | g- |  | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| MET | g+ | t | g+ |  | 0.73% | 0.06% | 0.58% | 0.06% | 2.28% | 1.37% | 0.79% |
| MET | g+ | t | t |  | 0.76% | 0.06% | 0.72% | 0.07% | 2.40% | 1.41% | 0.53% |
| MET | g+ | t | g- |  | 0.79% | 0.08% | 1.09% | 0.04% | 2.13% | 1.93% | 0.47% |
| MET | g+ | g- | g+ |  | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.01% | 0.00% |
| MET | g+ | g- | t |  | 0.08% | 0.00% | 0.04% | 0.00% | 0.25% | 0.23% | 0.02% |
| MET | g+ | g- | g- |  | 0.07% | 0.00% | 0.03% | 0.00% | 0.32% | 0.09% | 0.02% |
| MET | t | g+ | g+ |  | 1.92% | 2.44% | 2.27% | 0.98% | 1.55% | 1.11% | 1.30% |
| MET | t | g+ | t |  | 2.17% | 2.94% | 2.33% | 0.74% | 1.60% | 0.93% | 1.33% |
| MET | t | g+ | g- |  | 0.09% | 0.12% | 0.07% | 0.05% | 0.06% | 0.02% | 0.09% |
| MET | t | t | g+ |  | 4.50% | 5.14% | 3.35% | 2.06% | 5.26% | 3.07% | 5.00% |
| MET | t | t | t |  | 2.53% | 2.39% | 2.98% | 1.56% | 3.71% | 1.90% | 3.90% |
| MET | t | t | g- |  | 2.55% | 2.45% | 2.55% | 1.91% | 3.79% | 1.86% | 3.67% |
| MET | t | g- | g+ |  | 0.01% | 0.01% | 0.01% | 0.01% | 0.01% | 0.00% | 0.02% |
| MET | t | g- | t |  | 0.52% | 0.72% | 0.65% | 0.21% | 0.31% | 0.16% | 0.46% |
| MET | t | g- | g- |  | 1.16% | 1.80% | 0.99% | 0.15% | 0.54% | 0.29% | 0.62% |
| MET | g- | g+ | g+ |  | 0.72% | 0.33% | 1.14% | 1.47% | 1.04% | 1.27% | 1.25% |
| MET | g- | g+ | t |  | 0.83% | 0.65% | 1.11% | 1.09% | 0.94% | 1.08% | 1.30% |
| MET | g- | g+ | g- |  | 0.03% | 0.04% | 0.02% | 0.01% | 0.01% | 0.03% | 0.01% |

| Residue | χ1 | χ2 | χ3 | χ4 | Ind. | Dep. α-Helix R | Dep. Near α-Helix R | Dep. α-Helix L | Dep. β-Strand | Dep. $P_{IIL}$ | Dep. $P_{IR}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MET | g- | t | g+ |  | 16.40% | 15.74% | 14.54% | 18.44% | 17.15% | 17.31% | 18.70% |
| MET | g- | t | t |  | 15.82% | 14.04% | 13.67% | 18.81% | 17.96% | 18.65% | 19.40% |
| MET | g- | t | g- |  | 20.74% | 18.91% | 18.42% | 31.16% | 22.10% | 22.81% | 24.69% |
| MET | g- | g- | g+ |  | 0.94% | 1.36% | 0.57% | 0.30% | 0.29% | 0.65% | 0.16% |
| MET | g- | g- | t |  | 11.48% | 12.98% | 13.23% | 9.51% | 7.09% | 10.57% | 7.49% |
| MET | g- | g- | g- |  | 15.06% | 17.72% | 19.49% | 11.37% | 8.77% | 13.00% | 8.76% |
| PHE | g+ | g |  |  | 6.47% | 0.82% | 7.04% | 0.23% | 15.79% | 9.44% | 3.27% |
| PHE | g+ | t |  |  | 0.02% | 0.00% | 0.04% | 0.00% | 0.03% | 0.03% | 0.01% |
| PHE | t | g |  |  | 39.54% | 59.47% | 39.89% | 22.88% | 25.89% | 20.18% | 30.21% |
| PHE | t | t |  |  | 2.79% | 3.67% | 2.14% | 3.15% | 2.40% | 0.89% | 5.15% |
| PHE | g- | g |  |  | 45.99% | 29.79% | 47.36% | 69.48% | 52.54% | 63.16% | 58.75% |
| PHE | g- | t |  |  | 5.19% | 6.25% | 3.53% | 4.26% | 3.35% | 6.30% | 2.61% |
| PRO | g+ |  |  |  | 62.93% | 58.04% | 97.66% | 0.00% | 81.32% | 65.62% | 100.00% |
| PRO | g- |  |  |  | 37.07% | 41.96% | 2.34% | 100.00% | 18.68% | 34.38% | 0.00% |
| SER | g+ |  |  |  | 22.69% | 14.33% | 19.70% | 1.91% | 31.69% | 34.91% | 26.59% |
| SER | t |  |  |  | 2.05% | 1.30% | 1.98% | 2.39% | 3.91% | 1.95% | 4.36% |
| SER | g- |  |  |  | 75.25% | 84.37% | 78.31% | 95.70% | 64.40% | 63.14% | 69.04% |
| THR | g+ |  |  |  | 45.14% | 26.50% | 55.71% | 13.97% | 52.54% | 61.66% | 57.70% |
| THR | t |  |  |  | 1.71% | 0.12% | 0.93% | 0.36% | 4.84% | 1.46% | 4.78% |
| THR | g- |  |  |  | 53.15% | 73.38% | 43.36% | 85.67% | 42.63% | 36.88% | 37.52% |
| TRP | g+ | g+ |  |  | 6.57% | 4.05% | 14.85% | 0.05% | 11.08% | 5.98% | 1.85% |
| TRP | g+ | t |  |  | 0.51% | 0.06% | 0.16% | 0.00% | 0.86% | 1.38% | 0.37% |
| TRP | g+ | g- |  |  | 3.94% | 0.95% | 5.75% | 0.05% | 8.22% | 5.83% | 3.66% |
| TRP | t | g+ |  |  | 16.59% | 22.66% | 15.48% | 10.69% | 10.90% | 12.05% | 11.85% |
| TRP | t | t |  |  | 10.53% | 12.39% | 8.80% | 7.45% | 11.17% | 4.74% | 22.51% |
| TRP | t | g- |  |  | 13.28% | 21.20% | 10.92% | 2.88% | 8.47% | 5.17% | 6.12% |
| TRP | g- | g+ |  |  | 7.19% | 3.20% | 8.98% | 22.40% | 9.20% | 9.08% | 16.93% |
| TRP | g- | t |  |  | 14.98% | 14.92% | 12.51% | 17.46% | 11.15% | 20.57% | 11.69% |
| TRP | g- | g- |  |  | 26.40% | 20.56% | 22.55% | 39.01% | 28.94% | 35.20% | 25.03% |
| TYR | g+ | g |  |  | 8.27% | 1.44% | 8.56% | 0.70% | 18.84% | 11.34% | 4.28% |
| TYR | g+ | t |  |  | 0.02% | 0.00% | 0.00% | 0.00% | 0.04% | 0.04% | 0.08% |
| TYR | t | g |  |  | 37.99% | 57.87% | 35.08% | 26.01% | 24.75% | 21.51% | 25.61% |
| TYR | t | t |  |  | 2.85% | 3.88% | 1.67% | 2.40% | 2.38% | 0.98% | 5.63% |
| TYR | g- | g |  |  | 45.57% | 30.22% | 51.02% | 68.31% | 50.71% | 59.62% | 61.07% |
| TYR | g- | t |  |  | 5.29% | 6.60% | 3.66% | 2.58% | 3.29% | 6.52% | 3.33% |
| VAL | g+ |  |  |  | 1.36% | 0.71% | 1.19% | 0.18% | 2.32% | 1.41% | 1.33% |
| VAL | t |  |  |  | 55.53% | 81.10% | 43.65% | 89.09% | 50.17% | 26.78% | 48.76% |
| VAL | g- |  |  |  | 43.11% | 18.19% | 55.16% | 10.73% | 47.51% | 71.81% | 49.91% |

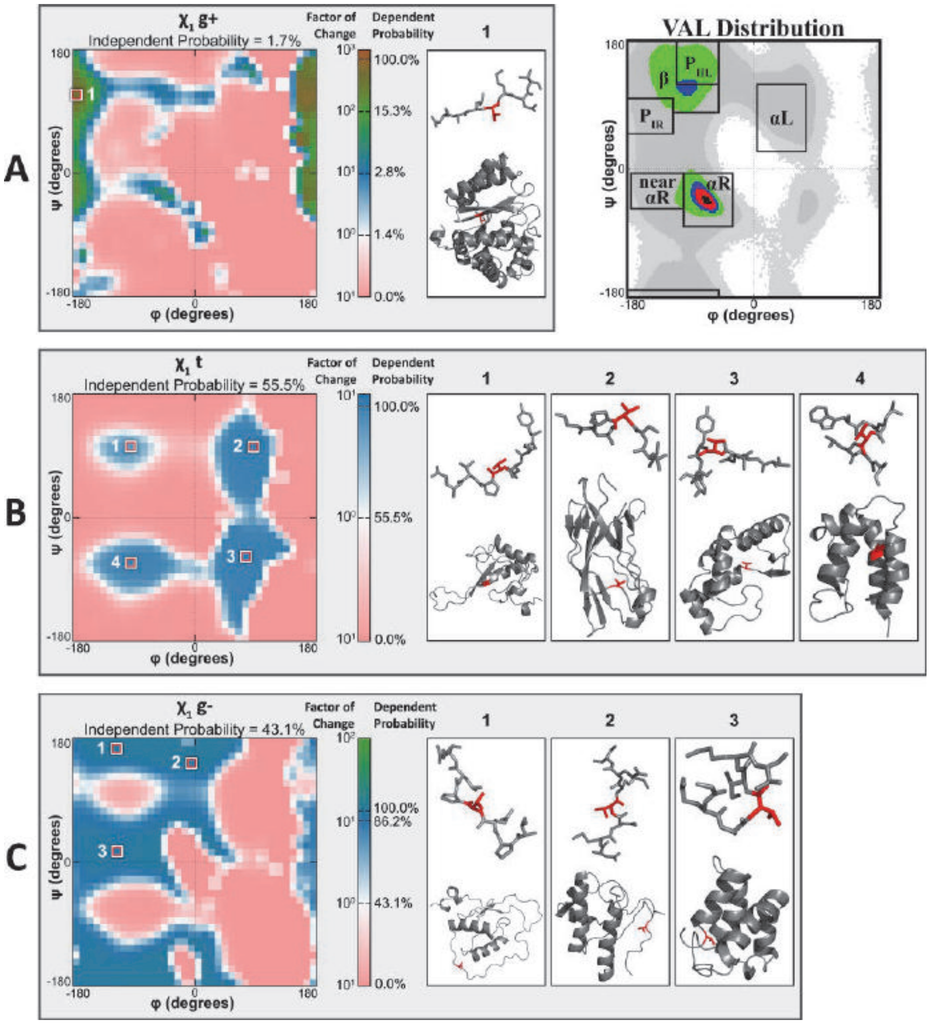


**Figure 6.1 Dependent rotamer probabilities and example structures for valine.**

The overall backbone-dependent distribution of valine residues in the Dynameomics 2009 release set is shown in the upper right-hand corner of the image with outlined secondary structure regions. Bins are colored by percentage of maximum bin size as follows: 0 < light gray ≤ 0.01, 0.01 < gray ≤ 0.05, 0.05 < green ≤ 0.2, 0.2 < blue ≤ 0.4, 0.4 < red ≤ 0.8, 0.8 < black. Panels A, B and C represent the gauche+ (g+), trans (t), and gauche- (g-) rotamers of valine, respectively. The plot on the left of each panel shows the distribution of backbone-dependent rotamer probabilities versus backbone-independent probabilities. The plots range from a 0% to 100% backbone-dependent probability on a logarithmic scale for improved contrast. Example rotamer conformations, with their source protein structure, are shown on the right portion of each panel and the numbers correspond to areas of interest in the distribution plots.

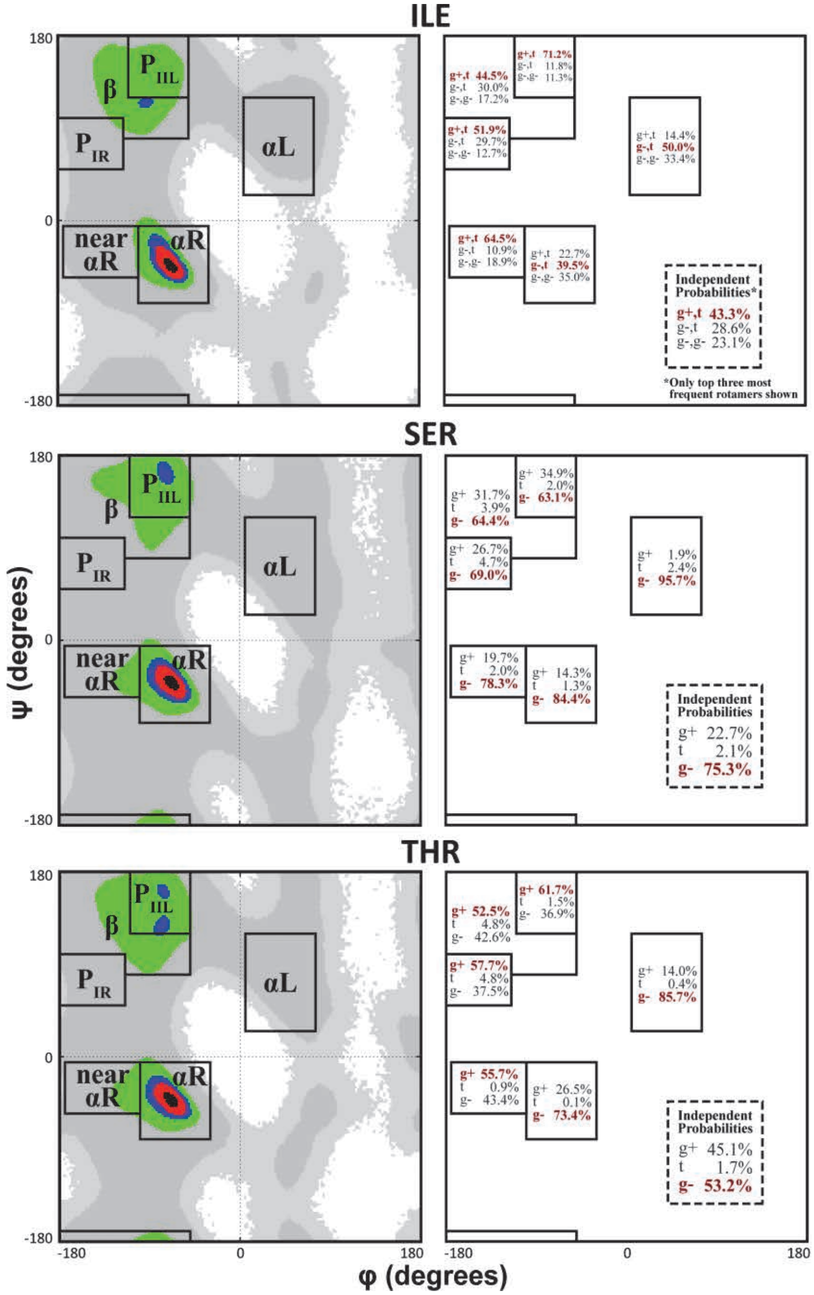


**Figure 6.2 Dependent probabilities per secondary structure for a selection of residues.**

Dependent probabilities per secondary structure for a selection of residues. The plots on the left depict the backbone-dependent distribution of the respective residue from simulations in the Dynameomics 2009 release set. The overlays highlight the secondary structure regions. Bins are colored by percentage of maximum bin size as follows: 0 < light gray ≤ 0.01, 0.01 < gray ≤ 0.05, 0.05 < green ≤ 0.2, 0.2 < blue ≤ 0.4, 0.4 < red ≤ 0.8, 0.8 < black. The panels on the right show the probabilities for the top three rotamers per secondary structure with the most probable rotamer highlighted in red. The complete probabilities can be found in Table 6.2.

# Chapter 7

# RELATED AND CONTINUING WORK

Scientific research extends beyond developing novel methods to test hypotheses and answer difficult questions. An important aspect of research is generalizing the methods, models and tools so that they can be reused and applied to new problems. Much of the research presented in this dissertation has been designed with the intent of sharing the methods, software, and resulting data. Here we present several ongoing projects that either extend the scope of the previously described research or capitalize on the existing frameworks and software to accelerate the research process. These works range in completeness from early stages of development to submitted articles of the completed research. In particular, the research in Sections 7.1 and 7.2 has been submitted for publication, the research in Section 7.3 has produced preliminary results, and the research in Sections 7.4 and 7.5 is ongoing work.

## *7.1 Evaluation of Cross-Linking Distances using Dynameomics*

*This research was led by Erik D. Merkley and submitted for publication in late 2013* (Merkley, Rysavy, et al., 2013).

Protein complexes are difficult to model with conventional approaches due to their size, dynamics, and complexity and can result in ambiguous models (Henderson et al., 2012; Lau et al., 2012; H.-W. Wang et al., 2009). These models can benefit from integrative structural modeling, the combination of computational protein structure prediction and experimentally derived data (Schneidman-Duhovny et al., 2012; Ward, Sali, & Wilson, 2013). One particular experimental method, chemical cross-linking mass spectrometry (Merkley, Cort, & Adkins, 2013; Sinz, 2006), provides three-dimensional distance restraints that can inform protein modelling efforts by specifying pairs of residue that are close to one another. However, there can be discrepancies between the residues' cross-linked distance and the distance calculated from a known structure. For this analysis we investigate a commonly used cross-linker, bis(sulfosuccinimidyl)suberate, which primarily targets lysine residues. We used simulations from the Dynameomics data warehouse to investigate the change in distance between pairs of lysine residues due to native state dynamics. These results in turn will inform researchers using cross-linking mass spectrometry to model protein complexes. Additional information will be published by Merkley *et al.* (Merkley, Rysavy, et al., 2013).

Distances between lysine-lysine atoms were calculated for all 807 native state simulations in the Dynameomics data warehouse using the Dynameomics API presented in Chapter 4. Of these 807 simulations, 766 contained two or more lysine residues. Distances were calculated between both Cα-Cα and Nζ-Nζ atoms pairs for all the lysine-lysine residue pairs at 100 ps intervals. This resulted in 4.6 x $10^7$ distance measurements from 43,354 lysine residues using the Dynameomics API.

## *7.2 Comparison of Native and Denatured State Protein Fold Space Coverage*

*This research was led by Clare Louise Towse and an article is near completion. This manuscript will be submitted for publication in early 2014* (Towse, Rysavy, & Daggett, 2014).

Dynameomics is an ongoing project to study the dynamics of all known protein structure folds (Beck et al., 2008). In order to characterize dynamics across protein fold space, a quantification of the space was needed. Our previous approach defined protein fold space via a consensus of classifications from three separate taxonomies: CATH (Pearl et al., 2003), Dali

(Holm & Sander, 1996), and SCOP (Andreeva et al., 2004). As a result, a comprehensive set of 807 protein structures, representing 95% of protein fold space, were selected for Dynameomics dataset using this consensus view of fold space (Day et al., 2003; Schaeffer et al., 2011). To extend this work, we implemented a recently introduced method called the Proteomic Ramachandran plot (Carugo & Djinović-Carugo, 2013) to further assess the space. We are now able to visually depict Dynameomics' coverage of protein fold space and confirm that our 807 representative proteins have comparable coverage to the PDB. Additionally, we can now detect and assess shifts in protein fold space for unfolding proteins across all Dynameomics simulations. Detailed information on this project will be published by Towse *et al.* (Towse et al., 2014).

The Dynameomics API (Chapter 4) was used to calculate and aggregate φ and ψ backbone angles for experimental structures in a subset of the PDB. The PDB set was generated using *PISCES* with a filter to only include X-ray crystal structures higher in resolution than 2.7 Å, less than 95% sequence identity, and having an R-factor of less than 3 Å (Gouli Wang & Dunbrack Jr., 2005; Guoli Wang & Dunbrack Jr., 2003). A command line interface program was created, referencing the Dynameomics API assemblies, to process all 23,080 protein chains in this PDB set.

### *7.3 Transition State Ensemble Prediction*

Defining the process by which a chain of amino acids (polypeptide) forms into a functional, three-dimensional protein structure is one of the biggest problems in molecular biology. This process, described as the folding/unfolding pathway, can be classified into four distinct regions: the native state, the transition state, the intermediate state, and the unfolded state (Fersht, 1999). The transition state (TS) is of particular interest due to its role in the protein folding pathway;

extended polypeptides must pass through this state in order to fold into the correct native state structure, which has even been demonstrated *in silico* (McCully, Beck, Fersht, & Daggett, 2010). We have developed a novel method to automatically determine the TS of a protein folding pathway in MD protein simulations using internal-coordinate based fragments. Using the Dynameomics data warehouse, we provide the promising results of this informatics-driven, high-throughput approach.

### 7.3.1 *Introduction and Background*

The Dynameomics data warehouse contains MD simulations of key protein structures representing the majority of all known protein folds. Each of these proteins is simulated at both 25º C and 225º C to mimic an *in-vitro* environment and thermal denaturation, respectively. These two types of simulations model the native state dynamics and protein folding/unfolding pathways. Identification of the TS ensemble in the denaturing simulations will greatly improve our understanding of the protein folding process. However, the Dynameomics data warehouse contains nearly 11,000 simulations totaling over 100TB in data storage space. This is prohibitively expensive for a manual or even semi-automatic processes (Toofanny, Jonsson, & Daggett, 2010); an automated, high-throughput method is necessary to identify TS ensembles for each of the protein simulations.

### 7.3.2 *Methods*

We identify TS ensembles by comparing a protein’s native state simulation with its denaturing simulation. The starting protein structure is first automatically parsed to determine regions of local structure, called secondary structure. Secondary structure elements are translated into fragment representations using the inter- and intra-atomic distances between non-hydrogen atoms in each fragment's terminal amino acids. Other fragments are similarly generated between these secondary structure elements, resulting in a collection of structurally representative fragments. Using an internal-coordinate comparison of these fragment distances, we gather statistics over the course of each simulation. The denaturing simulation statistics are analyzed against the native state simulation statistics for significant differences, which we use to mark the beginning and end of the TS.

The automated TS algorithm is based on internal coordinate (IC) fragment representations (see Chapter 5). These fragments are described by 45 inter- and intra-distances calculated between the heavy atoms in each terminal residue of the fragment (C, Cα, Cβ, N, O). Once obtained, these 45 distances can be compared between fragments using a root-mean-square deviation (RMSD) calculation to produce a structure similarity metric. The internal coordinate RMSD ($\text{RMSD}_{\text{IC}}$) results in similar structure comparison values as a Cα RMSD comparison, but has the advantage of being internally consistent; that is, $\text{RMSD}_{\text{IC}}$ calculations do not require structure alignments whereas Cα RMSD alignments do.

The lack of a pre-comparison structural alignment is the primary reason this algorithm is efficient. The method enables multiple, simultaneous fragment comparisons between protein structures within a trajectory. Using this concept, we created an algorithm that divides a protein structure into many fragments that are then compared over time. These comparisons are used to detect local and global structural variations. The specific steps of the algorithm are described in detail below.

1. **Determine secondary structure regions.** Using DSSP classifications from the Dynameomics data warehouse, we automatically determine the initial secondary structure assignments for the starting structure of the protein simulation. Specifically, regions of three or more contiguous 3-10 helix, α-helix, or π-helix residues are classified as α-helix structures. Regions of three or more contiguous parallel or anti-parallel β-strand regions are classified as β-strand structures. All other regions are classified as loop structures.
2. **Assign fragments.** Well-defined secondary structure elements (α-helix and β-strand regions) are labeled as secondary structure fragments. Loop regions are labeled as loop fragments. Fragments are also defined between each pair of well-defined secondary structure elements using the center of the secondary structure elements as terminal residues.
3. **Generate native state statistics.** Each fragment is compared to its starting structure counterpart using IC comparisons at predetermined time intervals. This list of resulting $\text{RMSD}_{\text{IC}}$ values is processed to ascertain the average variance, maximum variance, and standard deviation of variance for each fragment assigned to the protein structure.
4. **Generate unfolding statistics.** Similar to Step 3, statistics are gathered for each fragment over a predetermined region of the unfolding trajectory. Again, the average variance, maximum variance, and standard deviation of variance for each fragment are collected over this window.
5. **Compare native state and unfolding statistics.** The final step of the algorithm is to compare the native state fragment statistics with the unfolding fragment statistics. The

overall concept is to normalize the standard deviations between the two sets of fragment statistics to determine which fragments are most mobile. From this normalization, maximum $\text{RMSD}_{\text{IC}}$ values for native-like fragments are generated. We then define the start of the transition state when all fragments have surpassed their respective maximum $\text{RMSD}_{\text{IC}}$ value.

### 7.3.3 *Preliminary Results and Discussion*

We analyzed the TS ensembles of five protein simulations that also have experimentally determined data characterizing the TS structure, named Φ-values. We calculated the correlation between the experimental Φ-values and computationally-derived S-values to assess the TS ensembles' agreement with experiment. Preliminary results show that the automated method performs similarly to alternative TS-picking methods for four of the five simulations. However, this novel method for choosing TS ensembles from MD simulations is fully-automated and much less computationally-intensive. Refinement of the core algorithm should improve the results. Furthermore, this technique of comparing fragments using internal-coordinate comparisons may work well to determine other structural features in a protein folding/unfolding pathway as well.

## *7.4 Pentapeptide Structural Conformations*

Kabsch and Sander previously reported on the relationship between sequence and structure for peptides of five residues in length (pentapeptides) (W. Kabsch & Sander, 1984). They found, among other results, that pentapeptides with identical structures can have different secondary structure. This was an extremely important finding for protein structure prediction and modeling. However, the study was performed over 20 years ago on a mere 62 small protein structures, averaging approximately 160 residues per structure. We intend to mine the entirety of the native state simulations from the Dynameomics data warehouse to update this study with dynamic protein structure data and a more comprehensive set of proteins.

As described in Chapter 5, the fragment library schema is extensible. To address the pentapeptide research endeavor, we have created fragment tables populated with both sequence and structure data. Each row of these tables corresponds to a unique fragment and each row contains a residue name, a DSSP structure code (Wolfgang Kabsch & Sander, 1983), and a φ-ψ structure code per residue (Simms & Daggett, 2012). Using the Dynameomics API's sequence library, we have produced an overall structure assignment per fragment based on the residue structure values. These additional analysis tables are not restricted to pentapeptides, but can be generated for any length of fragment. Additionally, the analysis can be performed for Dynameomics or PDB-based fragment libraries (with the exception that DSSP values will not be available for PDB fragments).

Using the sequence software library in the Dynameomics API in conjunction with SQL, we can now query fragment sequences from any fragment library and aggregate the results based on secondary structure type. The sequence queries support basic Boolean logic for matching residues, so wildcard operators and classifications of residues can be used on instead of only strictly identical residue sequences.

### *7.5 D-Amino Acid Rotamer Library Comparison*

Amino acids exist in one of two isomeric forms, L- (left-handed) or D- (right-handed) chiral molecules, in which the side-chain atoms are mirrored structural conformations of one another about the Cα backbone atom. All proteins in nature are formed from L-amino acids.

However, D-amino acids are prevalent in nature outside of protein structures and have many important applications to areas such as antibiotic design (Wade et al., 1990) and peptide design (Hopping, Kellock, Caughey, & Daggett, 2013).  Since D-amino acids do not naturally occur in proteins, there is very little experimental structural data on their conformations or dynamics.  The Dynameomics API presented in Chapter 4 was used to produce a D-amino acid rotamer library from small peptide simulations containing guest D-amino acids.  We plan to evaluate the D-amino acid rotamer library and compare the contents with the L-amino acid rotamer library, described in detail in Chapter 6.

## BIBLIOGRAPHY

Andreeva, A., Howorth, D., Brenner, S. E., Hubbard, T. J. P., Chothia, C., & Murzin, A. G. (2004). SCOP database in 2004: refinements integrate structure and sequence family data. *Nucleic Acids Research*, *32*(suppl 1), D226–D229. doi:10.1093/nar/gkh039

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., … Sherlock, G. (2000). Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nature Genetics*, *25*(1), 25–29. doi:10.1038/75556

Baeten, L., Reumers, J., Tur, V., Stricher, F., Lenaerts, T., Serrano, L., … Schymkowitz, J. (2008). Reconstruction of Protein Backbones from the BriX Collection of Canonical Protein Fragments. *PLoS Comput Biol*, *4*(5), e1000083. doi:10.1371/journal.pcbi.1000083

Bahar, I., & Jernigan, R. L. (1996). Coordination geometry of nonbonded residues in globular proteins. *Folding and Design*, *1*(5), 357–370. doi:10.1016/S1359-0278(96)00051-X

Beauchamp, K. A., Lin, Y.-S., Das, R., & Pande, V. S. (2012). Are Protein Force Fields Getting Better? A Systematic Benchmark on 524 Diverse NMR Measurements. *Journal of Chemical Theory and Computation*, *8*(4), 1409–1414. doi:10.1021/ct2007814

Beck, D. A. ., Alonso, D. O. ., & Daggett, V. (2003). A microscopic view of peptide and protein solvation. *Biophysical Chemistry*, *100*(1–3), 221–237. doi:10.1016/S0301-4622(02)00283-1

Beck, D. A. ., & Daggett, V. (2004). Methods for molecular dynamics simulations of protein folding/unfolding in solution. *Methods*, *34*(1), 112–120. doi:10.1016/j.ymeth.2004.03.008

Beck, D. A. C., Alonso, D. O. V., & Daggett, V. (2000). *in lucem Molecular Mechanics (ilmm)*.

Beck, D. A. C., Jonsson, A. L., Schaeffer, R. D., Scott, K. A., Day, R., Toofanny, R. D., … Daggett, V. (2008). Dynameomics: mass annotation of protein dynamics and unfolding in water by high-throughput atomistic molecular dynamics simulations. *Protein Engineering Design and Selection*, *21*(6), 353–368. doi:10.1093/protein/gzn011

Benson, N. C., & Daggett, V. (2008). Dynameomics: Large-scale assessment of native protein flexibility. *Protein Science*, *17*(12), 2038–2050. doi:10.1110/ps.037473.108

Berman, H. M., Kleywegt, G. J., Nakamura, H., & Markley, J. L. (2013). How Community Has Shaped the Protein Data Bank. *Structure*, *21*(9), 1485–1491. doi:10.1016/j.str.2013.07.010

Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., … Bourne, P. E. (2000). The Protein Data Bank. *Nucleic Acids Research*, *28*(1), 235–242. doi:10.1093/nar/28.1.235

Bernstein, F. C, Koetzle, T. F., Williams, G. J. B., Meyer, E. F., Brice, M. D., Rodgers, J. R., … Tasumi, M. (1977). The protein data bank: A computer-based archival file for macromolecular structures. *Journal of Molecular Biology*, *112*(3), 535–542.

Bernstein, Frances C., Koetzle, T. F., Williams, G. J. B., Meyer Jr., E. F., Brice, M. D., Rodgers, J. R., … Tasumi, M. (1977). The protein data bank: A computer-based archival file for macromolecular structures. *Journal of Molecular Biology*, *112*(3), 535–542. doi:10.1016/S0022-2836(77)80200-3

Branden, C., & Tooze, J. (1991). *Introduction to protein structure* (Vol. 2). Garlnad New York.

Bromley, D., Anderson, P. C., & Daggett, V. (2013). Structural Consequences of Mutations to the α-Tocopherol Transfer Protein Associated with the Neurodegenerative Disease Ataxia with Vitamin E Deficiency. *Biochemistry, 52*(24), 4264–4273. doi:10.1021/bi4001084

Bromley, D., Rysavy, S. J., Su, R., Toofanny, R. D., Schmidlin, T., & Daggett, V. (2013). DIVE: A Data Intensive Visualization Engine. *Bioinformatics*, *30*(4), 593–595. doi:10.1093/bioinformatics/btt721

Bromley, D., Rysavy, S. J., Su, R., Toofany, R. D., Schmidlin, T., & Daggett, V. (2013). DIVE - A Data Intensive Visualization Engine.

Carugo, O., & Djinović-Carugo, K. (2013). A proteomic Ramachandran plot (PRplot). *Amino Acids*, *44*(2), 781–790. doi:10.1007/s00726-012-1402-z

Chang, G., Roth, C. B., Reyes, C. L., Pornillos, O., Chen, Y.-J., & Chen, A. P. (2006). Retraction. *Science*, *314*(5807), 1875–1875. doi:10.1126/science.314.5807.1875b

Choi, Y., & Deane, C. M. (2010). FREAD revisited: Accurate loop structure prediction using a database search algorithm. *Proteins: Structure, Function, and Bioinformatics, 78*(6), 1431–1440. doi:10.1002/prot.22658

Cino, E. A., Choy, W.-Y., & Karttunen, M. (2012). Comparison of Secondary Structure Formation Using 10 Different Force Fields in Microsecond Molecular Dynamics Simulations. *Journal of Chemical Theory and Computation*, *8*(8), 2725–2740. doi:10.1021/ct300323g

Cock, P. J. A., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., … de Hoon, M. J. L. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics (Oxford, England)*, *25*(11), 1422–1423. doi:10.1093/bioinformatics/btp163

Davis, I. W., Leaver-Fay, A., Chen, V. B., Block, J. N., Kapral, G. J., Wang, X., … Richardson, D. C. (2007). MolProbity: all-atom contacts and structure validation for proteins and nucleic acids. *Nucleic Acids Research*, *35*(suppl 2), W375–W383. doi:10.1093/nar/gkm216

Day, R., Beck, D. A. C., Armen, R. S., & Daggett, V. (2003). A consensus view of fold space: Combining SCOP, CATH, and the Dali Domain Dictionary. *Protein Science*, *12*(10), 2150–2160. doi:10.1110/ps.0306803

Dayhoff, M. O., & Schwartz, R. M. (1978). A model of evolutionary change in proteins. In *in Atlas of Protein Sequence and Structure* (pp. 345–352). Washington, DC.

Dunbrack, R. L., & Cohen, F. E. (1997). Bayesian statistical analysis of protein side-chain rotamer preferences. *Protein Science*, *6*(8), 1661–1681. doi:10.1002/pro.5560060807

Dunbrack, R. L., & Karplus, M. (1994). Conformational analysis of the backbone-dependent rotamer preferences of protein sidechains. *Nature Structural & Molecular Biology*, *1*(5), 334–340. doi:10.1038/nsb0594-334

Dyson, H. J. (2011). Expanding the proteome: disordered and alternatively folded proteins. *Quarterly Reviews of Biophysics*, *44*(04), 467–518. doi:10.1017/S0033583511000060

Eicken, C., Sharma, V., Klabunde, T., Lawrenz, M. B., Hardham, J. M., Norris, S. J., & Sacchettini, J. C. (2002). Crystal Structure of Lyme Disease Variable Surface Antigen VlsE of Borrelia burgdorferi. *Journal of Biological Chemistry*, *277*(24), 21691–21696. doi:10.1074/jbc.M201547200

Fersht, A. (1999). *Structure and Mechanism in Protein Science: A Guide to Enzyme Catalysis and Protein Folding*. Macmillan.

Fetrow, J. S. (1995). Omega loops: nonregular secondary structures significant in protein function and stability. *The FASEB Journal*, *9*(9), 708–717.

Gaile, G. L., & Burt, J. E. (1980). *Directional statistics*. Norwich: Geo Abstracts.

Glazer, D. S., Radmer, R. J., & Altman, R. B. (2009). Improving Structure-Based Function Prediction Using Molecular Dynamics. *Structure*, *17*(7), 919–929. doi:10.1016/j.str.2009.05.010

Goecks, J., Nekrutenko, A., Taylor, J., & The Galaxy Team. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, *11*(8), R86. doi:10.1186/gb-2010-11-8-r86

Gore, S., Velankar, S., & Kleywegt, G. J. (2012). Implementing an X-ray validation pipeline for the Protein Data Bank. *Acta Crystallographica Section D Biological Crystallography*, *68*(4), 478–483. doi:10.1107/S0907444911050359

Gough, J. J., & Gough, K. J. (2001). *Compiling for the .Net Common Language Runtime*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Hagarman, A., Mathieu, D., Toal, S., Measey, T. J., Schwalbe, H., & Schweitzer-Stenner, R. (2011). Amino Acids with Hydrogen-Bonding Side Chains have an Intrinsic Tendency to Sample Various Turn Conformations in Aqueous Solution. *Chemistry – A European Journal*, *17*(24), 6789–6797. doi:10.1002/chem.201100016

Henderson, R., Sali, A., Baker, M. L., Carragher, B., Devkota, B., Downing, K. H., … Lawson, C. L. (2012). Outcome of the First Electron Microscopy Validation Task Force Meeting. *Structure*, *20*(2), 205–214. doi:10.1016/j.str.2011.12.014

Henikoff, S., & Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences, 89*(22), 10915–10919.

Holm, L., & Sander, C. (1996). Mapping the Protein Universe. *Science, 273*(5275), 595–602. doi:10.1126/science.273.5275.595

Holmes, J. B., & Tsai, J. (2004). Some fundamental aspects of building protein structures from fragment libraries. *Protein Science*, *13*(6), 1636–1650. doi:10.1110/ps.03494504

Hopping, G., Kellock, J., Caughey, B., & Daggett, V. (2013). Designed Trpzip-3 β-Hairpin Inhibits Amyloid Formation in Two Different Amyloid Systems. *ACS Medicinal Chemistry Letters*, *4*(9), 824–828. doi:10.1021/ml300478w

Horrocks, I. (2008). Ontologies and the semantic web. *Communications of the ACM*, *51*(12), 58–67.

Jacobson, M. P., Friesner, R. A., Xiang, Z., & Honig, B. (2002). On the Role of the Crystal Environment in Determining Protein Side-chain Conformations. *Journal of Molecular Biology*, *320*(3), 597–608. doi:10.1016/S0022-2836(02)00470-9

Jones, E., Oliphant, T., & Peterson, P. (2001). *SciPy: Open source scientific tools for Python*. Retrieved from http://www.scipy.org/

Kabsch, W., & Sander, C. (1984). On the use of sequence homologies to predict protein structure: identical pentapeptides can have completely different conformations. *Proceedings of the National Academy of Sciences*, *81*(4), 1075–1078.

Kabsch, Wolfgang, & Sander, C. (1983). Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, *22*(12), 2577–2637. doi:10.1002/bip.360221211

Karplus, M., & Kuriyan, J. (2005). Molecular dynamics and protein function. *Proceedings of the National Academy of Sciences of the United States of America*, *102*(19), 6679–6685.

Kryshtafovych, A., Moult, J., Bales, P., Bazan, J. F., Biasini, M., Burgin, A., … Schwede, T. (2013). Challenging the state-of-the-art in protein structure prediction: Highlights of experimental target structures for the 10th Critical Assessment of Techniques for Protein Structure Prediction Experiment CASP10. *Proteins: Structure, Function, and Bioinformatics*, n/a–n/a. doi:10.1002/prot.24489

Larriva, M., & Rey, A. (2014). Design of a Rotamer Library for Coarse-Grained Models in Protein-Folding Simulations. *Journal of Chemical Information and Modeling*, *54*(1), 302–313. doi:10.1021/ci4005833

Lau, P.-W., Guiley, K. Z., De, N., Potter, C. S., Carragher, B., & MacRae, I. J. (2012). The molecular architecture of human Dicer. *Nature Structural & Molecular Biology*, *19*(4), 436–440. doi:10.1038/nsmb.2268

Leszczynski, J. F., & Rose, G. D. (1986). Loops in globular proteins: a novel category of secondary structure. *Science*, *234*(4778), 849–855. doi:10.1126/science.3775366

Levitt, M., Hirshberg, M., Sharon, R., & Daggett, V. (1995). Potential energy function and parameters for simulations of the molecular dynamics of proteins and nucleic acids in solution. *Computer Physics Communications*, *91*(1–3), 215–231. doi:10.1016/0010-4655(95)00049-L

Levitt, M., Hirshberg, M., Sharon, R., Laidig, K. E., & Daggett, V. (1997). Calibration and Testing of a Water Model for Simulation of the Molecular Dynamics of Proteins and Nucleic Acids in Solution. *The Journal of Physical Chemistry B*, *101*(25), 5051–5061. doi:10.1021/jp964020s

Lovell, S. C., Word, J. M., Richardson, J. S., & Richardson, D. C. (2000). The penultimate rotamer library. *Proteins: Structure, Function, and Bioinformatics*, *40*(3), 389–408. doi:10.1002/1097-0134(20000815)40:3<389::AID-PROT50>3.0.CO;2-2

McCully, M. E., Beck, D. A. C., Fersht, A. R., & Daggett, V. (2010). Refolding the Engrailed Homeodomain: Structural Basis for the Accumulation of a Folding Intermediate. *Biophysical Journal*, *99*(5), 1628–1636. doi:10.1016/j.bpj.2010.06.040

Merkley, E. D., Cort, J. R., & Adkins, J. N. (2013). Cross-linking and mass spectrometry methodologies to facilitate structural biology: finding a path through the maze. *Journal of Structural and Functional Genomics*, *14*(3), 77–90. doi:10.1007/s10969-013-9160-z

Merkley, E. D., Rysavy, S. J., Kahraman, A., Hafen, R. P., Daggett, V., & Adkins, J. N. (2013). Distance Restraints from Cross-Linking Mass Spectrometry: Mining a Molecular Dynamics Simulation Database to Evaluate Lysine-Lysine Distances. *Submitted for Publication*.

Microsoft Corporation. (2007a). .NET Framework (Version 3.5).

Microsoft Corporation. (2007b). SQL Server 2008 (Version 2008 Enterprise Edition R2 x64).

Microsoft Corporation. (2007c). Windows (Version 2008 Server R2 Enterprise Edition x64).

Montelione, G. T., Nilges, M., Bax, A., Güntert, P., Herrmann, T., Richardson, J. S., … Markley, J. L. (2013). Recommendations of the wwPDB NMR Validation Task Force. *Structure*, *21*(9), 1563–1570. doi:10.1016/j.str.2013.07.021

*OSGi Service Platform, Release 3*. (2003). IOS Press, Inc.

Otzen, D. E., & Fersht, A. R. (1995). Side-Chain Determinants of .beta.-Sheet Stability. *Biochemistry*, *34*(17), 5718–5724. doi:10.1021/bi00017a003

Outercurve Foundation. (2013). .NET Bio Framework (Version 1.1). Retrieved from http://bio.codeplex.com/

Pearl, F. M. G., Bennett, C. F., Bray, J. E., Harrison, A. P., Martin, N., Shepherd, A., … Orengo, C. A. (2003). The CATH database: an extended protein family resource for structural and functional genomics. *Nucleic Acids Research*, *31*(1), 452–455. doi:10.1093/nar/gkg062

Pettersen, E. F., Goddard, T. D., Huang, C. C., Couch, G. S., Greenblatt, D. M., Meng, E. C., & Ferrin, T. E. (2004). UCSF Chimera--a visualization system for exploratory research and analysis. *Journal of Computational Chemistry*, *25*(13), 1605–1612. doi:10.1002/jcc.20084

Read, R. J., Adams, P. D., Arendall III, W. B., Brunger, A. T., Emsley, P., Joosten, R. P., … Zwart, P. H. (2011). A New Generation of Crystallographic Validation Tools for the Protein Data Bank. *Structure*, *19*(10), 1395–1412. doi:10.1016/j.str.2011.08.006

Rysavy, S. J., Beck, D. A. C., & Daggett, V. (2014). Dynameomics: Data-Driven Methods and Models for Utilizing Large-Scale Protein Structure Repositories for Improving Fragment-Based Loop Prediction. *In Preparation*.

Rysavy, S. J., Bromley, D., & Daggett, V. (2014). DIVE: A Graph-Based Visual Analytics Framework for Big Data. *In Press*.

Rysavy, S. J., Towse, C.-L., & Daggett, V. (2014). Dynameomics: Comparative Data-Driven Analysis of the Correlation Between Rotameric States and Backbone Conformational Propensities. *In Preparation*.

Schaeffer, R. D., Jonsson, A. L., Simms, A. M., & Daggett, V. (2011). Generation of a consensus protein domain dictionary. *Bioinformatics*, *27*(1), 46–54. doi:10.1093/bioinformatics/btq625

Schneidman-Duhovny, D., Rossi, A., Avila-Sakar, A., Kim, S. J., Velázquez-Muriel, J., Strop, P., … Sali, A. (2012). A method for integrative structure determination of protein-protein complexes. *Bioinformatics*, *28*(24), 3282–3289. doi:10.1093/bioinformatics/bts628

Schroeder, W., Martin, K., & Lorensen, B. (1996). *The Visualization Toolkit: An Object-Oriented Approach to 3-D Graphics*. Upper Saddle River, NJ: Prentice Hall.

Schwartz, R. M., & Dayhoff, M. O. (1978). Matrices for detecting distant relationships. In *in Atlas of Protein Sequence and Structure* (pp. 353–358). Washington, DC.

Scouras, A. D., & Daggett, V. (2011). The dynameomics rotamer library: Amino acid side chain conformations and dynamics from comprehensive molecular dynamics simulations in water. *Protein Science*, *20*(2), 341–352. doi:10.1002/pro.565

Shannon, P., Markiel, A., Ozier, O., Baliga, N. S., Wang, J. T., Ramage, D., … Ideker, T. (2003). Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Research*, *13*(11), 2498–2504.

Shapovalov, M. V., & Dunbrack Jr., R. L. (2011). A Smoothed Backbone-Dependent Rotamer Library for Proteins Derived from Adaptive Kernel Density Estimates and Regressions. *Structure*, *19*(6), 844–858. doi:10.1016/j.str.2011.03.019

Shehu, A., & Kavraki, L. E. (2012). Modeling Structures and Motions of Loops in Protein Molecules. *Entropy*, *14*(12), 252–290. doi:10.3390/e14020252

Simms, A. M., & Daggett, V. (2012). Protein simulation data in the relational model. *The Journal of Supercomputing*, *62*(1), 150–173. doi:10.1007/s11227-011-0692-3

Simms, A. M., Toofanny, R. D., Kehl, C., Benson, N. C., & Daggett, V. (2008). Dynameomics: design of a computational lab workflow and scientific data repository for protein simulations. *Protein Engineering Design and Selection*, *21*(6), 369–377. doi:10.1093/protein/gzn012

Sinz, A. (2006). Chemical cross-linking and mass spectrometry to map three-dimensional protein structures and protein–protein interactions. *Mass Spectrometry Reviews*, *25*(4), 663–682. doi:10.1002/mas.20082

Skolnick, J., Zhou, H., & Brylinski, M. (2012). Further Evidence for the Likely Completeness of the Library of Solved Single Domain Protein Structures. *The Journal of Physical Chemistry B*, *116*(23), 6654–6664. doi:10.1021/jp211052j

Smock, R. G., & Gierasch, L. M. (2009). Sending signals dynamically. *Science*, *324*(5924), 198–203. doi:10.1126/science.1169377

Søndergaard, C. R., Garrett, A. E., Carstensen, T., Pollastri, G., & Nielsen, J. E. (2009). Structural Artifacts in Protein−Ligand X-ray Structures: Implications for the Development of Docking Scoring Functions. *Journal of Medicinal Chemistry*, *52*(18), 5673–5684. doi:10.1021/jm8016464

Spronk, C. A. E. M., Nabuurs, S. B., Krieger, E., Vriend, G., & Vuister, G. W. (2004). Validation of protein structures derived by NMR spectroscopy. *Progress in Nuclear Magnetic Resonance Spectroscopy*, *45*(3–4), 315–337. doi:10.1016/j.pnmrs.2004.08.003

Toofanny, R. D., & Daggett, V. (2012). Understanding protein unfolding from molecular simulations. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, *2*(3), 405–423. doi:10.1002/wcms.1088

Toofanny, R. D., Jonsson, A. L., & Daggett, V. (2010). A Comprehensive Multidimensional-Embedded, One-Dimensional Reaction Coordinate for Protein Unfolding/Folding. *Biophysical Journal*, *98*(11), 2671–2681. doi:10.1016/j.bpj.2010.02.048

Toofanny, R. D., Simms, A. M., Beck, D. A., & Daggett, V. (2011). Implementation of 3D spatial indexing and compression in a large-scale molecular dynamics simulation database for rapid atomic contact detection. *BMC Bioinformatics*, *12*(1), 334. doi:10.1186/1471-2105-12-334

Towse, C.-L., & Daggett, V. (2012). When a domain is not a domain, and why it is important to properly filter proteins in databases. *BioEssays*, *34*(12), 1060–1069. doi:10.1002/bies.201200116

Towse, C.-L., Rysavy, S. J., & Daggett, V. (2014). Dynameomics: Comparison of the Coverage of Protein Fold Space by Native and Denatured States. *In Preparation*.

Van der Kamp, M. W., Schaeffer, R. D., Jonsson, A. L., Scouras, A. D., Simms, A. M., Toofanny, R. D., … Daggett, V. (2010). Dynameomics: A Comprehensive Database of Protein Dynamics. *Structure*, *18*(4), 423–435. doi:10.1016/j.str.2010.01.012

Vanhee, P., Verschueren, E., Baeten, L., Stricher, F., Serrano, L., Rousseau, F., & Schymkowitz, J. (2011). BriX: a database of protein building blocks for structural analysis, modeling and design. *Nucleic Acids Research*, *39*(suppl 1), D435–D442. doi:10.1093/nar/gkq972

Verschueren, E., Vanhee, P., van der Sloot, A. M., Serrano, L., Rousseau, F., & Schymkowitz, J. (2011). Protein design with fragment databases. *Current Opinion in Structural Biology*, *21*(4), 452–459. doi:10.1016/j.sbi.2011.05.002

Wade, D., Boman, A., Wåhlin, B., Drain, C. M., Andreu, D., Boman, H. G., & Merrifield, R. B. (1990). All-D amino acid-containing channel-forming antibiotic peptides. *Proceedings of the National Academy of Sciences*, *87*(12), 4761–4765. doi:10.1073/pnas.87.12.4761

Wagner, G., Hyberts, S. G., & Havel, T. F. (1992). NMR Structure Determination in Solution: A Critique and Comparison with X-Ray Crystallography. *Annual Review of Biophysics and Biomolecular Structure*, *21*(1), 167–198. doi:10.1146/annurev.bb.21.060192.001123

Wang, Gouli, & Dunbrack Jr., R. L. (2005). PISCES: recent improvements to a PDB sequence culling server. *Nucleic Acids Research*, *33*(suppl 2), W94–W98. doi:10.1093/nar/gki402

Wang, Guoli, & Dunbrack Jr., R. L. (2003). PISCES: a protein sequence culling server. *Bioinformatics*, *19*(12), 1589–1591. doi:10.1093/bioinformatics/btg224

Wang, H.-W., Noland, C., Siridechadilok, B., Taylor, D. W., Ma, E., Felderer, K., … Nogales, E. (2009). Structural insights into RNA processing by the human RISC-loading complex. *Nature Structural & Molecular Biology*, *16*(11), 1148–1153. doi:10.1038/nsmb.1673

Ward, A. B., Sali, A., & Wilson, I. A. (2013). Integrative Structural Biology. *Science*, *339*(6122), 913–915. doi:10.1126/science.1228565

Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., … Goble, C. (2013). The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research*, *41*(W1), W557–W561. doi:10.1093/nar/gkt328

Wu, S.-J., & Dean, D. H. (1996). Functional Significance of Loops in The Receptor Binding Domain ofBacillus thuringiensisCryIIIA δ-Endotoxin. *Journal of Molecular Biology*, *255*(4), 628–640. doi:10.1006/jmbi.1996.0052

Xiang, Z., & Honig, B. (2001). Extending the accuracy limits of prediction for side-chain conformations. *Journal of Molecular Biology*, *311*(2), 421–430. doi:10.1006/jmbi.2001.4865

# Appendix A

# MOLECULAR DYNAMICS

Researchers commonly use molecular dynamics (MD) simulations (Toofanny & Daggett, 2012) to study protein structure and dynamics. Proteins are complex molecules consisting of amino acids (residues). Contacts between the constituent atoms exist when they're within a defined distance from one another.

Proteins are responsible for much of the functional and structural activity in living tissue. In the human body, protein function is involved in such areas as muscular structure, metabolism, immune response, and reproduction. So, understanding how proteins work is critical to advancing the science of human health. An interesting facet of protein biology is that structure equals function; what a protein does and how it does it is intrinsically tied to its 3D structure (see Figure A.1).

During an MD simulation, scientists simulate interatomic forces to predict motion among the atoms of a protein and its environment (see Figure A.1). In most cases, the environment is water molecules, although scientists can alter this to investigate different phenomena. The physical simulation is calculated using Newtonian physics; at specified time intervals, the simulation state is saved. This produces a trajectory, a series of structural snapshots reflecting the protein's natural behavior in an aqueous environment.

MD is useful for three primary reasons. First, like many *in silico* techniques, it allows virtual experimentation; scientists can simulate protein structures and interactions without the cost or risk of laboratory experiments. Second, modern computing techniques allow MD simulations to run in parallel, enabling virtual high-throughput experimentation. Third, MD simulation is the only protein analysis method that produces sequential time-series structures at both high spatial and high temporal resolution. These high-resolution trajectories can reveal how proteins move, a critical aspect of their functionality.

However, MD simulations can produce datasets considerably larger than what most structural-biology tools can handle. As computers become more powerful, MD simulations' size and resolution are increasing. So, the logistical challenges of storing, analyzing, and visualizing MD data require researchers to consider new analysis techniques.

At the University of Washington's Daggett laboratory, we're studying protein dynamics as part of the Dynameomics project (van der Kamp et al., 2010). This project aims to characterize the dynamic behaviors and folding pathways of topological classes of all known protein structures. So far, the project has generated hundreds of terabytes of data consisting of thousands of simulations and millions of structures, as well as their associated analyses. We store these data in a distributed SQL data warehouse. This warehouse currently holds $10^4$ times as many protein structures as the Protein Data Bank (Frances C. Bernstein et al., 1977), the primary repository for experimentally characterized protein structures. Dynameomics is currently the largest database of protein structures in the world.

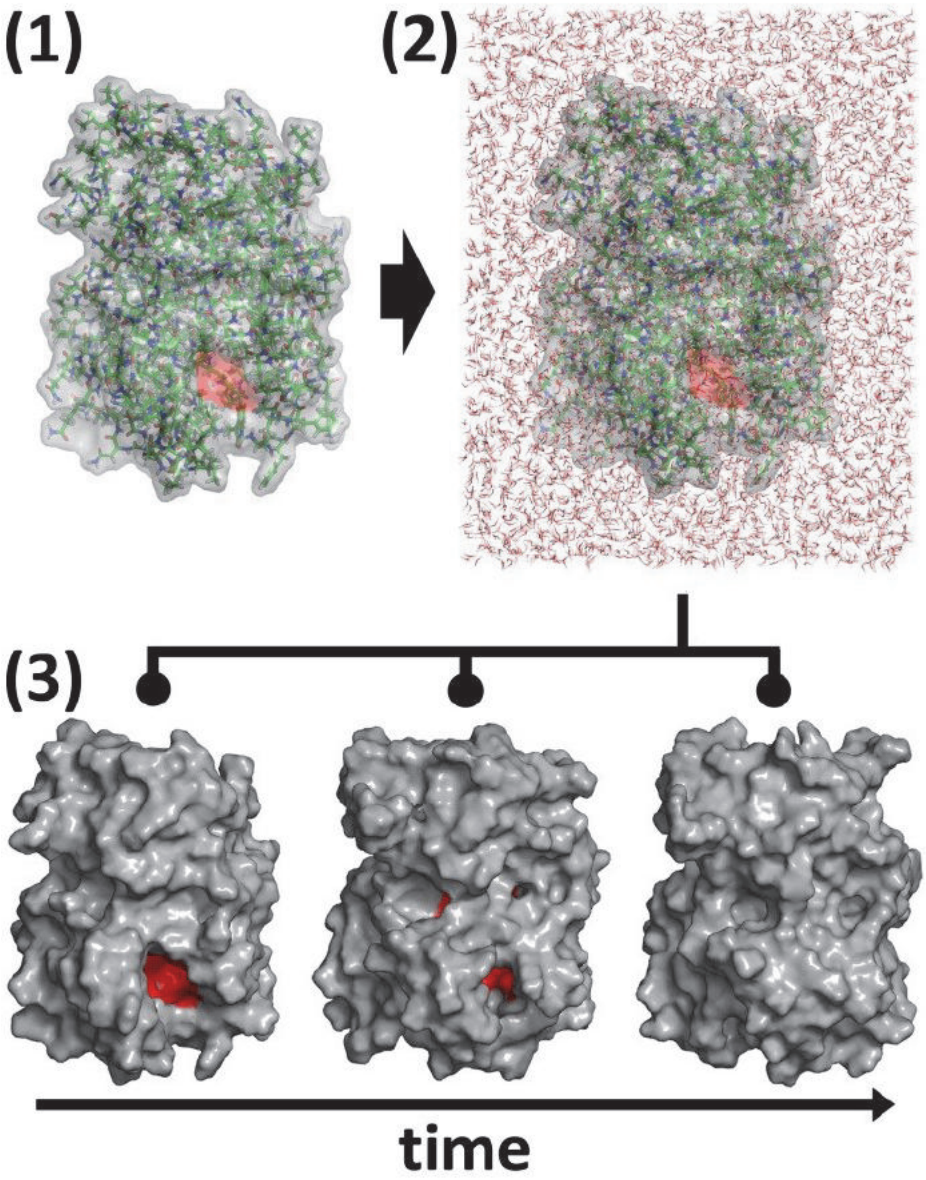


**Figure A.1 Solvating and simulating a protein using molecular dynamics.**

(1) An all-atom depiction of a protein with a transparent surface. (2) The same protein solvated and in a water box. (3) Three structures of interest selected from a trajectory containing more than 51,000 frames. The red area shows the protein's functional site and how it closes over time.

Appendix B

# PROVISIONAL PATENTS COVERING METHODS FOR EFFICIENT STREAMING OF STRUCTURED INFORMATION

## *B.1 Efficient Data Streaming into a Structured Ontology*

### B.1.1 *Background*

*Big data* is becoming increasingly present in many aspects of society and technology including health care, science, industry and government. Many of these data are highly complex and multi-dimensional and are best understood – and are made more valuable – when they are analyzed in a structured manner.

*Ontologies* are a way of representing data in a structured way. Although not new *per se*, the use of ontologies is growing in the presence of modern computer technologies; the semantic web is a very compelling, yet nascent and underdeveloped, example of this.

We believe that these two paradigms of data interaction will only become more important. Furthermore, we believe that these paradigms work well together and that using them in combination represents a powerful and potentially widespread usage scenario. An example of this is the field of *visual analytics*, the emerging field that uses interactive visual techniques to mine big data. Below, we describe an algorithm and data flow architecture for efficiently accessing, analyzing, visualizing and interacting with big data in an ontological structure.

### B.1.2 *Problem Invention Addresses*

Querying data from a big data source and representing them in a complex ontology can be slow. Interactively traversing different subsets of the big data is therefore slow because building each subset is slow. Unfortunately, traversing big data in an efficient manner is key to current and future big data interaction paradigms such as visual analytics.

### B.1.3 *Solution Invention Provides*

A pre-loader predicts user needs and preemptively caches frames (subsets) of data from the big data source. A data broker maps from a data frame to a set of pins; when the user (or code logic) switches from one data frame to another, all pins simultaneously re-map to the new frame. The ontology, in turn, maps node and edge properties to the pins. When the pins switch data frames, all ontology data is updated simultaneously.

### B.1.4 *Architecture Broken Down by Step*

Please see Figure B.1. Step numbers refer to circled numbers in the figure.

1. Numerous kinds of big data resources, both dynamic and static.
2. Preemptive loading and caching of frames (subsets) of the big data. Caching can take place locally or remotely. Caching could also be multi-tiered e.g. remote caching on a cloud database feeds local caching in computer RAM. Preemptive loading can reduce to on-demand loading of a specified frame if necessary.
3. Frames of cached data are stored where they can be quickly accessed e.g. computer RAM.

4. Data frame selection logic e.g. user input, programmatic logic, etc. This step switches the data pins to pull from a selected frame e.g. by changing an array pointer. This process is runs in constant time.
5. Data selection pins pull from the currently selected frame. Pins all switch together.
6. The data ontology consisting of arbitrary node types and arbitrary edges. Node and edge data and logic are fed by the pins. When the pins are switched, the data at every location in the ontology are therefore simultaneously updated. This step is the key to the invention.
7. Ontological data can be arbitrarily transformed before user interaction. Note that data transformation can take place anywhere in the pipeline.
8. Because of the pin-linked ontology, fed by a fast-switched dataset, in turn fed by preemptive data caching, complex and multi-dimensional data can be interacted with, analyzed, and visualized quickly.

## *B.2 Automated Parsing of Object-Oriented Assemblies into Dynamically Linked Ontologies*

### B.2.1 *Background*

*Ontologies* are a way of representing data in a structured way. This additional structure provides a powerful framework to programmatically access and traverse the data in interesting and semantically meaningful ways. Ontologies also enable formal analysis, which helps with semantic correctness, interoperability, and can bring much needed insight.

Ontologies are most useful when applied to complex, multi-dimensional, and/or large datasets. The development of data-specific, formal ontologies can be very difficult and may require collaborations between domain specialists and informaticians. However, the work of codifying these complex relationships is already implicitly done during the process of engineering software. Therefore it is possible to recover a formal ontology from the software itself.

As computational problems become bigger, more complex, and more data-intensive, ontologies will become necessary to gain insight from the data. Below we describe an invention for leveraging the pre-existing investment in software engineering to create formal ontologies for deep data analysis. This algorithm parses object-oriented assemblies into an automatically generated ontology. These ontologies can be output as a static definition or dynamically linked instances of the ontological structure.

### B.2.2 *Problem Invention Addresses*

Modern computational problems increasingly require formal ontological analysis. However, for most software formal ontologies do not exist. The generation of formal ontologies is time consuming, difficult, and may require several experts. Ontologies are often implicitly defined in code by software engineers, but there is currently no way to obtain a formal ontology from these definitions. Furthermore, once an ontology is created, the legacy code must be modified to interface with the ontology.

### B.2.3 *Solution Invention Provides*

An object-parser automatically traverses the object-oriented data structures within a provided assembly using code reflection. Using generalized rules to leverage the existing ontological structure, a formal ontology is generated from the existing relationships of the data structures within the code. The ontology is provided in either a static definition or by dynamically linking the ontological structure to the object instance. The dynamically linked ontology allows the underlying object instances to be modified through the context of the ontology without changes to the assembly itself. Alternatively, metadata tags may be added to the assembly to provide a richer ontology definition.

### B.2.4 *Architecture Broken Down by Step*

Please see Figure B.2. Step numbers refer to circled numbers in the figure.

1. An object or set of objects of interest is provided to the object parser via a code assembly.
2. Parameters are used to specify which semantic components will be parsed into an ontology, such as private objects, protected objects, static objects and interfaces.
3. The complete object hierarchy is recursively traversed using code reflection and expression trees. Using generalized, pre-defined rules, fields, properties, methods, classes, interfaces, etc. are parsed into ontological components. A simplified example is shown in Figure. This step is key to the invention.
4. The static definition is produced using a standardized ontology language.
5. An ontological structure is provided with dynamic links to the object(s) instance(s). This linking is done using delegate methods and lambda functions. This step is key to the invention.

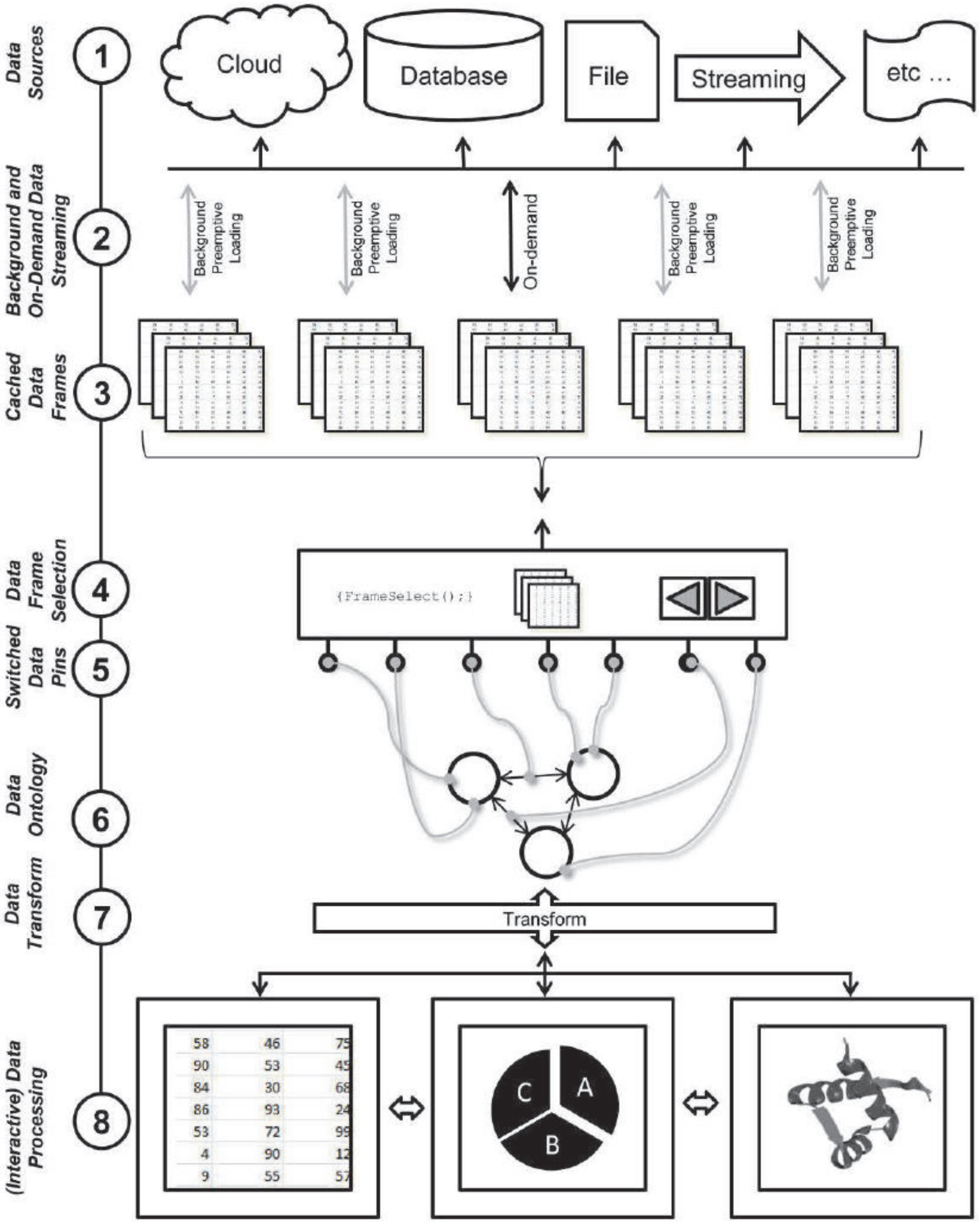


**Figure B.1 Steps for streaming data into a structured ontology.**

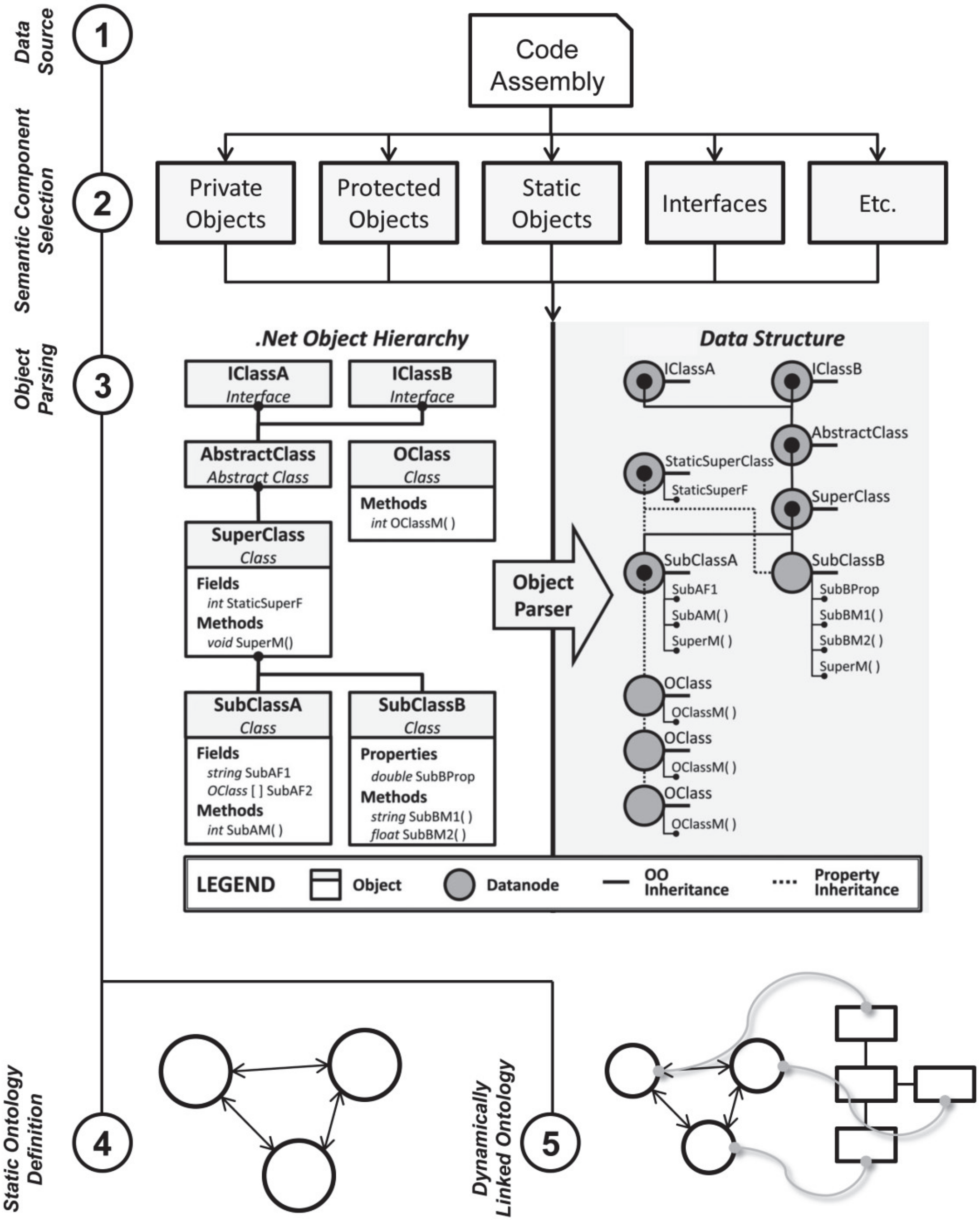


**Figure B.2 Steps for parsing assemblies into dynamically linked ontologies.**

# VITA

Steven Joseph Rysavy received his Bachelor’s of Science Degree in Computer Engineering from Iowa State University in 2002. He also earned a Master’s of Science Degree in Computer Science from San Francisco State University in 2008. In 2014 Steven earned his Doctor of Philosophy from the University of Washington’s in Biomedical and Health Informatics.