Visual Analytics Methods for Analyzing Molecular Dynamics Simulations of Mutant Proteins

Dennis N. Bromley

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

Peter Myler

Program Authorized to Offer Degree:
Biomedical Informatics and Medical Education

University of Washington

Abstract

Visual Analytics Methods for Analyzing Molecular Dynamics Simulations of Mutant Proteins

Dennis N. Bromley

Chair of the Supervisory Committee:
Professor Valerie Daggett
Bioengineering

The structural dynamics of proteins are integral to protein function; if these structural dynamics are altered by mutation, the function of the protein can be altered as well, potentially resulting in disease. Experimental structure-determination with x-ray crystallography and Nuclear Magnetic Resonance (NMR) can be useful in determining mutant protein structures, but detailed, high-resolution dynamics data can be difficult to ascertain. Molecular Dynamics (MD) simulation is a high temporal- and spatial-resolution *in silico* method for dynamic protein structure determination. Unfortunately, the data generated by MD simulations can be too large for standard analysis tools. Here I describe a novel visual-analytics tool called DIVE that was specifically created to handle large, structured datasets like those generated by MD simulations. Using DIVE, I analyzed MD simulation-data of disease-associated mutations to the α-Tocopherol Transfer Protein (α-TTP) and to the p53 tumor suppressor protein. In addition to

mutant structural-analysis and characterization, I also used DIVE to develop an algorithm for identifying regions of mutant proteins that are amenable to 'rescue', or ligand-mediated stabilization that can suppress the destabilizing effect of mutations. The results of these investigations highlight the utility of big-data, visual-analytics approaches to exploring MD simulation data.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would first like to thank Dr. Valerie Daggett for her guidance, mentorship and instruction throughout this process as well as her service on my thesis committee. I would also like to thank Dr. James Brinkley, Dr. Peter Myler and Dr. David Beck both for their service on my thesis committee and for their help, guidance and support. I am also grateful for the support provided by the Division of Biomedical and Health Informatics, the Department of Bioengineering, the National Library of Medicine, and the National Institutes of Health. As science has always been a collaborative effort, thanks also go to the members and alumni of the Daggett lab including Dr. Steve Rysavy, Dr. Noah Benson, Dr. Andrew Simms, Dr. Clare Towse, Dr. Gene Hopping, Dr. Michelle McCully, Dr. Tom Schmidlin, Dr. Amanda Jonsson, Dr. Dustin Schaeffer, Dr. Alex Scouras, Dr. Erik Merkley, Dr. Rudesh Toofanny, Dr. Sara Nowakowski, Jonathan Cheng, Matthew Childers, Peter Law and Robert Su. I am also grateful for the support from my BHI cohort, Dr. Wynona Black, Dr. Daniel Capurro, Dr. Walter Curioso, Dr. Rupa Patel and Melissa Clarkson.

DEDICATION

I would like to thank Allison FitzGerald for her constant love, support, humor, intelligence and patience through the myriad ups and downs of this process. I am also grateful to Dr. Elizabeth Whalley, whose support, patience and friendship throughout this process were both generous and indispensable. I would also like to thank my family, Dennis Bromley, Cathy Bromley, Darcy Bromley Harris and Chet Harris for their constant love and support. Ultimately, however, my work is dedicated to my son, Dennis London Bromley, whose love, smile and laughter turn any day into the best day of my life.

Chapter 1

VISUAL ANALYTICS METHODS FOR ANALYZING MOLECULAR-DYNAMICS SIMULATIONS OF MUTANT PROTEINS

1.1 Introduction

Proteins are biological macromolecules that perform many of the basic functions of life including reproduction, immunological response, and tissue construction (Alberts et al. 2008). Because proteins are DNA-transcription products, they can be affected by genetic mutations. These effects can cause structural changes relative to the wild-type protein; because protein structure and function are tightly coupled, these structural changes can result in a change in protein function. When the protein function is crucial to life, such as the tumor-suppressing functionality of the p53 transcription-factor protein, the results can be dramatic; it is estimated that approximately 50% of cancers are associated with mutations to the p53 protein (Olivier et al. 2002; Joerger et al. 2006) and the p53 Y220C mutation alone is associated with roughly 75,000 new cancer diagnoses every year (Olivier et al. 2002; Boeckler et al. 2008).

The structure of both wild type and mutant proteins can often be determined experimentally by x-ray crystallography or Nuclear Magnetic Resonance (NMR) (Alberts et al. 2008). Once determined, these structural data are often uploaded to a central repository such as the Protein Data Bank (Bernstein et al. 1977). Not all structures have had their structures experimentally-determined, however. In these circumstances, *in silico* techniques such a molecular mechanics and quantum mechanics can be used to model and predict the structures of

proteins (Wilson and Gisvold 2004). The Dynameomics project, discussed in the later chapters, uses molecular mechanics techniques to model the dynamic behaviors of proteins in solvent (Beck et al. 2008; Van der Kamp et al. 2010). These molecular dynamics (MD) simulations are typically bootstrapped with experimental protein structures; the *in silico* models are then subjected to physiologically-relevant conditions and the resulting data are analyzed to contribute to our understanding of protein structure and behavior. When simulations are performed of both wild type and mutant proteins, the simulations can be compared to gain insights into mutation-associated changes. When the mutations are associated with disease, these insights can pave the way for therapeutic approaches such as structure-based drug design (Wilson and Gisvold 2004).

High-resolution time-series data are a strength of MD. However, MD data can be very large, in keeping with an industry-wide growth in biomedical 'big data' (Martin-Sanchez and Verspoor 2014). For example, in Chapter 6 I describe the results of MD simulations of 21 p53 variants, each performed in triplicate for at least 100 ns. For per-picosecond, per-residue data such as solvent-accessible surface-area (SASA), this results in more than 1.2 billion data points. The three-dimensional atomic coordinates of these simulations, ignoring the solvent atoms, constitute 56.7 billion data points. Even in the presence of powerful data warehouses and computing resources, from a human-analysis perspective the absolute quantity of data can be daunting, particularly if the mutation-associated changes result in only subtle changes (Z. Wang and Moult 2001). As a result, data-analysis tools used to analyze MD simulations must be both powerful enough to handle the data loads and sensitive enough to identify small, yet important, changes.

I propose that high-performance interactive visual-analysis of MD data can facilitate mutant/wild-type comparative analysis and lead to more efficient protein structural-insights and more efficient drug-discovery research. In the following chapters and appendices I discuss 1) my work in developing interactive data-analysis tools capable of handling hundreds of gigabytes of data, 2) the application of those tools to disease-associated protein systems, 3) discoveries and conclusions regarding those protein systems, 4) *in silico* drug-discovery efforts informed by my interactive tools and 5) preliminary experimental-testing of putative pharmaceutical-chaperone molecules selected by my drug-discovery tools.

1.2 Visual Analytics

There are tools such as Pymol (DeLano 2002) and VMD (Humphrey et al. 1996) that are available for viewing protein structures. However, as MD simulation becomes more popular, the size of the data sets continue to grow, often beyond the capabilities of existing tools. The coordinates of a single 100 ns MD simulation, for example, can result in 15GB of data, more data than can fit into the memory of many computers. When these data are augmented by additional analysis-data and then scaled across multiple simulations, we see that fully-leveraging the high-resolution data generated by MD simulations requires commensurate high-resolution analysis tools.

While non-interactive tools such as command-line programs and SQL databases are capable of managing many of these tasks, data exploration is often facilitated by interactive visual analysis. *Visual analytics* (Thomas and Cook 2005; Keim et al. 2008) is a data-exploration paradigm that leverages human-guided data-analysis and powerful computer-processing. To

apply this approach to MD simulations and disease-research, I co-developed a visual analytics program called DIVE, a Data Intensive Visualization Engine (Bromley et al. 2014; Rysavy et al. 2014). As discussed below, DIVE is capable of streaming data from the Dynameomics data warehouse (Simms et al. 2008) and is therefore largely freed from the memory-limitations of the local computer. Moreover, DIVE facilitates integration of multiple analyses, supports both GUI and scripted interaction, and can be used as an application programming interface (API) for independent tool-development.

In Chapter 2, I discuss the technical features of DIVE as well as their implementations. These include interactive-speed data-streaming, a flexible scripting-paradigm called μScripting (microscripting), an ontologically-expressive data-representation, an extensible data-pipeline capable of visualizing and analyzing large data-streams, and a binary-parsing mechanism capable of importing unrelated software libraries as DIVE plugins. A case study involving the protein dashboard (also discussed in Chapter 3) illustrates how these pieces fit together. Appendix A contains supplementary materials for Chapter 2. The contents of Chapter 2 and the appendix were previously published in the journal *IEEE Computer Graphics and Applications* (Rysavy et al. 2014). In addition to the DIVE publication, a US patent application was filed in June of 2014 to protect several of the technologies invented during the development of DIVE. The contents of this patent application are contained in Appendix B. Furthermore, the DIVE software, documentation, examples, and developer guides are all freely distributed on the internet at www.dynameomics.org/dive.

In Chapter 3 I discuss various DIVE case studies. Most of the examples involve the Dynameomics data set; case studies include the Protein Dashboard, analysis of the protein Cu-Zn superoxide dismutase 1 (SOD-1, mutants of which are associated with amyotrophic lateral sclerosis (ALS, Lou Gehrig's Disease)) and analysis of the protein p53 (a tumor-suppressing transcription-factor whose mutations are associated with cancer). Another example demonstrates the interactive exploration of taxonomy data from the Gene Ontology (Ashburner et al. 2000) and another example includes DIVE data-integration with the Chimera protein-visualization tool. A final case-study demonstrating the interactive analysis of almost 200 years of professional baseball statistics illustrates the data-agnostic capabilities of the DIVE data pipeline. Appendix C contains supplementary materials for Chapter 3. The contents of Chapter 3 and the appendix were previously published in the journal *Bioinformatics* (Bromley et al. 2014).

1.3 Contact Analysis and the α-Tocopherol Transfer Protein

In Chapter 4 I discuss the results of analyzing MD simulations of the wild type and two mutants of the α-Tocopherol Transfer Protein (α-TTP). A notable feature of α-TTP is a hinged 'lid' that covers an internal hydrophobic cavity. This cavity is used to transport hydrophobic vitamin E through hydrophilic environments, helping maintain healthy vitamin E levels throughout the body. Mutations to α-TTP are associated with the neurodegenerative disease ataxia with vitamin E deficiency (AVED) (Meier et al. 2003).

To quantify and visualize the mutation-associated structural changes to α-TTP, I created a DIVE-based tool called Contact Walker (Bromley et al. 2013). Contact Walker calculates the mutation-associated changes to inter-residue contacts and generates visualizations that allow us

to compare and analyze gigabytes of data without losing structural detail. Using Contact Walker, we analyzed α-TTP and offered a structural description of the mutation-associated changes to both the internal hydrophobic cavity and the covering lid. We then hypothesized how these changes could impact the protein function. The contents of Chapter 4 were previously published in the journal *Biochemistry* (Bromley et al. 2013).

1.4 The Tumor-Suppressor Protein p53

The p53 protein is a transcription-factor that stimulates cell-cycle arrest and apoptosis in the presence of oncogenic factors (Vogelstein et al. 2000). Mutations to the DNA-binding domain of p53 can result in structural-destabilization and unfolding at physiological temperatures, resulting in a loss of tumor-suppressing function; it is estimated that half of human cancer cases are associated with mutations to the p53 protein (Joerger et al. 2006). It has been shown that small molecules can 'rescue' p53 by binding into pockets on the protein, stabilizing the protein and allowing it to resume tumor suppression (Boeckler et al. 2008). Small-molecule rescue has been shown to be effective in non-p53 protein systems as well (Leidenheimer and Ryder 2014). Despite being such an important and well-studied cancer target, only a few experimental structures of p53 DNA binding-domain mutants have been submitted to the Protein Data Bank. Without protein structural information, rational design of stabilizing rescue-ligands is hampered. Moreover, even with protein structural information, it is not clear where in the protein structure we would look to find an effective stabilization-pocket.

In Chapter 5, I discuss an *in silico* algorithm that I developed to address this problem. Prompted by the protein structure → protein function relationship, and using inter-residue

contacts as proxies for protein structure, I hypothesized that, in the presence of mutation-associated contact changes, the ligand-mediated re-assertion of wild type-like contacts could re-assert wild type-like function. As a corollary to this, I also hypothesized that protein regions amenable to binding a stabilizing ligand could be identified as those regions with a large degree of mutant-associated contact destabilization. There is support for this general approach (Leidenheimer and Ryder 2014) and initial analysis of published p53-stabilizing ligands (Wilcken et al. 2012) showed good overlap with contact-disruption analysis. As the Y220C mutation is well-studied (Wilcken et al. 2012; Boeckler et al. 2008), I selected it as a positive control. By analyzing MD simulations of the p53 wild type and Y220C mutant, I selected an MD protein conformation containing a pocket that I hypothesized could bind a stabilizing ligand. I then used *in silico* drug-docking techniques to identify potential stabilizing ligands. *In silico* comparison of the final pocket and ligand structures showed good agreement with the published positive-control data. The putative stabilizing ligands were then submitted for experimental testing (experimental testing courtesy of Dr. Matthias Bauer and Dr. Alan Fersht). As discussed below, initial experimental results were weak but positive; preliminary data from $^{1}H/^{15}N$ HSQC NMR experiments and thermal shift assays indicated that one of the three *in silico*-identified ligands showed weak interaction with the intended pocket region and small but positive dose-dependent increases in melting temperature. I conclude that the results are encouraging but not yet conclusive, and that the algorithm may improve with future refinement.

In Chapter 6, I discuss an enlarged p53 MD simulation effort undertaken to better characterize the effects of various p53 mutations. We simulated 19 different tumorigenic p53

mutants, a p53 mutant with known stabilizing mutations, and the p53 wild type. Analysis of these simulations suggested several things. First, simulation data suggested that p53 may be amenable to α-sheet secondary-structure formation. While still a hypothesis, it has been suggested that α-sheet may play a role in protein aggregation (Armen, Alonso, et al. 2004; Daggett 2006). Together with a recent re-characterization of cancer as an amyloid disease (Xu et al. 2011), these MD data offer an atomic-level hypothesis for the structure of the aggregating species. Second, the destabilizing effect of p53 mutants may arise from exacerbating native structural tendencies already inherent in the wild type. This is similar to the findings from the α-TTP work discussed earlier (Bromley et al. 2013). Third and lastly, I conclude that addressing structural motifs common to multiple p53 mutants may be an efficient drug-design approach.

1.5 Conclusions and Future Work

The data presented in the following chapters offer a perspective on drug discovery that is difficult to obtain through experimental means. Using MD simulations and DIVE analysis tools, I was able to analyze protein structural data in great depth and great breadth simultaneously. As a result, I was able to quantify, analyze, and compare to wild-type two different protein systems including more than six million high-resolution, physically-realistic structures of 20 different p53 mutants. Analysis of these data suggest that it may be more efficient for mutant-rescue drug-design efforts to focus on a relatively small set of common, multi-mutant structural-disruption motifs rather than focusing on a multitude of individual mutants. Moreover, given that more than 25 diseases are currently linked to protein misfolding (Gavrin et al. 2012), these findings are likely relevant to more diseases than just p53-associated cancers.

The work presented here can be extended and continued in multiple different ways. From a visual analytics perspective, DIVE would benefit from improved cloud-distribution functionality. For example, a common goal of distributed computing is to perform calculations as close to the data as possible, avoiding the multiple costs associated with transporting big data sets over a network. To do this, analysis modules and their associated computational resources must be present at or near the data storage location and also be capable of communicating the analysis results back over the network to the user. This scenario becomes more complex when a single analysis involves data present in multiple locations. SQL queries that centrally aggregate machine-specific sub-queries are an example of this analysis model.

We have proofs-of-concept that DIVE analyses can be distributed over a network; I developed a DIVE plugin that converts data into BSON (binary JSON) objects and sends them over a network socket rather than a DIVE output pin. These BSON data can then be intercepted and interpreted by receiving software. Two DIVE pipelines were developed to investigate DIVE distributed analysis. The first involved Dynameomics MD structural data streaming over a network from DIVE to the popular protein analysis program Chimera (Pettersen et al. 2004); a custom python library loaded into Chimera received the DIVE data over a network socket, translated the BSON data into Chimera-appropriate data, and prepared them for visualization. The second DIVE pipeline involved an 'upstream' DIVE pipeline sending generic DIVE data over a network socket connection to a 'downstream' DIVE pipeline. The data were sent, as before, from the BSON DIVE plugin in one DIVE pipeline and received on the other end by another instance of the BSON DIVE plugin in a separate DIVE pipeline. The elegance of this

approach is that the BSON plugins were the only parts of the DIVE pipeline that were aware of network activity; the other DIVE plugins received only generic structured DIVE data arriving at their input pins and leaving via their output pins – they were neither aware nor impacted by the integration of network data-streaming.

This latter point is what will allow future versions of DIVE to scale into the cloud. Multiple DIVE instances could be connected in a large virtual DIVE pipeline with network data-sources and data-sinks replacing the local data-source and data-sink pins. The resulting DIVE pipeline could have an analysis node in each database of a SQL warehouse and an aggregation node on the user's local computer, for example. The result would be faster, more responsive analyses distributed across available computer resources without any material change in user experience or plugin/analysis-module usage.

From a structural-biology standpoint, the work begun in Chapter 5 should be continued to more conclusively test the *disrupted-region-as-rescue-region* hypothesis. Although initial results are encouraging, true validity will only be demonstrated by results that meet or exceed the capabilities of current approaches. To begin, testing larger numbers of *in silico*-identified ligands would be appropriate; other published studies (Wassman et al. 2013; Boeckler et al. 2008) that followed similar docking protocols tested between one and two orders of magnitude more compounds with favorable results. Next, more extensive validation of the rescue-region hypothesis may be facilitated by using additional protein systems as positive controls; it may be that p53 is an uncharacteristically problematic system to analyze computationally and a consensus analysis spanning multiple systems would more effectively assess the general validity

of this approach. There are many alternative positive-control protein systems available for this analysis; transthyretin (TTR), Cu-Zn superoxide dismutase 1 (SOD-1), lysozyme, prion protein, and α-1-antitrypsin are all disease-associated misfolding protein systems appropriate to this task (Gavrin et al. 2012). TTR, SOD-1, and prion protein have already been simulated as part of the Dynameomics project (Armen, Alonso, et al. 2004; Schmidlin et al. 2009; Chen et al. 2014). Notably, TTR and SOD-1 both belong to the same β-sandwich protein fold family as p53.

An increase in protein conformational data would also be relevant to the work presented in Chapter 5 and Chapter 6. This could be achieved by longer and/or additional MD simulations. While additional conformational data are not always necessary – a 2 ns simulation played a pivotal role in developing the first FDA-approved HIV-1 integrase inhibitor (Schames et al. 2004) – the 10^4 reduction in conformational space performed by the pocket search tool reduced the pocket set under consideration to only 98 pockets, a small number in absolute terms. Increasing the overall pool of pockets would likely give us more pockets and conformations from which to choose, increasing our ability to select multi-mutant pockets, or pockets with more favorable structural characteristics such as native-like conformations or favorable conformational entropies.

Once these steps have been taken, the application of the pocket-finding algorithm from Chapter 5 to the regional-disruption findings of Chapter 6 is the logical end-goal of this work, potentially identifying a stabilization-capable multi-mutant pocket for p53. Small-molecule protein rescue of either mutant or misfolded-wild type represents an enormous opportunity for p53-associated cancer research; initial *in silico* and experimental results published in the

academic literature (Leidenheimer and Ryder 2014; Wilcken et al. 2012; Boeckler et al. 2008; Conn et al. 2014; Gavrin et al. 2012) and in the commercial sector (Conn 2005; Boeckler et al. 2011) indicate that such an approach is feasible.

Experimental testing, however, can be both expensive and difficult. And while, as discussed above, experiment remains the necessary gold standard for research, a great deal of progress can be made using *in silico* tools like those described here. The increasing power, sophistication and accessibility of *in silico* techniques and tools effectively democratizes pharmaceutical research, opening up opportunities for orphan-disease researchers, researchers studying restricted pathogens, researchers studying unstable systems, students for whom experimental work is unfeasible or expensive, and even citizen scientists (Khatib et al. 2011). Even in the presence of abundant resources, many biomedical disciplines are undergoing a paradigm shift from *hypothesis-driven* to *data-driven* investigation (Martin-Sanchez and Verspoor 2014). Taken together, it is apparent that big-data analyses, approaches and tools such as those discussed in this dissertation are having a profound and material effect on biomedical research.

Chapter 2

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

The contents of this chapter and Appendix A were previously published in the journal *IEEE Computer Graphics & Applications* (Rysavy et al. 2014). Steven Rysavy and I were co-first authors of this paper. As a result, this previously-published content appears in his PhD dissertation as well. In addition to sharing the overall design of the DIVE framework, my primary contributions were in the core DIVE kernel, encompassing the ontological data-structure, μscripting, the DIVE pipeline and associated pipeline-plugin technologies, the DIVE GUI, the majority of the visualization plugins, interactive SQL, data-streaming protocols, and the development of internal analytical software-libraries such as mathematical libraries and signal processing libraries.

The University of Washington applied for a patent for several of the technologies invented during the development of DIVE. The contents of the DIVE patent-application are contained in Appendix B.

2.1 Abstract

As the need for data-centric scientific tools grows, scientists are increasingly adopting computational approaches. DIVE (Data Intensive Visualization Engine) was developed to help scientists deal with big data. DIVE is a data-agnostic, ontologically expressive visual analytics software framework that can stream and analyze large datasets at interactive speeds.

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 et al. 2001); specialized visualization tools such as Chimera (Pettersen et al. 2004); and scientific workflow tools such as Taverna (Wolstencroft et al. 2013), Galaxy (Goecks et al. 2010), and the Visualization Toolkit (VTK) (Schroeder et al. 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.

2.2 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).

2.2.1 *Data Representation*

Ontologies (see Appendix A) are gaining popularity as a powerful way to organize data. 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 data-edges, 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 et al. 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 et al. 2014) (Chapter 3).) 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.

2.2.2 *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 (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.

2.2.3 *The 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 naïve 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.

2.2.4 *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, because .NET supports 64-bit computations and simple parallelism, DIVE can 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 the "Molecular Dynamics" sidebar), 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.

2.3 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.

2.4 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.

2.5 Data Streaming

DIVE supports the following two SQL data-streaming approaches.

2.5.1 *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. Interactive SQL employs user

input to build an SQL query at runtime. DIVE facilitates this; for example, DIVE events can be thrown in response to mouse clicks or slider bar movements. Upon receiving these events, a DIVE component can construct the appropriate SQL query (which can consist of both data queries and analysis-function execution), send it to the SQL database, and parse the resulting dataset. Depending on the query's size and complexity, this approach can result in user-controlled SQL analysis through the GUI at interactive rates.

2.5.2 *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.

2.6 A Case Study

A major research focus in the University of Washington's Daggett laboratory 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 re-implementing 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, 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 can be accessed through the data's inheritance hierarchy) change over time.

2.7 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.

2.7.1 *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 ontologies 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.

2.7.2 *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.

2.7.3 *Streaming Structured Data*

The most obvious benefit of DIVE is big-data accessibility through data streaming. Interactive SQL's flexibility effectively provides a visualization frontend 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.

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 (Data Intensive Visualization Engine) 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 an** *Atom*

Table 2.2 µScripting example

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`	`{`
		`  int hour = DateTime.Now.Hour;`
		`  return hour < 12;`
		`}`
			Any .NET code is allowed, including complex, multi-statement functions.

`datanode[]`	`Dynamic set`	`from dn in dns`
		`group dn by Math.Round(dn.X, 2) into g`
		`select new`
		`{`
		`  bin = g.Key, population = g.Count()`
		`};`
			This creates a histogram based on the datanode objects' "X" property.

		`from dn in dns`
		`where dn.X > Math.PI`
		`&& dn.is_Superclass`
		`&& dn.Func() == true`
		`select dn;`
			This filters a subset of datanodes on the basis of properties, methods, and inherited type.

		`from dn1 in dnSet1`
		`join dn2 in dnSet2 on dn1.X equals dn2.X`
		`select new {X = dn1.X, Y = dn2.Y}`
			DIVE can act as an object-oriented database by joining multiple potentially disparate datasets.

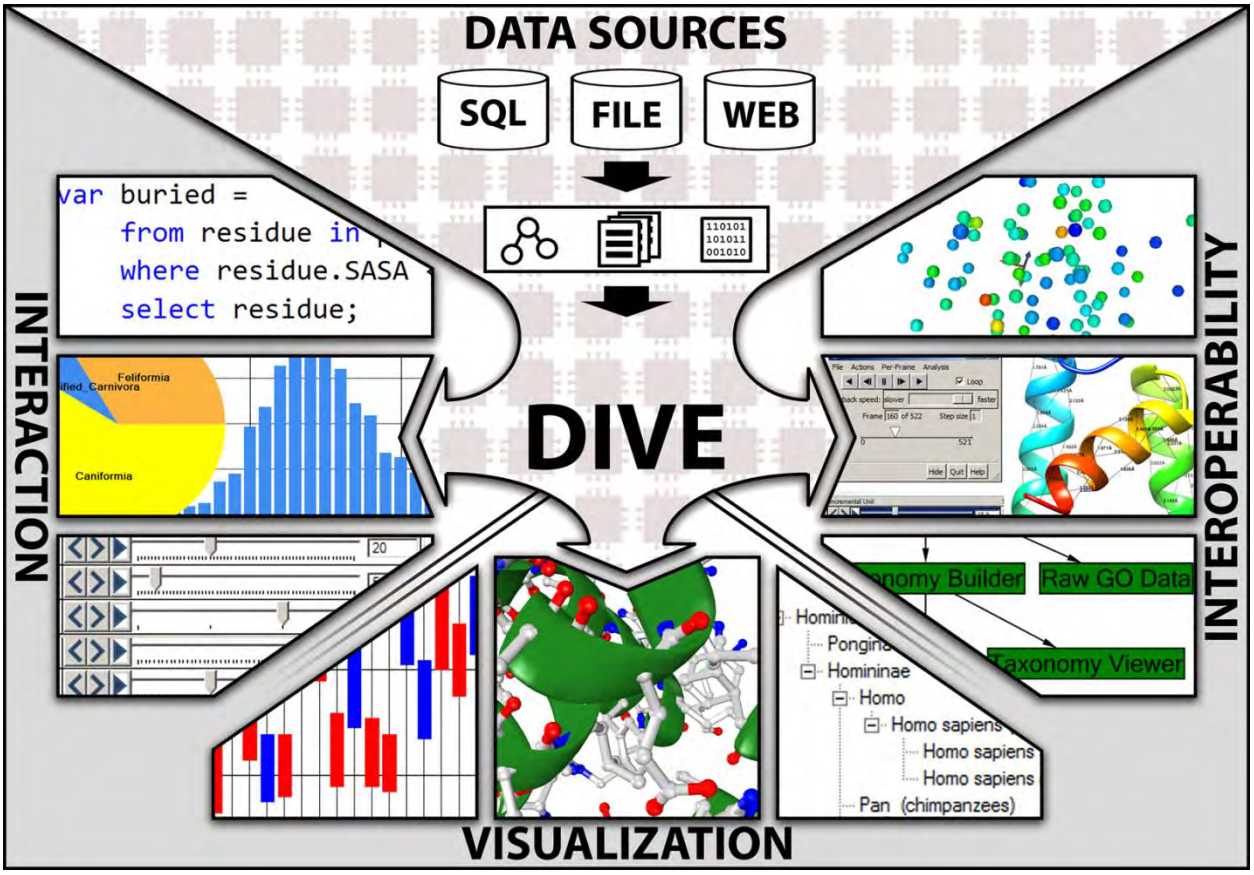


Figure 2.1 An overview of DIVE (Data Intensive Visualization Engine), with screenshots. Users can access and structure data in various ways, including interactive and real-time data streaming. DIVE allows various types of interoperability, including interoperability with existing software libraries, 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 taxonomy from the Gene Ontology. Interaction scenarios include scripted data manipulation, GUI interaction via charts and graphs, and event-driven data loading.

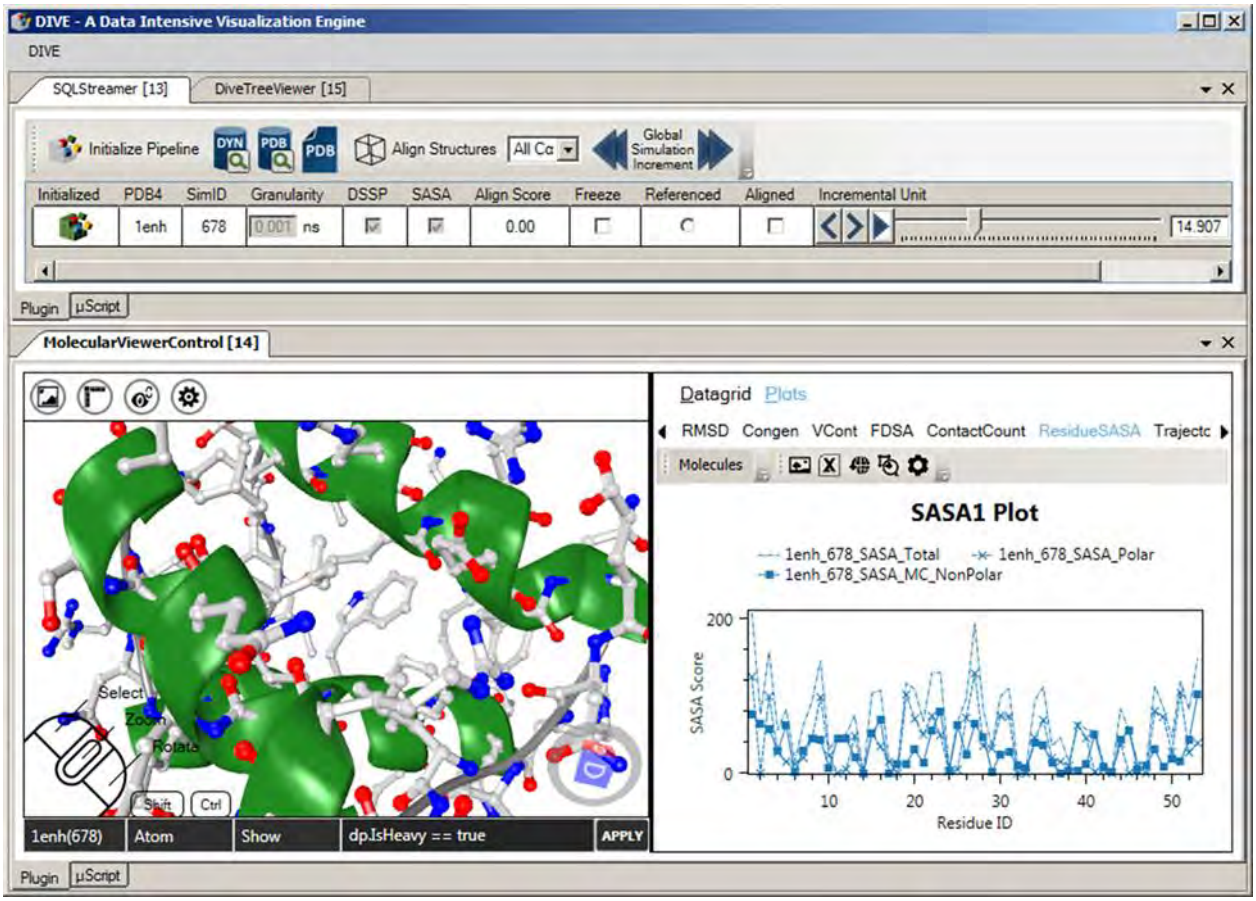


Figure 2.2 The DIVE GUI with the Protein Dashboard pipeline loaded.
At the top is a data loader with which users can load and interact with protein structures and molecular-dynamics trajectories (see the "Molecular Dynamics" sidebar) from different sources. On the lower left is an interactive 3D rendering 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.

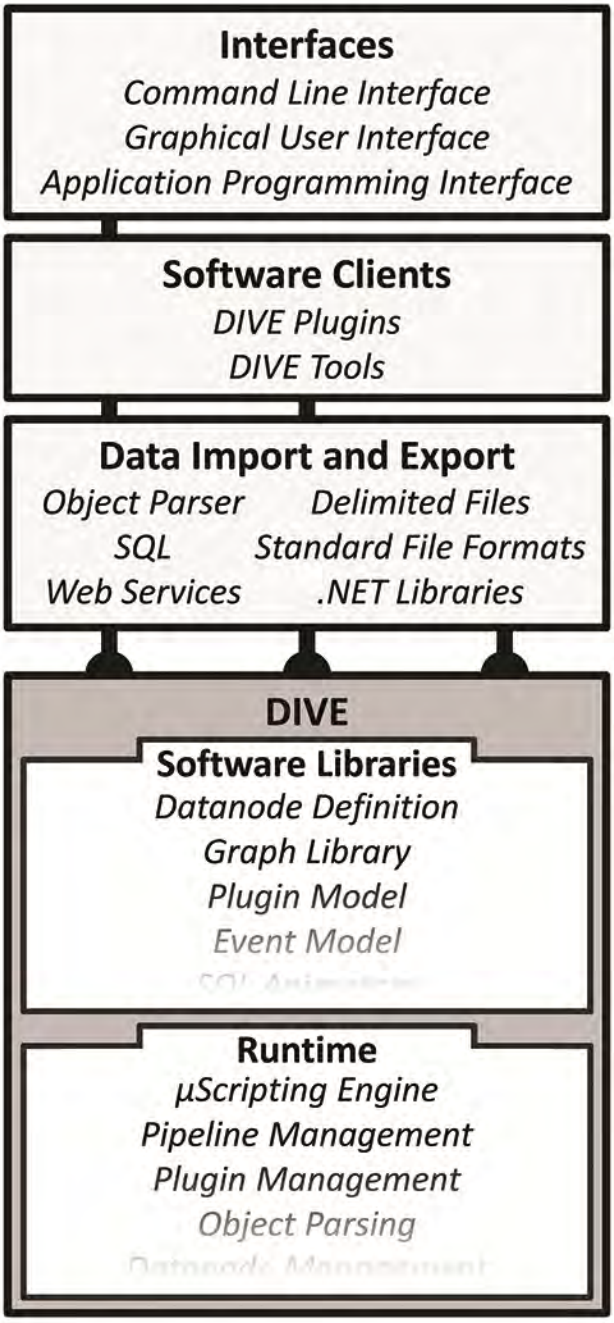


Figure 2.3 The DIVE architecture.
The DIVE kernel acts as both a software library and runtime environment. In both cases, DIVE can import and export data and functionality from a variety of sources. Pipeline plug-ins use DIVE primarily as a software library, exploiting DIVE's data-handling capabilities. DIVE tools are applications that instantiate and launch a DIVE pipeline for a specific analysis task. DIVE supports multiple types of interfaces.

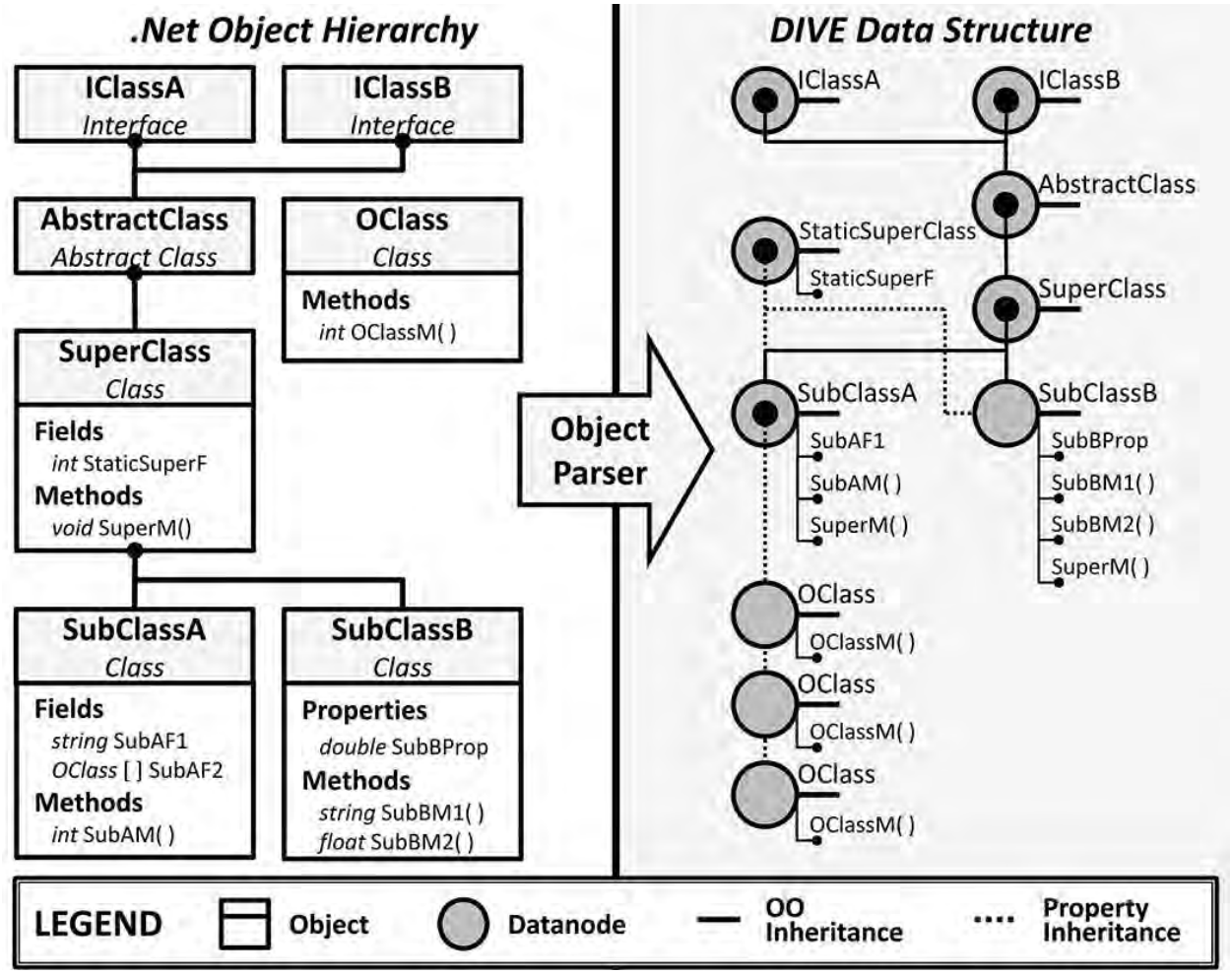


Figure 2.4 A mapping of a datanode-ontology from a third-party .NET assembly.
On the left, a generic .NET class hierarchy contains interfaces; class inheritance; and member fields, properties, and methods. On the right, the automatically generated ontology replicates the strongly typed objects and relationships 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.

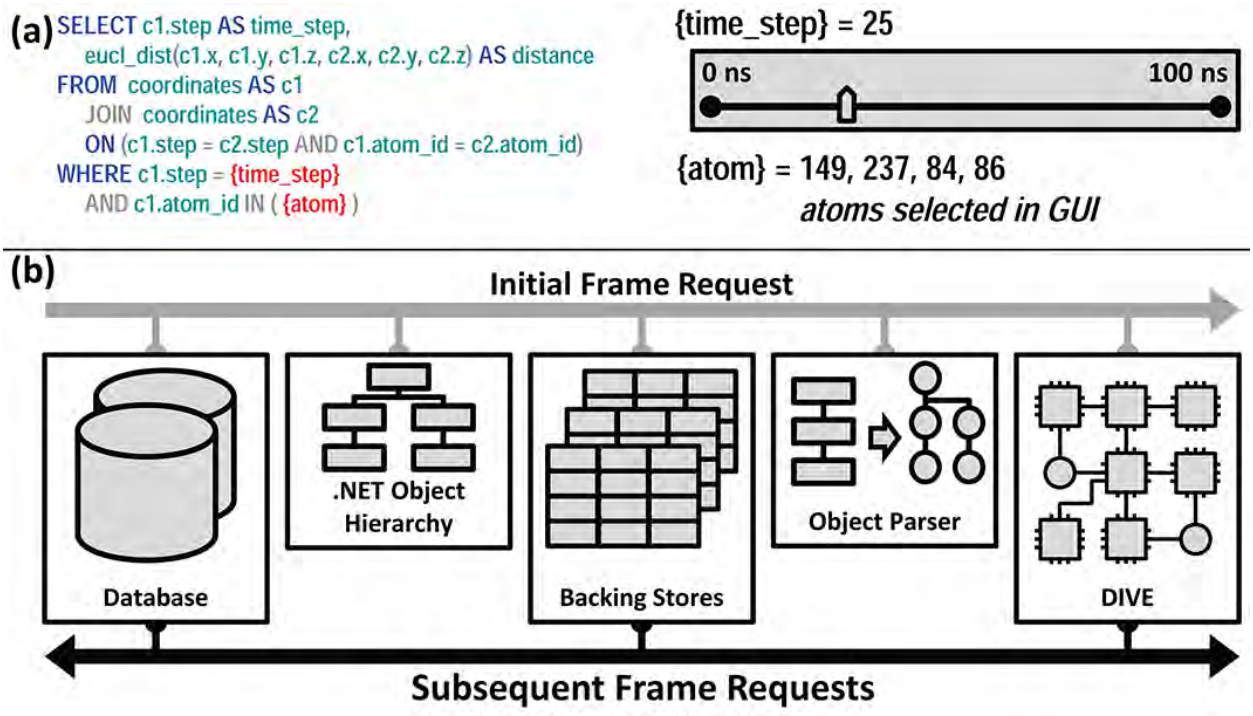


Figure 2.5 SQL streaming in DIVE.
(a) Interactive SQL. On the left is an SQL template with tags for time_step and atom. This approach replaces the tags with input from GUI elements, and the final query calculates the distances between all user-selected atoms at the specified time. (b) Pass-through SQL. 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.

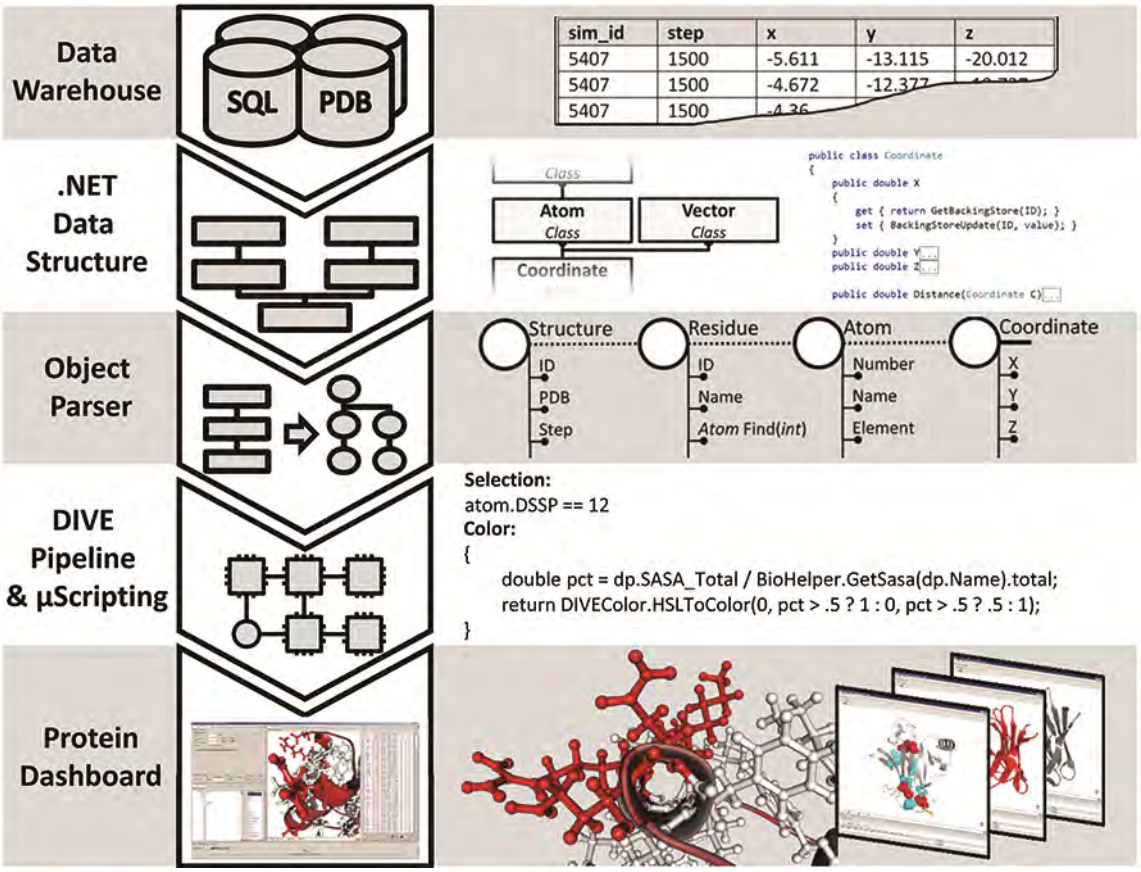


Figure 2.6 The Protein Dashboard case study.
First, data are parsed in from the Dynameomics SQL warehouse or the Protein Data Bank (PDB), populating the Dynameomics API's backing stores. DIVE then parses these data structures and creates corresponding datanodes and dataedges available to the DIVE pipeline. 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

The contents of this chapter and Appendix C were previously published in the journal *Bioinformatics* (Bromley et al. 2014). Steven Rysavy and I were co-first authors of this paper. As a result, this previously-published content appears in his PhD dissertation as well. In addition to sharing the overall design of the DIVE framework, my primary contributions were in the core DIVE kernel, encompassing the ontological data structure, μscripting, the DIVE pipeline and associated pipeline-plugin technologies, the DIVE GUI, the majority of the visualization plugins, interactive SQL, data-streaming protocols, and the development of internal analytical software-libraries such as mathematical libraries and signal processing libraries.

3.1 Abstract

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. Binaries and documentation are available at http://www.dynameomics.org/DIVE/DIVESetup.exe.

3.2 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 (Appendix Figure C.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.

3.3 System and Implementation

DIVE provides a rich ontologically expressive data representation and a flexible modular streaming-data architecture or pipeline (Appendix Figure C.2). It is accessible through an application programming interface, 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 although 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 (Appendix C). A thorough description of the novel technical contributions of DIVE is provided elsewhere (Rysavy et al. 2014).

3.4 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 and Daggett 2012) and file system environment distributed over multiple servers containing hundreds of terabytes of data and $>10^4$

times as many structures as the Protein Data Bank (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 the p53 DNA binding domain is responsible for destabilizing the core (Joerger et al. 2006), leading to ~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 3.1a 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.

As another example of the use of DIVE, we have >300 simulations of 106 disease-associated mutants of SOD1 (Schmidlin et al. 2009). Through extensive studies of A4V mutant SOD1 simulations, Schmidlin *et al* 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 Appendix Figure C.1).

The top panel of Figure 3.1b maps secondary structure for different variants as an example of DIVE's charting tools. This chart is quickly generated, contains results for >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 3.1b). These data are in turn linked to protein structure modules, allowing interactive visualization of >60,000 structures from each of the 300 simulations, all streamed from the Structured Query Language (SQL) data warehouse (Figure 3.1b). 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 (Appendix Figure C.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 Appendix C. One example shows an interaction with the Gene Ontology (Ashburner et al. 2000), and another example explores professional baseball statistics.

3.5 Conclusions

Overall, DIVE provides an interactive data-exploration framework that expands on 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.

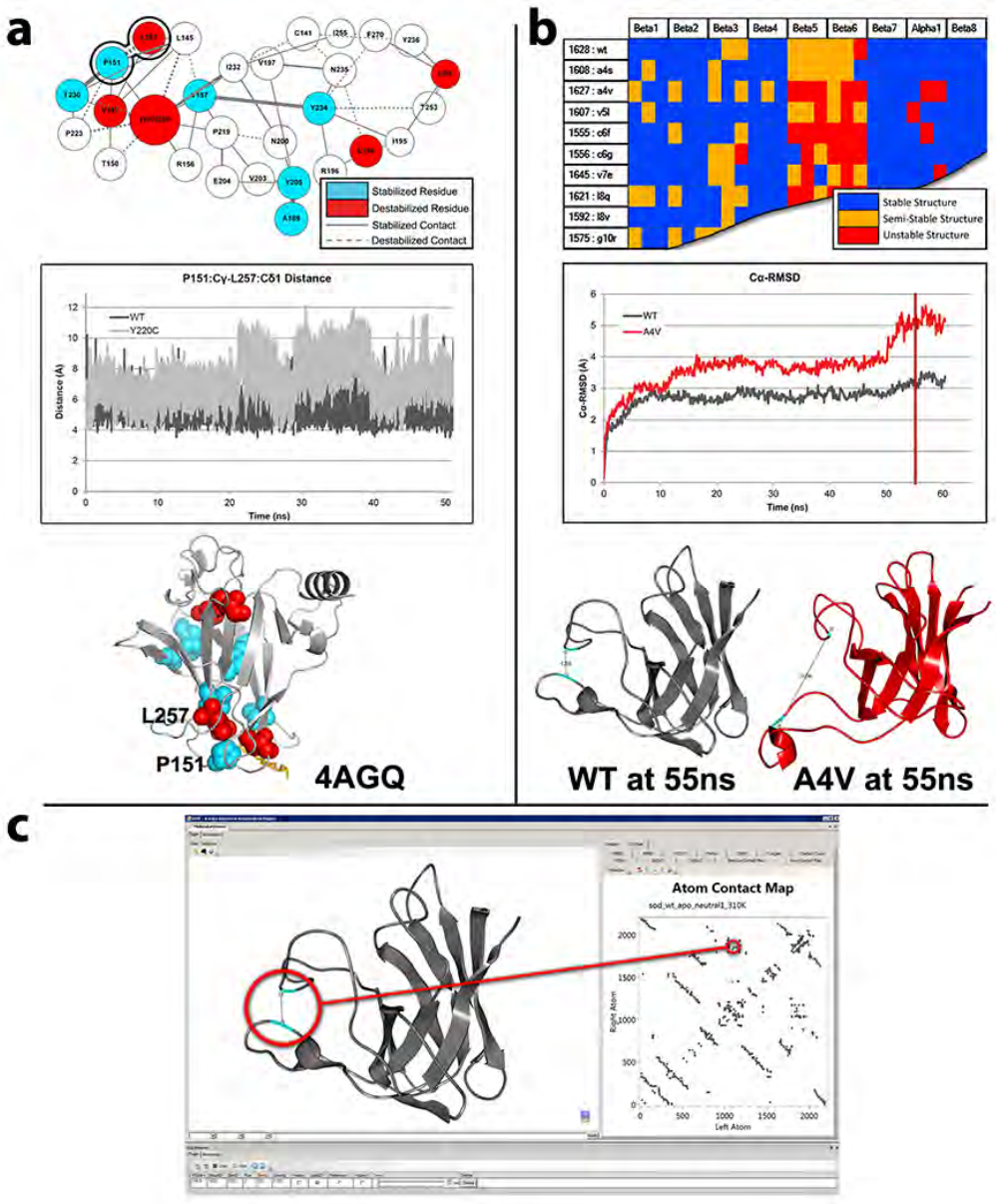


Figure 3.1 Interactive visualizations in DIVE
(a) The p53 analysis visualizations. Top, ContactWalker summary of contact differences between wild type and Y220C simulations. The highlighted residues have contacts with ≥50% occupancy change. Middle, distances between P151 and L257, outlined in black in the map above. Bottom, p53 with ligand (stick figure at bottom) (Protein Data Bank code 4AGQ) in proximity to disrupted colored residues. (b) SOD1 analysis visualizations. Top, aggregated secondary structural data from mutant simulations. Middle, plot of the Cα root-mean-squared (RMS) deviation of the wild-type and A4V mutant simulations. Bottom, MD structures. (c) Protein dashboard application showing a viewer and interactive contact map.

Chapter 4

STRUCTURAL CONSEQUENCES OF MUTATIONS TO THE α-TOCOPHEROL TRANSFER PROTEIN ASSOCIATED WITH THE NEURODEGENERATIVE DISEASE ATAXIA WITH VITAMIN E DEFICIENCY

The contents of this chapter were previously published in the journal *Biochemistry* (Bromley et al. 2013).

4.1 Abstract

The α-tocopherol transfer protein (α-TTP) is a liver protein that transfers α-tocopherol (vitamin E) to very low-density lipoproteins (VLDLs). These VLDLs are then circulated throughout the body to maintain blood a-tocopherol levels. Mutations to the a-TTP gene are associated with ataxia with vitamin E deficiency, a disease characterized by peripheral nerve degeneration. In this study, molecular dynamics simulations of the E141K and R59W disease-associated mutants were performed. The mutants displayed disruptions in and around the ligand-binding pocket. Structural analysis and ligand docking to the mutant structures predicted a decreased affinity for

a-tocopherol. To determine the detailed mechanism of the mutation-related changes, we developed a new tool called Contact Walker that analyzes contact differences between mutant and wild-type proteins and highlights pathways of altered contacts within the mutant proteins. Taken together, our findings are in agreement with experiment and suggest structural explanations for the weakened ability of the mutants to bind and carry a-tocopherol.

4.2 Introduction

α-Tocopherol transfer protein (α-TTP) transfers α-tocopherol, a strong antioxidant and a form of vitamin E, to very-low-density lipoproteins (VLDLs). These lipoproteins, located in the liver, are then circulated throughout the body to maintain blood α-tocopherol levels. Vitamin E occurs in multiple forms, including α-, β-, γ-, and δ-tocopherol. Each form has a hydrophobic tail with three chiral centers, and α-TTP preferentially binds RRR-α-tocopherol (Figure 4.1a) (Meier et al. 2003).

Patients with mutations in the α-TTP gene have a weakened ability to transfer α-tocopherol to VLDLs and, as a result, are unable to maintain blood α-tocopherol levels. This condition, known as ataxia with vitamin E deficiency (AVED), is associated with peripheral nerve degeneration. Although the link between α-tocopherol deficiency and AVED-associated nerve damage is not fully understood, it is likely that α-tocopherol protects tissues from oxidative stress. Increasing the level of dietary consumption of vitamin E can be an effective treatment for AVED (Di Donato et al. 2010).

α-TTP belongs to the SEC14-like family of proteins and is composed of two domains: an N-terminal three-helix bundle and a C-terminal CRAL-TRIO domain (Figure 4.1b). The latter

domain is used for transporting hydrophobic molecules through hydrophilic environments; other SEC14-like proteins include human supernatant protein factor and cellular retinaldehyde-binding protein (CRALBP). The CRAL-TRIO domain has a ligand binding pocket, entry to which is controlled by a swinging "lid". Investigations of SEC14 and CRALBP indicate that the lid is "hinged" by two regions of residues: residues 242-244 and 263-265 for CRALBP (Liu et al. 2005) and residues 212 and 213 and 239, 240, and 242 for Sec14 (Ryan et al. 2007). On the basis of sequence alignment of these proteins (Liu et al. 2005), these regions fall in the vicinity of residues 194-201 and 220-224 for a-TTP. These regions are termed hinge 1 (H1) and hinge 2 (H2), respectively (Figure 4.1b). In α-TTP, the lid is formed by residues 198-221 (Meier et al. 2003), the central structure of which is helix α14.

Meier et al. crystallized α-TTP in both the lid-open (PDB entry 1OIZ) (Figure 4.1b) and lid-closed (PDB entry 1OIP) (Figure 4.1c) forms (Meier et al. 2003). When α-TTP is not bound to a-tocopherol, the lid is open and the ligand-binding pocket is accessible. When α-TTP is bound to α-tocopherol, the lid is closed and the binding pocket is separated from the external environment. The open and closed forms are overlaid in Figure 4.1d.

In this study, we investigated two mutants of α-TTP: one replaces a Glu with a Lys at position 141 (E141K), and the other replaces an Arg with a Trp at position 59 (R59W). Both mutants alter strongly conserved residues (Panagabko et al. 2003) and are associated with severe clinical effects (Meier et al. 2003; Cavalier et al. 1998; Morley et al. 2004). Although experimental binding, ligand transfer, and urea denaturation data have been reported for these mutants (Morley et al. 2004), experimentally determined structures of these mutants are not available.

Consequently, we have used all-atom molecular dynamics (MD) simulations to investigate the structural effects of these mutations. This protein has not been the subject of previous MD simulations to the best of our knowledge, but Ryan et al (Ryan et al. 2007) and Schaff et al (Schaaf et al. 2011) have performed MD simulations of related Sec14 and Sfh1 proteins. In addition to standard structural analyses of the α-TTP simulations, docking calculations were also performed to assess the impact of mutations on the protein's ability to bind RRR-α-tocopherol.

Finally, to characterize how effects are propagated between the mutation site and the binding site, we created a new tool called Contact Walker. Analysis of these pathways provides a molecular description of the mutation-associated structural changes resulting from the E141K and R59W mutations.

4.3 Methods

4.3.1 *Structural Models*

The 1.88 Å resolution crystal structure of α-TTP was obtained from the PDB (Bernstein et al. 1977) (entry 1OIZ) (Meier et al. 2003) and was used for the starting structure for the wild-type and mutant simulations. The unbound, apo structure was used so we could investigate mutation-induced changes in the intrinsic structural and dynamic behavior of the protein and to avoid ligand-induced structural bias. Starting structures for both mutants were constructed by making the appropriate amino-acid substitutions to the wild-type protein. The energies of the resulting structures were minimized for 100 steps in vacuo using the ENCAD simulation package (Levitt 1990) and the Levitt et al. force field (Levitt et al. 1995). A different 1.95 Å resolution crystal structure of α-TTP (PDB entry 1OIP) (Meier et al. 2003) crystallized with bound α-tocopherol and

the lid "closed" was used as a ligand docking baseline. The model of RRR-α-tocopherol used for docking studies was extracted from the 1OIP crystal structure.

4.3.2 *Molecular Dynamics Simulations and Analysis*

MD simulations were performed using the *in lucem* molecular mechanics (*il*mm) software package (Beck et al. 2000-2014) using a previously published protocol and potential energy function (Levitt et al. 1995; Beck and Daggett 2004; Levitt et al. 1997; Beck et al. 2005). Starting structures were prepared for MD using 1000 steps of steepest descent minimization and then solvated with flexible F3C water molecules in a periodic box with walls located at least 10 Å from all protein atoms. The solvent density was set to the experimentally determined value for water at 37 °C, 0.993 g/mL (Kell 1967). The solvent energy was minimized for at least 500 steps before the solvent was heated again for 1 ps. The solvent energy was then minimized for an additional 500 steps and followed by an energy minimization of the entire system for 500 steps. Atomic velocities were assigned from a Maxwellian distribution at low temperatures and then brought to the target temperature of 37 °C (310 K). Thereafter, we used the NVE microcanonical ensemble, in which the box volume, number of particles, and total energy are fixed. A force-shifted nonbonded cutoff range of 10 Å was used for nonbonded interactions (Beck et al. 2005), and the interaction list was updated every two steps. Simulation steps were 2 fs, and the structures were saved every 1 ps. The temperature was set to 37 °C, and the simulations were performed at neutral pH (Asp and Glu negatively charged, Arg and Lys positively charged, and His neutral). Three independent simulations of the wild type and each mutant were performed for at least 51 ns. These simulations were performed as part of the SNP (single-nucleotide polymorphism)

thrust of our Dynameomics project (Beck et al. 2008; Van der Kamp et al. 2010), an ongoing project to determine the native-state dynamics and unfolding pathways of all known protein folds. a-TTP's three-helix bundle N-terminal domain is represented in fold rank 89 and the C-terminal CRAL-TRIO domain in fold rank 1251 in our 2009 Consensus Domain Dictionary (CDD) (Day et al. 2003; Schaeffer et al. 2011). Protein images were created using Pymol (DeLano 2002) and UCSF Chimera (Pettersen et al. 2004).

4.3.3 *Inter-Residue Contacts*

Inter-residue contact occupancies were calculated at 1 ps granularity using heavy-atom contacts from time steps greater than 25 ns. Contact distance thresholds were 5.4 Å for carbon-carbon contacts and 4.6 Å for all other heavy-atom contacts; residues were considered to be in contact if at least one interatomic contact was within the appropriate threshold. Contacts between neighboring residues were ignored. The "residue occupancy difference" was calculated as the difference in occupancy between each wild-type residue and the corresponding mutant residue. As a result, negative values indicated greater prevalence in the mutant (mutant stabilization) and positive values reflected greater prevalence in the wild type (mutant destabilization).

4.3.4 *Contact Walker*

Because of the dynamic nature of proteins, interatomic contact distances change over time and even stable contacts oscillate around some mean distance. As a result, searching for structurally-important contact changes between wild-type and mutant proteins can be

challenging. To address this challenge, we built Contact Walker, a tool that measures the difference in interatomic contact occupancies between wild type and mutant proteins.

Contact Walker calculates contact occupancy differences between a set of one or more wild-type simulations and a set of one or more mutant simulations. It uses the minimal demonstrated occupancy change between the wild-type and mutant simulations to avoid overestimating the significance of the change in occupancy. For example, the R54:Q145 contact had occupancy values of 24, 53, and 61% for the three wild-type simulations and 100, 100, and 98% for the three E141K simulations. The minimal demonstrated occupancy change between the wild type and the E141K mutant was -37% (61% - 98%). Using the minimal demonstrated occupancy change provided the most conservative measure of significant changes in the strength of protein contacts.

4.3.5 *Contact Networks*

A residue whose contacts have been altered, either by stabilization or by destabilization, is considered to be a disrupted residue. Preliminary observations found that disrupted residues tended to exist in connected networks rather than in isolation. Contact Walker can be used to visualize these networks, highlighting pathways of disrupted contacts (disruption pathways) and building on discretized, non-interactive network depictions such as those reported by Schmidlin et al (Schmidlin et al. 2009).

To do this, Contact Walker builds a connected graph to represent all the contact changes between the wild-type and mutant proteins. Graph nodes represent residues, and graph edges

represent nonbonded contacts with edge weights proportional to the change in contact occupancy. Backbone peptide bonds are inserted with an occupancy change value of zero.

Following construction of the contact graph, Contact Walker identifies the target residues. These are residues among which Contact Walker looks for networks of disrupted contacts. Residues can be manually specified if a region of interest is already known, or they can be automatically specified by selecting those residues with changes in contact occupancy that lie outside of some threshold. If the automatic method is used, additional manual targets may be specified as well.

Once the target residues have been identified, the user specifies the search parameters. These include options such as maximal search depth (the maximal number of contacts to search before trying another path) and a minimal edge value to determine if an edge is eligible for search. A cutoff in terms of absolute occupancy may also be specified. Backbone peptide bonds are always available for traversal.

Next, Contact Walker performs the actual search. A depth-first search (DFS) is performed between every pair of target nodes to find pathways of significant occupancy change. A DFS is a graph-traversal algorithm that searches a graph by moving from parent node to child node, often limited to a maximal number of steps, before backing out and trying a new path. In this scenario, this limits the search to a maximal number of inter-residue contacts between significant residues.

In the final step, the output graph is laid out and rendered. The output graph is a subset of the original connectivity graph containing only those edges and nodes that meet the user's

significance criteria. Graph images are rendered using the Graphviz graph layout tool (Gansner and North 2000) and the built-in scalable force-directed placement graph layout algorithm (Hu 2005). The graph layout is determined by the connectivity patterns of significantly changed edges, not necessarily the proteins' three-dimensional structure. Approximate structural regions of the protein are highlighted on the graph. Edge colors indicate the degree of occupancy change along a green (mutant stabilized) to orange (mutant destabilized) spectrum, and edge size indicates the magnitude of the occupancy change. Node color designates secondary structure per the legend. ContactWalker also outputs a Pymol script to visualize the contact change data mapped onto the protein structure.

4.3.6 *Ligand Docking*

To predict the effect of distortions of the binding site on the binding affinity of α-tocopherol, we performed docking calculations of RRR-α-tocopherol on MD structures taken between 25 and 51 ns. Docking energies and conformational pose calculations were made using AutoDock Vina (Trott and Olson 2010). Of the 26000 structures between 26 and 51 ns, 260 (1%) were selected using simple random sampling without replacement. The sample mean, the 95% confidence interval (CI), and the sample standard deviation were calculated using a two-tailed t-test, $\alpha = 0.05$, and either 260 or 780 degrees of freedom for individual simulation data or aggregated data, respectively. Docking energies are reported as the sample mean ± the standard deviation. Additionally, a baseline docking energy was established by docking RRR-α-tocopherol into the 1OIP crystal structure.

4.3.7 *Cα RMSD and Cα RMSF*

Cα RMSD values were calculated using only α-carbons and excluded both termini (residues 9-23 and 266-274) and the α14 region (residues 201-213). Cα RMSF data were calculated using MD structures beginning at 25 ns, used only α-carbons, and also excluded both termini. The α14 region was included in the Cα RMSF calculations because visual inspection of the trajectories showed that the lid had closed by approximately 10 ns.

α-Tocopherol headgroup-coordinating residues were defined as those residues with at least one atom within 5.4 Å of the headgroup of the vitamin E molecule. Water-coordinating residues were defined as residues with a nitrogen or oxygen atom within 4.6 Å of a water molecule. Calculations were performed using Pymol and the structure of PDB entry 1OIP of α-TTP.

4.4 Results and Discussion

In general, all simulations maintained the overall binding pocket structure and did not undergo significant unfolding; this is consistent with the urea denaturation experiments reported by Morley et al (Morley et al. 2004) that showed a 0.4 M change in C50 for the R59W and E141K mutants relative to that of the wild type (WT). The Cα RMSD of all three sets of simulations indicated that the proteins were relatively stable between 2 and 3 Å Cα RMSD, with the exceptions that each mutant had one simulation with Cα RMSD values of 4.0 Å (simulation 1 in each case) (Figure 4.2). Cα RMSF analysis of all three proteins showed a similar trend (Figure 4.3); each protein demonstrated Cα RMSF values around 1 Å in regions of secondary structure,

and in each case, the simulation with higher Cα RMSD values also showed higher Cα RMSF values, often in the vicinity of α13 and α14.

α-Tocopherol docking studies resulted in a range of docking energies from -6.5 to -10.2 kcal/mol (Figure 4.4). The 95% confidence intervals for the mean potential energies were 0.1 and 0.0 kcal/mol for the per-simulation and aggregated docking data, respectively. For reference, the potential energy of docking α-tocopherol into the 1OIP closed structure was -10.8 kcal/mol, and the Cα RMSD between the 1OIZ and 1OIP crystal structures excluding α14 was 0.5 Å. Thus, this control suggests that the method can correctly identify and recover both the experimentally determined structure of the complex and its correct binding interactions. Per-simulation α-tocopherol docking analyses demonstrated that both wild type and mutant proteins were capable of a range of docking energies (Figure 4.4a-c). In aggregate, the wild type and the E141K mutant had the same mean docking energy (-8.5 ± 0.7 and -8.5 ± 0.6 kcal/mol, respectively), and the R59W mutant performed less favorably (-8.1 ± 0.6 kcal/mol) (Figure 4.4d). Converting these potential energies to dissociation constants (K_d) resulted in values of 1000 nM for the wild type and E141K mutant and 1900 nM for the R59W mutant. This is approximately in keeping with the findings of Morley et al (Morley et al. 2004), who reported 2- and 4-fold increases in K_d for the E141K and R59W mutants, respectively.

4.4.1 *Wild-Type Simulations*

All three wild-type trajectories stabilized between 2.5 and 3 Å Cα RMSD (Figure 4.2a), and simulation 2 displayed more structural fluctuation than the other two (Figure 4.3b and Figure 4.5). In all three wild-type simulations, α14 underwent a swinging motion from the lid-open

position to the lid-closed position; Ryan et al (Ryan et al. 2007) reported a similar conformational change in MD simulations of the related Sec14 protein. The β-sheet wall of the binding pocket remained stable in all simulations, while the α-helix wall exhibited more dynamic behavior (Figure 4.3b and Figure 4.5). Despite these similarities, the three wild-type simulations each behaved slightly differently; to better characterize the range of behaviors of the wild-type protein, we analyzed each simulation separately.

As shown in Figure 4.5a, structures from simulation 1 did not deviate far from the minimized crystal structure with the exceptions of the α3-α4 loop, α5, α7, α13, N227, and the α17-α18 loop. Simulation 1 also provided the most favorable docking conformations of the three wild-type simulations with a mean potential energy of -9 ± 0.5 kcal/mol (Figure 4.4a). Because of its stability and favorable docking performance, wild-type simulation 1 was used as a structural baseline for subsequent contact comparisons.

Simulation 2 was more dynamic than either simulation 1 or simulation 3, particularly near α14 (Figure 4.3b and Figure 4.5b), and the mean docking energy (-8.4 ± 0.6 kcal/mol) was between the energies of the other two simulations (Figure 4.4a). Contact occupancy comparisons of simulations 1 and 2 indicated that the structures in simulation 2 were missing several binding pocket contacts that were present in simulation 1, including E82:K178 (100% difference) and I83:K178 (66% difference) contacts (Figure 4.5b). These two contacts tethered α5 to α13 and helped maintain the structure of the α-helix wall of the binding pocket. Both E82 and I83 established alternative stabilizing contacts, including 100% occupancy contacts with R134. K178 did not establish significant alternative stabilizing contacts and remained destabilized relative to

simulation 1 (1.5 Å Cα RMSF in simulation 2 vs 0.75 Å Cα RMSF in simulation 1). Structures from simulation 2 showed, relative to simulation 1, a general lack of contacts between α12-α14 lid and hinge regions. Of the 19 contacts in this region with occupancy differences of >50%, six were stabilized and 13 were destabilized. This difference in occupancy is reflected in the increased mobility and fluctuation of α14 in simulation 2 (Figure 4.3b and Figure 4.5b). The internal structure of α14 saw a similar destabilization; of the 13 contacts within α14 with occupancy differences of >50%, three contacts were stabilized and ten contacts were destabilized.

Simulation 3 had relatively low Cα RMSF values, similar to those of simulation 1 (Figure 4.3b and Figure 4.5c), but it also had the least favorable docking energy of the three wild-type simulations (-8.1 ± 0.5 kcal/mol) (Figure 4.4a). In total, 27 contacts had occupancy differences of >99% relative to simulation 1, the majority of which were in the vicinity of the hinge and lid region formed by α12-α14. Also, like simulation 2, the I83:K178 contact (66% occupancy) was not present in simulation 3. However, unlike simulation 2, the E82:K178 contact was intact, retaining some of the α13 tethering. Furthermore, also unlike simulation 2, K178 formed alternative stabilizing contacts with L183 (93%). Thus, K178 was not completely disconnected from E82 and I83, nor was it left in a destabilized state, resulting in slightly lower Cα RMSF values (1.1 Å vs 1.5 Å in simulation 2).

4.4.2 *E141K Simulations*

Overall, the E141K simulations were fairly stable and did not undergo any significant unfolding. As indicated by the Cα RMSD calculations (Figure 4.2b), simulations 2 and 3

stabilized between 2 and 2.5 Å, while simulation 1 deviated from the starting structure by as much as 4 Å. Consistent with the Cα RMSD data, the Cα RMSF data indicated that simulation 1 of the E141K mutant was more dynamic than simulation 2 or 3 (Figure 4.3c and Figure 4.6).

Comparing contact occupancies of all three E141K simulations against those of the wild-type simulation 1 baseline, we found 15 contacts with a minimal demonstrated occupancy change of >95%. Of these, three were stabilizations of the hinge 2 region (K217:R221, K217:E220, and E216:E220), five were destabilizations that included the lid (F213:K217, F213:E216, L214:K217, P212:E216, and K211:T215), one tethered α13 to α14 (T184:T215), and one untethered α5 from α13 (W76:S186). All 10 of these contact changes were also observed in the wild-type simulations. However, their presence in the E141K mutant was both more common (all three E141K simulations) and, in the case of the hinge 2 region and α13-α14 tethering, larger in magnitude (Table 4.1).

To isolate the specific effects of the E141K mutation from the inherent propensities of the protein, we compared all three E141K simulations against all three wild-type simulations. Only nine contacts were observed that had a minimal demonstrated occupancy difference of >20% (Table 4.2). Of these, four of the nine contacts were in or near hinge 2. Disruption pathways between the mutation site and the hinge 2 region (Figure 4.7) were found by increasing the sensitivity of Contact Walker (significance threshold of 9%, allowable threshold of 6%, search depth of 2, absolute occupancy cutoff of 0.01). The primary contact change at the mutation site was the R54:K141 contact (92% loss). As indicated in the figure, the disruption pathway included α5, α10, α13, and the hinge 2 region.

Zhang et al. reported that membrane binding is dependent on several hydrophobic residues, including F165 and F169 (Zhang et al. 2011). While these residues did not experience appreciable contact disruption, they did have average Cα RMSF and Cα RMSD values different from those of WT; the Cα RMSD values were 2.9 and 2.6 Å in WT and 1.9 and 1.9 Å in E141K for residues F165 and F169, respectively, and the Cα RMSF values for F169 were 1.4 Å in WT and 1.2 Å in E141K.

Several headgroup-coordinating residues were disrupted: F133, L137, V182, L183, F187, and L198 (Figure 4.7). Although docking studies with MD structures did not indicate significant disruption to α-tocopherol binding for this mutant, these contact changes may be indicative of binding pocket disruptions that take place over time scales longer than those currently available with molecular dynamics. Also, the 1OIP crystal structure shows four crystallographic water molecules involved with the α-tocopherol headgroup binding. Three of the disrupted headgroup-coordinating residues, F133, V182, and F187, also help to coordinate these waters. Data reported by Schaaf et al (Schaaf et al. 2011) suggest that water coordination within the analogous hydrophobic ligand binding pocket of the related Sec14 and Sfh1 proteins may be important to the functioning of those proteins.

In addition to possible binding site disruption, these results indicate that a structural effect of the E141K mutation was to disrupt the hinge 2 region. This is reflective of the *in silico* findings of Ryan et al. that the G266D mutant of the related Sec14 protein disrupts hinge function (Ryan et al. 2007), although we did not observe significant disruption to the gating module reported in that same study. The ubiquity of these disruptions within the mutant simulations

suggests an explanation for part of the increase in K_d relative to that of the wild type reported by Morley et al. (Morley et al. 2004); the destabilization caused by the E141K mutation was enough to disrupt the hinge and lid region of the protein, but not enough to severely disrupt the actual α-tocopherol binding pocket (although, as discussed above, some loss of binding ability may be attributable to pocket disruption). As a result, the E141K *in silico* docking energies were almost identical to those of the wild type, while the *in vitro* experiments, which were dependent on the wild-type-like gate and hinge behavior, demonstrated an increase in K_d.

4.4.3 *R59W Simulations*

The R59W mutant simulations, like the E141K simulations, did not undergo significant unfolding. Simulations 2 and 3 were relatively stable with Cα RMSD values stabilizing between 2.5 and 3 Å (Figure 4.2c). Simulation 1, however, was considerably more dynamic with Cα RMSD values as high as 4.5 Å and Cα RMSF values often twice those of the other simulations (Figure 4.3d).

Comparison of all three R59W simulations against the wild-type simulation 1 baseline revealed four disrupted contacts in the lid region: K211:T215 (83%), P212:E216 (86%), F213:E216 (100%), and F213:K217 (100%) Four contacts that tethered α13 in place were also destabilized: W76:F187 (58%), Y73:P188 (83%), W59:T184 (87%), and W76:S186 (91%).

Comparison of contact occupancies between all three wild type simulations and all three R59W simulations resulted in low minimal demonstrated occupancy change values. Only 18 contacts demonstrated minimal occupancy changes of >20%. Of these, only six contacts exhibited occupancy change values of >50%. Notably, the W59 mutation site did not see

significant occupancy change; in the aggregate, the largest minimal demonstrated occupancy change for a W59 contact was a 6% contact occupancy increase with R54.

R59W simulations 1 and 3, considered on their own, exhibited less favorable α-tocopherol docking energies than simulation 2 (Figure 4.4 and Figure 4.8). They were also structurally similar to wild-type simulation 2, displaying a characteristic separation of α13 from α5 and α10 (Figure 4.5b). Contact comparison of only these two simulations against wild-type simulation 1 revealed additional disruptions to T184:K217 (98% destabilization), L214:K217 (100% destabilization), and K217:R221 (99% stabilization) contacts. The T184:K217 and L214:K217 contact disruptions were also present in the poorer-docking wild-type simulations and in the E141K simulations. Contact analysis also revealed stabilization in the hinge 2 region, similar to the E141K hinge 2 stabilizations.

Given these data, we took R59W simulation 1 and 3 conformations to be representative of mutation-induced disruptions. To isolate the structural disruptions associated with weakened docking performance, R59W simulations 1 and 3 were compared against R59W simulation 2; 50 contacts with >90% occupancy change were identified, of which 72% were destabilized (Figure 4.9). Contacts responsible for holding α13 in place were destabilized, including E82:K178 (100%), E141:F187 (99%), Y73:S186 (100%), N72:D185 (99%), W76:V182 (97%), N72:S186 (96%), and F133:L183 (91%) contacts. Of the stabilized contacts, three contacts strengthened the connection between the C-terminus of α13 and the N-terminal three-helix bundle: W59:F187 (93%), L65:S186 (93%), and A58:L189 (98%). Notably, one of these (W59:F187) included the mutated residue. Furthermore, the hinge 1 region contained three significant destabilizing

interactions [E199:N227 (91%), P200:H225 (95%), and I197:P200 (94%)]; the hinge 2 region contained one significant stabilizing interaction [E216:E220 (94%) as well as others in the near vicinity], and the α14 lid contained one significant destabilization [K211:E216 (97%)]. The gating module reported by Ryan et al (Ryan et al. 2007). was disrupted only in the H101:S136 contact.

The R59W mutation strengthened the tendency of the protein to adopt conformations that were unfavorable for ligand docking; relative to the wild-type simulation 1 baseline, the lid and hinge regions were disrupted and α13 became disrupted more often in the R59W simulations than in the wild-type simulations. Furthermore, the *in silico* docking was less favorable than that of the wild type, reflecting structural changes to the binding pocket in addition to disruptions to the lid and hinge regions. Like the E141K mutation, the R59W mutation disrupted several headgroup-coordinating residues, including F133, S136, V182, L183, F187, and L189 (Figure 4.9). All of these residues except L183 also help to coordinate crystallographic waters in the ligand binding pocket of the 1OIP (closed) crystal structure.

Finally, like the E141K mutant, the R59W mutant did not cause significant contact disruption to F165 and F169, the residues reported by Zhang et al. to be important to membrane binding (Zhang et al. 2011). These simulations did, however, show a lower average Cα RMSD relative to the starting structure than did the WT residues (2.9 and 2.6 Å in the WT vs 2.3 and 2.2 Å in R59W for residues F165 and F169, respectively). Also, F165 had a larger average Cα RMSF in the R59W simulations (1.2 Å in WT vs 1.5 Å in R59W).

4.5 Conclusions

In aggregate, the mutants had very few large occupancy changes relative to the wild type, indicating that their observed behaviors were already present in the wild type. The primary effect of both mutations was to exacerbate preexisting tendencies; in the case of the E141K mutant, we observed changes in the hinge 2 region. In the case of the R59W mutant, we observed a widening of the ligand binding pocket. In both cases, the mutant proteins remained folded and relatively stable, indicating that these mutants' weakened α-tocopherol binding is due to a shift in the population with an increase in conformers with altered binding regions.

Genetic diseases can have severe clinical effects while causing only subtle changes to protein structure. As demonstrated by the E141K and R59W α-TTP mutations, the cause of these disruptions can be masked by the dynamic propensities of the protein. In the case of the E141K mutant, finding that the hinge was disrupted with only minor disruptions to the actual ligand binding pocket highlights the complexity and subtlety of protein dynamics. Occupancy comparison was a useful tool for characterizing these dynamic systems because we were able to establish a wild-type baseline against which we could compare mutation-associated structural changes. We were also able to characterize large-scale structural changes without losing single contact data resolution. Visualization of these contact changes, together with docking studies and traditional analyses, facilitated the exploration of gigabytes of MD data and provided both quantitative and qualitative accounts of the changes that occurred as a result of mutations to α-TTP. These findings offer a structural description of the E141K and R59W mutants and suggest a molecular basis for their attenuated ability to transport α-tocopherol.

Table 4.1 Occupancy Differences in Contacts Common to Both E141K and Wild Type

Residue 1	Residue 2	WT1 vs. <u>WT2</u>[a]	WT1 vs. <u>WT3</u>[a]	WT1 vs. <u>E141K (all)</u>[a]
W76	S186	68%	97%	97%
K211	T215	96%	97%	95%
P212	E216	99%	99%	98%
F213	E216	100%	100%	100%
F213	K217	100%	100%	100%
L214	K217	100%	98%	100%
T184	T215	-43%	-49%	-96%
K217	E220[b]	-24%	-4%	-99%
K217	R221[b]	-86%	-57%	-99%
E216	E220[b]	-20%	(0% in both WT1 and WT3)	-100%

[a]Negative numbers indicate stabilization in the compared structure (underlined). [b] Hinge 2 region.

Table 4.2 E141K Occupancy Change Magnitudes > 20%

Residue 1	Residue 2	Occupancy Difference
E216	E220	-80%
K217	E220	-75%
A10	D60	-53%
T184	T215	-47%
R54	Q145	-37%
L183	I218	-37%
A10	F61	-27%
R77	L137	73%
R54	K141	92%

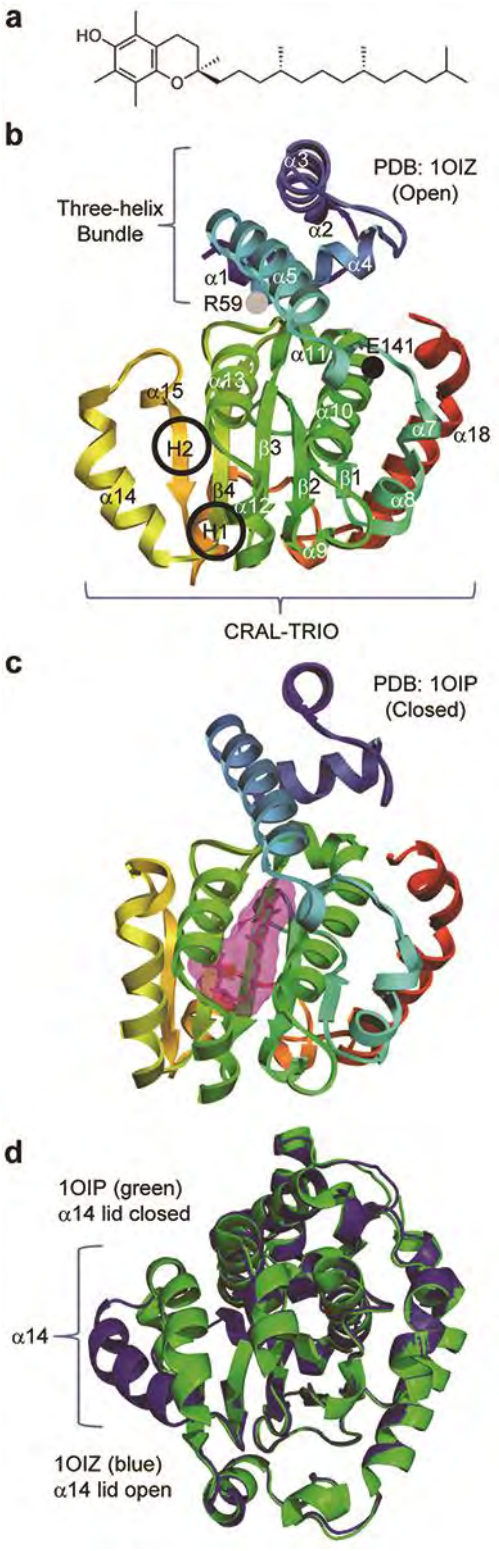


Figure 4.1 Molecular structures of vitamin E and the different forms of α-TTP
(a) RRR-α-tocopherol (vitamin E). (b) In the unbound structure (PDB code: 1OIZ), a 'lid' formed by the α14 region is open, allowing entrance to a hydrophobic binding pocket. The CRAL-TRIO domain is indicated in brackets. The black and gray circles indicate the E141K and R59W mutation regions, respectively. The two hinge regions are circled and labeled as H1 and H2. (c) In the bound conformation (PDB code: 1OIP), the lid is closed, enclosing the binding pocket. (d) 1OIZ (blue) and 1OIP (green) overlaid and aligned by Cα RMSD. The Cα RMSD between the two structures excluding α14 was 0.5 Å.

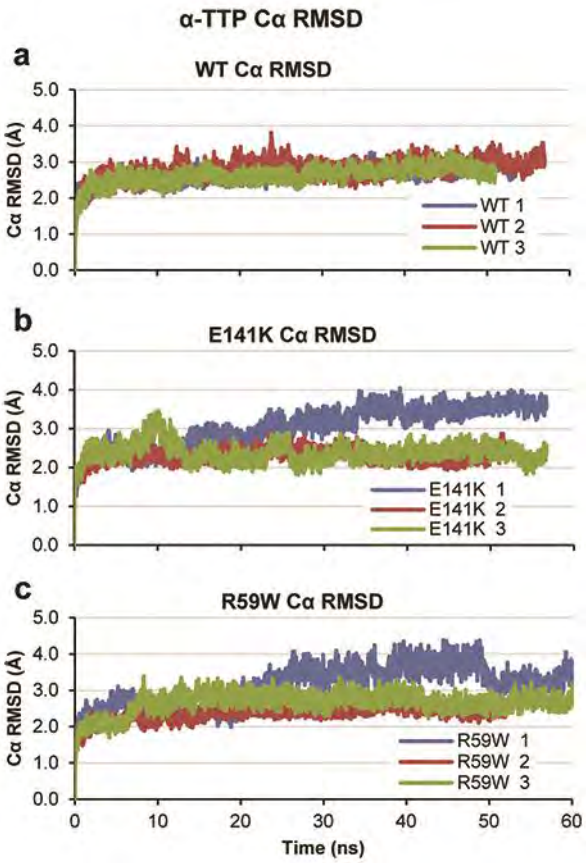


Figure 4.2 Cα RMSD values over time for all three proteins
Values were calculated using α-carbons excluding both termini and α14. (a) Wild type. (b) E141K. (c) R59W.

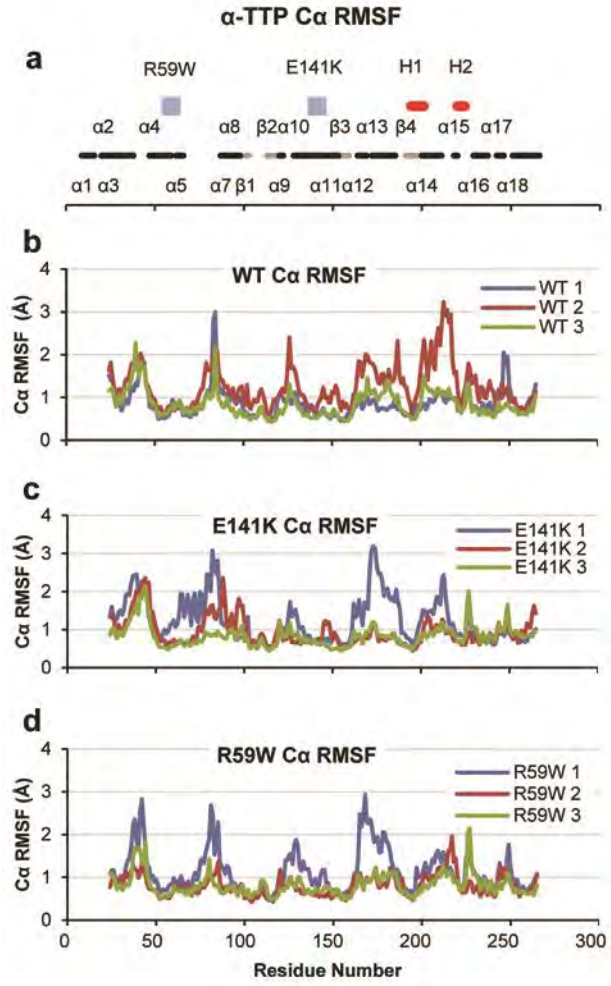


Figure 4.3 Cα RMSF for all three proteins
Cα RMSF calculated were calculated using α-carbons and excluding both termini. (a) Black and gray bands indicate α-helices and β-sheets respectively, blue squares indicate mutation sites, and red bands indicate hinge regions 1 and 2 (labeled H1 and H2). (b) Wild type. (c) E141K mutant. (d) R59W mutant.

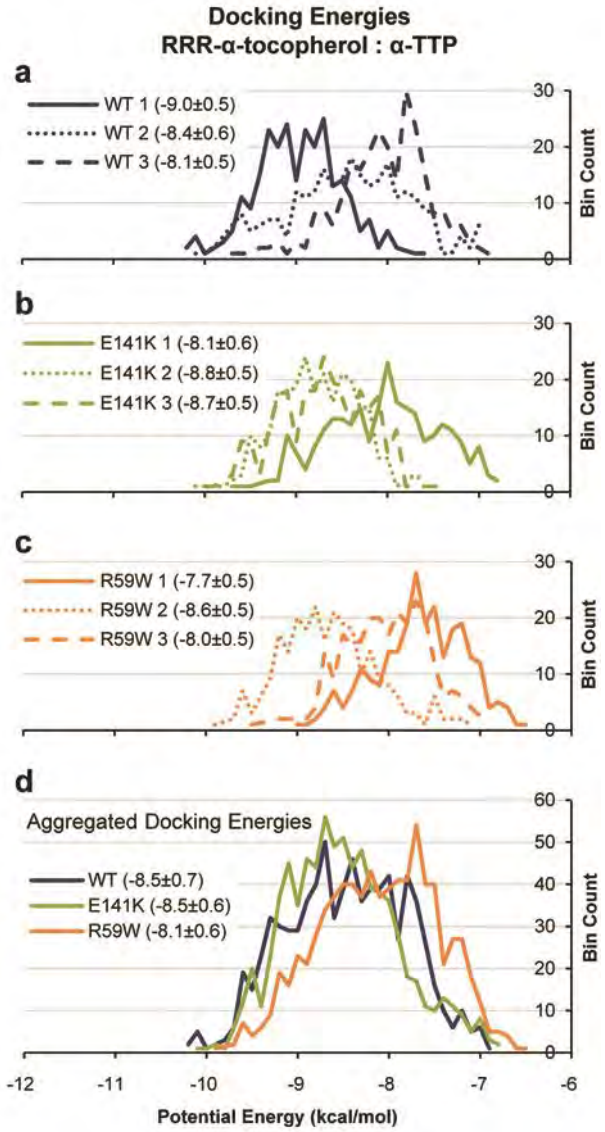


Figure 4.4 Histograms of α-tocopherol docking potential energies
AutoDock Vina was used to dock α-tocopherol into 260 randomly-selected structures from each simulation. Mean potential energies and standard deviations are shown in the legends. (a, b, c) Per-simulation histograms of wild type, E141K mutant, and R59W mutant, respectively. (d) Aggregated energies for each protein.

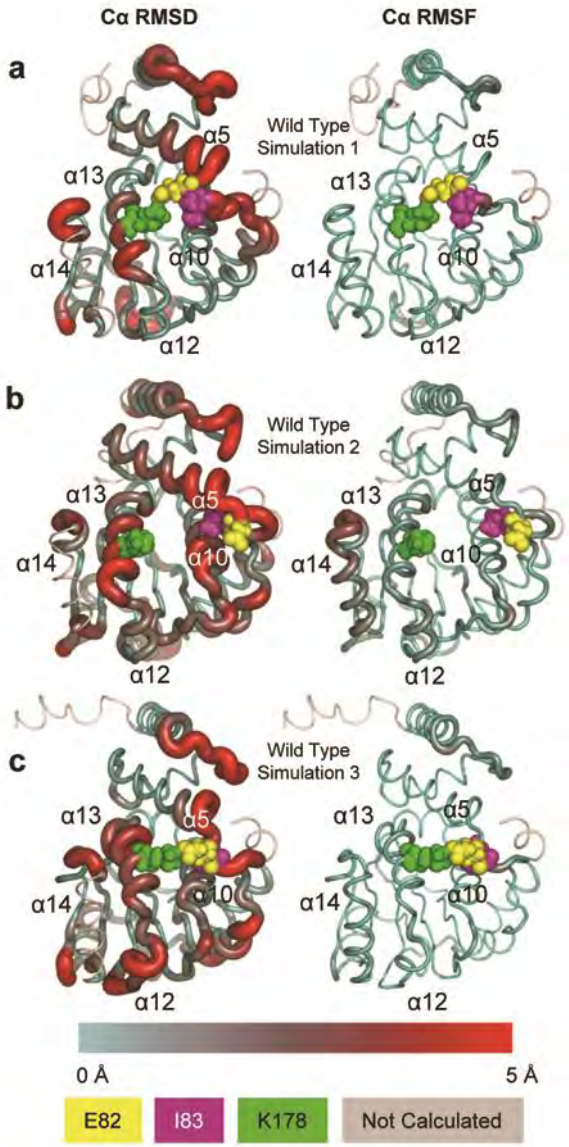


Figure 4.5 Representative wild-type structures rendered as tubes
Tube radius and color reflect Cα RMSD (left column) and Cα RMSF (right column). E82, I83 and K178 are colored per the legend and rendered as spheres. (a) Simulation 1. (b) Simulation 2. (c) Simulation 3. Note the separation of K178 from E82 and I83 in simulation 2.

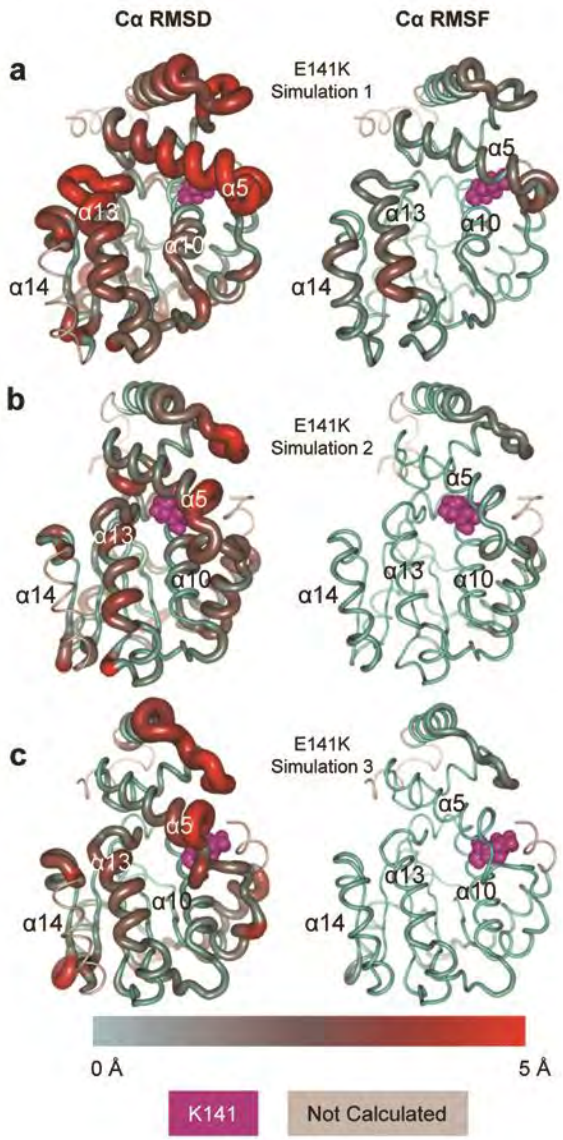


Figure 4.6 Representative E141K structures rendered as tubes
Tube radius and color reflect Cα RMSD (left column) and Cα RMSF (right column). The K141 mutation site is rendered as magenta spheres. (a) Simulation 1. (b) Simulation 2. (c) Simulation 3.

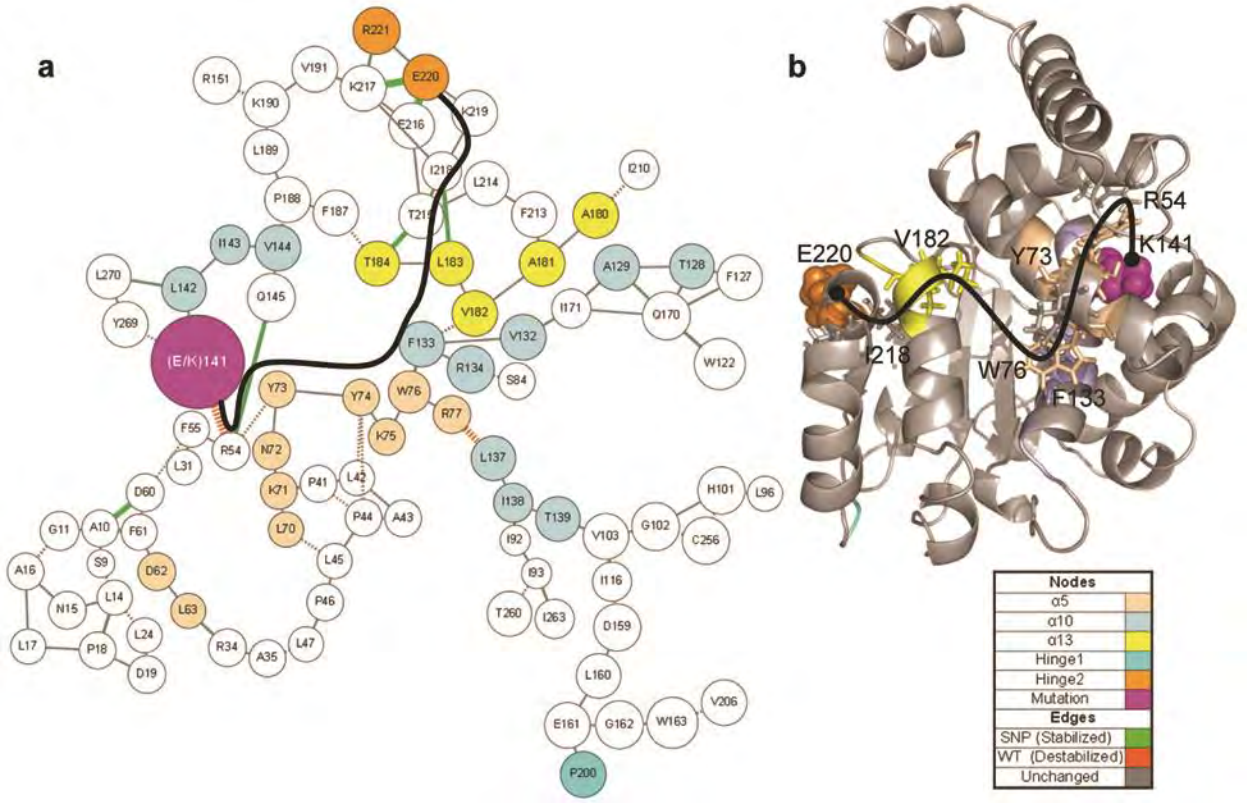


Figure 4.7 Contact disruption pathways in the E141K mutant
Contact occupancy differences were calculated between all wild-type simulations and all E141K simulations with a significance threshold of 0.09, an allowable traversal threshold of 0.06, and a search depth of 2. (a) ContactWalker disruption pathways. Line thickness indicates magnitude of occupancy change, green lines indicate SNP stabilization, and orange hashed lines indicate SNP destabilization. Nodes are colored per the legend. The black line highlights a pathway between the mutation site and the hinge 2 region. (b) 51 ns structure of E141K simulation 1. The black line indicates the same pathway that was labeled in (a). Pathway residues are rendered as sticks and labeled for orientation. The K141 mutation site is rendered as magenta spheres. The final residue of the pathway (E220) is rendered as orange spheres.

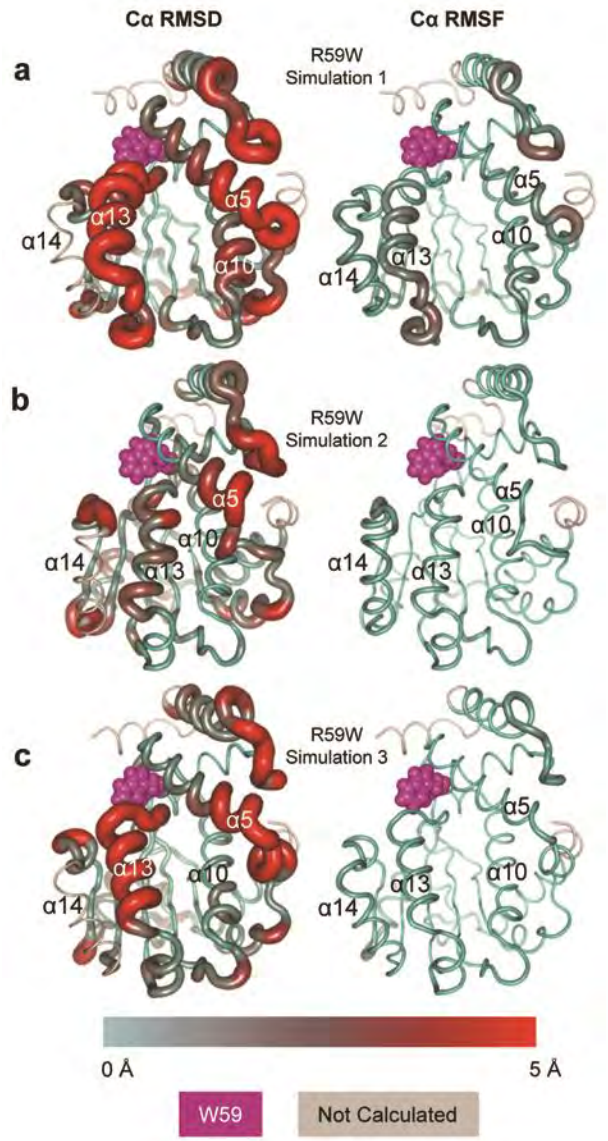


Figure 4.8 Representative R59W structures rendered as tubes
Tube radius and color reflect Cα RMSD (left column) and Cα RMSF (right column). The W59 mutation site is rendered as magenta spheres. (a) Simulation 1. (b) Simulation 2. (c) Simulation 3.

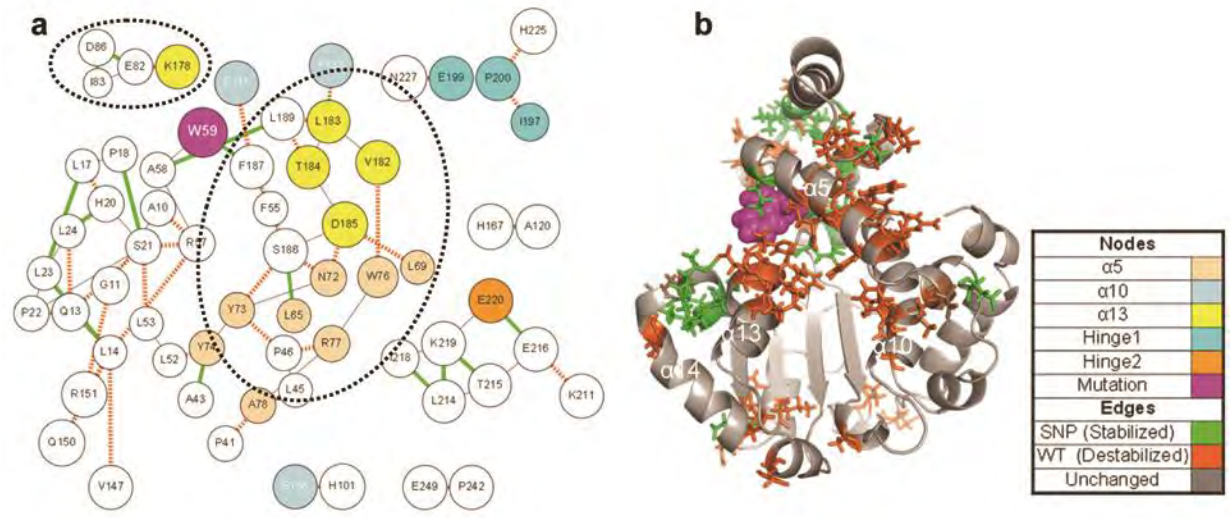


Figure 4.9 Contact disruption pathways in the R59W mutant
Contact occupancy difference was calculated between simulation 2 and simulations 1 and 3 with a significance threshold of 0.9, an allowable traversal threshold of 0.9, and a search depth of 1. (a) ContactWalker disruption pathways. Line thickness indicates magnitude of occupancy change, green lines indicate SNP stabilization, and orange hashed lines indicate SNP destabilization. Nodes are colored per the legend. The dotted circles highlight destabilized α13 residues. (b) 51 ns structure of R59W simulation 1. Significant nodes are rendered as sticks and colored based on overall stability (green) or instability (orange). Note the prevalence of destabilized residues between α5, α10, and α13 coincident with the expansion of the ligand binding pocket.

Chapter 5

PRELIMINARY RESULTS FROM A PROPOSED *IN SILICO* ALGORITHM FOR IDENTIFYING STABILIZING POCKETS IN THE Y220C MUTANT OF THE p53 TUMOR SUPPRESSOR PROTEIN

5.1 Abstract

The p53 tumor suppressor protein is a DNA-binding protein that performs an important role in stimulating apoptosis and cell cycle arrest in the presence of various oncogenic factors. The function of p53 can be compromised by mutation, leading to increased risk of cancer; approximately 50% of cancers are associated with mutations to p53, and the majority of those are in the core p53 DNA-binding domain. The Y220C mutation of p53, for example, destabilizes the p53 core domain by 4 kcal/mol, leading to increased protein unfolding. The associated reduction in tumor suppressor functionality is associated with approximately 75,000 new cancer cases every year. It has been shown that destabilized p53 mutants can be 'rescued' and their function restored; binding a small molecule into a pocket on the surface of mutant p53 can stabilize the protein and allow it to resume its wild-type structure and function. Here we describe an *in silico* algorithm for identifying potential rescue-pockets, including the algorithm's integration with the Dynameomics molecular-dynamics data warehouse and the DIVE visual-analytics engine. We discuss the results of our method's application to the p53 Y220C mutant, describe a putative rescue pocket, and detail a subsequent search for stabilizing ligands to dock into the putative rescue pocket. We then compare our *in silico* rescue-pocket and rescue-ligand

results with published experimental data. Finally, we discuss the results of preliminary experimental testing of the putative rescue ligands.

5.2 Introduction

The p53 tumor suppressor protein (Figure 5.1) is a DNA-binding protein that initiates cell-cycle arrest and apoptosis in the presence of various oncogenic factors (Vogelstein et al. 2000). It is a well-studied cancer target as 50% of human cancers involve a mutation to this protein (Joerger et al. 2006) with the majority of those mutations located in the central DNA-binding domain. These mutations can be broken out into two general categories: "contact" mutations affect the DNA contacts, and "structural" mutations affect the protein's structural integrity (Cho et al. 1994). The p53 protein is already only moderately stable at physiological temperatures (Bullock et al. 2000) and mutations that affect its structural integrity can destabilize it further, causing it to unfold and lose its tumor-suppressing functionality.

It has been shown that destabilized p53 mutants can be 'rescued' i.e. re-stabilized such that they can resume their anti-tumor functionality; this has become a point of interest in the search for new cancer therapies. Experimental evidence indicates that mutants can be rescued both by specific rescue mutations (Brachmann et al. 1998) (mutations that stabilize, rather than destabilize, the protein) and by rescue ligands, small molecules that dock into pockets on the surface of the protein and stabilize the protein (Boeckler et al. 2008). However, it is not clear where on the protein these rescue-regions and rescue-pockets exist. Protein regions amenable to stabilization by a ligand have been uncovered by visual analysis, analyzing known stabilizing

ligands or stabilizing mutations, as discussed above, and, as shown by Wassman et al, by analyzing homologous proteins and associated stabilizing ligands (Wassman et al. 2013).

p53 stabilization is a validated approach to rescuing and reactivating p53 mutants, yet there is no general method for identifying rescue pockets that are capable of protein-stabilization, particularly if the pockets are not present in an experimental structure. Here, we propose an *in silico* algorithm for identifying potential rescue pockets given molecular dynamics (MD) simulation data. MD is an *in silico* technique that computationally simulates the forces involved in protein dynamics and produces time-series snapshots of protein structures, similar to frames in a movie. The MD data used in this work were part of the Dynameomics project, an ongoing molecular dynamics project designed to elucidate the unfolding pathways and native dynamics of all known protein folds (Van der Kamp et al. 2010; Beck et al. 2008).

Our hypothesis was that potential protein rescue-regions were those regions of the protein whose native contacts had become destabilized due to mutation. This hypothesis was supported and motivated by our original observations that the most disrupted region of the Y220C mutant simulations coincided with an experimental pocket capable of docking stabilizing ligands (discussed below). Using contacts as proxies for structure, and leveraging the basic tenants of protein structure → protein function, we postulated that re-asserting native-like contacts, possibly via ligand-mediation, could re-assert native-like function. We used MD simulation data and our in-house contact-analysis tool Contact Walker (Bromley et al. 2013) to compare the inter-residue contact data of mutant and wild-type p53. We then computationally identified regions of

the mutant protein that were destabilized relative to wild type and were therefore, if the destabilized contacts could be re-stabilized, putative rescue-regions.

Once a destabilized region was identified, it was necessary to find a pocket or crevice that permitted access to that region. Proteins are dynamic molecules and, as such, the surface topology of any given region changes continuously. Quantifying and analyzing this dynamic topology is a strength of MD. Because our primary data originated in MD simulations, we had access to hundreds of thousands of protein structures, each with multiple pockets. To make pocket-data analysis tractable, we stored the data in a pocket database and made them accessible from a front-end software tool.

To test our ability to find rescue pockets, we analyzed MD simulations of the core DNA-binding domain of the p53 wild-type and Y220C mutant proteins. The Y220C mutation was selected for analysis because experimental work has identified pockets on its surface that dock stabilizing ligands (Joerger et al. 2006; Boeckler et al. 2008); as a result, it was able to act as a positive control to validate our method's ability to predict and identify rescue pockets. After identifying a putative rescue pocket, we identified three putative rescue ligands and compared both the pocket and the ligands to experimentally-validated rescue pockets and ligands. Finally, we performed experimental testing on the selected ligands to determine 1) if any of the ligands were able to successfully interact with the putative rescue pocket and 2) if any of the ligands were capable of exerting a stabilizing effect on the p53 Y220C mutant. Although there are many examples of using MD simulations to study p53 (Calhoun and Daggett 2011; Basse et al. 2010; Wassman

et al. 2013; Lukman et al. 2013; Barakat et al. 2011), we believe that using multi-simulation contact-disruption analysis to identify rescue-regions and rescue-pockets is novel.

5.3 Methods

5.3.1 *Molecular Dynamics Simulations*

MD simulations were performed using *in lucem* molecular mechanics (*il*mm) (Beck et al. 2000-2014), our in-house modeling package, with the Levitt *et al* force field and established methods (Beck et al. 2000-2014; Levitt et al. 1995; Beck and Daggett 2004; Levitt et al. 1997). Starting structures of the wild-type p53 DNA-binding region were obtained using the Protein Data Bank (PDB) (Berman et al. 2000) crystal-structure 2ocj (Y. Wang et al. 2007) (2.05Å resolution, chain A). Structures for the Y220C and R175H mutants were created by *in silico* point mutation. As zinc has been shown to be significantly disassociated from p53 at physiological conditions as well as in the presence of destabilizing mutations (Butler and Loh 2003; Loh 2010), we focus primarily on the 310K zinc-free (*apo*) simulations here. However, we also simulated the *holo* wild type at 298K for experimental comparison. Each simulation was performed in triplicate, in explicit solvent and at neutral pH (negative Asp and Glu, positive Arg and Lys, and neutral His). The *apo* 310K simulations were performed for at least 51 ns and the *holo* 298K wild-type simulations were performed for at least 100 ns.

First, models were minimized for 1000 steps (1 step = 2 fs) using steepest-descent minimization. Models were placed in a periodic box with walls at least 10Å from all protein atoms, and then solvated using the F3C water model. Solvent density was 0.993 or 0.997 g/mL, the experimental values for water density at 310K or 298K, respectively (Kell 1967). Water

minimization followed for another 1000 steps, and was then followed by water-only dynamics at the simulation temperature for another 500 steps. This was followed by 500 steps of solvent minimization and then followed again by 500 steps of minimization of both the solvent and the protein.

Initial atomic velocities were assigned from a low-temperature Maxwellian distribution, after which the temperature was increased to 298K or 310K, appropriately. The non-bonded cutoff was set to 10Å and the interaction list was updated every 2 steps. The number of particles, energy, and volume was fixed using the NVE microcanonical ensemble. Structures were saved every 500 steps (1 ps). The p53 DNA-binding domain is an immunoglobulin-like β-sandwich fold, which is represented by Rank 1 of our Consensus Domain Dictionary (CDD) (Day et al. 2003; Schaeffer et al. 2011).

5.3.2 *Simulation Analysis*

Contact analysis, Cα root-mean-squared-fluctuation (RMSF), average properties, and other aggregate analyses were performed over the last 35 ns of the 310K *apo* simulations to allow for equilibration. The 298K *holo* wild-type simulations were slower to equilibrate and NOE and SASA data for comparison with experiment were calculated over the last 25 ns of the 100 ns simulations.

Structure alignment for Cα root-mean-squared-deviation (RMSD) and Cα RMSF analysis used the β-strand residues. Contact analysis was heavy atom-only and used a 5.4Å cutoff for carbon-carbon contacts and a 4.6Å cutoff for all other contacts. Only one inter-atomic contact was required for residue:residue contact and contacts between adjacent residues were skipped.

Secondary-structure comparison in the pocket search tool used DSSP (Kabsch and Sander 1983) secondary-structure assignment. Images were created using Pymol (DeLano 2002) and Accelrys Discovery Studio 3.5 Visualizer (http://accelrys.com/products/discovery-studio/visualization.html).

5.3.3 *Pocket Database and Search Tool*

Initial pocket-finding was performed at 1 ps granularity on the 310K *apo* simulations of p53 wild type and the Y220C mutant; pocket finding was performed using the EPOS BALLPass software (http://gepard.bioinformatik.uni-saarland.de/software/epos-bp). To make this data set more usable, the pocket data were loaded into a relational database. This database first cleans the data and then organizes them in a pocket-centric manner. Each pocket is identified with a unique number, associated with pocket properties such as volume and polarity, and linked via simulation identifier, structure identifier, and simulation time step back to the original Dynameomics data warehouse (Beck et al. 2008; Van der Kamp et al. 2010; Simms et al. 2008).

A second set of data tables, linked by the unique pocket identifier, details which atoms are involved in each pocket. These data include each atom's name and type, containing residue number and type, and flags indicating if an atom is a heavy atom and/or if it is part of the residue main chain or side chain. Finally, the atoms are assigned atom identifiers from the original Dynameomics MD trajectories, allowing each atom and residue to be analyzed in its original dynamic context, and in association with the standard Dynameomics analyses such as solvent accessible surface area (SASA), Cα RMSF, and dihedral angle analysis. The database was built using Microsoft SQL Server (www.microsoft.com/sql).

A search-tool based on the DIVE visual analytics system (Bromley et al. 2014; Rysavy et al. 2014) was built to provide a front-end user-interface to the pocket database. Given one or more simulation identifiers, one or more residue numbers, and optional minimum and maximum pocket volumes, the tool searches the pocket database and returns pockets from the given simulations whose pocket-lining atoms belong to the specified residues and whose volumes lie within the specified volume-range. It then aligns and clusters these pockets based on heavy-atom RMSD of the specified search residues and produces a hierarchical-clustering dendrogram plot, useful for identifying inter-pocket structural-similarities. Experimental data can be integrated into this search by specifying one or more external Protein Data Bank (PDB) files. These structures are aligned and clustered along with the dynamic pockets, allowing researchers to identify the dynamic pockets whose structures most closely match known experimental pockets.

The search tool can also optionally output docking files suitable for analysis by Autodock Vina (Trott and Olson 2010). These files include the protein structures converted to the required file format, optional flexible side chains, and a configuration file specifying the search box around the selected pocket. The only remaining information necessary for performing Vina docking is a ligand structure file. Finally, the search tool outputs a comma-separated-value (CSV) file detailing the pockets in each cluster and where the pocket structures can be found in the database, in the Dynameomics data warehouse, and/or on disk. The pocket search-results also include the pocket volumes, the time in the trajectory at which they occurred, their degree of

secondary-structure overlap with the simulation starting-structure, and the total SASA of the specified residues. This pocket analysis pipeline is illustrated in Figure 5.2.

5.3.4 *Contact Analysis*

Searching the pocket database requires a list of pocket-associated residues. In the presence of experimental data, these residues can be specified by analyzing which residues make contact with a particular ligand, for example, or by identifying the residues that line a known pocket cavity. In the absence of experimental data, however, pocket residues must be identified in some other manner.

The Contact Walker tool (Bromley et al. 2013) was originally designed to identify regions of a protein that had become destabilized by mutation. Briefly, it uses molecular-dynamics trajectory data and compares the inter-residue heavy-atom contact-occupancy between wild type and mutant proteins. Contact Walker then produces a connected-graph diagram that shows which contacts have become disrupted as a result of the mutation and whether the disruption stabilized them (increased occupancy) or destabilized them (decreased occupancy). Contact Walker can aggregate data from multiple simulations to show the minimum demonstrated occupancy change between wild type and mutant simulations i.e. for each contact, it shows the most conservative estimate of occupancy change.

5.3.5 *Ligand Docking*

Ligand docking was performed using Autodock Vina version 1.1.2 and UCSF Dock Blaster (Irwin et al. 2009). UCSF Dock Blaster uses UCSF DOCK (Lorber and Shoichet 1998; Lorber

and Shoichet 2005) version 3.6 to perform high-throughput virtual screening of the entire ZINC (Irwin and Shoichet 2005) 'leads-now' database into a user-specified pocket. At the time of this writing, the ZINC 'leads-now' database contains 2,395,805 lead-like molecules ('lead-like' indicates a molecular weight between 250 and 350 g/mol, xlogp ≤ 3.5, and 7 or fewer rotatable bonds). The Dock Blaster service returns the top 500 docking hits. Molecule file conversion was performed using Autodock Tools (Morris et al. 2009) and OpenBabel (O'Boyle et al. 2011). Ligand-protein polar contacts were identified using Pymol with a cutoff of 3.3Å (Dock Blaster default).

5.3.6 *Small Molecule Probing*

Small-molecule energy probing was used to determine if the selected pocket would admit a benzene moiety into the pocket volume vacated when Y220 was mutated away. Minimization and energy calculations for small-molecule probing was performed *in vacuo* using the *il*mm molecular dynamics package (Beck et al. 2000-2014) and the Levitt et al force field (Levitt 1990; Levitt et al. 1997). The specific probe algorithm has been described previously (Bernard 2006). Briefly, benzene molecules were placed in a regular 1Å grid around the pocket residues; grid points that were more than 4.2Å away from a protein residue or closer than 2.4Å to a protein residue were discarded. At each probe point, the benzene molecule was rotated evenly around a sphere to calculate energies in different positions. For each rotation, steepest-descent minimization was performed for 10 steps. The probe location was reported as the geometric center of the benzene atoms.

5.3.7 *NOE comparison*

Nuclear Overhauser Effect (NOE) analysis was performed using 298K NMR data (PDB code: 2fej (Cañadillas et al. 2006)). An NOE was satisfied if the average inter-hydrogen d^{-6} distance was less than 5Å or the distance in the NOE constraint file, whichever was larger.

5.4 Results and Discussion

5.4.1 *Comparison of 298K holo simulations with experiment*

Because experimental structural data are not available for 310K *apo* p53, we validated our *in silico* protein system by simulating the p53 *holo* wild type at 298K and comparing those simulations to experimental 298K *holo* NMR data. The specifics of this comparison are detailed in Chapter 6. In summary, native structures were maintained throughout the three simulations and no unfolding was observed. β-structure in the three-residue strands S1 and S5 was fragmented or lost in all three simulations, one simulation adopted α-sheet secondary structure (Daggett 2006) between strands S3 and S8, and S6 β-structure was lost in one simulation. In every case, however, the protein main-chain retained its basic structure and no significant restructuring occurred. H1, H2, and the remaining β-strands were all maintained throughout the simulations. The largest consistent deviations from starting structure occurred in the S7/S8 loop. Wild-type 298K *holo* simulations satisfied 89% of published 298K *holo* NOEs and 45% of residue pairs containing an NOE violation also contained at least one NOE satisfaction. Comparison of the solvent accessible surface area (SASA) of the experimental NMR structures and the wild-type simulations yielded a correlation coefficient of $R = 0.79$.

5.4.2 *Analysis of 310K apo simulations*

The average Cα Root-Mean-Squared Deviation (RMSD) from starting structure for the three 310K *apo* wild-type simulations were 3.7 ± 0.3Å, 3.9 ± 0.3Å, and 3.6 ± 0.4Å. The per-residue Cα RMSD and Cα RMSF values are shown in Figure 5.3. The primary regions of deviation from the starting structure were the loop-sheet-helix region, L2 and L3, particularly around the zinc-binding residues, and the S7/S8 loop. Cα RMSF followed a similar pattern with the largest fluctuation values demonstrated in the loop-sheet-helix region, L2, L3, the S7/S8 loop and the S6/S7 loop. Secondary-structure analysis indicated that while most structures in the larger S6/S7/S4/S9/S10/S2'/S2 β-sheet were maintained, the smaller S1/S3/S8/S5 β-sheet demonstrated loss of structure, as did the H1 helix. In one simulation, S1 and S3 demonstrated adoption of the α-sheet secondary structure, which has been proposed to play a role in protein aggregation (Armen, Alonso, et al. 2004; Daggett 2006), and the L3 loop gained helical content around residue G245. In another simulation, the region around H168 gained helical structure, although it should be noted that the crystal structure of this region contains enough helical content that some software packages characterize it as helical and others do not. In a third simulation, distance measurements between the Cα atoms of the K120 and R280 DNA contacts indicated that L1 separated from H2 by 22Å, disrupting the loop-sheet-helix DNA-binding region.

5.4.3 *Validation of Search Tool Functionality*

The pocket search tool was designed to search a database and recover pockets with certain structural characteristics such as volume and residue composition (i.e. certain residues play a role in the structure of the pocket). To validate the search-capabilities of the tool, we

searched the Y220C simulations for a pocket similar to the experimentally-validated ligand-docking pocket found in the x-ray crystal structure of the p53 Y220C mutant (PDB: 4AGQ (Wilcken et al. 2012)). If our MD simulations contained the pocket, the pocket search tool was expected to recover it given the experimental input. In the stabilized crystal-structure, residues 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 220, 221, 222, 223, 228, 229, and 230 were identified as having heavy atoms within 5.4Å of the stabilizing ligand. We directed our search tool to find simulation pockets involving these residues. The pocket search tool discovered 170 MD structures that contained a pocket matching these specifications. Approximately 150,000 structures and 2 million pockets were searched to arrive at this set. The final pockets were all less than 3Å heavy-atom RMSD from the experimental crystal structure (RMSD was calculated using heavy atoms from the 18 specified residues). Figure 5.4 shows a selection of the discovered pockets.

Our original hypothesis was that Contact Walker could identify potential rescue-regions by identifying native contacts that had become destabilized. There has been a great deal of work done analyzing the Y220C pocket originally published by Joerger *et al* (Joerger et al. 2006)*;* indeed, it was analysis of this pocket that initially motivated our hypothesis. More recently, a rescue-pocket for the p53 R175H mutant was identified by Wassman *et al* (Wassman et al. 2013), providing us an opportunity to further test our hypothesis. Wassman *et al* used both experimental and *in silico* techniques to identify a region of the protein around residues L114 and C124 that had stabilizing potential. They then went on to show that stictic acid was capable of docking into a pocket in that region, increasing the melting temperature of the R175H mutant by 4.6° C.

To test our hypothesis that Contact Walker could identify rescue regions by their destabilized contacts, we analyzed three 310K *apo* simulations of the R175H mutant. By comparing these mutations to wild type, Contact Walker highlighted those regions of the mutant protein that were consistently disrupted across the ensemble of simulations. Contact Walker indicated that the most consistently disrupted contacts in the R175H simulations were L114:T125 and L114:Y126. As shown in Figure 5.5, these contacts and residues are immediately adjacent to the rescue-pocket identified by Wassman *et al*.

5.4.4 *Y220C Mutant Simulation Analysis*

The average Cα RMSD values for the three 310K *apo* Y220C mutant simulations were 3.8 ± 0.3Å, 3.2 ± 0.2Å, and 3.5 ± 0.2Å. The average per-residue Cα RMSD and Cα RMSF values are shown in Figure 5.3. Like the wild type, the Y220C mutant demonstrated the largest deviations from starting structure in the loop-sheet helix region, L2, L3, and the S7/S8 loop. Also like the wild type, Cα RMSF values followed the same pattern as Cα RMSD with the largest fluctuations being demonstrated in the loop-sheet-helix region, L2, L3, the S7/S8 loop, and the S6/S7 loop. Relative to wild type, one Y220C simulation also demonstrated a slight increase (1Å) in fluctuation in the S5/S6 loop. Secondary-structure analysis showed a larger departure from native secondary structures than was demonstrated by wild type. Like wild type, most of the larger S6/S7/S4/S9/S10/S2'/S2 β-sheet was well maintained. However, one simulation showed novel helical-content in the L1 loop, disrupting the loop-sheet-helix region, another simulation showed loss of β-structure in S1 and S5 and an increase in helical structure around H168, and a third simulation showed adoption of the α-sheet secondary structure (Armen,

Alonso, et al. 2004; Daggett 2006) between S1, S3 and S8. One simulation showed a 25Å separation between L1 and H2 (distance measured between the Cα atoms of DNA-contacts K120 and R280).

Contact Walker analysis of the p53 Y220C mutant is shown in Figure 5.6. To illustrate reading a Contact Walker diagram, this figure indicates that L145, V147, P151, C220, P223, L257, and L265 (magenta circles) were all highly destabilized. It shows that C220 lost contact with V147 (magenta line) and that P151 lost contact with L257 and L265 (magenta lines) but gained alternative contacts with T230 and E221 (green lines). Note that E221 is colored gray despite gaining two alternative contacts. This is because E221 has lost very little contact-occupancy (this property-coloring scheme can be controlled via DIVE 'micro-scripting' (Rysavy et al. 2014)). We are focusing here on contact loss because our work focused on stabilizing native contacts, not destabilizing non-native contacts. Note also that this figure shows the *minimum demonstrated occupancy difference* among three wild-type simulations and three mutant simulations. Thus, if a given contact has occupancy values of 10%, 20%, and 30% in the wild-type simulations and occupancy values of 70%, 80%, and 90% in the mutant simulations, the reported contact difference will be 30% - 70% = -40%, or a net mutation-associated stabilization of 40% occupancy. This difference value represents the most conservative estimation of disruption possible given the demonstrated occupancies.

5.4.5 *Pocket Selection Algorithm and Application*

As discussed above, Contact Walker identified the most disrupted region of the protein as the region near the C220 mutation site (Figure 5.6). After analyzing molecular-dynamics data

for this region, the residues L145, V147, P151, C220, L257, and L265 were selected to identify potential rescue pockets. While this is the region that was discussed earlier as correlating with a known rescue pocket, this analysis was performed in the absence of experimental data, using only inter-residue contact-information from molecular dynamics simulations. Pockets containing these six residues were present 1.5% of the simulation time.

Querying the pocket database for pockets in this region with volumes > 600 $Å^3$ returned 98 potential rescue pockets, a four order-of-magnitude data-reduction from the more than two million pockets stored in the pocket database. These 98 candidate pockets were then manually inspected and filtered based on the following criteria:

1. The pocket needed to provide a space in which an aromatic ring, positioned similarly to the tyrosine benzene-moiety lost by the mutation, could dock. Essentially, we wanted to provide an opportunity to replace what was lost without introducing additional foreign material.
2. The pocket should not be overly invasive. The mutation site and the lost benzene moiety lie near the surface, so replacing what was lost should not require binding a ligand deep into the hydrophobic core. It was the core that we were trying to stabilize and injecting a large foreign presence could easily introduce non-favorable contacts.
3. The protein conformation containing the pocket needed to retain as much wild type-like structure as possible, particularly around the hydrophobic core. We were trying to rescue a protein from unfolding, so we wanted the hydrophobic core of the stabilized conformation to be as wild type-like as possible. As discussed previously by Joerger *et al*

(Joerger and Fersht 2010), ligands that preferentially stabilize the native state over the denatured state should drive the unfolding equilibria toward the native state.

This entire process is shown in Figure 5.7 and the final selected putative rescue-pocket is shown in Figure 5.8. This pocket met the basic criteria of being in the disrupted region, providing a solvent-accessible opportunity to introduce a tyrosine-like benzene moiety, and being located in a protein structure with a relatively intact beta-core. Small-molecule energy probing indicated that the rescue pocket could accommodate a benzene moiety in the desired position (Figure 5.9).

Dihedral-angle analysis as well as visual inspection indicated that the pocket-lining residues did not undergo large conformational changes over the 1 ns window surrounding the rescue pocket. The residues with the most side-chain movement were S149, C220 (the mutation site), and E221. S149 and E221 both faced outwards into the solvent, but C220 was not solvent accessible aside from the putative rescue pocket. Cα RMSF analysis of the rescue-pocket simulation indicated that the motion of the pocket-lining residues all fell within wild-type ranges with the exception of H115 which exhibited approximately 1.7Å larger Cα RMSF than wild type. Analysis of these residues in the other two Y220C simulations indicated similar wild type-like stability, although in one simulation residues V225 and C229 exhibited Cα RMSF values between 1 and 2Å greater than wild type.

5.4.6 *Ligand Selection*

To confirm that the selected pocket had rescue-potential, it was necessary to confirm that it could interact with a stabilizing ligand. Therefore, once the putative rescue pocket was selected, we identified ligands that could potentially dock into it and stabilize the protein. To begin this search, we docked the entire ZINC (Irwin and Shoichet 2005) 'leads-now' database containing 2.4M ligands into the rescue pocket using the UCSF Dock Blaster (Irwin et al. 2009) service. After initial docking, the top 500 ligands were manually inspected for two different characteristics:

1. Ligands needed to have a benzene moiety in roughly the position where the wild-type tyrosine would be. This was in keeping with the goal of simply replacing what had been removed by mutation rather than attempting to bind more aggressively and risk introducing new structural artifacts. Large ligands were not perceived as necessary to stabilize a protein; for example, the N239Y rescue mutation rescues the G245S mutation by introducing a tyrosine side-chain that stabilizes the local hydrophobic packing (Joerger et al. 2004). Such a side chain is similar in scope to a small-molecule stabilizing ligand.
2. Ligands needed to support a strong contact to L145. Originally, this was not a required criterion. However, several of the final 500 ligands contained such a contact and thus it became an option in the refinement process. The L145 side-chain position within the rescue pocket was wild-type like and thus a contact in that position was expected to not only help secure the stabilizing ligand in place, but also help stabilize L145, part of the β-core, in a wild type-like position.

Of the 500 ligands returned by Dock Blaster, only three ligands met all of these criteria. For additional docking-validation, these three ligands were re-docked into the putative rescue pocket using Autodock Vina. Our docking energy point-of-reference was -6.5 kcal/mol, calculated by using Autodock Vina to dock the known rescue ligand PhiKan5196 (Wilcken et al. 2012) into its holo crystal position (PDB: 4AGQ). Vina energy estimates of the Dock Blaster poses were -5.2 kcal/mol for ZINC 67692870, -5.5 kcal/mol for ZINC 95476490 and -4.4 kcal for ZINC 19565304. For this last ligand, Vina found a -6.0 kcal/mol pose that was very similar to the Dock Blaster pose with comparable positioning of the benzene moiety and contact with L145. This process is illustrated in Figure 5.10.

A histogram of the overall binding-energy distribution is shown in Figure 5.11 with an inset showing only the favorable (negative) binding energies. The two ligands with the least favorable energies reported by Dock Blaster were ZINC 3904140 (+2.2M kcal/mol) and ZINC 6653655 (+45K kcal/mol). The three selected rescue-ligands described above had negative potential energies three and four standard deviations less than the mean negative binding-energy (-22.5 ± 3.6 kcal/mol). 99% of compounds were able to favorably bind somewhere in the vicinity of the selected pocket. However, re-docking of the two worst binders with Autodock Vina indicated that simple negative energies did not necessarily imply successful pocket-docking; sometimes, ligands would dock only peripherally at the surface or in small adjacent clefts that inadvertently fell into the specified rectangular docking-volume. Both of these scenarios placed the ligands far away from the intended docking site. More specifically, docking conformations such as these placed the ligands far away from the destabilized residues that were

hypothesized to provide stabilizing contacts. These observations underscored the notion that understanding *which* residues the ligands should contact and *why* the ligands should contact them was an important part of our pocket-selection algorithm and the subsequent ligand selection.

5.4.7 *In silico Analysis and Comparison to Existing Experimental Data*

The three final ligand picks are illustrated in Figure 5.12. As discussed above, both the pocket and the selected ligands were discovered using strictly *in silico* methods. However, the final rescue pocket and the final ligand selections had considerable structural and pharmacophore similarities to experimentally-validated rescue pockets and ligands. Given these similarities, it is worth comparing the selected pocket and ligands with their experimental counterparts.

To begin, the selected pocket was in the same region that is stabilized by the PhiKan5196 ligand. Indeed, it was the colocation of the PhiKan5196 docking site and Contact Walker's analysis of disrupted contacts that motivated our disrupted-region-as-rescue-region hypothesis. Despite this location overlap, however, the actual surface-accessible pocket geometries were different. Consider the two ligands illustrated in Figure 5.13. Both ligands interacted with the same region of the protein (i.e. the disrupted region specified by Contact Walker), but they did so from pockets with different surface geometries; the PhiKan5196 pocket contained a long surface trench with a subsurface cavity (Figure 5.13a) and the putative rescue pocket selected by Contact Walker and the pocket database (Figure 5.13b) was primarily an extension of a similar subsurface cavity, but without the trench.

Despite these pocket differences, the ligands themselves had an overlap of pharmacophore groups, notably the Y220 'replacement' aromatic group and a strong contact to

L145 (Figure 5.12 and Figure 5.13). In addition to being present in validated rescue-ligands, both of these features as well as the polar contacts near D228 (Figure 5.12) have been shown in fragment-binding experiments (Basse et al. 2010) to promote docking. PhiKan5196 extends further in the direction of the C220 mutation site than any of the putative rescue ligands, which may contribute some degree of stability. However, PhiKan083 (Boeckler et al. 2008), a stabilizing molecule docked into this same mutant (PDB: 2VUK), is not extended in this manner, suggesting that such an extension is not necessary for protein rescue. Similarly, PhiKan083 does not support a strong contact with L145, a feature that was present in other known rescue ligands and each of the suggested ligands. Table 5.1 compares the ligand contacts of PhiKan083, PhiKan5196, and the three putative rescue ligands (heavy-atom only, 5.4Å cutoff). Finally, the side-chain conformation of the six selected disrupted residues in the 2VUK crystal structure was similar to that of the selected rescue pocket (Figure 5.14), suggesting that the pocket selected by the pocket search tool presented a docking opportunity similar to that of a known rescue pocket.

5.4.8 *Results of Experimental Testing*

The three putative stabilizing compounds, as well as the two worst dockers (used here as decoys) were experimentally tested for their abilities to stabilize the Y220C mutant (experimental testing and data courtesy of Dr. Matthias Bauer and Dr. Alan Fersht). Initial thermal shift assay results showed that one of the putative stabilizing ligands (ZINC: 95476490) had a small and potentially dose-dependent stabilization of the Y220C mutant (Figure 5.15, highlighted box). *In silico* analysis of ligand 95476490 showed two polar contacts with D228 as well as contacts with both Q110 and H115 (Figure 5.12 and Table 5.1). The only other

compound to show a positive response was one of the decoy compounds, which showed less dose-dependent behavior than did the putative stabilizing ligand. $^{1}H/^{15}N$ HSQC NMR analysis of compound 95476490 showed slight chemical shifts, and initial peak assignments indicated that the chemical environments near several residues in the putative rescue pocket were altered in the presence of the ligand (Figure 5.16).

5.5 Conclusions

Preliminary experimental results were not sufficient to show conclusively that our algorithm was effective. Aspects of the results were encouraging, however, and, given that the preliminary data show the basic hallmarks of stabilizing ligands – correct interaction location, positive stabilization, and dose-dependent action - future refinement and application may achieve more substantial results. Our algorithm aims to select small-molecules whose contacts will mimic, or at least serve as proxies for, lost native-contacts. In some cases, this may not work; for example, some mutations alter DNA-contacts, while in others small→large mutations can cause repacking around a buried residue. However, absent these or similar conditions, our hypothesis is that re-establishing native-like contacts may help re-establish native-like functionality.

In silico analysis of the putative rescue pocket and ligand showed a considerable degree of overlap with published experimental evidence. Although experiment is the final assessment, multiple aspects of structural and pharmacophore agreement with well-established *in silico* tools suggest that the algorithm can successfully produce results that fall within these tools' predictive ranges. This suggests that the next step is to either refine our pocket and ligand selections using

additional tools or criteria, or to sample the ligand space more extensively by testing more ligands.

The goal of developing this algorithm was to provide a general means of discovering drug leads for protein targets that are not amenable to structural characterization. There is a need for such a method, underscored by recognizing that p53 plays a crucial role in cancer biology, yet only a few experimentally-discovered structures exist. Fundamentally, our algorithm was intended to find stabilization pockets; confirmation of the stabilizing potential of a pocket can only truly be assessed, however, by confirming that it can interact successfully with a stabilizing ligand. Thus, we conclude that this algorithm holds potential, but is in need of more extensive testing and future refinement.

Table 5.1 Ligand/protein heavy atom contacts.

	109	110	115	145	146	147	148	149	150	151	152	153	154	155	220	221	222	223	227	228	229	230
2VUK (PhiKan083)	X			X	X	X			X	X					X	X	X	X	X	X	X	X
4AGQ (PhiKan 5196)				X		X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X
ZINC 67692870				X	X	X			X	X					X	X	X	X	X	X	X	X
ZINC 95476490		X	X	X	X	X	X		X	X					X	X	X	X	X	X	X	X
ZINC 19565304				X	X	X	X		X	X					X	X	X	X		X	X	X

Ligands are listed down the left and residue numbers are listed across the top. The mutation site is highlighted in magenta. Contacted residues are marked with an 'X' and highlighted in pink.

Figure 5.1 The DNA-binding core domain of the p53 tumor suppressor protein
The DNA-binding region is shown at the top and the Y220C mutation site discussed herein is shown at the bottom of the beta-sandwich structure.

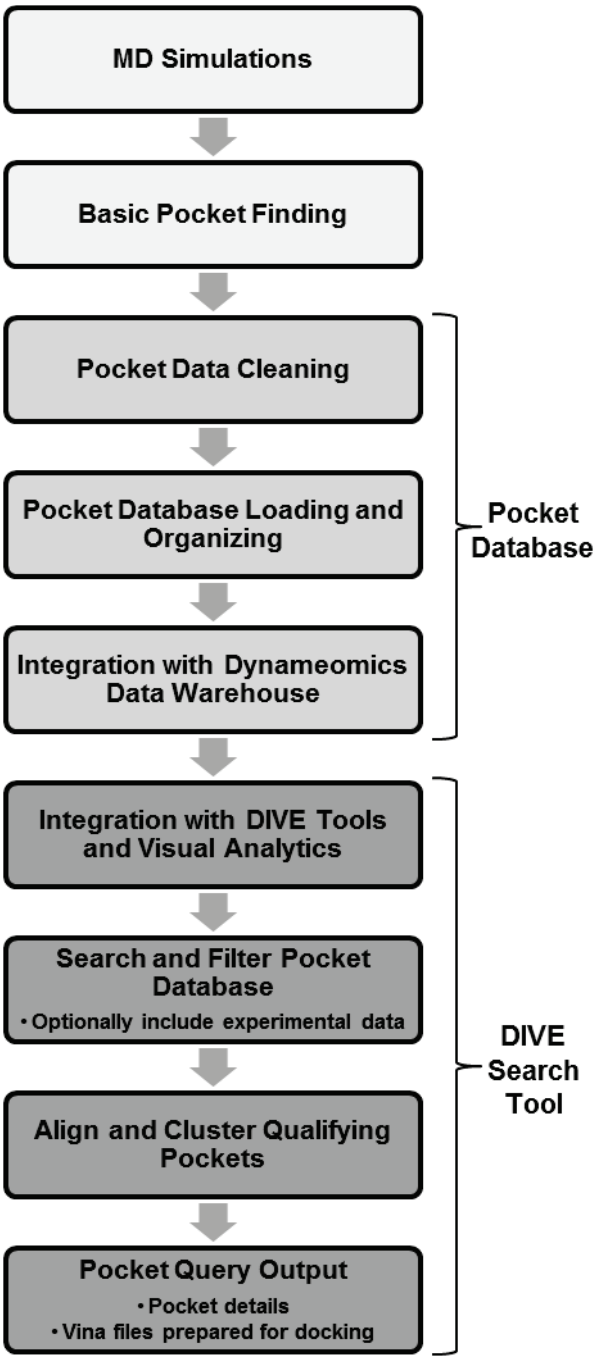


Figure 5.2 Overview of pocket discovery pipeline

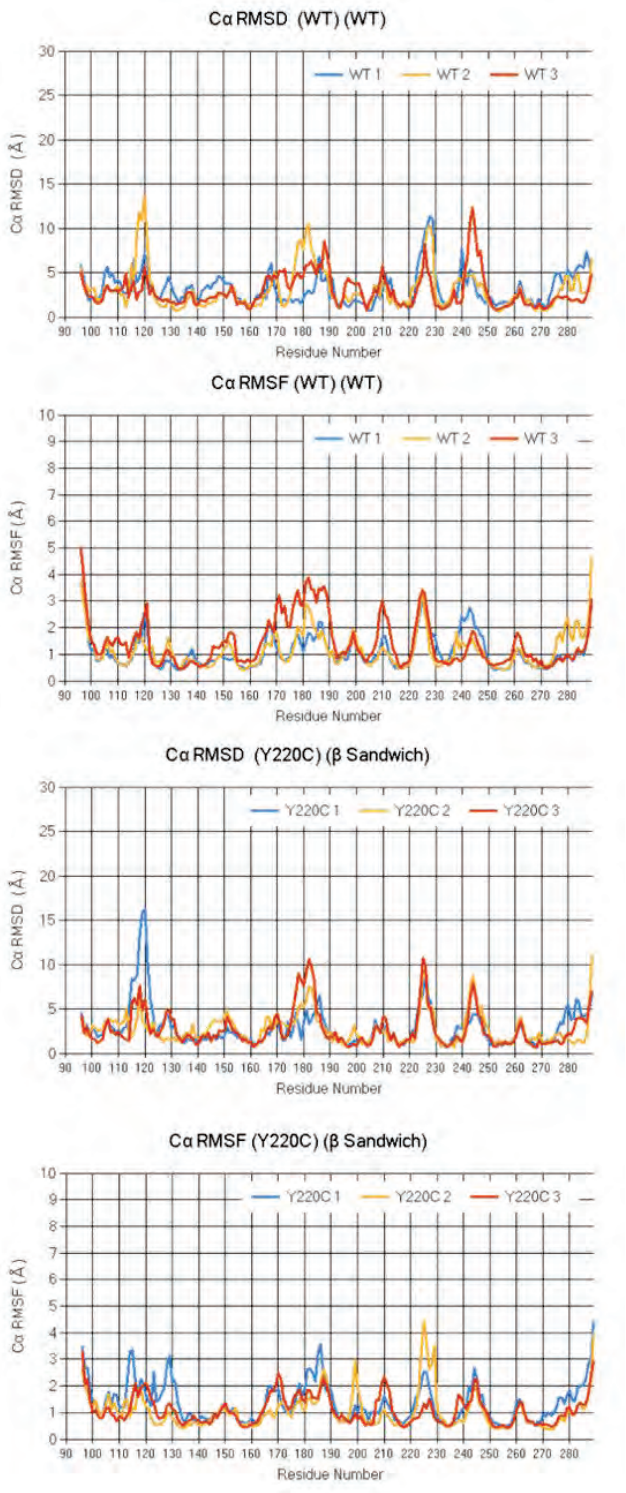


Figure 5.3 Per-residue average Cα RMSD and Cα RMSF of the p53 wild type and Y220C mutant.

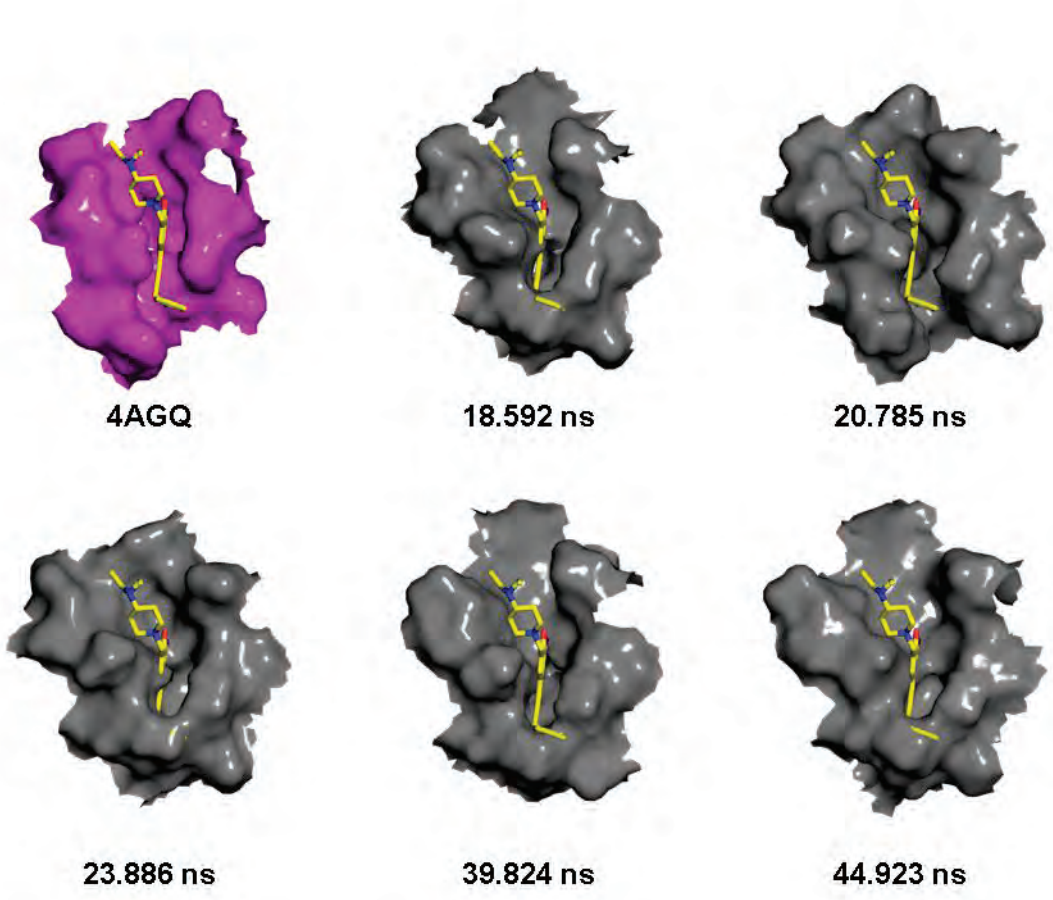


Figure 5.4 *in silico* search tool recovery of experimental stabilizing pocket
Experimental structure (magenta) and five recovered MD pockets (gray). The experimental ligand is overlaid on each structure for orientation. Simulation time points are shown for the MD structures.

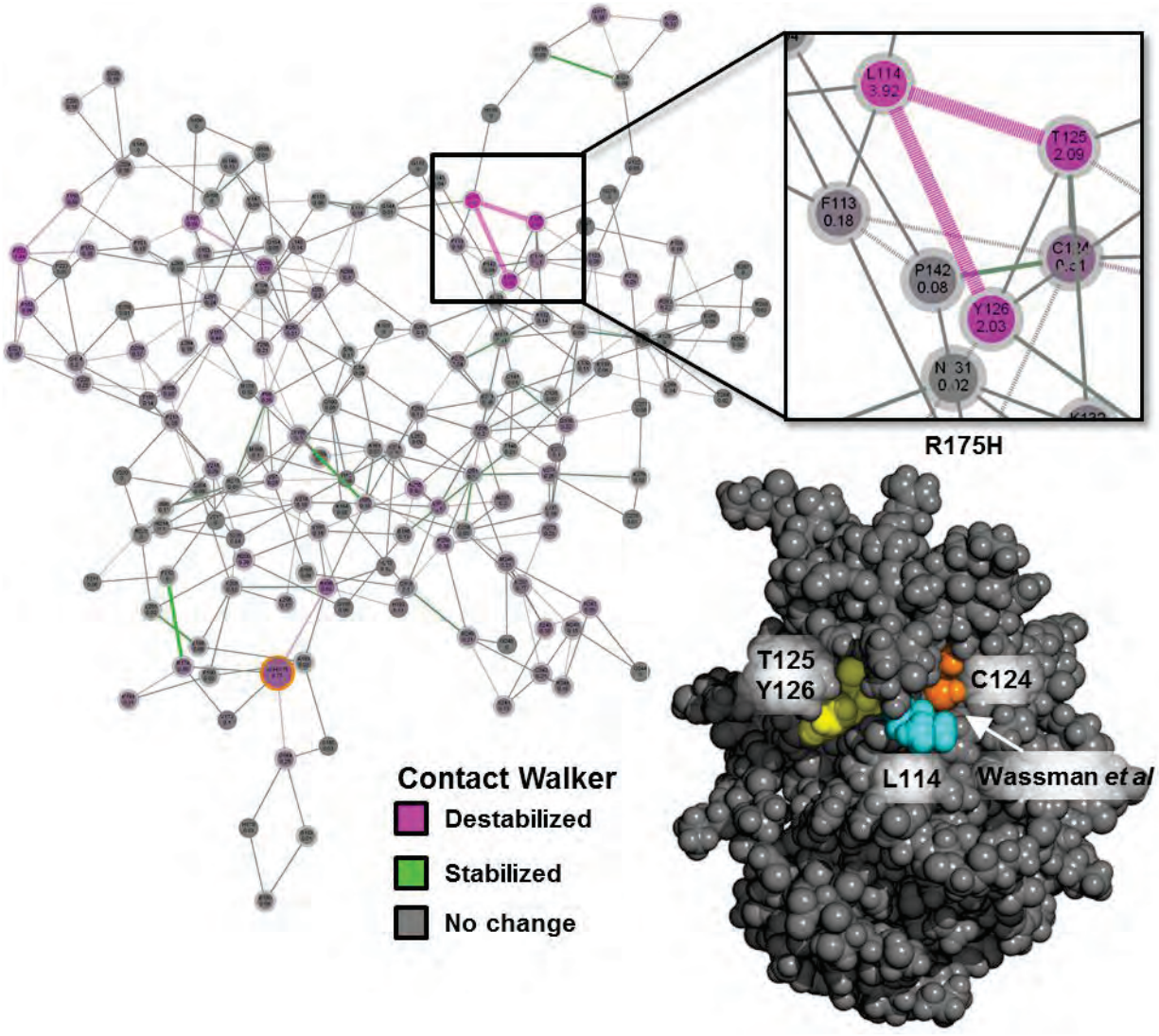


Figure 5.5 R175H destabilized region
Upper left: Contact Walker analysis of R175H indicated that L114 had lost contact with T125 and Y126 (magenta lines) and that this region was a potential rescue region. Right: Protein structure published by Wassman *et al* showing putative destabilized region and R175H stabilization pocket.

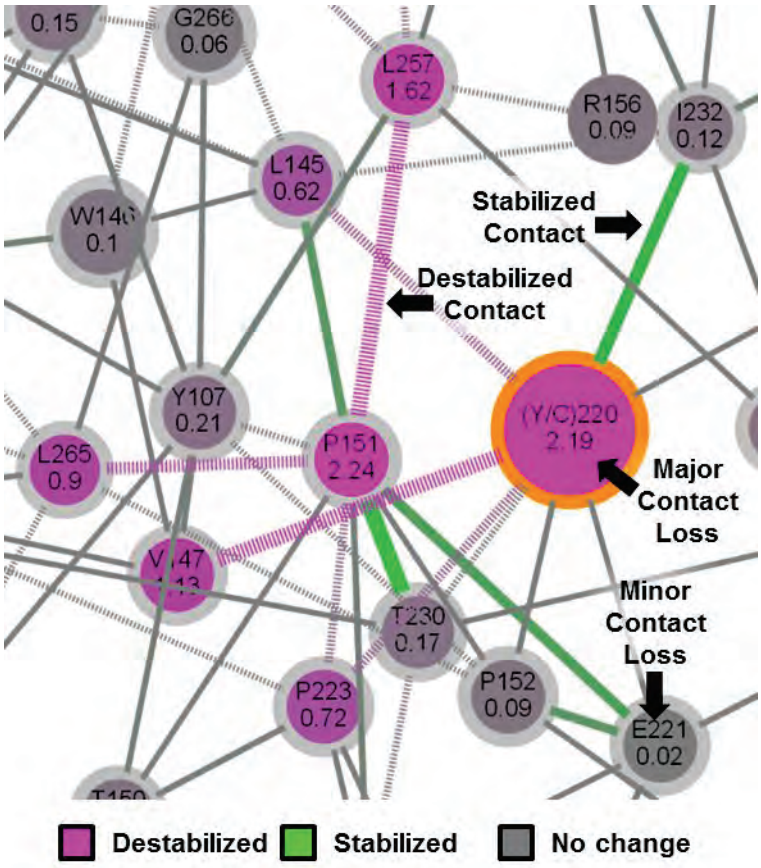


Figure 5.6 Example Contact Walker diagram for the p53 Y220C mutant showing mutation-associated contact disruptions

Residues are represented by circles and inter-residue contacts are represented by lines. The orange circle indicates the mutation site. Only peptide bonds and highly disrupted non-bonded contacts are shown. Contact color indicates destabilization (magenta), stabilization (green), or no change (gray). Contact line-width indicates disruption magnitude. Residue color indicates cumulative contact-occupancy loss; magenta indicates major occupancy loss, gray indicates minor occupancy loss. All color ranges are scaled linearly between the minimum and maximum values for each protein analyzed. Residue name, number, and occupancy-loss value is notated within each residue. This diagram incorporates three wild-type simulations and three Y220C mutant simulations. As such, each contact shows the *minimum demonstrated occupancy change.*

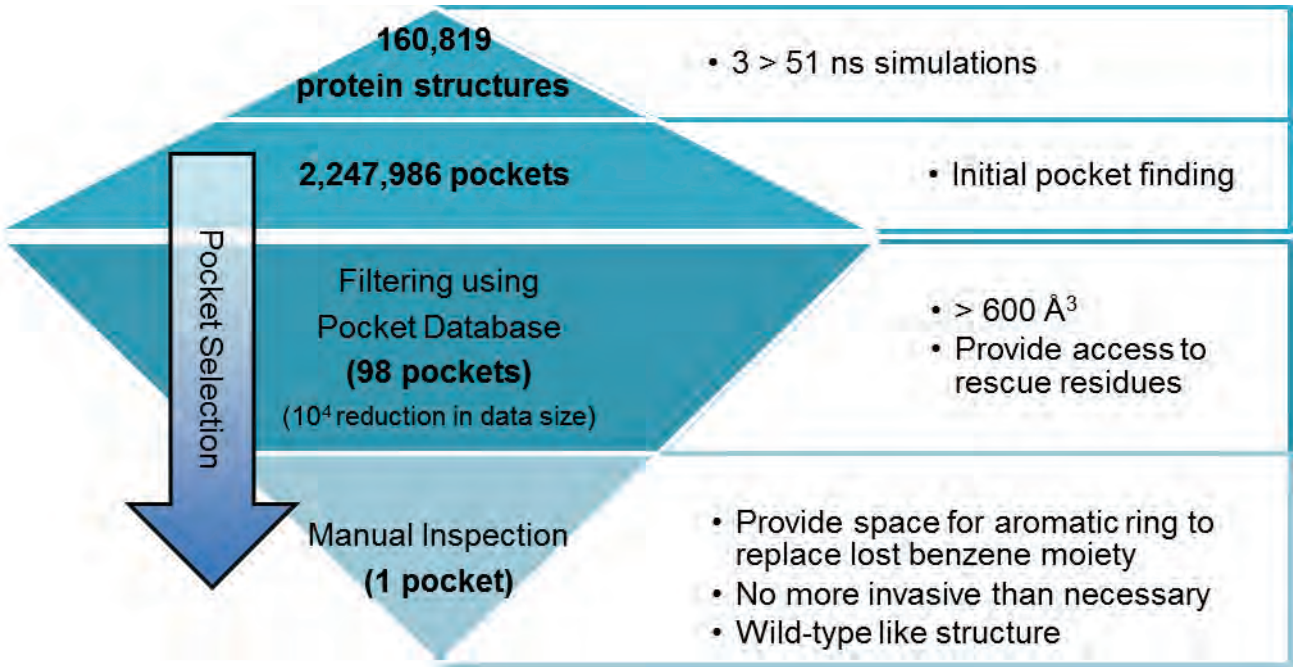


Figure 5.7 Pocket search and refinement process
More than 160K protein structures resulted in over 2.2 million pockets. Use of the pocket database and pocket search tool reduced the search space to 98 pockets. These 98 pockets were manually analyzed and a final rescue pocket was selected.

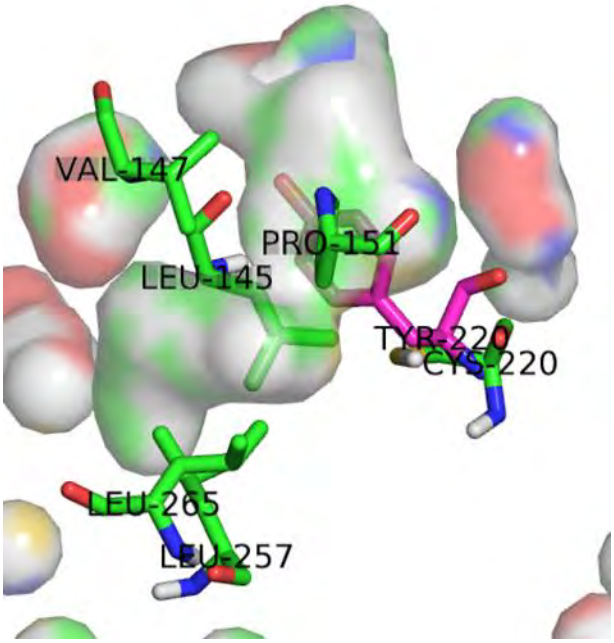


Figure 5.8 Final pocket selected from the 98 candidate pockets
Disrupted residues are shown in green, aligned wild-type Y220 is shown in magenta. Pocket was solvent-accessible from the top, and the middle of the pocket lay roughly adjacent to the wild-type Y220 benzene moiety.

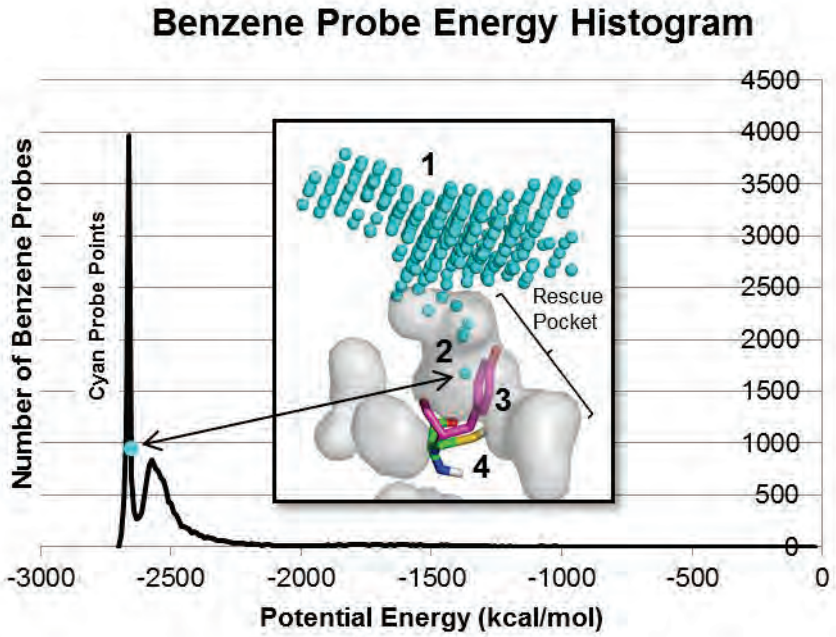


Figure 5.9 Histogram showing benzene probe points with negative energies
Energy profile was a bimodal distribution; probe points from the large peak on the left are shown as cyan spheres in the inset image. (1) Surface probe points. (2) Internal pocket probe points. The arrow indicates where a particular probe point lies on the energy spectrum. This probe point was in the vicinity of the wild-type tyrosine that was lost to the Y220C mutation. The presence of this probe point indicated that benzene had both the spatial freedom and energetic ability to bind into the pocket volume vacated by the benzene moiety of the wild-type tyrosine. (3) Wild-type Y220 (magenta) aligned in the putative rescue pocket, indicating a possible binding location for a stabilizing benzene moiety. (4) Mutant C220 (green) aligned in the putative rescue pocket.

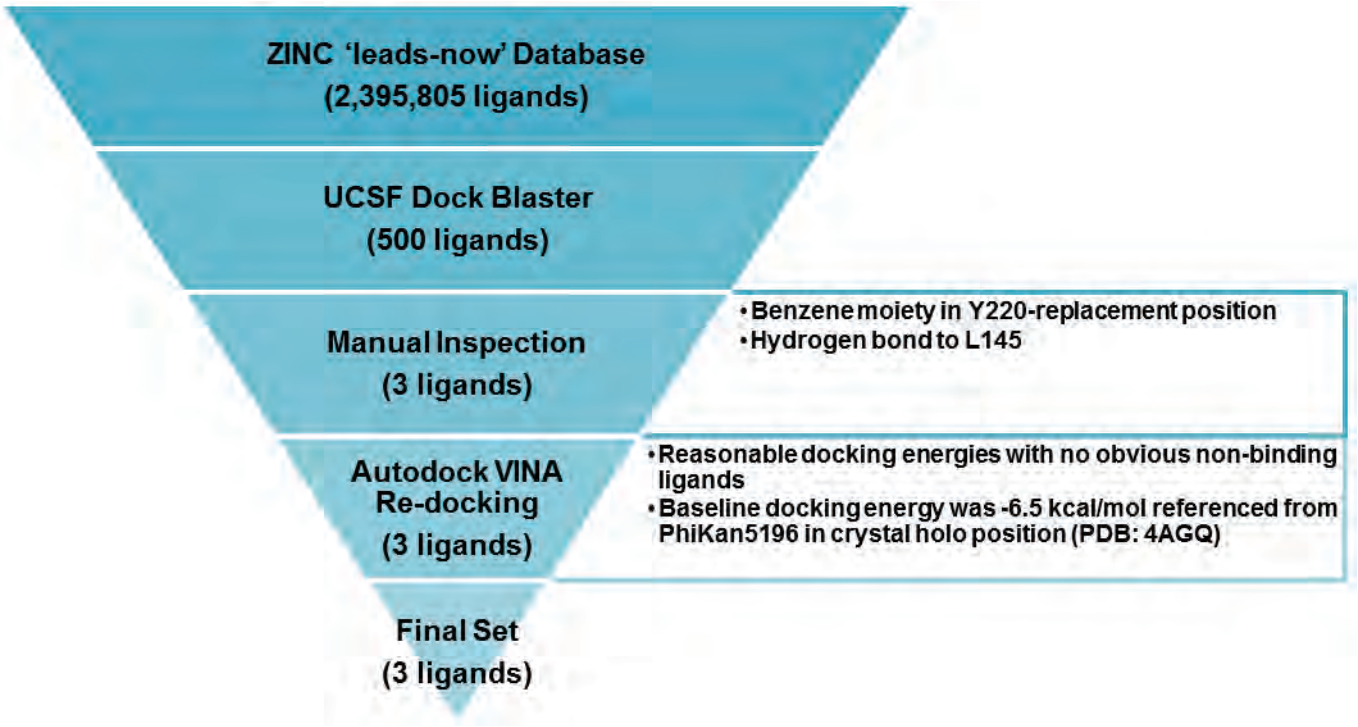


Figure 5.10 Ligand refinement process
2.4 million lead-like ligands were screened using UCSF Dock Blaster. This reduced the number of ligands under consideration to 500. These were then inspected manually for various structural and pharmacophore characteristics. The final three ligands were re-docked into the pocket using Autodock Vina as a consensus check to remove any obvious non-binding ligands.

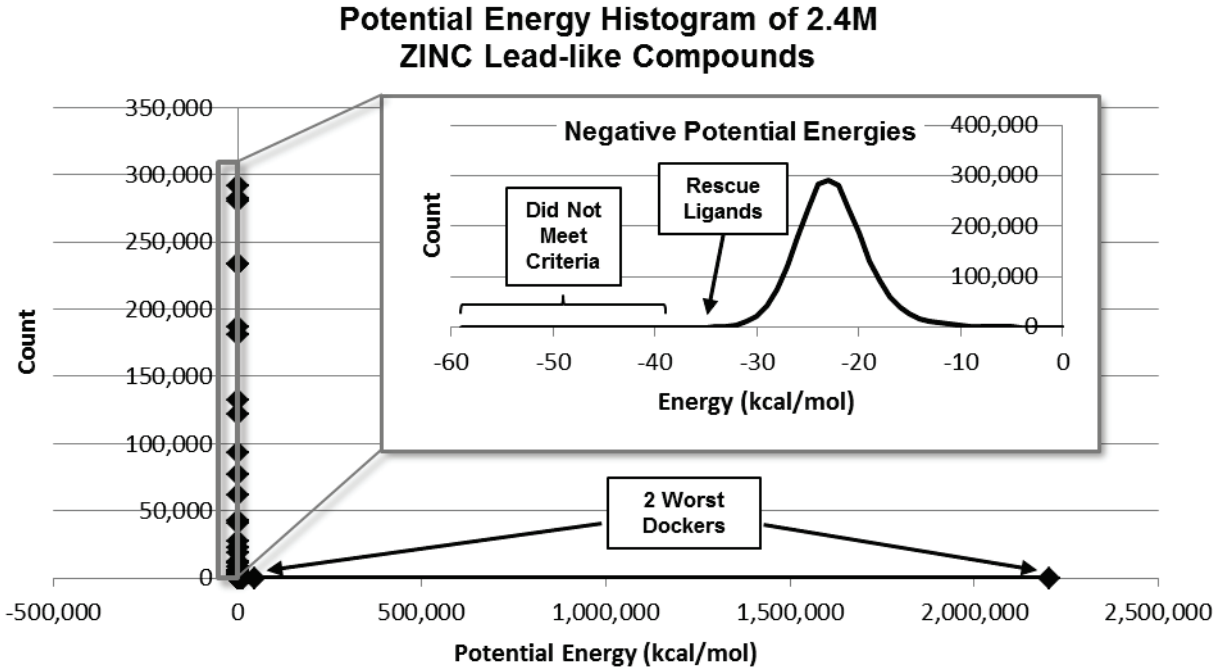


Figure 5.11 Potential energy distribution of 2.4M ligands docked by UCSF Dock Blaster
The distribution shape was a tall, narrow bell curve just below zero with a shallow tail extending to the right and a remote data outlier far to the right. 99% of the compounds were able to bind somewhere in the pocket vicinity with a potential energy less than zero, although Autodock Vina re-docking of the two worst dockers indicates that negative binding energies did not necessarily indicate significant contact with the desired cavity. Inset shows a close up of only the negative values. Mean ± standard deviation of the negative values was -22.5 ± 3.6 kcal/mol. The binding energies of the putative rescue ligands were located three and four standard deviations to the left of the mean.

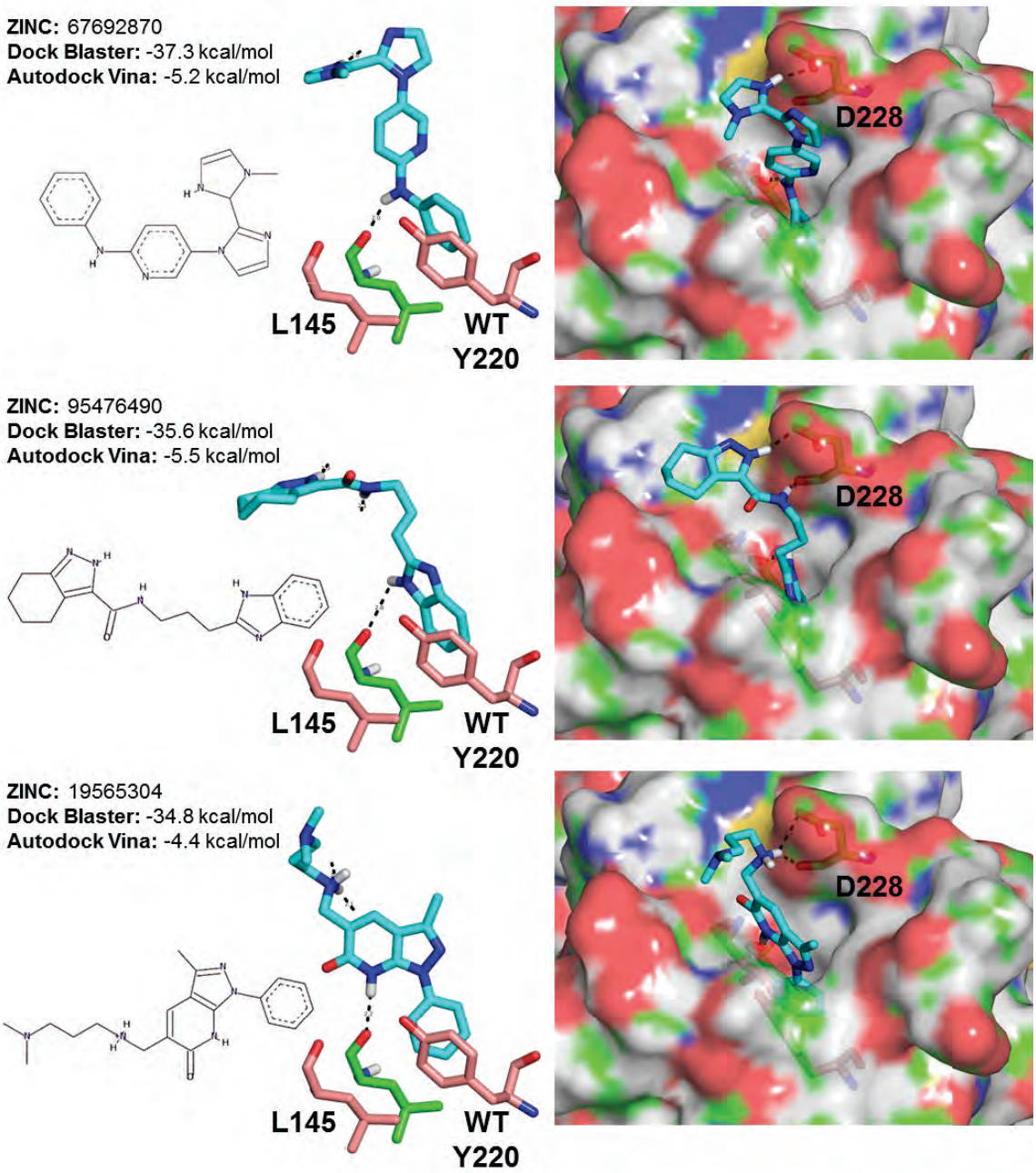


Figure 5.12 Final ligand picks
The three final ligands and specific binding data are shown from top to bottom. On the left is a molecular drawing and docking energies. In the middle is the ligand in its docked pose. The ligand is blue and L145 is green. Wild type L145 and Y220 are pink. On the right is a surface perspective showing the ligand in the rescue pocket. Dotted lines indicate polar contacts. Note that all three ligands have at least one contact with D228 at the surface.

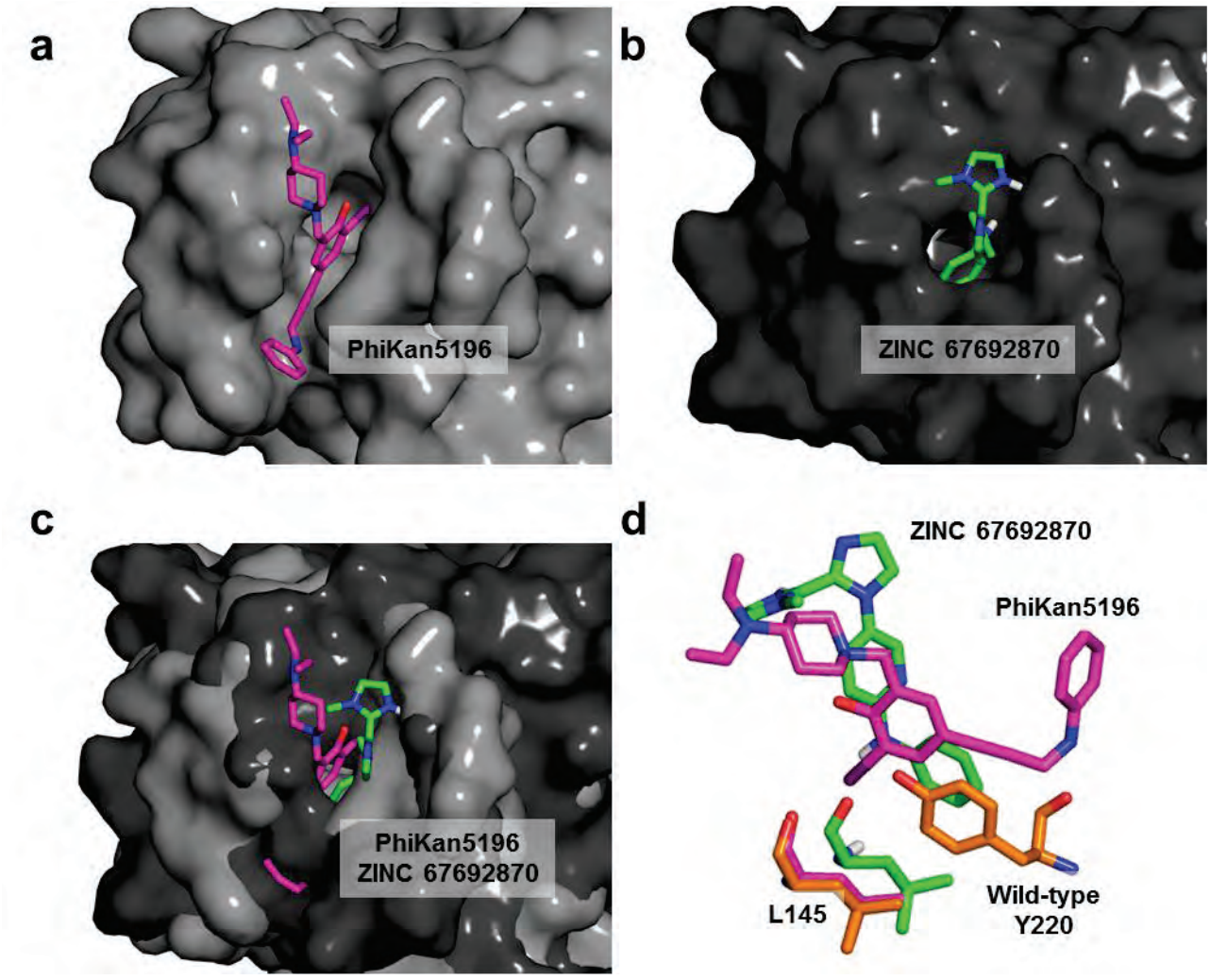


Figure 5.13 Overlap of known rescue ligand and putative rescue ligand addressing the same protein region from pockets with different surface geometries.
(a) PhiKan5196 docked in 4AGQ crystal structure (b) ZINC 67692870 (example putative rescue ligand) docked into selected rescue pocket. Note that the geometry of this pocket lacks the surface trench of the PhiKan5196 binding pocket. (c) Overlap of the PhiKan5196 and ZINC 67692870 shows that they were both docking into the same disrupted region using different surface geometries. (d) Side view of PhiKan5196 and ZINC 67692870 shows similarity of pharmacophore groups, particularly the tyrosine-replacing aromatic ring and a strong L145 contact. Wild-type Y220 is orange, PhiKan5196 and associated L145 side chain are magenta, ZINC 67692870 and associated L145 side chain are green.

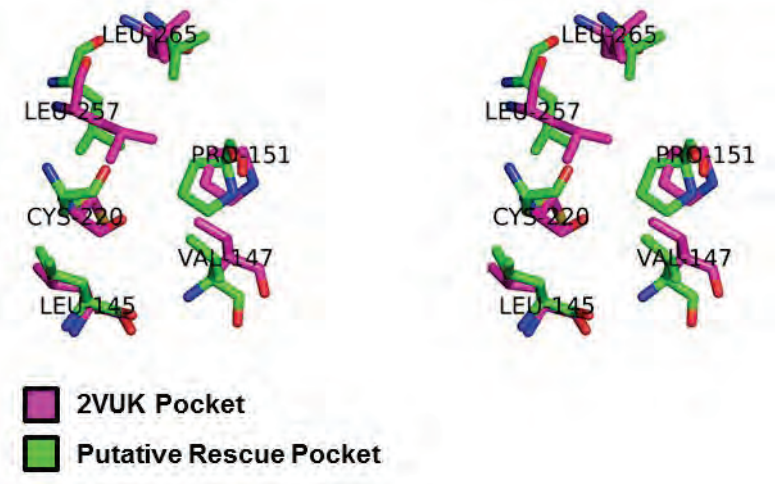


Figure 5.14 Stereo image showing overlap of putative rescue pocket and the 2VUK crystal structure.

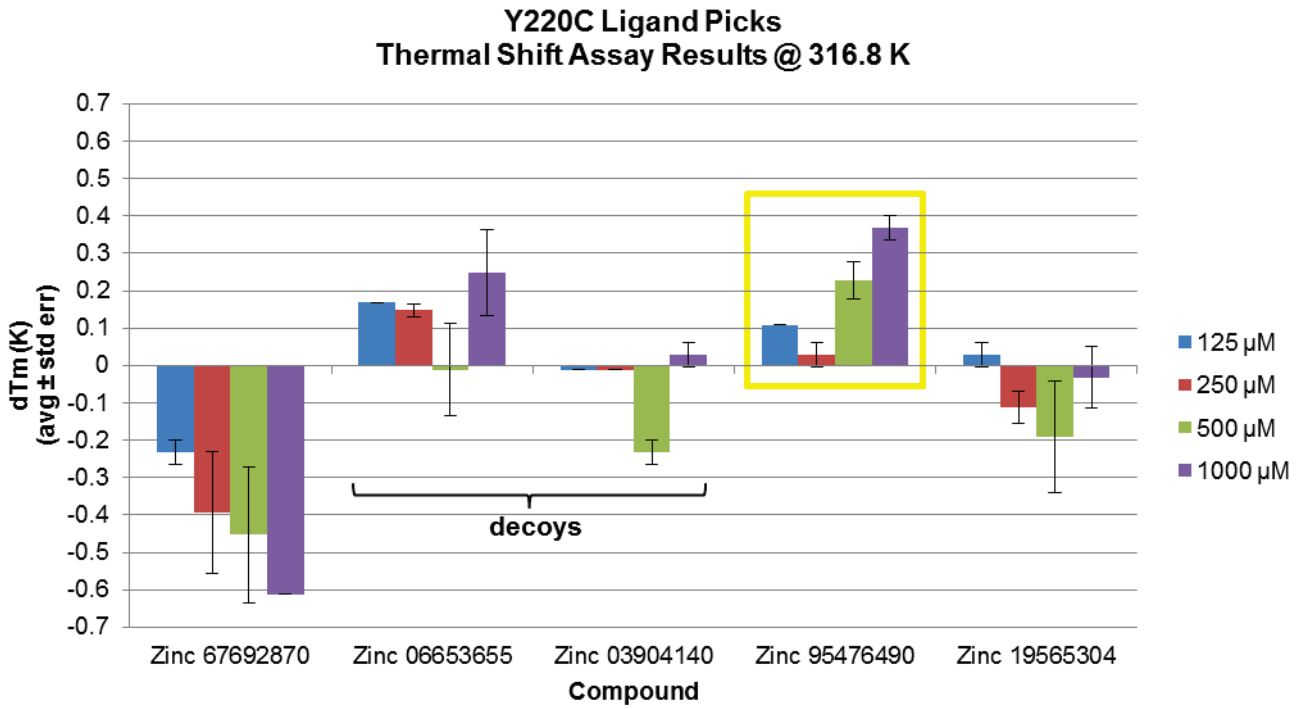


Figure 5.15 Initial thermal shift assay results.
One of the non-decoy ligands (Zinc ID: 95476490) showed a small, possibly dose-dependent stabilizing effect on the Y220C mutant (yellow highlight).

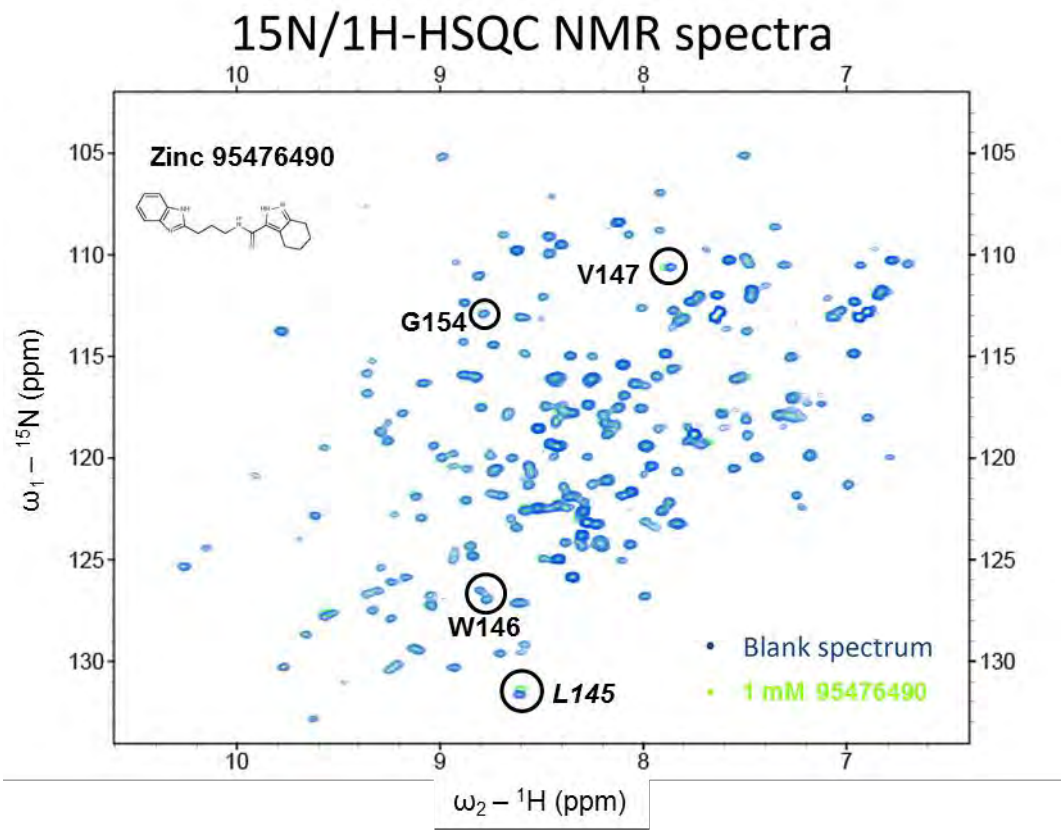


Figure 5.16 Initial NMR chemical-shift data.
Peak assignments suggest that the ligand capable of small thermal stabilizations (Zinc ID: 95476490) was also binding in or near the identified putative rescue pocket. (L145 assignment (italic) is preliminary).

Chapter 6

INSIGHTS INTO THE STRUCTURAL DYNAMICS OF CANCER: MOLECULAR DYNAMICS SIMULATIONS OF WILD TYPE p53 AND 20 DIFFERENT MUTANTS SHOW COMMON STRUCTURAL-DISRUPTION MOTIFS INCLUDING HYPOTHESIZED AMYLOIDOGENIC CONFORMATIONS

6.1 Abstract

The p53 protein is a commonly-studied cancer target because of its role in tumor suppression. Unfortunately, it is susceptible to mutation-associated loss of function; approximately 50% of cancers are associated with mutations to p53, the majority of which are located in the central DNA-binding domain. Here we report molecular dynamics simulations of wild-type p53 and 20 different mutants, including a stabilized pseudo-wild type mutant. Our findings indicate that p53 mutants tend to exacerbate latent structural-disruption tendencies, or vulnerabilities, already present in the wild type protein, suggesting that it may be possible to develop cancer therapies by targeting a relatively small set of structural-disruption motifs rather than a multitude of effects specific to each mutant. In addition, α-sheet secondary structure formed in almost all of the proteins. α-sheet has been previously hypothesized to play a role in amyloidogenesis, and its presence in the reported p53 simulations coincides with the recent re-consideration of cancer as an amyloid disease.

6.2 Introduction

The p53 transcription factor (Vogelstein et al. 2000) is one of the most-studied cancer targets because of its prominent role in tumor suppression. Roughly half of all cancers are associated with mutations to p53 (Joerger et al. 2006) with most of those located in the central DNA-binding domain. Many of these mutations have been studied and characterized, and they commonly display reduced DNA-binding ability, reduced structural integrity, or both (Bullock et al. 2000). Extensive experimental characterizations have been performed on p53, leading to a deeper understanding of the mechanisms involved, in some cases proceeding to compounds that are capable of stabilizing the mutants and restoring function (Boeckler et al. 2008; Wilcken et al. 2012), even to the point of progressing to clinical trials of potential cancer therapeutics (Bykov et al. 2002; Lehmann et al. 2012). Figure 6.1 shows the structure of p53, a schematic overview of its secondary structure, and the locations of the 19 destabilizing mutations whose simulations are presented here. The p53 protein belongs to the immunoglobulin-like β-sandwich fold represented in Rank 1 of our Consensus Domain Dictionary (Day et al. 2003; Schaeffer et al. 2011). It consists of two helices H1 and H2, a smaller β-sheet comprised of strands S1, S3, S8, and S5 and a larger β-sheet comprised of strands S6, S7, S4, S9, S10, S2', and S2.

In the work presented here, we have performed molecular dynamics (MD) simulations (Beck and Daggett 2004; Levitt et al. 1995; Levitt et al. 1997) of the DNA-binding domain of wild-type (WT) p53, 19 different destabilizing p53 mutants, and a pseudo-wild-type (pWT) mutant that is stabilized by 2.65 kcal/mol via four mutations (Joerger et al. 2004). MD is a powerful complement to experimental work because it provides protein structural-data with sub-picosecond, sub-angstrom resolution.

Our purpose in analyzing simulations of p53 was to understand the structural ramifications of known oncogenic mutations and, optimally, to identify conformational motifs that would be amenable targets for structure-based drug design. To our knowledge, we present here the most extensive set of p53 simulations to date (Wassman et al. 2013; Calhoun and Daggett 2011; Basse et al. 2010; Lukman et al. 2013; Duan and Nilsson 2006; Barakat et al. 2011; Demir et al. 2011). These simulations involve all six "hot spot" residues (Harris and Hollstein 1993; Cho et al. 1994) and represent eight of the ten most common mutations and 21% of the listings in the IARC TP53 Database (Petitjean et al. 2007) (version R16). The most common cancer topologies associated with these mutations are the uterus, rectum, colon, liver, brain, stomach, hematopoietic and reticuloendothelial systems, pancreas, bone, ovary, lymph nodes, esophagus, breast, mouth, larynx, prostate, soft tissue, bladder, head and neck, lung and skin (Petitjean et al. 2007). The mutations discussed here are distributed throughout the protein (Figure 6.1c) and in most cases have experimental data available (Bullock et al. 2000). These simulations were performed as part of our Dynameomics project (Van der Kamp et al. 2010; Beck et al. 2008) (www.dynameomics.org). The Dynameomics project uses MD to characterize the folding and unfolding pathways as well as native-state dynamics of all known protein folds.

Here we report our findings, including descriptions of the structural-disruption motifs that were common to multiple mutants. In addition to large-scale tertiary-structure disruptions, one of the most interesting structural-disruption motifs that we identified was the presence of the α-sheet secondary-structure in almost all of the proteins. It has been proposed that α-sheet plays a role in amyloidogenesis (Armen, Alonso, et al. 2004; Armen, DeMarco, et al. 2004; Daggett 2006) and our

findings, together with a recent re-characterization of cancer as an amyloid disease (Ano Bom et al. 2012; Silva et al. 2013), suggest that α-sheet may be another target in the design of future cancer therapies.

6.3 Methods

6.3.1 *Molecular Dynamics Simulations*

A 2.05Å resolution Protein Data Bank (PDB) (Berman et al. 2000) x-ray crystal structure of the DNA-binding domain of human p53 (PDB code: 2ocj, chain A (Y. Wang et al. 2007; Berman et al. 2000)) was used as the wild-type starting structure. The same structure was used to create the 19 destabilized mutant structures by *in silico* point mutation to the appropriate residue. A stabilized wild type-like quad-mutant with mutations M133L, V203A, N239Y and N268D was also simulated using chain A of the 1.9Å PDB crystal structure 1uol (Joerger et al. 2004). Because zinc is significantly disassociated from p53 at physiological temperatures (Butler and Loh 2003) and because mutations are associated with zinc loss (Loh 2010), wild type and mutant simulations were performed *apo* at 310K. We also performed additional 298K *holo* simulations of wild-type p53 for comparison to experimental NMR data.

Molecular dynamics simulations were performed using our in-house modeling package *in lucem* molecular mechanics (*il*mm) with the Levitt *et al* force field and established methods (Beck et al. 2000-2014; Levitt et al. 1995; Beck and Daggett 2004; Levitt et al. 1997). Protein structures were first minimized with steepest descent minimization for 1000 steps (1 step = 2 fs). Structures were then solvated using the F3C water model (Levitt et al. 1995) in a periodic box with walls no closer than 10Å from any protein atom. The solvent density was set to 0.993 or 0.997 g/mL, the

experimental densities for water at 310K and 298K, respectively (Kell 1967). Water was minimized for 1000 steps followed by water-only dynamics at 298K or 310K for 500 steps. The solvent was again minimized for 500 steps and then both the solvent and protein were minimized for 500 steps.

Production simulations were performed in triplicate for 100 ns at 298K (*holo*, WT) or 310K (*apo*, WT and all mutants) for a total simulation time of 6.6 μs. A Maxwellian distribution at low temperature was used to assign initial atomic velocities after which the temperature was increased to 298K or 310K. A 10Å non-bonded cutoff was used and the interaction list was updated every 2 steps. The NVE microcanonical ensemble was used with constant number of particles, energy, and volume. Simulations were performed at neutral pH (neutral His, positive Arg and Lys, and negative Asp and Glu). Structures were saved every 1 ps.

6.3.2 *Simulation Analysis*

Solvent-accessible surface area (SASA), Cα Root-Mean-Squared Deviation (RMSD), Cα Root-Mean-Squared Fluctuation (RMSF), secondary structure (using the Define Secondary Structure of Proteins (DSSP) secondary-structure assignment algorithm (Kabsch and Sander 1983)), atomic-contact analyses, and Nuclear Overhauser Effect (NOE) analysis were performed using *il*mm (Beck et al. 2000-2014) analysis modules. SASA values for NMR structures were also calculated using *il*mm. Contact analysis was performed using heavy atoms with a 5.4Å cutoff for carbon-carbon contacts and 4.6Å cutoff for all other heavy-atom contacts. Only one atomic contact was necessary for residues to be considered in contact; contact-occupancy was calculated as the percentage of time two residues were in contact. Contacts between adjacent residues were

skipped. The Contact Walker analysis tool was used for mutant/wild-type contact comparison (Bromley et al. 2013). Cα RMSD and Cα RMSF were performed using the β-strand residues as reference (for analysis purposes, the β-strand residues were considered to constitute the core of the protein). The equation $RMSF = \sqrt{3\beta/(8\backslash\pi^2)}$ was used to compare Cα RMSF values against crystallographic B factors. Aggregate analyses such as contact analyses, NOE analyses, Cα RMSF, per-residue averages, and percent time in secondary structure were performed over the last 25 ns (75-100 ns) of the simulations to ensure equilibration. Mutant secondary-structure contacts were deemed to have changed relative to wild type if the mutant and wild-type total contact-occupancy avg ± stdev ranges did not overlap. Calculations involving DNA-contacting residues used residues 120, 241, 248, 273, 276, 277, 280, and 283 (Cho et al. 1994). The L1/H2 distance was measured between the Cα atoms of K120 and R280. The L2/L3 distance was measured between the Cα atoms of C176 and C242. The L2/S5 distance was measured between the Cα atoms of D186 and G199.

6.3.3 *Additional Analyses*

Molecular images were created in Pymol (DeLano 2002), UCSF Chimera (Pettersen et al. 2004), and VMD (Humphrey et al. 1996). Electrostatic surface visualizations were created using Adaptive Poisson-Boltzmann Solver (APBS) software (Baker et al. 2001) hosted at the National Biomedical Computation Resource (NBCR) (http://nbcr.ucsd.edu) and visualized using UCSF Chimera. Additional interactive and aggregated data analyses were performed using the DIVE visual analytics platform (Bromley et al. 2014; Rysavy et al. 2014).

6.4 Results

6.4.1 *Experimental Validation and Simulation Quality Control*

Comparison of 298K holo simulations with experiment

As there are no 310K *apo* experimental structural data for p53, we validated our protein system by comparing 298K *holo* simulations with 298K *holo* NMR data (PDB: 2fej (Cañadillas et al. 2006)). In all three 298K *holo* simulations, the smaller S1/S3/S8/S5 β-sheet demonstrated disruptions in S1 and S5, and one simulation adopted α-sheet secondary structure (Daggett 2006) between strands S3 and S8. In all cases, however, the protein retained its general β-sandwich structure and no significant β-core disruptions or unfolding occurred. With the exception of the loss of S6 β-structure in one simulation, the larger S6/S7/S4/S9/S10/S2'/S2 β-sheet was consistently stable. One simulation demonstrated a 21Å separation between L1 and H2. H1 and H2 helical structure were maintained in all simulations.

The 298K *holo* wild-type simulations satisfied 89% of experimental NOEs with an average violation distance of 2.0 ± 1.9Å. 88% of the NOE violations involved side-chain atoms and most violations involved at least one loop residue. 33% of the violations were associated with nine residues, eight of which were in the S1 and L1 regions. These residues had an average violation distance of 3.3 ± 2.1Å. The remaining residues had an average violation distance of 1.4 ± 1.3Å. Almost half (45%) of the residue pairs containing a violation also contained a satisfied NOE. Average solvent-accessible surface area (SASA) of the 298K *holo* simulation structures and the 298K *holo* NMR structures was correlated with R = 0.79. Cα root-mean-squared

fluctuation (RMSF) behavior was similar between the NMR structures and the 298K *holo* simulations (Appendix Figure D.1a).

310K apo WT simulations

Per-residue Cα root-mean-squared-fluctuation (RMSF) analysis of the WT 310K *apo* simulations also showed general agreement with the WT 298K *holo* NMR data (Appendix Figure D.1b). There were small increases in residue fluctuation in the loop region immediately preceding S1, the S3/S4 loop, the S9/S10 loop, and H2. Not surprisingly, the largest increases in fluctuation occurred near the *apo* zinc-binding residues C176, H179, C238 and C242. The wild-type simulations had average Cα RMSD values of 4.7 ± 0.2Å, 4.7 ± 0.2Å, 4.5 ± 0.3Å; the values for the β-core were 2.8 ± 0.3, 1.9 ± 0.2Å, and 2.6 ± 0.2Å. The majority of the deviation from the starting structure occurred in L1, L2, the S7/S8 loop, L3, and H2 (Appendix Figure D.2).

Secondary-structure analysis identified sporadic loss of S1 β content, gain of α-sheet secondary structure between S1 and S3 and between S6 and S7, loss of H1 helical content, and gain of helical content in the H168 region. One WT simulation (#2) demonstrated a 20Å separation between L1 and H2, disrupting the loop-sheet-helix region while another WT simulation (#3) demonstrated a 15Å separation between L2 and L3.

The per-residue Cα RMSF and experimental crystallographic B factors were similar to the NMR Cα RMSF discussed above (Appendix Figure D.3). Wild type was compared to the crystal structures 2ocj (Y. Wang et al. 2007) (original simulation structure) and 2xwr (Natan et al. 2011) (wild-type structure with a five-residue extended N-terminus). Experimental crystal structures were found for ten mutants (PDB codes: 1uol (Joerger et al. 2004), 4ibq (Eldar et al. 2013),

4ijt (Eldar et al. 2013), 4ijs (Eldar et al. 2013), 2j1y (Joerger et al. 2006), 2bin (Joerger et al. 2005), 3d05 (Suad et al. 2009), 3d06 (Suad et al. 2009), 3d07 (Suad et al. 2009), 2j21 (Joerger et al. 2006), 2j1w (Joerger et al. 2006), 4kvp (Wallentine et al. 2013), and 2j1x (Joerger et al. 2006)). In some cases, the only mutant structures available for comparison also contained the four additional stabilizing mutations discussed above (Joerger et al. 2005; Joerger et al. 2006).

The Cα RMSF data were generally in agreement between the simulations and the crystallographic B factors, although the crystallographic amplitudes were typically lower, particularly in the loop regions. The largest differences occurred in both termini and the L1, L2, S7/S8 and, to a lesser degree, L3 loop regions. Higher simulation Cα RMSF values can be rationalized due to the lack of zinc, temperature differences (310K vs 100K), and the different environments (solvated vs. crystal). In Appendix Figure D.3, missing B factor data are circled in green and the individual PDB files are labeled with the number of missing residues. Missing B factor data were concentrated in the L1 and L2 loop regions, although the terminal residues were occasionally missing data as well.

6.4.2 *Mutant Analysis*

Quad-Stabilized Pseudo-Wild Type Mutant

The quad-stabilized pseudo-wild type p53 protein (PDB: 1uol) contains four known stabilizing mutations: M133L, V203A, N239Y, and N268D. The mutant has been shown to maintain wild type functionality while being stabilized by 2.65 kcal/mol (Nikolova et al. 1998). The average Cα RMSD values of our simulations were 5.1 ± 0.2Å, 4.1 ± 0.2Å, and 3.8 ± 0.4Å and β-core Cα RMSD values were 3.1 ± 0.2Å, 1.9 ± 0.2Å, and 1.7 ± 0.2Å. In every simulation, the

S7/S8 loop was the region with the highest Cα RMSD values with average deviations from the starting structure of 10.2 ± 1.0Å, 9.3 ± 1.2Å, and 6.1 ± 0.7Å in the three simulations. Most Cα RMSD values were near or below wild-type ranges, although one simulation had values for S1, L1, and S3 that exceeded wild-type values by 2-5Å and two simulations had values in L2 that exceeded wild-type values by 5Å. Cα RMSF values were also near or below wild-type ranges except for the L1 region which exceeded wild-type values by 2Å and the H1 region which was lower than wild-type by approximately 1Å. Secondary-structure analysis indicated a loss of H1 helical-structure in one simulation, the loss of S1 β-structure in two simulations, and the gain of α sheet between S3 and S8 in one simulation and between S1 and S3 in another simulation. L1 and H2 did not separate in any of the three simulations and both L2/L3 separation and L2/S5 separation distance values were less than wild type.

To analyze the effects of the four stabilizing mutations to the p53 protein, we compared the average contact-occupancy of WT contacts with the average contact-occupancy of pseudo-wild type contacts. From 1706 residue:residue contacts, we selected those contacts whose wild-type and pseudo-wild-type occupancy differed by at least 50%. This resulted in 81 contacts, 41 of which were more stable in pseudo-wild type and 40 of which were more stable in wild type. The location of these contacts and the specific residues involved are detailed in Figure 6.2 and in Appendix Table D.1 and Appendix Table D.2. Contact changes were primarily located in the loop-sheet-helix region, the DNA-binding region, and the S5/S6 region. Additional contact changes included a few stabilizations across the S1/S3/S8/S5 β-sheet and a few destabilizations in the N-terminus of S10 and in the N-terminal loop.

19 Disease-Associated Mutants

Individual analyses of the 19 destabilizing mutants are included in Appendix D. In general, mutation-location was not indicative of specific structural behaviors. Over the time-frames simulated, tertiary-disruptions from the starting structure were common, but significant structural loss of the β-core was not observed. The Cα RMSD of the β-core was always lower than the full-protein, and the Cα RMSD of the larger S6/S7/S4/S9/S10/S2'/S2 β-sheet was almost always lower than that of the smaller S1/S3/S8/S5 β-sheet. The largest Cα RMSD values were typically seen outside of the β-core, particularly in loops L2 and L3 (as expected without the stabilizing zinc-contact, and in agreement with experiment (Butler and Loh 2003)), the loop-sheet-helix region, the loops in the S5/S6/S7 region, and the S7/S8 loop. In general, the largest Cα RMSF values were in the same regions. Most of the time, the Cα RMSD and Cα RMSF values fell within or very close to wild-type ranges.

These same regions also showed the greatest departure from crystal secondary structures. As discussed below, novel helical structures were observed in the N-terminal loop, L1, L2 and L3 while helical content was often lost in H1. The β-content of the larger β-sheet was usually well-maintained, but the β-content was of the smaller sheet was often lost. In several instances, the N-terminal loop was able to form a novel β-strand aligned antiparallel to S10. The appearance of α-sheet structure, discussed in more detail below, was also a common occurrence. Common tertiary-structure changes involved L2 moving away from L3, which disrupted the zinc- and DNA-binding regions, and L1 moving away from H2, which disrupted the loop-sheet-helix region.

6.4.3 *Aggregate Analysis*

The region around H168 was able to support helical content

The region around H168 demonstrated both helical and loop structures; wild type and 12 mutants (15 simulations in total) demonstrated helical content around H168 for at least 50% of the time. This is consistent with published data; Protein Data Bank DSSP analysis of the H168 region indicates a loop for the 2ocj WT crystal structure and a 3/10 helix for the 2fej WT NMR structure (www.pdb.org).

The smaller β sheet was more prone to structural disruption than the larger β sheet

In general, the smaller S1/S3/S8/S5 β sheet lost more secondary structure than the larger S6/S7/S4/S9/S10/S2'/S2 β sheet. This may have been because the shorter strands in the S1/S3/S8/S5 sheet provided fewer stabilizing contacts. It may also have been due to environmental conditions; the average residue in the shorter sheet had 1.4 ± 0.2 times the solvent exposure of a residue in the larger sheet. The solvent-accessible surface of the wild-type crystal structure is shown in Figure 6.3. This figure shows that the shorter β sheet is considerably more solvent exposed than the larger β sheet, with S1, S3, and S8 all contiguously exposed. Much of the larger β sheet is shielded from solvent by the N-terminal loop, the N-terminal half of L2, the S6/S7 loop, and the S9/S10 loop. The primary points of solvent exposure in the larger β sheet are S4, S6 and S10. S4 is exposed at the N-terminal end near the S3/S4 loop. S6 is exposed for much of its length, but it is exposed on its edge, and S6 itself is the edge of the larger β sheet, so in general the strand:strand hydrogen bonds in the larger β sheet are shielded from competitive hydrogen bonding from solvent. S10 is exposed for most of its ten-residue length, from the

S9/S10 loop, past S1, and then into the loop-sheet-helix region. Two of these regions, the S1 and the loop-sheet-helix regions, demonstrated significant disruption from the original starting structure in most of the simulations. Notably, the point where both S10 and S1 are exposed lies at the N268/L111contact point, the same point where the N268D mutation in the pseudo-wild type crystal structure establishes a novel hydrogen bond between S10 and S1 (Joerger et al. 2004).

L1 separation from H2 was a common loop-sheet-helix structural-disruption motif

The loop-sheet-helix region of p53 is involved in DNA binding (Cho et al. 1994), and disruption of this region could compromise that ability. K120 and R280 are DNA-contact residues in L1 and H2, respectively. The experimentally-determined distance between the Cα atoms of these two residues ranges between 4.9-7.1Å (for crystal structures 2ocj and 1tsr (Cho et al. 1994)) and 5.6-9.1Å (for NMR structure 2fej). Simulations of the wild type, pseudo-wild type, and several mutants established stable conformations that placed these atoms between 5Å and 10Å of each other. However, the wild-type protein and 13 of the 19 mutants all had at least one simulation where these atoms and, by proxy, L1 and H2, were separated by at least 20Å. The stabilized pseudo-wild type did not show such a separation; pseudo-wild-type L1 and H2 separation was stable at 5Å across all three simulations.

In total, there were 17 simulations that adopted a conformation that separated L1 and H2 by at least 20Å. There were 122 contacts involving at least one loop-sheet-helix residue that were common to all of these conformations; to identify which contacts might be contributing to the L1/H2 separation, we compared them against a wild-type simulation with a stable crystal-like L1/H2 distance and minimal Cα RMSD in the loop-sheet-helix region (2.4 ± 0.2Å). Comparing

the wild-type contact occupancy with the average contact occupancy across the 17 simulations, we identified 47 contacts that had at least a 90% occupancy difference with the wild type. These contacts, common to all L1/H2 separations, are shown in Figure 6.4 and Appendix Table D.3. Of these 47 contacts, 46 of them were weakened or lost relative to wild-type, suggesting that these contacts may play a role in maintaining the structure of the loop-sheet-helix region by holding L1 and H2 in a crystal-like conformation.

Another disruption common to the loop-sheet-helix region was the gain of helical content in L1, resulting in a distortion of the local structure. Nine simulations, representing nine different mutants, adopted L1 helical content between 10% and 100% of the time.

L2 separation from S5 was a common zinc-region structural-disruption motif

The L2 loop region stretches between residues 164 and 194 and contains the H1 helix between residues 177 and 180. Zinc-binding residues C176 and H179 are contained within H1. A common structural-disruption motif in this region was a separation of L2, particularly the region of L2 C-terminal to H1, from the S5/S6 loop. To quantify the ubiquity of this separation, we measured the distance between the Cα atoms of D186 in L2 and G199 in the S5/S6 loop. Experimental distances for these atoms range between 9.8-10.6Å (for crystal structures 2ocj and 1tsr) and 11.0-15.6Å (for NMR structures 2fej); investigation of the simulations showed that 13 of the 19 mutants demonstrated a separation between these atoms of at least 20Å in at least one simulation. Neither the wild type nor the stabilized pseudo-wild type demonstrated this degree of separation, although the wild-type distances went as high as 19Å.

In total, there were 14 simulations in which L2 and S5 were separated by at least 20Å. There were 113 contacts involving at least one L2, S5, or S5/S6 residue that were common to all of these conformations. To identify which contacts might be contributing to the L2/S5 separation, we compared them against a wild-type simulation with a stable crystal-like L2/S5 distance and minimal Cα RMSD in the L2 region (3.3 ± 0.3Å). Comparing the wild-type contact occupancy with the average contact occupancy across the 14 simulations, we identified 49 contacts that had at least a 90% occupancy difference with the wild type. These contacts, common to all L2/S5 separations, are shown in Figure 6.5 and Appendix Table D.4. 48 of the 49 contacts were weakened or lost relative to wild type; only one contact between L3 and S5 was found to be commonly stabilized among the mutants.

L3 helical-propensity was associated with DNA-region disruption

L3 adopted helical conformations in multiple simulations, distorting the DNA-binding region. Residue S241, which is responsible for contacting the DNA backbone (Cho et al. 1994), was contained within the novel helical structure; in the R273C mutant (Figure 6.6), the Cα atom of R241 was displaced from the starting structure by 7Å and the side chain of that residue was reoriented.

In most instances, a complete helix did not form, but an alternative α-sheet-like structure formed instead. α-sheet is comprised of residues in alternating α_R/α_L positions on a Ramachandran plot with the backbone carbonyl oxygens aligned uniformly along the strands (see discussion below). In these instances, the residues on either side of the L3 turn aligned in α-sheet position and the residues in the turn were typically some mix of α_R and α_L, along with other

more distributed loop-like Ramachandran positions. This MD conformation was similar to the 2ocj crystal starting-structure and the 2fej NMR structure (Figure 6.7).

In total, secondary-structure analysis indicated that four simulations representing four different mutants adopted L3 helical structure at least 50% of the time; three of these were right-handed helices, while one (R273H) entered a right-handed helical turn, reversed direction, and completed a full left-handed turn. Similar analyses indicated that 34 simulations representing wild type, pseudo-wild type, and 18 of the 19 mutants adopted α sheet-like structure in the L3 turn at least 50% of the time. The only mutant not represented was V143A which fell below the threshold at 21%.

Most proteins adopted stable α-sheet content

The wild type, pseudo-wild type, and 18 of the 19 mutants adopted α-sheet in at least one simulation for at least 10% of the time (the remaining F134L mutant adopted a novel N-terminal α-strand for only 1% of the time, discussed below). α sheets were typically three residues in length, two strands wide and centered around residues R110 or L111 in S1 and Q144 or L145 in S3 (Figure 6.8). These two-strand, three-residue sheets typically showed two residues in α_R/α_L positioning and one residue in β position on a Ramachandran plot. Although not as common, some α-sheets were able to elongate beyond the original β-strand residues; one G245S α-sheet extended from residue 109 to residue 113 in S1 and from residue 144 to residue 149 in S3 (Figure 6.9). α sheets were also able to extend to strands S8 and S5, although these were less frequent, and α-sheet residues were able to align while oriented in either direction (left or right with respect to the α-strand).

In addition to the S1/S3/S8/S5 α sheet, α-strand conformation was observed at the N-terminus, in L1, in L3 (discussed above), in the S7/S8 loop, and between strands S6 and S7. This latter case occurred less frequently; 8 mutants (including M237I, which did not exhibit S1/S3 α-sheet) demonstrated S6/S7 α-sheet, but all less than 35% of the time. One wild-type simulation, however, exhibited S6/S7 α sheet for 91% of the time (Figure 6.10).

To establish that the α-sheets were legitimate constructs and not simply unstable residues moving between structured and unstructured conformations, we looked at the last 25 ns of the simulations and, for each α-sheet residue pair (e.g. L111:Q144), we correlated the per-picosecond secondary-structure assignments to α-conformation. If two α-strand residues were acting as a single α-sheet, they should be in α conformation, or not, at every picosecond; if they were simply unstructured, there should be no correlation. We also calculated the percentage of time that the residues spent in α-conformation and (with the exception of the F134L mutant) only analyzed those α-sheets that were present for at least 10% of the time. A summary of the major identified α-sheets is shown in Table 6.1, aggregated into inter-secondary-structure α-sheet contacts.

With the exception of L3, all α-sheets had average structural correlation values $R \geq 0.90$. Most α-sheets were in the S1/S3/S8 region, accompanied by several less-frequent occurrences and several single-mutant α-sheets. All of them, however, had strong structural correlations between the two α-strands.

The α-sheet occurring in the turn in the L3 loop had a lower average correlation coefficient ($R = 0.74 \pm 0.22$) than the other regions, despite being present for more of the time

and for more of the mutants than any of the other regions. The N-terminal α-sheet was only present for 1% of the time in a single mutant, but with a structural correlation coefficient of R = 0.91. This sheet was the only α-sheet involving a novel piece of secondary structure and the only sheet to convert a residue in the larger β-sheet. It is also worth noting that the converted S10 residue G266 neighbors the rescue-mutation residue N268 (Figure 6.11).

A cascade of rotations and rearrangements was common during α-sheet formation (Figure 6.9). This sequence often began with an initiating residue, such as L111, rotating into an α-sheet position and creating an unstable situation for a residue on an adjacent strand, such as Q144. If the initiating residue was sufficiently stable in this position, the adjacent residue rotated into a complementary α-position. In a three-stranded α-sheet such as that adopted by the Y220C mutant, this cascade of rotations continued into the third strand. During this cascade, the strands often rearranged to optimize their relative positions; in the G245S mutant (Figure 6.9), this was achieved by S1 and S3 sliding past each other almost one full residue length.

The N-terminal residues were capable of forming novel secondary structures

In most of the simulations, the residues N-terminal to S1 remained unstructured. As discussed above, however, the N-terminal loop was capable of forming α-sheet with S10. In addition to this novel α-content, 11 simulations representing 8 different mutants adopted novel β-content in the N-terminal residues for at least 10% of the time and as much as 83% of the time. These non-native strands were typically centered on Y103, ran anti-parallel to S10, and were three to four residues in length. Analysis of ϕ/ψ angles indicated that T102 and Y103 were usually in the β-quadrant of the Ramachandran plot while Q104 was less structured as the

residues curved around to form S1. An example of a novel N-terminal β strand is shown in Figure 6.12.

In addition to creating a novel β-strand, the N-terminal loop was also capable of adopting helical character. Two simulations adopted novel helices from residues 104-108 for a significant percentage of time (M237I: 48% of the time, R273C: 86% of the time) (Figure 6.13). Other mutants demonstrated N-terminal helical-content less frequently, typically < 5% of the time.

Aggregate contact analysis of all destabilizing mutant simulations identified three primary regions of disruption

Appendix Figure D.4a shows a histogram of aggregated contact-difference data from all 57 (3x19) simulations of destabilizing p53 mutants. In general, most contacts retained wild type-like contact occupancies. The top two most destabilized contacts (L114:T125 and L114:Y126) overlapped with the pocket discovered by Wassman *et al* (Wassman et al. 2013) that was shown to be capable of stabilizing the R175H tumorigenic mutant. Appendix Figure D.4b shows the 31 contacts whose occupancy-difference relative to wild type was > 0.3 or < -0.3 (the left and right extremes of Appendix Figure D.4a). Analysis of these data identified three primary disruption-regions, overlapping with the secondary- and tertiary-structure analyses discussed above: the loop-sheet-helix region, the N-terminal loop, and the L2/S8/S5/S6/S7 region. Contacts in these regions whose simulations demonstrated at least 50% occupancy loss relative to wild type included 100:252 (12 of 19 mutants), 114:125 and 114:126 (16 of 19 mutants), 200:232 (14 of 19 mutants), 171:249 (15 of 19 mutants), and 165:249 (16 of 19 mutants).

6.5 Discussion

6.5.1 *α-Sheet May Play a Role in p53 Aggregation*

Cancer is in some respects an amyloid disease; p53 mutations have been shown to be statistically correlated with p53 accumulation in tumor cells and long-term patient survival is statistically lower with aggregating mutants (Xu et al. 2011). Aggregation of p53 is accelerated by increased temperature (Bullock et al. 1997), increased pressure (Ishimaru et al. 2003), destabilizing mutations (Xu et al. 2011; Bullock et al. 1997), demetallation (Butler and Loh 2003), denaturants such as urea (Bullock et al. 1997), and inflammatory agents such a formaldehyde (Lasagna-Reeves et al. 2013); aggregation can also be induced in native wild-type protein by seeding it with mutant p53 or with wild-type p53 that has been converted to an amyloidogenic conformation (Ano Bom et al. 2012). Accordingly, the dominant-negative effect long known to be present with p53 mutants is hypothesized to be at least partly attributable to aggregation-prone mutant p53 converting native wild-type p53 to an aggregating species (Bullock and Fersht 2001; Xu et al. 2011).

p53 can also induce aggregation in other proteins, namely the p53 analogues p63 and p73 (Xu et al. 2011), suggesting that there is a common aggregation mechanism at work that is not p53-specific. This is further supported by the report that the A11 antibody binds soluble p53 (Levy et al. 2011); the A11 antibody is known to bind multiple amyloidogenic targets, in particular the soluble oligomeric forms, and the inclusion of the p53 protein into that set supports the hypothesis that there is a common aggregation mechanism that generalizes beyond the p53 family (Glabe and Kayed 2006).

Based on MD simulations of aggregating proteins, we have hypothesized that the general oligomeric aggregation mechanism involves the α-sheet secondary structure (Armen, DeMarco, et al. 2004; Armen, Alonso, et al. 2004; Daggett 2006), which was predicted by Pauling and Corey and termed 'polar-pleated sheet' (Pauling and Corey 1951). Although rare, small regions of α-strand have been demonstrated experimentally (Di Blasio et al. 1994; Daggett 2006). α-sheet has been identified in MD simulations of the aggregating proteins transthyretin (TTR), β2-microglobulin, human prion protein, lysozyme, and polyglutamine (Armen, Alonso, et al. 2004; Daggett 2006; Armen, DeMarco, et al. 2004). α-sheet was identified again in the p53 simulations presented here; notably, both TTR and β2-microglobulin are β-sandwich proteins in the same fold family as p53. In fact, alignment of the p53 crystal structure (PDB: 2ocj) with the TTR crystal structure (PDB: 1tta (Hamilton et al. 1993)) (DeepAlign (S. Wang et al. 2013)) indicates that the two primary α sheet-prone strands in p53 (strands S1and S3) align with the two primary α sheet-prone strands in TTR (strands G and A, respectively) (Figure 6.14). α-sheets also demonstrated solvent-exposed charge separation, one of the structural characteristics hypothesized to facilitate aggregation (Daggett 2006) (Figure 6.9). Additionally, previously published *in silico* analyses predicted that many of the putative p53 α-sheet residues presented here are also aggregation-prone (Rangel et al. 2014). Consequently, we hypothesize that α-sheet may play a role in p53 dysfunction.

Recent work by Hopping *et al* (Hopping et al. 2014) offers supporting experimental evidence for the presence of α-sheet in toxic oligomeric structures. Hopping *et al* designed peptides with α-sheet structure complementary to the hypothesized oligomer α-sheet structure, reasoning that the complementary structures should bind and inhibit amyloidosis. Biophysical measurements of

the designed peptides are consistent with the presence of α-sheet, and the peptides successfully inhibit aggregation of both transthyretin and β-amyloid peptide, preferentially binding the toxic species over the non-toxic species of those proteins.

Hopping *et al* also showed that α-sheet has a strong Fourier Transform Infrared (FTIR) absorbance band around 1680 cm^{-1} and a weaker absorbance band around 1640 cm^{-1}, making it separable from β-sheet, α-helix, and turns. With this in mind, it is possible that previously published work has measured but not identified the presence of α-sheet in amyloid analysis. For example, Xu *et al* published FTIR difference spectra for one non-aggregating p53 mutant and three aggregating mutants (Xu et al. 2011). Relative to wild type, the aggregating species has a large absorbance band at 1683 cm^{-1} and a smaller absorbance band at 1615 cm^{-1} whereas the non-aggregating species shows essentially no difference. The authors attribute the 1683 cm^{-1} absorbance band to an aggregation-correlated increase in β-structure; what we found interesting was that the strong high-frequency and weak low-frequency absorption pattern is consistent with the α-sheet FTIR spectra published by Hopping *et al*, supporting the hypothesis that α-sheet plays a role in p53 aggregation. If α-sheet is present, aggregation inhibitors like those designed by Hopping *et al* may represent a new class of cancer therapeutics.

Cancer has only recently begun to be considered an amyloid disease and more work is needed before the role of α-sheet, if any, becomes clear. However, between existing experimental data, *in silico* predictions, and engineered α-sheet binders, there is increasing evidence for a common aggregation-prone p53 species whose conformation differs from the

expected β-sheet structure. Our simulations, in conjunction with the previous work described above, offer an atomic-level hypothesis for the structure of this species.

6.5.2 *Common structural disruptions suggest the possibility of common structural rescue mechanisms*

As reported above, the largest structural disruptions observed in the simulations were common to many, and sometimes most, of the analyzed proteins, including both the wild type and the pseudo-wild type. A similar finding was observed by Wassman *et al* (Wassman et al. 2013). These observations suggest that 1) p53 has native disruption propensities that were exercised and/or exacerbated by the destabilizing mutations and 2) future cancer therapies may be able to address multiple p53 mutations at once by addressing common structural disruptions, or, as suggested by the aggregation inhibitors discussed above, by addressing common structural-disruption pathways.

Several pieces of experimental evidence support our finding that both the wild type and mutant proteins express the same structural disruptions. The first was reported by Butler and Loh (Butler and Loh 2003) who found that zinc is 30% disassociated from wild type at 310K. This suggests that the physiologically-relevant native state of p53 involves a significant amount of *apo* protein. NMR experiments showed that this *apo* conformer was structurally different from the native *holo* conformer; not only was it structurally different, it, like many reported mutants, was destabilized by 3 kcal/mol and was capable of inducing aggregation in wild-type p53. Moreover, the zinc-region mutations capable of destabilizing the *holo* protein by 2-4 kcal/mol only destabilized the *apo* protein by fractional amounts. If demetallation resulted in different

structural disruptions than the mutations, one would expect the mutation to have at least a partially additive destabilizing effect. This effect was not seen in the zinc-region mutants, suggesting that the *apo* wild type and the *apo* zinc-region mutations have similar disruption patterns. Additive disruption was, however, seen in the DNA-region mutant R282Q, suggesting that the zinc-region disruption found in *apo* wild type and the loop-sheet-helix region disruption found in *apo* R282Q mutant have at least partially-different disruption mechanisms.

A second piece of evidence comes from Ishimaru *et al* (Ishimaru et al. 2003) who used high-pressure, low-temperature denaturation to show that wild-type p53 could be converted to a stable, aggregation-prone conformer whose aggregation and denaturing characteristics were very similar to the R248Q mutant. Chemical-shift analysis showed that this conformer had altered structure in many of the same regions as the simulations reported here, with the largest shift in the S1/L1 region followed by smaller shifts in the S5/S6/S7 region, the S9 region, and the H2 region. Further support was presented by Ano Bom *et al* whose data indicated that p53 wild type and the R248Q mutant demonstrate similar molten-globule states when exposed to low pH (Ano Bom et al. 2010) and by Gogna *et al* (Gogna et al. 2012) whose experiments showed that wild-type p53 in hypoxic solid-tumor environments had similar immunoprecipitation results to mutant p53.

Additional evidence for a common disrupted conformational-state was reported by Baroni *et al* (Baroni et al. 2004) who showed that suppressor-mutations at codons 235, 239, and 240 were capable of suppressing the effects of 16 of the 30 most common p53 mutations, suggesting that there is a commonly-rescuable conformer endemic to many p53 mutants. The final piece of evidence was reported by Ano Bom *et al* (Ano Bom et al. 2012) who showed that wild-type and

mutant aggregates were equally cytotoxic. Taken together, these data suggest that there is a common toxic conformation and both the wild-type and the mutant proteins are capable of adopting it, even if the pathways to do so vary from protein to protein.

There is also evidence that supports the notion of a common rescue-mechanism. It has been shown that specific mutations can suppress the disruptive effects of multiple p53 mutations (Brachmann et al. 1998) and, as discussed above, that some suppressor-mutation locations hold more multi-mutant stabilization-potential than others (Baroni et al. 2004). Indeed, Baroni *et al* speculated that there was a common rescue-mechanism because the suppressor mutations at codons 235, 239, and 240 that were capable of rescuing more than half of the tested mutants were dominated by only one or two amino acid changes. In addition to mutations, small molecules, similar in scope to suppressor-mutant side chains, have been shown to be effective at stabilizing and rescuing p53 mutations (Boeckler et al. 2008; Wilcken et al. 2012; Wassman et al. 2013). There are also examples of multi-mutant p53 rescue-compounds such as PRIMA-1 (Bykov et al. 2002) (which has proceeded on to clinical trials (Lehmann et al. 2012)) and p53R3 (Weinmann et al. 2008) and, as discussed above, aggregation inhibitors capable of addressing multiple proteins (Hopping et al. 2014). In addition to rescuing mutant p53, it has been shown that wild-type p53, having undergone mutant-like conformational changes in hypoxic solid tumor environments, can re-assert native wild-type conformations by re-oxygenation or by chaperone-stabilization by native wild-type p53 (Gogna et al. 2012). Taken together, these data support our findings that p53 mutants share common structural-disruption motifs both among themselves and with wild type and suggest that a common therapeutic approach to multiple p53 mutations is possible. They also

suggest that targeting a relatively small set of structural-disruption motifs rather than a multitude of specific mutants may be an effective approach for cancer-drug research.

6.6 Conclusions

Our goal in performing these simulations was to offer specific, atomic-resolution data complementary to experimental data. To this end, we have performed molecular dynamics simulations of wild-type p53 and 20 mutants and provided dynamic, atomic-level details not accessible by experiment. Based on similar disruption tendencies observed for the p53 wild type and mutants, we propose that p53 mutants can adopt one or more common tumorigenic conformations and that at least one of those conformations - α-sheet - plays a role in p53 aggregation. We also propose that drug-design efforts targeting these common tumorigenic conformations will advance the search for single-drug multi-mutant cancer therapies.

Table 6.1: α-sheet Summary

	NT/S10	S1/S3	S3/S8	S8/S5	L1 Loop	S6/S7	S7/S8 Loop	L3 Loop
R[£]	0.91	0.95 ± 0.11	1.00 ± 0.00	1.00	0.97	1.00 ± 0.00	0.99	0.74 ± 0.22
% Time[†]	1	71 ± 28	73 ± 28	23	25	37 ± 29	23	84 ± 25
Simulations	1	23	6	1	1	5	1	38
Mutants[‡]	1	16	5	1	1	4	1	15
WT	no	yes	no	no	no	yes	no	yes
Quad	no	yes	yes	yes	no	no	no	yes
Notes[§]	F134L			G245S	V157F		L145Q	

£The average and standard deviation values were calculated using the correlation coefficients from the α-sheets in each group. Each individual correlation coefficient was calculated at picosecond resolution between 75 ns and 100 ns (n=25,000). †α sheets were required to be present for at least 10% of the time. Because of its relevance to the α-sheet analysis and because of its high degree of structural correlation, the F134L NT/S10 α sheet is included here despite only being present 1% of the time. ‡The pseudo-wild type is called out separately and is not included in this group. §Mutant is called out if the α-sheet appeared in only one simulation.

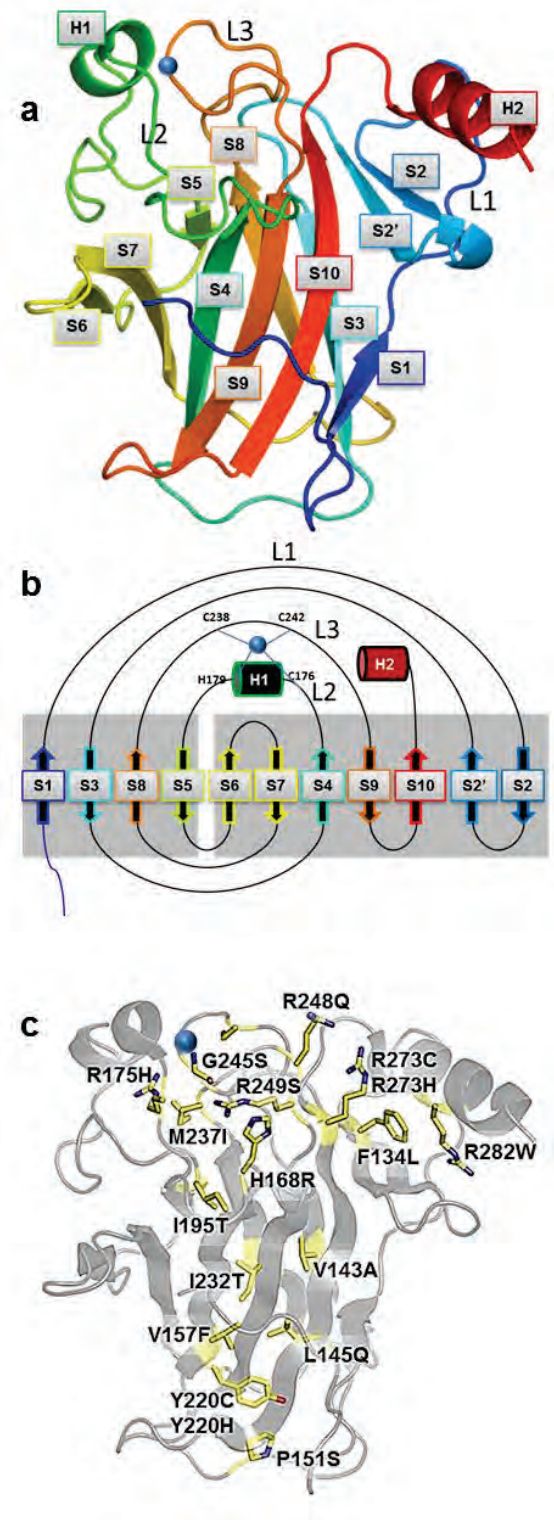


Figure 6.1 p53 crystal structure
(a) p53 crystal structure (PDB: 2ocj) labeled with secondary-structure. Zinc is depicted with a blue ball. (b) Schematic overview of the p53 secondary structures in (a). (c) p53 crystal structure overlaid with the mutations presented here.

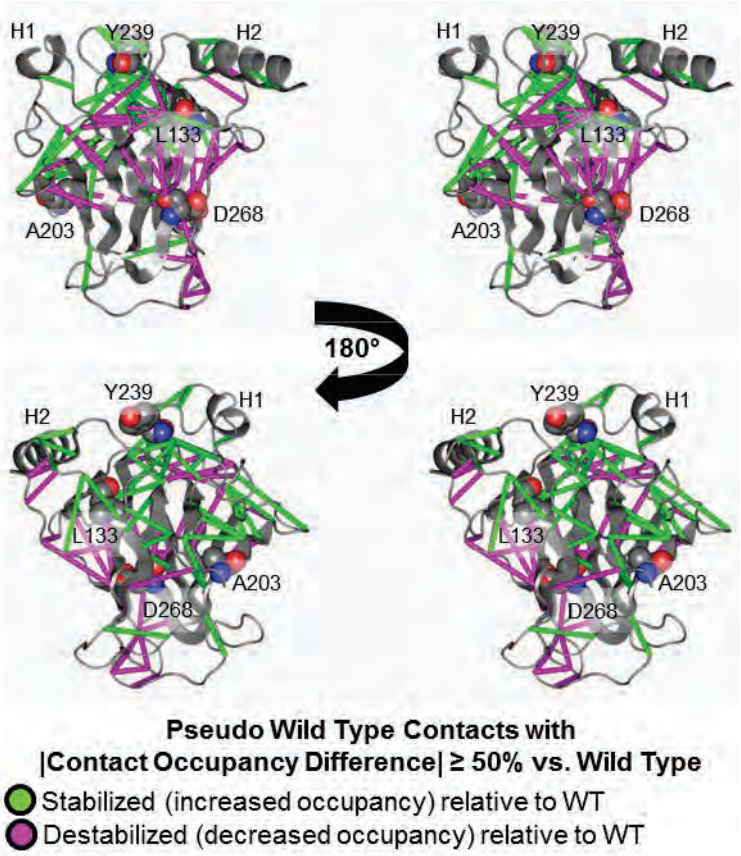


Figure 6.2 Contact differences between wild type and the pseudo-wild type
(stereo view) Crystal structure of stabilized pseudo-wild type (PDB: 1uol) overlaid with the 81 contacts whose occupancy differed by at least 50% from wild type. The four quad-stabilizing mutation sites (M133L, V203A, N239Y, N268D) are shown as spheres.

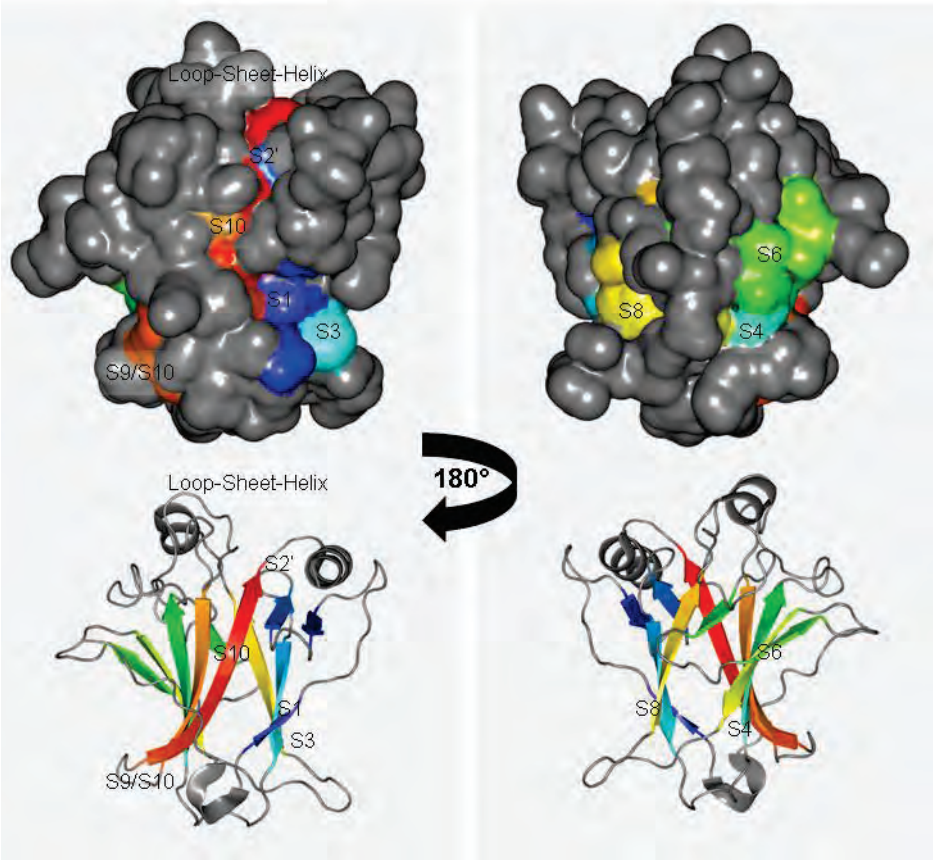


Figure 6.3 p53 crystal-structure solvent exposure
Rotated views of the β-sheet solvent exposure of the 2ocj crystal structure. β-sheets are colored and the remainder of the structures are gray. (top) Solvent-accessible surface area (bottom) Cartoon view.

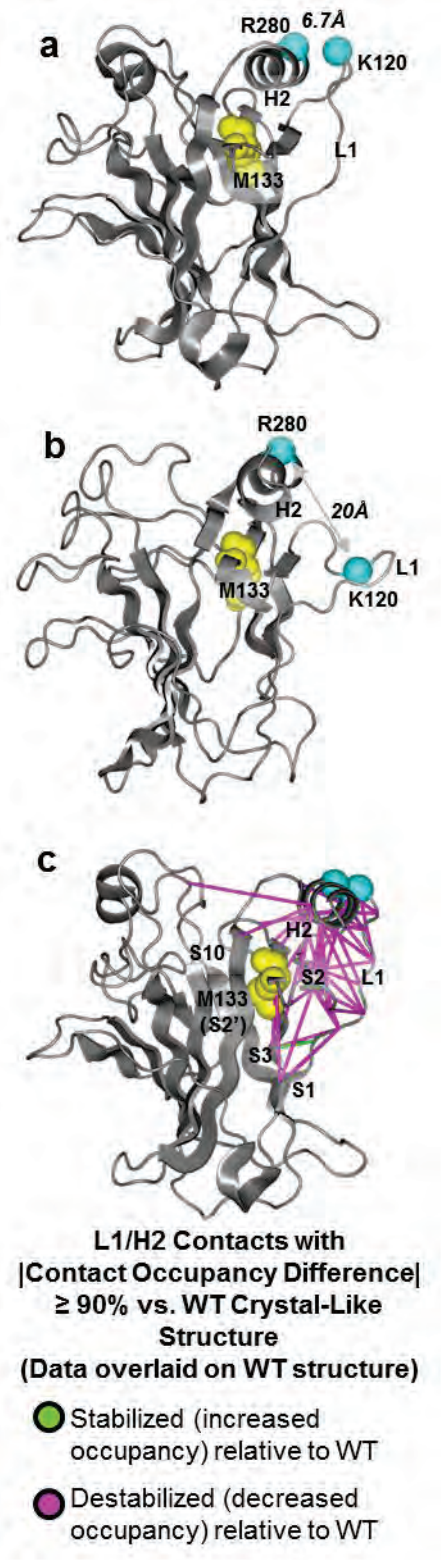


Figure 6.4 p53 structures highlighting disrupted loop-sheet-helix region contacts
Cα atoms of K120 and R280 are colored cyan, stabilizing mutation site M133 is colored yellow. (a) p53 crystal structure (2ocj). K120/R280 distance is 6.7Å. (b) p53 wild-type MD structure at 80 ns. K120/R280 distance is 20Å. (c) p53 crystal structure overlaid with contacts common to all 17 L1/H2-separated conformations. Green lines indicate contacts that were present in the separated conformations but not in the crystal-like wild-type conformation. Magenta lines indicate contacts that were not present in the separated conformations and were present in the crystal-like wild-type conformation. Appendix Table D.3 specifies which contacts were present in the separated and non-separated conformations.

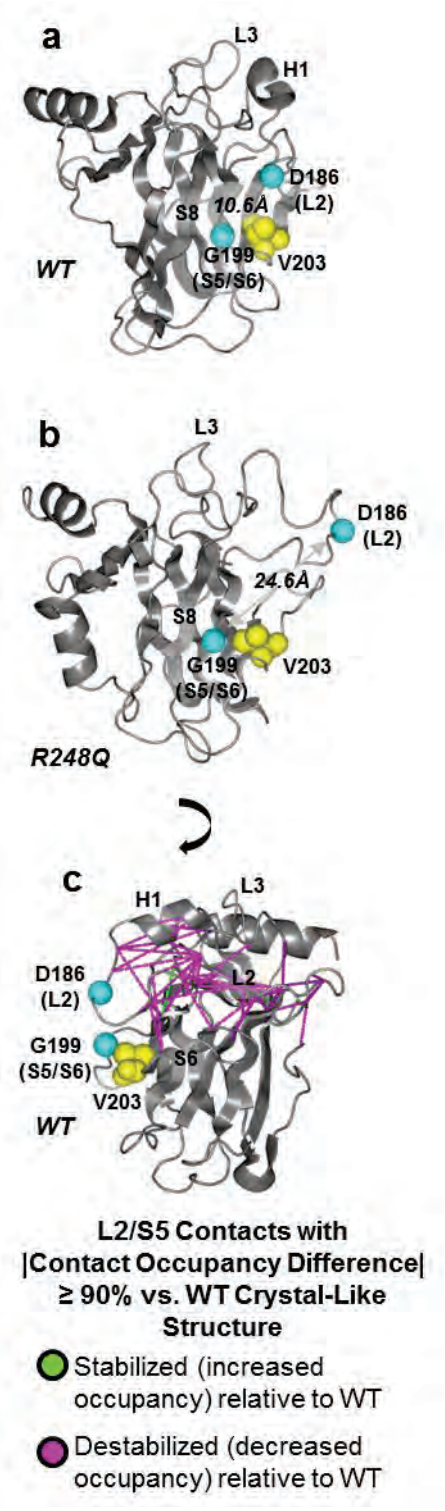


Figure 6.5 p53 structures highlighting disrupted L2 and S5 region contacts
Cα atoms of D186 and G199 are colored cyan, stabilizing mutation site V203 is colored yellow. (a) p53 crystal structure (2ocj). D186/G199 distance is 10.6Å. (b) p53 R248Q mutant MD structure at 80 ns. D186/G199 distance is 24.6Å. (c) p53 crystal structure overlaid with contacts common to all 14 L2/S5-separated conformations. Green lines indicate contacts that were present in the separated conformations but not in the crystal-like wild-type conformation. Magenta lines indicate contacts that were not present in the separated conformations and were present in the crystal-like wild-type conformation. Appendix Table D.4 specifies which contacts were present in the separated and non-separated conformations.

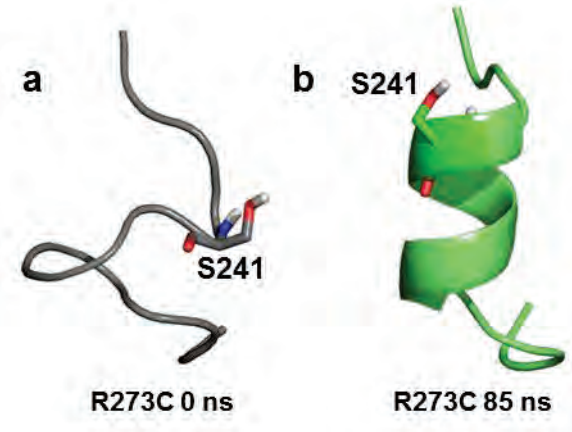


Figure 6.6 Highlight of novel helix in L3 region
L3 region (residues 237-250) of R273C mutant. (a) 0ns (b) 85ns. S241 Cα atom was displaced by 7Å between the two time points.

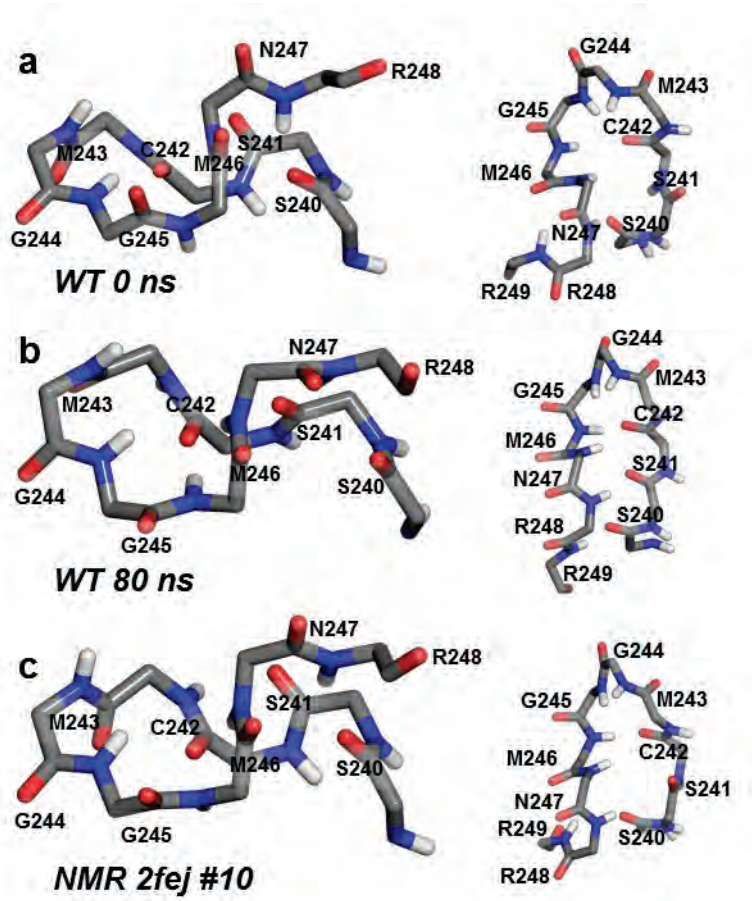


Figure 6.7 L3 main chain atoms highlighting α-sheet-like orientation of residues
Different perspectives of the same residues are shown on the left and right. Labels refer to the backbone carbonyl oxygens. (a) Wild-type starting structure (b) Wild type at 80ns (c) NMR structure 2fej model 10.

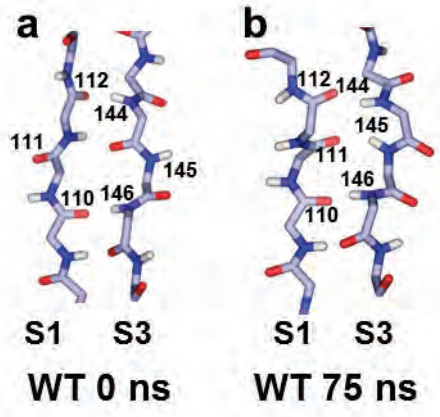


Figure 6.8 Wild-type simulation displaying α-sheet conformation
(a) Strands S1 and S3 displaying β-sheet conformation at 0 ns. (b) Strands S1 and S3 displaying α-sheet conformation at 75 ns.

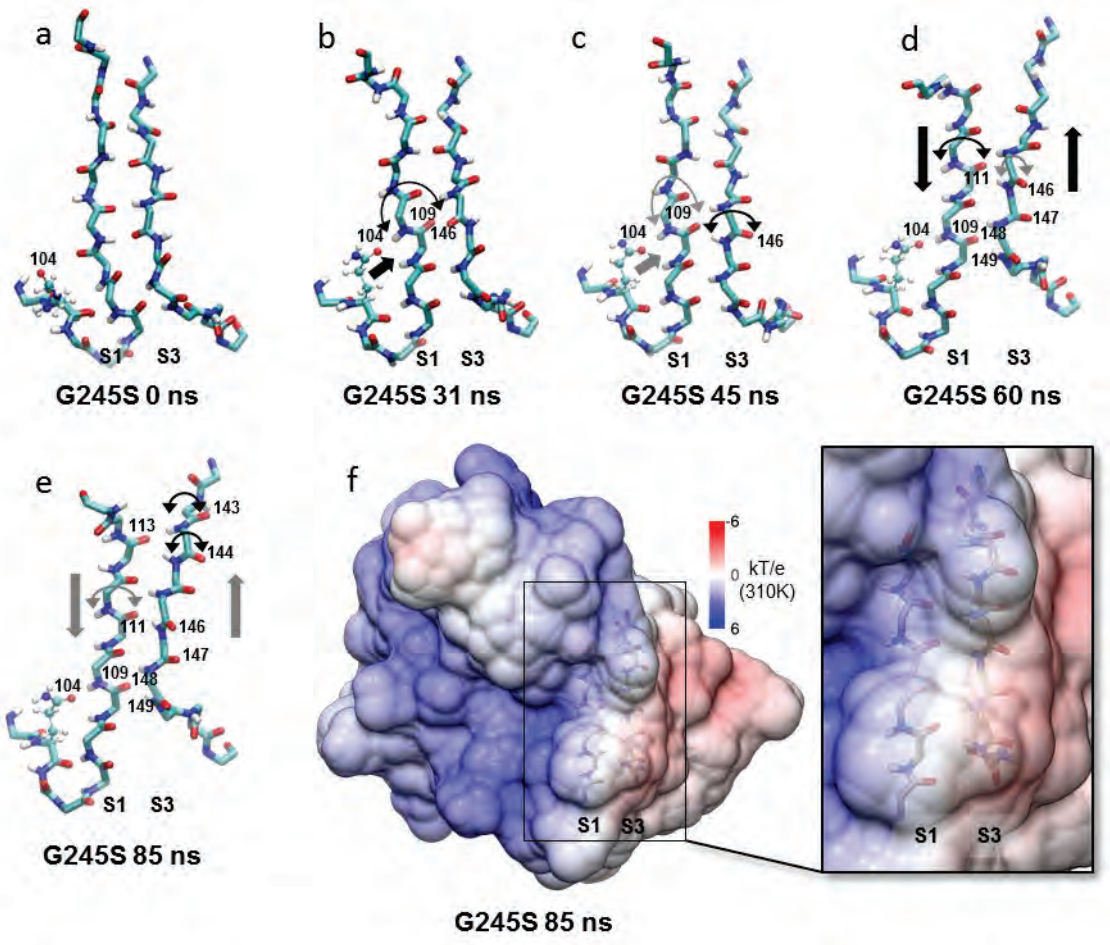


Figure 6.9 Proposed G245S α-sheet formation model
Arrows indicate new motion and fade over time. (a) 0 ns (b) 31ns. N-terminal Q104:OE1 orients to F109:O, causing it to rotate and face W146:O. Main chain hydrogens of F109 and R110 now face Q104:OE1. (c) 45 ns. W146 rotates away from F109. (d) 60ns. L111:O rotates inward and S1 and S3 slide past each other, finding a stable conformation one residue offset from previous conformation. (e) 85 ns. Q144 and V143 rotate and move toward F113. Note that Q104:OE1, F109:H and R110:H, once in position, remain stable for the entire process. (f) Electrostatic surface showing solvent-accessible charge-separation across the α-sheet.

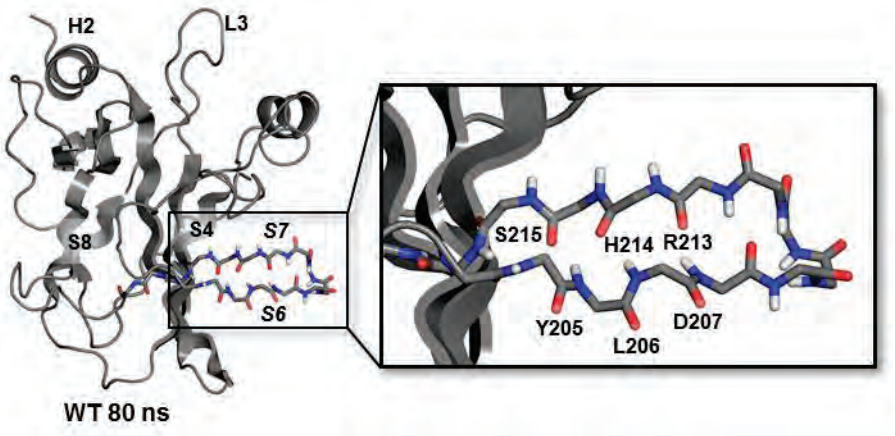


Figure 6.10 Wild type S6/S7 α-sheet
S6/S7 α-sheet was present for 91% of the time. Structure shown at 80 ns.

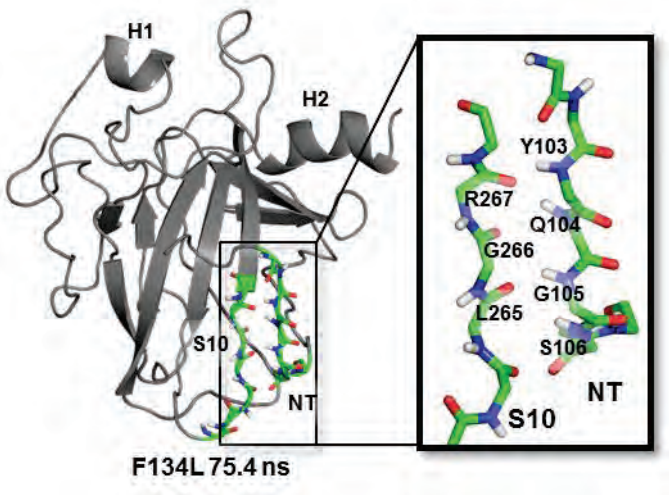


Figure 6.11 Novel N-terminal α-sheet
F134L mutant at 75.4 ns. α-sheet formed for 1% of the time. G105 and G266 α conformations were correlated with R = 0.91.

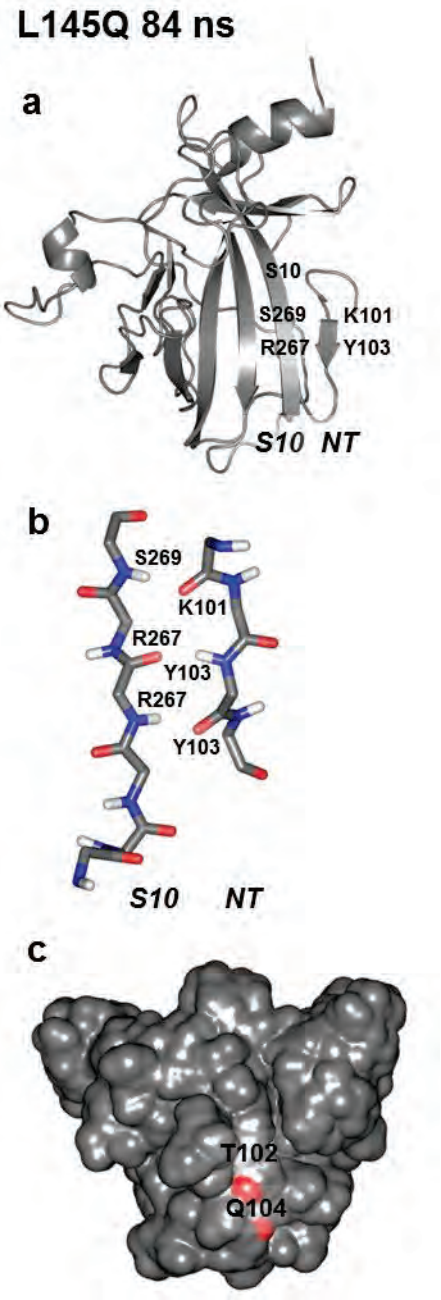


Figure 6.12 Novel N-terminal β-sheet
L145Q mutant at 84 ns with novel N-terminal β-strand spanning residues 101-103 and aligned with S10. (a) Overview showing location of novel strand in protein. (b) View showing alignment of β-strand backbone atoms. (c) View showing solvent accessible edge of the novel N-terminal β strand.

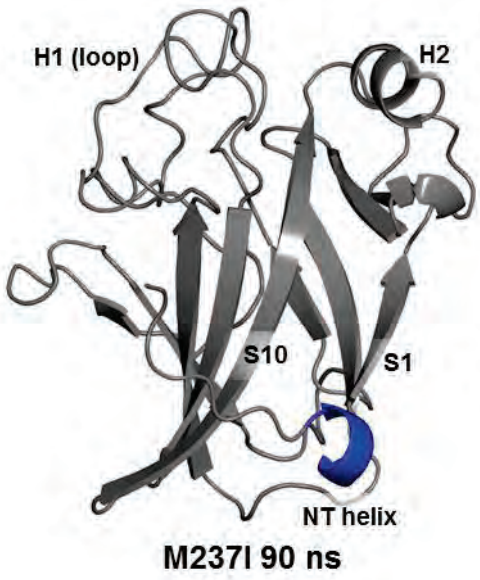


Figure 6.13 Novel N-terminal helix
M237I mutant at 90 ns. N-terminal helix is colored blue; other structures are noted for orientation; note that H1 has lost helical content.

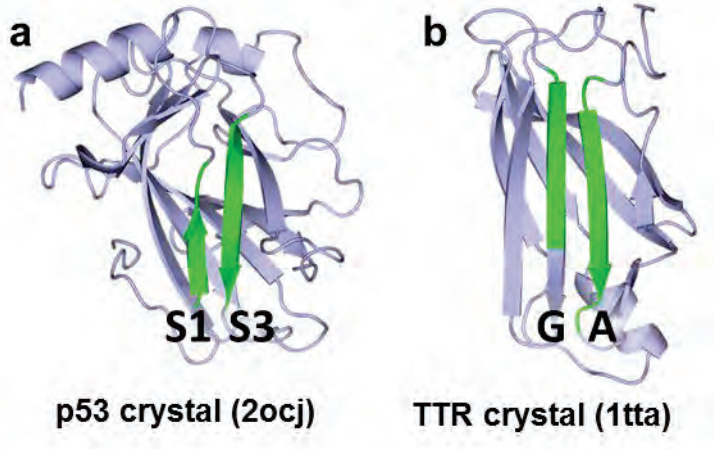


Figure 6.14 p53 and transthyretin (TTR)
p53 S1 aligns with TTR strand G and p53 S3 aligns with TTR strand A. Aligned α-strand residues are colored green. (a) p53 crystal (PDB: 2ocj) (b) TTR crystal (PDB: 1tta).

BIBLIOGRAPHY

Alberts, B. et al. 2008. *Molecular Biology of the Cell*. 5th edition. NY, NY: Garland Science.

Ano Bom, A. P. D. et al. 2010. "The p53 Core Domain Is a Molten Globule at Low pH Functional Implications Of A Partially Unfolded Structure." *Journal of Biological Chemistry* 285 (4): 2857–66.

Ano Bom, A. P. D. et al. 2012. "Mutant p53 Aggregates into Prion-like Amyloid Oligomers and Fibrils Implications for Cancer." *Journal of Biological Chemistry* 287 (33): 28152–62.

Armen, R. S., M. L. DeMarco, et al. 2004. "Pauling and Corey's α-Pleated Sheet Structure May Define the Prefibrillar Amyloidogenic Intermediate in Amyloid Disease." *Proceedings of the National Academy of Sciences of the United States of America* 101 (32): 11622–27.

Armen, R. S., D. O. V. Alonso, et al. 2004. "Anatomy of an Amyloidogenic Intermediate: Conversion of β-Sheet to α-Sheet Structure in Transthyretin at Acidic pH." *Structure* 12 (10): 1847–63.

Ashburner, M. et al. 2000. "Gene Ontology: Tool for the Unification of Biology." *Nature Genetics* 25 (1): 25–29.

Baker, N. A. et al. 2001. "Electrostatics of Nanosystems: Application to Microtubules and the Ribosome." *Proceedings of the National Academy of Sciences of the United States of America* 98 (18): 10037–41.

Barakat, K. et al. 2011. "Effects of Temperature on the p53-DNA Binding Interactions and Their Dynamical Behavior: Comparing the Wild Type to the R248Q Mutant." *PLoS ONE* 6 (11): e27651.

Baroni, T. E. et al. 2004. "A Global Suppressor Motif for p53 Cancer Mutants." *Proceedings of the National Academy of Sciences of the United States of America* 101 (14): 4930–35.

Basse, N. et al. 2010. "Toward the Rational Design of p53-Stabilizing Drugs: Probing the Surface of the Oncogenic Y220C Mutant." *Chemistry & Biology* 17 (1): 46–56.

Beck, D. A. C. et al. 2000-2014. "ilmm, in lucem Molecular Mechanics."

Beck, D. A. C. et al. 2005. "Cutoff Size Need Not Strongly Influence Molecular Dynamics Results for Solvated Polypeptides." *Biochemistry* 44 (2): 609–16.

Beck, D. A. C. et al. 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): 353–68.

Beck, D. A. C., and V. Daggett. 2004. "Methods for Molecular Dynamics Simulations of Protein Folding/Unfolding in Solution." *Methods* 34 (1): 112–20.

Berman, H. M. et al. 2000. "The Protein Data Bank." *Nucleic Acids Research* 28 (1): 235–42.

Bernard, B. 2006. *Incorporation of Protein Flexibility in Structure-Guided Drug Design*. Seattle, WA, USA: Thesis MSB - University of Washington.

Bernstein, F. C. et al. 1977. “The Protein Data Bank: A Computer-Based Archival File for Macromolecular Structures.” *Journal of Molecular Biology* 112 (3): 535–42.

Boeckler, F. M. et al. 2008. “Targeted Rescue of a Destabilized Mutant of p53 by an in silico Screened Drug.” *Proceedings of the National Academy of Sciences of the United States of America* 105 (30): 10360–65.

Boeckler, F. M. et al. 2011. “Compounds for Use in Stabilizing p53 Mutants.” Patent US20110059953 A1.

Brachmann, R. K. et al. 1998. “Genetic Selection of Intragenic Suppressor Mutations That Reverse the Effect of Common p53 Cancer Mutations.” *The EMBO Journal* 17 (7): 1847–59.

Bromley, D. et al. 2013. “Structural Consequences of Mutations to the α-Tocopherol Transfer Protein Associated with the Neurodegenerative Disease Ataxia with Vitamin E Deficiency.” *Biochemistry* 52 (24): 4264–73.

Bromley, D. et al. 2014. “DIVE: A Data Intensive Visualization Engine.” *Bioinformatics* 30 (4): 593–95.

Bullock, A. N. et al. 1997. “Thermodynamic Stability of Wild-Type and Mutant p53 Core Domain.” *Proceedings of the National Academy of Sciences of the United States of America* 94 (26): 14338–42.

Bullock, A. N. et al. 2000. “Quantitative Analysis of Residual Folding and DNA Binding in Mutant p53 Core Domain: Definition of Mutant States for Rescue in Cancer Therapy.” *Oncogene* 19 (10): 1245–56.

Bullock, A. N., and A. R. Fersht. 2001. “Rescuing the Function of Mutant p53.” *Nature Reviews Cancer* 1 (1): 68–76.

Butler, J. S., and S. N. Loh. 2003. “Structure, Function, and Aggregation of the Zinc-Free Form of the p53 DNA Binding Domain.” *Biochemistry* 42 (8): 2396–2403.

Bykov, V. J. N. et al. 2002. “Restoration of the Tumor Suppressor Function to Mutant p53 by a Low-Molecular-Weight Compound.” *Nature Medicine* 8 (3): 282–88.

Calhoun, S., and V. Daggett. 2011. “Structural Effects of the L145Q, V157F, and R282W Cancer-Associated Mutations in the p53 DNA-Binding Core Domain.” *Biochemistry* 50 (23): 5345–53.

Cañadillas, J. M. P. et al. 2006. “Solution Structure of p53 Core Domain: Structural Basis for Its Instability.” *Proceedings of the National Academy of Sciences of the United States of America* 103 (7): 2109–14.

Cavalier, L. et al. 1998. “Ataxia with Isolated Vitamin E Deficiency: Heterogeneity of Mutations and Phenotypic Variability in a Large Number of Families.” *The American Journal of Human Genetics* 62 (2): 301–10.

Chen, W. et al. 2014. “Structural and Dynamic Properties of the Human Prion Protein.” *Biophysical Journal* 106 (5): 1152–63.

Cho, Y. et al. 1994. “Crystal Structure of a p53 Tumor Suppressor-DNA Complex: Understanding Tumorigenic Mutations.” *Science* 265 (5170): 346–55.

Cock, P. J. A. et al. 2009. “Biopython: Freely Available Python Tools for Computational Molecular Biology and Bioinformatics.” *Bioinformatics* 25 (11): 1422–23.

Conn, P. M. 2005. “Pharmacoperones for Correcting Disease States Involving Protein Misfolding.” Patent US7842470 B2.

Conn, P. M. et al. 2014. “Transitioning Pharmacoperones to Therapeutic Use: In Vivo Proof-of-Principle and Design of High Throughput Screens.” *Pharmacological Research*, Pharmacological Chaperones: On the Frontier of 21st Century Therapeutics, 83 (May): 38–51.

Daggett, V. 2006. “α-Sheet: The Toxic Conformer in Amyloid Diseases?” *Accounts of Chemical Research* 39 (9): 594–602.

Day, R. et al. 2003. “A Consensus View of Fold Space: Combining SCOP, CATH, and the Dali Domain Dictionary.” *Protein Science* 12 (10): 2150–60.

DeLano, W. L. 2002. “The PyMOL Molecular Graphics System.” *http://www.pymol.org*.

Demir, O. et al. 2011. “Ensemble-Based Computational Approach Discriminates Functional Activity of p53 Cancer and Rescue Mutants.” *PLoS Computational Biology* 7 (10): e1002238.

Di Blasio, B. et al. 1994. “A Crystal Structure with Features of an Antiparallel α-Pleated Sheet.” *Biopolymers* 34 (11): 1463–68.

Di Donato, I. et al. 2010. “Ataxia with Vitamin E Deficiency: Update of Molecular Diagnosis.” *Neurological Sciences* 31 (4): 511–15.

Duan, J., and L. Nilsson. 2006. “Effect of Zn2+ on DNA Recognition and Stability of the p53 DNA-Binding Domain.” *Biochemistry* 45 (24): 7483–92.

Eldar, A. et al. 2013. “Structural Studies of p53 Inactivation by DNA-Contact Mutations and Its Rescue by Suppressor Mutations via Alternative Protein-DNA Interactions.” *Nucleic Acids Research* 41 (18): 8748–59.

Gansner, E. R., and S. C. North. 2000. “An Open Graph Visualization System and Its Applications to Software Engineering.” *Software Practice and Experience* 30 (11): 1203–33.

Gavrin, L. K. et al. 2012. “Small Molecules That Target Protein Misfolding.” *Journal of Medicinal Chemistry* 55 (24): 10823–43.

Glabe, C. G., and R. Kayed. 2006. “Common Structure and Toxic Function of Amyloid Oligomers Implies a Common Mechanism of Pathogenesis.” *Neurology* 66 (2 suppl 1): S74–78.

Goecks, J. et al. 2010. “Galaxy: A Comprehensive Approach for Supporting Accessible, Reproducible, and Transparent Computational Research in the Life Sciences.” *Genome Biology* 11 (8): R86.

Gogna, R. et al. 2012. “Chaperoning of Mutant p53 Protein by Wild-Type p53 Protein Causes Hypoxic Tumor Regression.” *Journal of Biological Chemistry* 287 (4): 2907–14.

Hamilton, J. A. et al. 1993. “The X-Ray Crystal Structure Refinements of Normal Human Transthyretin and the Amyloidogenic Val-30→Met Variant to 1.7-Å Resolution.” *The Journal of Biological Chemistry* 268 (4): 2416–24.

Harris, C. C., and M. Hollstein. 1993. “Clinical Implications of the p53 Tumor-Suppressor Gene.” *New England Journal of Medicine* 329 (18): 1318–27.

Hopping, G. et al. 2014. “Designed α-Sheet Peptides Inhibit Amyloid Formation by Targeting Toxic Oligomers.” *eLife* 3 (July): e01681.

Horrocks, I. 2008. “Ontologies and the Semantic Web.” *Communications of the ACM* 51 (12): 58–67.

Humphrey, W. et al. 1996. “VMD: Visual Molecular Dynamics.” *Journal of Molecular Graphics* 14 (1): 33–38.

Hu, Y. 2005. “Efficient, High-Quality Force-Directed Graph Drawing.” *Mathematica Journal* 10 (1): 37–71.

Irwin, J. J. et al. 2009. “Automated Docking Screens: A Feasibility Study.” *Journal of Medicinal Chemistry* 52 (18): 5712–20.

Irwin, J. J., and B. K. Shoichet. 2005. “ZINC-a Free Database of Commercially Available Compounds for Virtual Screening.” *Journal of Chemical Information and Modeling* 45 (1): 177–82.

Ishimaru, D. et al. 2003. “Conversion of Wild-Type p53 Core Domain into a Conformation that Mimics a Hot-Spot Mutant.” *Journal of Molecular Biology* 333 (2): 443–51.

Joerger, A. C. et al. 2004. “Crystal Structure of a Superstable Mutant of Human p53 Core Domain Insights Into The Mechanism Of Rescuing Oncogenic Mutations.” *Journal of Biological Chemistry* 279 (2): 1291–96.

Joerger, A. C. et al. 2005. “Structures of p53 Cancer Mutants and Mechanism of Rescue by Second-Site Suppressor Mutations.” *Journal of Biological Chemistry* 280 (16): 16030–37.

Joerger, A. C. et al. 2006. “Structural Basis for Understanding Oncogenic p53 Mutations and Designing Rescue Drugs.” *Proceedings of the National Academy of Sciences of the United States of America* 103 (41): 15056–61.

Joerger, A. C., and A. R. Fersht. 2010. “The Tumor Suppressor p53: From Structures to Drug Discovery.” *Cold Spring Harbor Perspectives in Biology* 2 (6): a000919.

Jones, E. et al. 2001. “SciPy: Open Source Scientific Tools for Python.” *http://www.scipy.org/*.

Kabsch, W., and C. Sander. 1983. “Dictionary of Protein Secondary Structure: Pattern Recognition of Hydrogen-Bonded and Geometrical Features.” *Biopolymers* 22 (12): 2577–2637.

Keim, D. A. et al. 2008. “Visual Analytics: Scope and Challenges.” In *Visual Data Mining: Theory, Techniques and Tools for Visual Analytics*, edited by Simeon J. Simoff et al., 4404:76–90. Lecture Notes in Computer Science (LNCS). Berlin Heidelberg, Germany: Springer-Verlag.

Kell, G. S. 1967. “Precise Representation of Volume Properties of Water at One Atmosphere.” *Journal of Chemical and Engineering Data* 12 (1): 66–69.

Khatib, F. et al. 2011. “Crystal Structure of a Monomeric Retroviral Protease Solved by Protein Folding Game Players.” *Nature Structural & Molecular Biology* 18 (10): 1175–77.

Lasagna-Reeves, C. A. et al. 2013. “Dual Role of p53 Amyloid Formation in Cancer; Loss of Function and Gain of Toxicity.” *Biochemical and Biophysical Research Communications* 430 (3): 963–68.

Lehmann, S. et al. 2012. "Targeting p53 in Vivo: A First-in-Human Study With p53-Targeting Compound APR-246 in Refractory Hematologic Malignancies and Prostate Cancer." *Journal of Clinical Oncology* 30 (29): 3633–39.

Leidenheimer, N. J., and K. G. Ryder. 2014. "Pharmacological Chaperoning: A Primer on Mechanism and Pharmacology." *Pharmacological Research*, Pharmacological Chaperones: On the Frontier of 21st Century Therapeutics, 83 (May): 10–19.

Levitt, M. 1990. *ENCAD-Energy Calculations and Dynamics. Molecular Applications Group*. Palo Alto, CA and Rehovot, Israel: Stanford University and Yeda.

Levitt, M. et al. 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): 215–31.

Levitt, M. et al. 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): 5051–61.

Levy, C. B. et al. 2011. "Co-Localization of Mutant p53 and Amyloid-like Protein Aggregates in Breast Tumors." *The International Journal of Biochemistry & Cell Biology* 43 (1): 60–64.

Liu, T. et al. 2005. "Structural Insights into the Cellular Retinaldehyde Binding Protein (CRALBP)." *Proteins: Structure, Function, and Bioinformatics* 61 (2): 412–22.

Loh, S. N. 2010. "The Missing Zinc: p53 Misfolding and Cancer." *Metallomics* 2 (7): 442–49.

Lorber, D. M., and B. K. Shoichet. 1998. "Flexible Ligand Docking Using Conformational Ensembles." *Protein Science* 7 (4): 938–50.

Lorber, D. M., and B. K. Shoichet. 2005. "Hierarchical Docking of Databases of Multiple Ligand Conformations." *Current Topics in Medicinal Chemistry* 5 (8): 739–49.

Lukman, S. et al. 2013. "Mapping the Structural and Dynamical Features of Multiple p53 DNA Binding Domains: Insights into Loop 1 Intrinsic Dynamics." *PLoS ONE* 8 (11): e80221.

Martin-Sanchez, F., and K. Verspoor. 2014. "Big Data in Medicine Is Driving Big Changes." *Yearbook of Medical Informatics* 9 (1): 14–20.

Meier, R. et al. 2003. "The Molecular Basis of Vitamin E Retention: Structure of Human α-Tocopherol Transfer Protein." *Journal of Molecular Biology* 331 (3): 725–34.

Morley, S. et al. 2004. "Molecular Determinants of Heritable Vitamin E Deficiency." *Biochemistry* 43 (14): 4143–49.

Morris, G. M. et al. 2009. "AutoDock 4 and AutoDockTools 4: Automated Docking with Selective Receptor Flexibility." *Journal of Computational Chemistry* 30 (16): 2785–91.

Musen, M. A. et al. 2012. "The National Center for Biomedical Ontology." *Journal of the American Medical Informatics Association* 19 (2): 190–95.

Natan, E. et al. 2011. "Interaction of the p53 DNA-Binding Domain with Its N-Terminal Extension Modulates the Stability of the p53 Tetramer." *Journal of Molecular Biology* 409 (3): 358–68.

Nikolova, P. V. et al. 1998. "Semirational Design of Active Tumor Suppressor p53 DNA Binding Domain with Enhanced Stability." *Proceedings of the National Academy of Sciences of the United States of America* 95 (25): 14675–80.

O'Boyle, N. M. et al. 2011. "Open Babel: An Open Chemical Toolbox." *Journal of Cheminformatics* 3 (1): 1–14.

Olivier, M. et al. 2002. "The IARC TP53 Database: New Online Mutation Analysis and Recommendations to Users." *Human Mutation* 19 (6): 607–14.

OSGi Service Platform, Release 3. 2003. IOS Press, Inc.

Panagabko, C. et al. 2003. "Ligand Specificity in the CRAL-TRIO Protein Family." *Biochemistry* 42 (21): 6467–74.

Pauling, L., and R. B. Corey. 1951. "The Pleated Sheet, a New Layer Configuration of Polypeptide Chains." *Proceedings of the National Academy of Sciences of the United States of America* 37 (5): 251–56.

Petitjean, A. et al. 2007. "Impact of Mutant p53 Functional Properties on TP53 Mutation Patterns and Tumor Phenotype: Lessons from Recent Developments in the IARC TP53 Database." *Human Mutation* 28 (6): 622–29.

Pettersen, E. F. et al. 2004. "UCSF Chimera - A Visualization System for Exploratory Research and Analysis." *Journal of Computational Chemistry* 25 (13): 1605–12.

Rakhit, R., and A. Chakrabartty. 2006. "Structure, Folding, and Misfolding of Cu,Zn Superoxide Dismutase in Amyotrophic Lateral Sclerosis." *Biochimica et Biophysica Acta (BBA) - Molecular Basis of Disease* 1762 (11–12): 1025–37.

Rangel, L. P. et al. 2014. "The Aggregation of Mutant p53 Produces Prion-like Properties in Cancer." *Prion* 8 (1): 75–84.

Ryan, M. M. et al. 2007. "Conformational Dynamics of the Major Yeast Phosphatidylinositol Transfer Protein Sec14p: Insight into the Mechanisms of Phospholipid Exchange and Diseases of Sec14p-Like Protein Deficiencies." *Molecular Biology of the Cell* 18 (5): 1928–42.

Rysavy, S. J. et al. 2014. "DIVE - A Graph-Based Visual Analytics Framework for Big Data." *IEEE Computer Graphics and Applications* 34 (2): 26–37.

Schaaf, G. et al. 2011. "Resurrection of a Functional Phosphatidylinositol Transfer Protein from a Pseudo-Sec14 Scaffold by Directed Evolution." *Molecular Biology of the Cell* 22 (6): 892–905.

Schaeffer, R. D. et al. 2011. "Generation of a Consensus Protein Domain Dictionary." *Bioinformatics* 27 (1): 46–54.

Schames, J. R. et al. 2004. "Discovery of a Novel Binding Trench in HIV Integrase." *Journal of Medicinal Chemistry* 47 (8): 1879–81.

Schmidlin, T. et al. 2009. "Structural Changes to Monomeric CuZn Superoxide Dismutase Caused by the Familial Amyotrophic Lateral Sclerosis-Associated Mutation A4V." *Biophysical Journal* 97 (6): 1709–18.

Schroeder, W. et al. 1996. *The Visualization Toolkit: An Object-Oriented Approach to 3-D Graphics*. Upper Saddle River, NJ: Prentice Hall.

Schuurman, N., and A. Leszczynski. 2008. "Ontologies for Bioinformatics." *Bioinformatics and Biology Insights* 2 (March): 187–200.

Shannon, P. et al. 2003. "Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks." *Genome Research* 13 (11): 2498–2504.

Silva, J. L. et al. 2013. “Expanding the Prion Concept to Cancer Biology: Dominant-Negative Effect of Aggregates of Mutant p53 Tumour Suppressor.” *Bioscience Reports* 33 (4): 593–603.

Simms, A. M. et al. 2008. “Dynameomics: Design of a Computational Lab Workflow and Scientific Data Repository for Protein Simulations.” *Protein Engineering Design and Selection* 21 (6): 369–77.

Simms, A. M., and V. Daggett. 2012. “Protein Simulation Data in the Relational Model.” *The Journal of Supercomputing* 62 (1): 150–73.

Suad, O. et al. 2009. “Structural Basis of Restoring Sequence-Specific DNA Binding and Transactivation to Mutant p53 by Suppressor Mutations.” *Journal of Molecular Biology* 385 (1): 249–65.

Thomas, J. J., and K. A. Cook. 2005. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. Vol. 54. Los Alamitos: IEEE CS Press.

Toofanny, R. D., and V. Daggett. 2012. “Understanding Protein Unfolding from Molecular Simulations.” *Wiley Interdisciplinary Reviews: Computational Molecular Science* 2 (3): 405–23.

Trott, O., and A. J. Olson. 2010. “AutoDock Vina: Improving the Speed and Accuracy of Docking with a New Scoring Function, Efficient Optimization, and Multithreading.” *Journal of Computational Chemistry* 31 (2): 455–61.

Van der Kamp, M. W. et al. 2010. “Dynameomics: A Comprehensive Database of Protein Dynamics.” *Structure* 18 (4): 423–35.

Vogelstein, B. et al. 2000. “Surfing the p53 Network.” *Nature* 408 (6810): 307–10.

Wallentine, B. D. et al. 2013. “Structures of Oncogenic, Suppressor and Rescued p53 Core-Domain Variants: Mechanisms of Mutant p53 Rescue.” *Acta Crystallographica Section D: Biological Crystallography* 69 (10): 2146–56.

Wang, S. et al. 2013. “Protein Structure Alignment beyond Spatial Proximity.” *Scientific Reports* 3 (1448): 1–7.

Wang, Y. et al. 2007. “Structure of the Human p53 Core Domain in the Absence of DNA.” *Acta Crystallographica Section D: Biological Crystallography* 63 (3): 276–81.

Wang, Z., and J. Moult. 2001. “SNPs, Protein Structure, and Disease.” *Human Mutation* 17 (4): 263–70.

Wassman, C. D. et al. 2013. “Computational Identification of a Transiently Open L1/S3 Pocket for Reactivation of Mutant p53.” *Nature Communications* 4 (1407): 1–9.

Weinmann, L. et al. 2008. “A Novel p53 Rescue Compound Induces p53-Dependent Growth Arrest and Sensitises Glioma Cells to Apo2L/TRAIL-Induced Apoptosis.” *Cell Death & Differentiation* 15 (4): 718–29.

Whetzel, P. L. et al. 2011. “BioPortal: Enhanced Functionality via New Web Services from the National Center for Biomedical Ontology to Access and Use Ontologies in Software Applications.” *Nucleic Acids Research* 39 (2): W541–45.

Wilcken, R. et al. 2012. “Halogen-Enriched Fragment Libraries as Leads for Drug Rescue of Mutant p53.” *Journal of the American Chemical Society* 134 (15): 6810–18.

Wilson, C., and O. Gisvold. 2004. *Wilson and Gisvolds Textbook of Organic Medicinal and Pharmaceutical Chemistry*. Edited by John M Beale and John H Block. 11th edition. Philadelphia: Lippincott Williams & Wilkins.

Wolstencroft, K. et al. 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): W557–61.

Xu, J. et al. 2011. "Gain of Function of Mutant p53 by Coaggregation with Multiple Tumor Suppressors." *Nature Chemical Biology* 7 (5): 285–95.

Zhang, W. X. et al. 2011. "The Contribution of Surface Residues to Membrane Binding and Ligand Transfer by the α-Tocopherol Transfer Protein (α-TTP)." *Journal of Molecular Biology* 405 (4): 972–88.

Appendix A

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

The contents of this appendix were supplementary to a previously-published paper in the journal *IEEE Computer Graphics & Applications* (Rysavy et al. 2014).

A.1 Ontologies

An *ontology* is a semantically and syntactically formal structure for organizing information (Horrocks 2008). The need for formal semantics and syntax has grown in recent decades as the size and complexity of organized datasets have grown. In particular, the need for such formalisms is driven by the desire to handle these large and complex datasets programmatically. Ontologies enforce a strict formalism that guarantees structured information to be both meaningful and extensible; once established, such information can be clearly reasoned with, built upon and discussed. An ontology can be represented as a graph comprised of nodes representing specific concepts and edges representing specific relationships.

Efforts like the semantic web hold the promise of establishing a global formal ontology of everything.(Horrocks 2008). While desirable, there is still a great deal of work to be done in smaller, more localized knowledge domains; in biology, for example, over 300 ontologies are currently indexed at the National Center for Biomedical Ontology's BioPortal (Whetzel et al. 2011; Musen et al. 2012; Schuurman and Leszczynski 2008) (http://bioportal.bioontology.org). Projects like

BioPortal demonstrate that ontologies are becoming increasingly popular frameworks for structured information, and that modern data analysis tools must be capable of handling large

A.2 Molecular Dynamics

Molecular dynamics (MD) simulation is a computational method for studying the dynamic behaviors of molecules (Toofanny and Daggett 2012). This method is commonly used to study protein structure and dynamics. Proteins are complex molecules comprised of amino acids (residues), which are themselves comprised of atoms. Contacts exist between atoms when they are within a defined distance from one another.

Proteins are responsible for much of the functional and structural activity in living tissue; protein function within the human body ranges from muscular structure to metabolism to immune response to reproduction. Proteins are fundamental to the human body and understanding how they 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 a protein's three-dimensional structure (see Appendix Figure A.1).

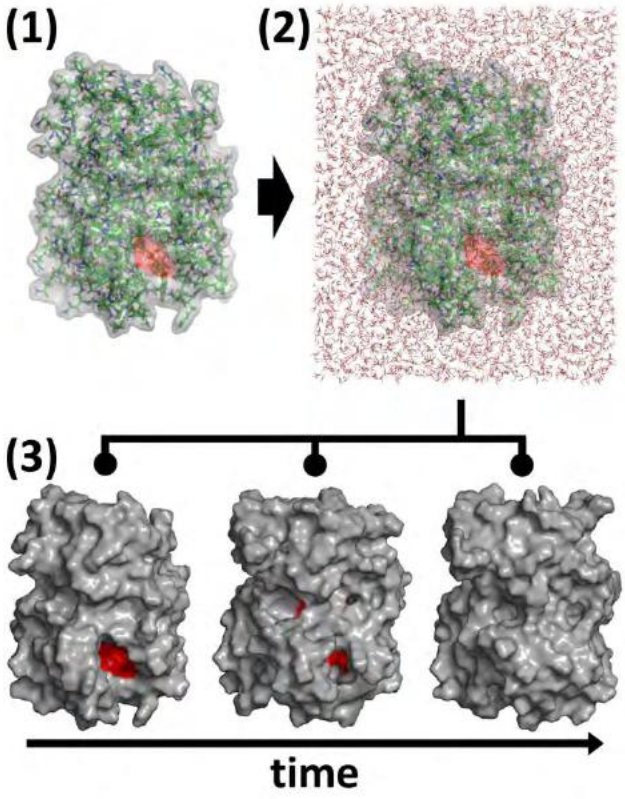
During an MD simulation, we simulate interatomic forces to predict motion among the atoms that comprise a protein and its environment (see Appendix Figure A.1). In most cases, the environment consists of water molecules, although this can be altered to investigate different phenomena. The physical simulation is calculated using Newtonian physics; at specified time intervals, the simulation state is saved, resulting in a series of structural snapshots. Together, this

series is referred to as a *trajectory* and is reflective of 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; protein structures and interactions can be simulated without the cost or risk incurred by laboratory experiments. Second, modern computing techniques allow MD simulations to be run in parallel, enabling virtual high-throughput experimentation. Third, molecular dynamics 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 show us how proteins move, a critical aspect of their functionality. However, MD simulations can produce datasets considerably larger than what most structural biology tools are designed to handle. As computers become more powerful, MD simulations are increasing in size and resolution, and the logistical challenges of storing, analyzing, and visualizing MD data are requiring researchers to consider new analysis techniques.

In the Daggett laboratory at the University of Washington, we study protein dynamics as part of the Dynameomics project (Van der Kamp et al. 2010). The Dynameomics project characterizes the dynamic behaviors and folding pathways of topological classes of all known protein structures. So far, this effort has generated hundreds of terabytes of data consisting of thousands of simulations and millions of structures, as well as their associated analyses. These data are stored in a distributed SQL data warehouse; this warehouse currently holds 10^4 as many protein structures as the Protein Data Bank (Bernstein et al. 1977), the primary repository for

experimentally-characterized protein structures. Dynameomics is currently the largest database of protein structures in the world.



Appendix Figure A.1 Process of solvating and simulating a protein using MD.
(**1**) All-atom depiction of a protein with a transparent surface. (**2**) Same protein solvated and shown in a water box. (**3**) Three structures of interest selected from a trajectory containing over 51,000 frames. The red area shows the functional site of the protein and how it dynamically closes over time.

Appendix B

US PATENT APPLICATION: METHODS FOR EFFICIENT STREAMING OF STRUCTURED INFORMATION

A patent application entitled "Methods for Efficient Streaming of Structured Information" was filed with the US Receiving Office on June 27, 2014. The technologies discussed in this patent were invented over the course of developing DIVE. The US patent application number was PCT/US14/44683. The listed inventors were Dennis Bromley, Steven Rysavy, and Valerie Daggett. As discussed above, Steven Rysavy and I were co-first authors of DIVE. As a result, his PhD dissertation also contains material from this patent application. In addition to sharing the overall design of the DIVE framework, my primary contributions were in the core DIVE kernel and encompassed such technologies as the ontological data structure, μ-scripting, the DIVE pipeline and associated pipeline-plugin technologies, the DIVE GUI, the majority of the visualization plugins, interactive SQL, data-streaming protocols, and internal analytical software-libraries such as mathematical libraries and signal processing libraries. The contents of the DIVE-technology patent application are enclosed here.

Images and written content contained in the patent application were originally published in the two DIVE publications (Bromley et al. 2014; Rysavy et al. 2014), the contents of which are contained in Chapter 2 and Chapter 3.

MBHB Case No. 14-883-PCT

UW Case No. 46518.02WO2

METHODS FOR EFFICIENT STREAMING OF STRUCTURED INFORMATION

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] The present application claims priority to U.S. Provisional Patent Application No. 61/840,617, entitled "Methods for Efficient Streaming of Structured Information", filed June 28, 2013, which is entirely incorporated by reference herein for all purposes.

STATEMENT OF GOVERNMENT RIGHTS

[0002] This invention was made with government support under Grant Nos. 5T15LM007442 and GM50789, awarded by the National Institutes of Health. The government has certain rights in the invention.

BACKGROUND

[0003] The advent of massive networked computing resources has enabled virtually unlimited data collection, storage and analysis for projects such as low-cost genome sequencing, high-precision molecular dynamics simulations and high-definition imaging data for radiology, to name just a few examples. The resulting large, complex datasets known as "big data" make data processing difficult or impossible using database management software from one computer. Big data are becoming increasingly present in many aspects of society and technology including

health care, science, industry and government. Many of these large, complex data sets are best understood when analyzed in a structured manner.

[0004] One such structured manner is to use an ontology for a data set, which is a structured representation of the data in that data set. Although not new *per se*, the use of ontologies is growing in the presence of modern computer technologies. For example, the semantic web is a very compelling, yet nascent and underdeveloped, example use of ontologies for data sets. The paradigms of big data and ontologies are likely to become more important. These paradigms have worked well together, such as in the field of *visual analytics*, which uses interactive visual techniques to interact with big data.

[0005] Ontologies also enable formal analysis, which helps with semantic correctness, interoperability, and can bring much needed insight. Ontologies can be applied to complex, multi-dimensional, and/or large data sets. But the development of data-specific, formal ontologies can be very difficult.

SUMMARY

[0006] In one aspect, a method is provided. A computing device receives data from one or more data sources. The computing device generates a data frame based on the received data. The data frame includes a plurality of data items. The computing device determines a data ontology, where the data ontology includes a plurality of datanodes. The computing device determines a plurality of data pins. A first data pin of the plurality of data pins includes a first reference and a second reference. The first reference for the first data pin refers to a first data item in the data frame and the second reference for the first data pin refers to a first datanode of the plurality of

datanodes. The first datanode is related to the first data item. The computing device obtains data for the first data item at the first datanode of the data ontology via the first data pin. The computing device provides a representation of the data ontology.

[0007] In another aspect, a computing device is provided. The computing device includes a processor and a tangible computer readable medium. The tangible computer readable medium is configured to store at least executable instructions. The executable instructions, when executed by the processor, cause the computing device to perform functions including: receiving data from one or more data sources; generating a data frame based on the received data, the data frame including a plurality of data items; determining a data ontology, where the data ontology includes a plurality of datanodes; determining a plurality of data pins, where a first data pin of the plurality of data pins includes a first reference and a second reference, where the first reference for the first data pin refers to a first data item in the data frame, where the second reference for the first data pin refers to a first datanode of the plurality of datanodes, and where the first datanode is related to the first data item; obtaining data for the first data item at the first datanode of the data ontology via the first data pin; and providing a representation of the data ontology.

[0008] In another aspect, a tangible computer readable medium is provided. The tangible computer readable medium is configured to store at least executable instructions. The executable instructions, when executed by a processor of a computing device, cause the computing device to perform functions including: receiving data from one or more data sources; generating a data frame based on the received data, the data frame including a plurality of data items; determining

a data ontology, where the data ontology includes a plurality of datanodes; determining a plurality of data pins, where a first data pin of the plurality of data pins includes a first reference and a second reference, where the first reference for the first data pin refers to a first data item in the data frame, where the second reference for the first data pin refers to a first datanode of the plurality of datanodes, and where the first datanode is related to the first data item; obtaining data for the first data item at the first datanode of the data ontology via the first data pin; and providing a representation of the data ontology.

[0009] In another aspect, a device is provided. The device includes means for receiving data from one or more data sources; means for generating a data frame based on the received data, the data frame including a plurality of data items; means for determining a data ontology, where the data ontology includes a plurality of datanodes; means for determining a plurality of data pins, where a first data pin of the plurality of data pins includes a first reference and a second reference, where the first reference for the first data pin refers to a first data item in the data frame, where the second reference for the first data pin refers to a first datanode of the plurality of datanodes, and where the first datanode is related to the first data item; means for obtaining data for the first data item at the first datanode of the data ontology via the first data pin; and means for providing a representation of the data ontology.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] Figure 1 shows a high-level representation of the DIVE system receiving data from data sources, in accordance with an embodiment.

[0011] Figure 2 shows an example architecture for the DIVE system, in accordance with an embodiment.

[0012] Figure 3 shows a pipeline between data sources and the DIVE system, in accordance with an example embodiment.

[0013] Figure 4 shows a DIVE object parser having converted a software object hierarchy to a DIVE data structure, in accordance with an embodiment.

[0014] Figure 5 shows a scenario where the DIVE object parser has translated a code assembly into two ontologies, in accordance with an example embodiment.

[0015] Figure 6 shows examples of interactive SQL streaming and pass-through SQL streaming, in accordance with an example embodiment.

[0016] Figure 7 shows an example protein simulated using molecular dynamics, in accordance with an embodiment.

[0017] Figure 8 shows a data flow using the DIVE system for the Dynameomics project, in accordance with an example embodiment.

[0018] Figure 9 shows an example view of a Protein Dashboard, in accordance with an embodiment.

[0019] Figures 10A and 10B show visualizations related to the respective p53 and SOD1 proteins provided by the DIVE system, in accordance with an embodiment.

[0020] Figure 11 shows example DIVE pipelines, in accordance with an embodiment.

[0021] Figure 12 is a block diagram of an example computing network, in accordance with an embodiment.

[0022] Figure 13A is a block diagram of an example computing device, in accordance with an embodiment.

[0023] Figure 13B depicts an example cloud-based server system, in accordance with an embodiment.

[0024] Figure 14 is a flow chart of an example method, in accordance with an embodiment.

DETAILED DESCRIPTION

Efficient Streaming of Structured Information

[0025] Many modern large-scale projects, such as scientific investigations for bioinformatics research, are generating big data. The 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. Data analysis and integration in large data storage environments can challenge even experienced scientists.

[0026] Many of these large datasets are complex, heterogeneous, and/or incomplete. Most existing domain-specific tools designed for complex heterogeneous datasets are not equipped to visually analyze big data. For example, while powerful scientific toolsets are available, including software libraries such as SciPy, specialized visualization tools such as Chimera, and scientific workflow tools such as Taverna, Galaxy, and the Visualization Toolkit (VTK), some toolsets cannot handle large datasets. Other toolkits have not been updated to handle recent advances in data generation and acquisition.

[0027] DIVE (Data Intensive Visualization Engine) was designed and developed to help fill this technological gap. DIVE includes a software framework intended to facilitate analysis of big data and reduce the time to derive insights from the big data. DIVE employs an interactive, extensible, and adaptable data pipeline to apply visual analytics approaches to heterogeneous, high-dimensional datasets. 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.

[0028] DIVE provides a rich ontologically expressive data representation and a flexible modular streaming-data architecture or pipeline. The DIVE pipeline is accessible to users and software applications through an application programming interface, 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 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.

[0029] A core feature of DIVE's framework is the flexible graph-based data representation. DIVE data are stored as datanodes in a strongly typed ontological network defined by the data. These data can range from a set of unordered numbers to a complex object hierarchy with inheritance and well-defined relationships. Datanodes are software objects that can update both their values and structures at runtime. Furthermore, the datanodes' ontological context can change during runtime. So, DIVE can explore dynamic data sources and handle the impromptu user interactions commonly required for visual analysis.

[0030] Data flow through the system explicitly as a set of datanodes passed down the DIVE pipeline or implicitly as information transferred and transformed through the data relationships. Data from any domain may enter the DIVE pipeline, allowing DIVE to operate on a wide variety of datasets, such as, but not limited to, protein simulations, gene ontology, professional baseball statistics, and streaming sensor data.

[0031] 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, ontologies from disparate sources can be merged. Each ontology can be represented as DIVE datanodes and dataedges. Then, the ontologies can be merged through property inheritance. This allows ontologies to inherit definitions from each other, resulting in a new, merged ontology compatible with multiple data sources and amenable to new analytical approaches.

[0032] DIVE includes a DIVE object parser with the ability to parse a .NET object or assembly distinct from the DIVE framework. Use of the DIVE object parser can circumvent

addition of DIVE-specific code to existing programs. Further, the DIVE object parser can augment those programs with DIVE capabilities such as graphical interaction and manipulation. In one example (the Dynameomics API), the underlying data structures and the streaming functionality were integrated into a Protein Dashboard tool using the DIVE object parser without modifying the existing API code base, enabling reuse of the same code base in the DIVE framework and in Structured Query Language (SQL) Common Language Runtime implementations and other non-DIVE utilities.

[0033] DIVE supports two general techniques for data streaming: interactive SQL and pass-through SQL. Interactive SQL can effectively provide a flexible visualization frontend for an SQL database or data warehouse. However, for datasets not immediately described by the underlying database schema or other data source, the pass-through SQL approach can be used to stream complex data structures. Use of the pass-through SQL approach can enable use of very large scale datasets. For example, the pass-through SQL approach allowed DIVE to make hundreds of terabytes of structured data immediately accessible to users in a Dynameomics case study. These data can be 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.

[0034] 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.

[0035] Overall, DIVE provides an interactive data-exploration framework that expands on conventional analysis paradigms and self-contained tools. 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, fast, flexible big data visual analytics solutions, such as the herein-described DIVE, can provide a new perspective for projects using large, complex data sets.

DIVE Architecture

[0036] Figure 1 shows a high-level representation of the DIVE system 100 receiving data from data sources 110, in accordance with an embodiment. DIVE system 100 can provide interaction, interoperability, and visualization of data received from data sources 110. DIVE system 100 includes an API whose primary component is the data pipeline, for streaming, transforming, and visualizing large, complex datasets at interactive speeds. The pipeline can be extended with plug-ins; each plug-in can operate independently on the data stream of the pipeline.

[0037] Figure 1 shows example data sources 110 can be accessed using the SQL format. Data sources 110 can include both local data sources (*e.g.,* a data source whose data is stored on a computer running software for DIVE system 100) and remote data sources, such as databases

and files with data. In some embodiments, DIVE system 100 can support additional and/or different languages than SQL to access data in data sources 110; *e.g.,* Contextual Query Language (CQL), Gellish English, MQL, NoSQL for key/value databases, MDX for online analytical processing (OLAP) data, Object Query Language (OQL), RDQL, SMARTS.

[0038] Interaction can be provided by DIVE system 100 providing visual analytics and /or other tools for exploration of data from data sources 110. Interoperability can be provided by DIVE system 100 providing data obtained from data sources 110 in a variety of formats to DIVE plug-ins, associated applications, and DIVE tools.

[0039] These plug-ins, applications, and tools can be organized via the data pipeline. As one example, a DIVE tool can start a DIVE pipeline to convert data in a data frame into an ontological representation using a first DIVE plug-in, an application can generate renderable data from the ontological representation, and then a second DIVE plug-in can enable interaction with the renderable data.

[0040] The DIVE pipeline can be used to arrange components in a sequence of pipeline stages. An example three-stage DIVE pipeline using the above-mentioned components can include:

Stage 1 - the first DIVE plug-in receives data from data sources 110, generates corresponding ontological representations, and outputs the ontological representations onto the pipeline.

Stage 2 - the application receives the ontological representations as inputs via the pipeline, generates renderable data, and outputs the renderable data onto the pipeline.

Stage 3 - the second DIVE plug-in can receive the renderable data via the pipeline and present the renderable data for interaction.

Additional DIVE pipeline examples are possible as well -- some of these additional examples are discussed herein.

[0041] DIVE is domain independent and data agnostic. The DIVE pipeline accepts data from any domain, provided an appropriate input parser is implemented. Some example data formats supported by DIVE include, but are not limited to, SQL, XML, comma- and tab-delimited files, and several other standard file formats. In some embodiments, DIVE can utilize functionality from an underlying software infrastructure, such as a UNIX™-based system or the .NET environment.

[0042] Ontologies are gaining popularity as a powerful way to organize data. DIVE system 100's core data representation using datanodes and dataedges was developed with ontologies in mind. The fundamental data unit in DIVE is the datanode, where datanodes can be linked using dataedges.

[0043] Datanodes somewhat resemble traditional object instances from object-oriented (OO) languages such as C++, Java, or C#. For example, datanodes are typed, contain strongly typed properties and methods, and can exist in an inheritance hierarchy. Datanodes extend the traditional model of object instances, as datanodes can exist outside of an OO environment; *e.g.,* in an ontological network or graph, and can have multiple relationships beyond simple type inheritance. DIVE system 100 implements these relationships between datanodes using dataedges to link related datanodes. Dataedges can be implemented by datanode objects and

consequently might contain properties, methods, and inheritance hierarchies. Because of this basic flexibility, DIVE system 100 can represent arbitrary, typed relationships between objects, objects and relationships, and relationships and relationships.

[0044] Datanodes are also dynamic; every method and property can be altered at runtime, adding flexibility to DIVE system 100. The DIVE pipeline contains various data integrity mechanisms to prevent unwanted side effects. 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.

[0045] Datanodes of DIVE system 100 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, *e.g.,* to allow data manipulation.

[0046] Dataedges can be used to implement multiple inheritance models. Besides the traditional is-a relationship in object-oriented (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:

- With *OO inheritance*, which is identical to OO languages such as C++ and Java, subclasses inherit the parent's type, properties, and methods; *e.g.,* a triangle is a polygon.
- With *type inheritance*, subclasses inherit only the type; type inheritance is used to implement OO languages.

- With *property inheritance*, subclasses inherit only the properties and methods; *e.g.,* a polygon contains line segments.

[0047] 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 system 100 supports not only multiple inheritance but also multiple kinds of inheritance, casting can involve 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. Then, graph-theoretical methods can be applied to analyze both the data structures and ontologies represented in DIVE system 100. This graph-theoretical approach has already proved useful in some examples; *e.g.,* application of DIVE system 100 to structural biology.

[0048] DIVE system 100 supports code and tool reuse. 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, modules can be connected so long as the analyzed datanodes have the expected properties, methods, or types.

[0049] Data-type handling is a challenge in modular architectures. For examples, Taverna uses typing in the style of MIME (Multipurpose Internet Mail Extensions), the VTK uses strongly typed classes, and Python-based tools, such as Biopython and SciPy, often use Python's dynamic typing.

[0050] For DIVE system 100, 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. 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. Datanodes and dataedges themselves can be strongly typed objects to facilitate programmatic manipulation of the dataflow itself. Although each typing approach has its strengths, typing by DIVE system 100 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.

[0051] Data can be imported into DIVE system 100 to make the data accessible to the DIVE pipeline. In some cases, DIVE system 100 includes built-in functionality for importing data. For tabular data or SQL data tables, DIVE system 100 can construct 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. Once DIVE obtains the data for data nodes, DIVE can establish relationships between datanodes using dataedges.

[0052] The DIVE pipeline can utilize plug-ins to create, consume, or transform data. A plug-in can include a compiled software library whose objects inherit from a published interface to the DIVE pipeline. Plug-ins can 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 states as saved pipelines and also share saved pipelines. DIVE system 100 can provide subsequent plug-in connectivity, pipeline instantiation, scripting, user interfaces, and other aspects of plug-in functionality.

[0053] When DIVE system 100 sends a datanode object through a branching, multilevel transform pipeline, correctness of the datanode's property value(s) should be maintained at each pipeline stage. For example, a simple plug-in that scaled its incoming values could scale all incoming data values everywhere in the pipeline. One option to ensure datanode correctness is to copy all datanodes at every pipeline stage. This option can be computational-resource intensive and can delay a user from interacting with the datanodes.

[0054] Another option to address the correctness problem is to create a version history of each transformed value of a datanode. For example, DIVE system 100 can use read and write contexts to maintain the version history; *e.g.,* values of a datanode can be saved before and after writing by the pipeline. The version history can be keyed on each pipeline stage. Then, each plug-in can reads only the appropriate values for its pipeline stage and does read values from another pipeline stage or branch. The use of version histories can be fast and efficient because upstream graph traversal is linear and each value lookup in a read or write context is a constant time operation. Use of version histories maintains data integrity in a branching transform pipeline as well as being parallelizable. Further, the use of read and write contexts can accurately track a property value at every stage in the pipeline with a minimum of memory use.

[0055] In some embodiments, DIVE system 100 can utilize the Microsoft Windows platform including the .NET framework, as this platform includes dynamic-language runtime, expression

trees, and Language-Integrated Query (LINQ) support. The .NET framework can provide coding features such as reflection, serialization, threading, and parallelism for DIVE. These capabilities can affect DIVE's functionality and user experience. Support for dynamic languages allows flexible scripting and customization. LINQ can be useful in a scripted data-exploration environment. Expression trees and reflection can provide object linkages for the DIVE object parser. DIVE streaming can use the .NET framework's threading libraries. DIVE system 100 can use 64-bit computations and parallelism supported by .NET to scale as processor capabilities scale. In other embodiments, DIVE can utilize one or more other platforms that provide similar functionality as described as being part of the Windows platform and .NET framework.

[0056] The platform can support several software languages; *e.g.,* C#, Visual Basic, F#, Python, and C++. Such platform support enables authoring DIVE plug-ins in the supported languages. In addition, the supported languages can be used for writing command-line, GUI, and programmatic tools for DIVE system 100. DIVE can use external libraries that are compatible with the platform, including molecular visualizers, clustering and analysis packages, charting tools, and mapping software; *e.g.,* the VTK library wrapped by the ActiViz .NET API. In some embodiments, DIVE can draw on data base support provided by the platform; *e.g.,* storing data in a Microsoft SQL Server data warehouse.

[0057] Figure 2 shows an example architecture for DIVE system 100, in accordance with an embodiment. DIVE system 100 can include both software libraries and a runtime environment, as shown in the bottom of Figure 2. DIVE system 100 can import and export data and

functionality from a variety of sources, such as the DIVE object parser, SQL, Web Services, files, file formats, and libraries, as shown in the middle of Figure 2.

[0058] Software clients of DIVE system 100 can include DIVE plug-ins and DIVE tools, as shown in Figure 2, DIVE plug-ins can use DIVE software libraries to exploit DIVE's data handling capabilities. DIVE tools can include applications that manage a DIVE pipeline to solve one or more tasks; that is, the DIVE tool can instantiate, launch, and close a DIVE pipeline. In conjunction, DIVE tools can manage and build DIVE pipelines using DIVE plug-ins, applications, and perhaps even other DIVE tools associated with DIVE system 100. DIVE system 100 provides both user interfaces; *e.g.,* command line interfaces (CLIs), graphical user interfaces (GUIs), and programmatic interfaces for software; *e.g.,* one or more DIVE application programming interfaces (APIs).

[0059] Figure 3 shows a pipeline 300 between data sources 110 and DIVE system 100, in accordance with an example embodiment. In pipeline 300, data from data sources 110 is first received at DIVE system 100 by pre-loader 310. Pre-loader 310 for DIVE system 100 can facilitate big data operations. Traversing big data in an efficient manner is important for current and future big data interaction paradigms such as visual analytics. However, many big data operations can be slow; *e.g.,* querying data from a big data source, representing data from big data source(s) in a complex ontology, and building subsets of represented data for visualization.

[0060] To speed big data operations, pre-loader 310 can predict user needs, perform on-demand and/or pre-emptive loading of corresponding data frames 320; *e.g.,* subsets of data from one or more of data sources 110, and subsequent caching of loaded data frames 320. Each data

frame of data frames 320 can include one or more data items, where each data item can include data in the subset(s) of data from one or more of data sources 110. For example, if an pre-loader 310 is loading data from data sources 110 related to purchases at a department store into data frame DF1 of data frames 320, each data frame, including DF1, can have data items (values) for data having data types such as “Purchased Item”, “Quantity”, “Item Price”, “Taxes”, “Total Price”, “Discounts”, and “Payment Type”.

[0061] Preemptive loading can reduce to on-demand loading of a specified frame, if necessary. Caching can be take place locally or remotely and can be single- or multi-tiered. For example, caching can include remote caching on a cloud database, which feeds local caching in local computer memory. In some embodiments, the local computer memory can include random access memory (RAM) chips, processor or other cache memory, flash memory, magnetic media, and/or other memory resident on a computing device executing software of DIVE system 100.

[0062] Loaded and cached data from data sources 110 can be stored by pre-loader 310 as data frames 320. Data frames 320 can be stored where they can be quickly accessed by the local computer memory.

[0063] Data frame selection logic 330 can include logic for switching relationships between data frames 320 and data pins 332. For example, data selection logic 330 can switch some or all of data pins 332 to reference data from a selected frame of frames 320. Data frame selection logic 330 can be provided by user input, programmatic logic, etc. In some embodiments, a pin-switching process for switching data pins 332 between frames of data frames 320 is O(1).

[0064] Once switched to a frame, data pins 332 can pull data, such as data items, from one or more selected data frames. In some embodiments, all pins reference one data frame, while in other embodiments, pins can reference two or more data frames; *e.g.,* a first bank, or subset, of data pins 332 can reference the selected data frame F1 and a second bank of data pins 332 can reference a previously selected frame. Then, when a new data frame F2 is selected, the first bank of data pins 332 can reference the new frame F2 and the second bank of data pins 332 can reference the previously selected frame F1, or perhaps some other previously selected frame.

[0065] In some examples, one or more data pins of data pins 332 can be designated as a control pin. The control pin can indicate a control, or one or more data items of interest of the plurality of data items. For example, if data frames are each associated with a time, a control pin can indicate a time of interest a control, two control pins can respectively indicate a beginning time of interest and an ending time of interest for a time-range control, and multiple control pins can indicate multiple time/ranges of interest. As another example, if data frames are each associated with unique identifiers (IDs) such as serial numbers, VINs, credit card numbers, etc., a control pin can specify an ID of interest as a control. As another example, if data frames are each associated with a location, the location for the data frame can be used as a control. Many other examples of controls and control pins are possible as well.

[0066] In some examples, the control pin can be writeable so a user could set the control pin data; *e.g.,* specify the control associated with the control pin (e.g., specify a time or ID). Then, once a control has been specified, DIVE system 100 can search or otherwise scan the data from data sources 100 for data related to the control. In other examples, the control pin can be read-

only; that is, indicate a value of the control in a data frame, without allowing the control to be changed.

[0067] Data in data frames 320 can be organized according to data ontology 340, which can include arbitrary node types and arbitrary edges. Data ontology 340, in turn, can map node and edge properties; *e.g.,* datanodes and dataedges, to data pins 332. When data pins 332 are switched between frames, data throughout ontology 340 that refers to data pins 332 can be simultaneously updated. For example, suppose data pin #1 referred to a data item having a data type of "name" in a data frame of data frames 320, and suppose that the data item accessible via data pin #1 is "Name11". Then, if data pins 332 are all switched to refer to a new data frame with a name of"Name22", the reference in data ontology 340 to data pin #1 would refer to the switched data item "Name22". Many other examples are possible as well.

[0068] If data ontology 340 changes, references from data pins 332 into data ontology 340 can be changed as well. That is, each of data pins 332 can include at least two references: one or more references into data frames 320 for data item(s) and one or more references into data ontology 340 for node/edge data/logic. Then, changes in the structure, format, and/or layout of data frames 320 can be isolated by data pins 332 (and perhaps data frame selection logic 330) from data ontology 340 and vice versa.

[0069] In some embodiments, all pins switch together. Then, when data pins 332 indicate a data frame of data frames 320 has been switched, all references to data within data ontology 340 made using data pins 332 are updated simultaneously, or substantially simultaneously. If data ontology 340 changes, references from data pins 332 into data ontology 340 can be changed as

well, thereby changing references to data made available by data pins 332. For example, if ontology 340 referred to data pin #1 to access a data type of "name" but changed to refer to a "first name" and a "last name", the reference to data pin #1 may change; *e.g.,* to refer to data pin #1 and #2 or some other data pin(s) of data pins 332.

[0070] In other embodiments, upon arrival of a new frame, some data pins 332 may not switch; *e.g.,* a bank of data pins 332 referring to a first-received frame may not switch after the first data frame is received.

[0071] Ontological data from data ontology 340 can be arbitrarily transformed via transform 350 before providing data interactions 360. Because of the pin-linked ontology, fed by a fast-switched data set, in turn fed by preemptive data caching, pipeline 300 can use DIVE system 100 to provide quick interaction, analysis, and visualization of complex and multi-dimensional data.

DIVE Object Parsing

[0072] Modern computational problems increasingly require formal ontological analysis. However, for some software hierarchies, formal ontologies do not exist. The generation of formal ontologies can be time consuming, difficult, and require expert attention. Ontologies are often implicitly defined in code by software engineers and so code, such as object hierarchies, can be parsed for conversion into a formal ontology.

[0073] For example, an object-parser can traverse object-oriented data structures within a provided assembly using code reflection. Using generalized rules to leverage the existing ontological structure, a formal ontology can be generated from the existing relationships of the

data structures within the code. The ontology can be a static ontology defining an ontological structure or can be a dynamic ontology; that is, a dynamic ontology can include links between the ontological structure (of a static ontology) and object instances of the provided code assembly. The dynamic ontology can allow the underlying object instances to be modified through the context of the ontology without changes to the code assembly. In other examples, metadata tags can be added to the assembly to provide definitions for (generated) ontologies, and so provide a richer ontology definition.

[0074] DIVE system 100 can include a DIVE object parser, which can automatically generate datanodes and dataedges of a DIVE data structure from a software object hierarchy, such as a .NET object or assembly. Using reflection and expression trees, the DIVE object parser can consume object instances of the software object hierarchy and translates the object instances into propertied datanodes and dataedges of a DIVE data structure. For example, standard objects can be created by library-aware code. Then, those standard objects can be parsed by the DIVE object parser into a DIVE data structure, which can be injected into a DIVE pipeline as a data ontology.

[0075] The DIVE object parser can make software object hierarchies available for ontological data representation and subsequent use as DIVE plug-ins written without prior knowledge of DIVE. The DIVE object parser can include generic rules to translate between a software object hierarchy and a DIVE data structure. These generic rules can include:

- Complex objects in the software object hierarchy, such as classes, can be translated into datanodes of a DIVE data structure.

- Interfaces, virtual class, and abstract class objects in the software object hierarchy can be translated into datanodes of the DIVE data structure.
- Built-in system objects, primitive fields, primitive properties, and methods with primitive return types in the software object hierarchy can be translated into properties on datanodes of the DIVE data structure.
- Inheritance and member relationships objects in the software object hierarchy can be interpreted as object and property inheritance dataedges in the DIVE data structure, respectively; these dataedges can then connect the datanode hierarchy.

[0076] Additional rules beyond the generic rules can handle other program constructs:

- The DIVE object parser can translate static members of the software object hierarchy into a single datanode in the DIVE data structure.
- Multiple object instances with the same static member of the software object hierarchy can reference a single, static datanode instance in the DIVE data structure.
- Public objects and members can always be parsed
- Private members, static objects, and interfaces can be parsed based on parameters provided to the DIVE object parser and/or via other user-controllable data.
- More, different, and/or other rules that these generic rules and additional rules for parsing software object hierarchies into DIVE data structures/ontologies are possible as well.

[0077] Throughout a parse, no data values are copied to datanodes or dataedges. Instead, dynamically created virtual properties link all datanode properties to their respective software object hierarchy members. So, any changes to runtime object instances are reflected in their

corresponding representations in DIVE data structures. Similarly, any changes to datanode or dataedge properties in DIVE data structures propagate back to their software object instance counterparts.

[0078] Using this approach, the generic rules, and additional rules, the DIVE object parser can recursively produce an ontological representation of the entire software object hierarchy. With object parsing, users can import and use software object hierarchies within DIVE without special handling, so that software applications can be parsed and readily exploit DIVE capabilities. For example, assume L1 is a nonvisual code library that dynamically simulates moving bodies in space. A DIVE plug-in, acting as a thin wrapper, can automatically import library L1 and add runtime visualizations and interactive analyses. As the simulation progresses, the datanodes will automatically reflect the changing property values of the underlying software object instances. Through a DIVE interface, the user of the DIVE pipeline that imported L1 could change a body's mass. This change would propagate back to the runtime instance of L1 and appear in the visualization. Many other examples are possible as well.

[0079] Figure 4 shows an example where DIVE object parser 400 has converted software object hierarchy 410 to DIVE data structure 420, in accordance with an embodiment. In the example shown in Figure 4, software object hierarchy 410 includes a .NET assembly with interfaces IClassA, IClassB and classes AbstractClass, OClass, SuperClass, SubClassA and SubClassB arranged using object inheritance, shown in Figure 4 using solid lines between classes, into an object hierarchy. Some classes in software object hierarchy 410 include methods; *e.g.,* class OClass has method OClassM(), class SuperClass has method SuperM(), class

SubClassA has method SubAM(), and class SubClassB has methods SubBM1() and SubBM2(). Other classes have fields; *e.g.,* class SuperClass has field StaticSuperF and class SubClassA has fields SubAF1 and SubAF2, while class SubClassB has property SubBProp.

[0080] Similarly, DIVE data structure 420 has datanodes for interfaces and classes IClassA, IClassB, Abstract Class, OClass, SuperClass, SubClassA and SubClassB, methods OClassM(), SuperM(),SubAM(),SubBM1() and SubBM2(), fields StaticSuperF, SubAF1, and SubAF2, and property SubBProp. Relationships between datanodes in DIVE data structure 420 are shown using both solid and dashed lines representing dataedges.

[0081] DIVE object parser 400 can parse software object hierarchy 410 for translation into a data ontology and/or DIVE data structure. In other examples, other software hierarchies than .NET assemblies can be input to DIVE object parser 400 for parsing. In the example shown in Figure 4, DIVE object parser 400 can parse software object hierarchy 410 using the above-mentioned generic and additional rules to translate hierarchy 410 into DIVE data structure 420. DIVE data structure 420 can replicate the strongly typed objects and relationships indicated by the structure of software object hierarchy 410.

[0082] In the example shown in Figure 4, DIVE data structure 420 represents a data ontology corresponding to software object hierarchy 410, with data nodes (shown in Figure 4 as circles) corresponding to objects in software object hierarchy 410 and data edges (shown in Figure 4 as lines) corresponding to relationships between objects in software object hierarchy 410. Figure 4 shows some data edges in DIVE data structure 420 as solid lines, corresponding to object inheritance relationships in software object hierarchy 410. Other data edges in DIVE data

structure 420 are shown in Figure 4 as dashed lines, corresponding to property inheritance relationships in software object hierarchy 410.

[0083] Instance-specific data of software object hierarchy 410 are maintained on the subclass data nodes in DIVE data structure 420; that is, data for super classes is not stored with superclass data nodes. The original fields, properties, and methods of software object hierarchy 410 are accessible through the data nodes of DIVE data structure 420 by virtual properties.

[0084] In DIVE data structure 420, each instance of a class can be represented. For example, Figure 4 shows DIVE data structure 420 with one instance of all classes except for class OClass. In this example, Class OClass has three instances, which are shown as three separate data nodes DIVE data structure 420.

[0085] Figure 5 shows scenario 500, where DIVE object parser 400 has translated code assembly 510 to ontologies 520, 530, in accordance with an example embodiment. Scenario 500 begins with code assembly 510 being provided to DIVE object parser 400. As indicated in Figure 5, code assembly 510 can include private objects, protected objects, static objects, interfaces, and other software entities ("Etc."). For example, code assembly 510 can be a software object hierarchy, such as software object hierarchy 410 discussed above in the context of Figure 4.

[0086] In scenario 500, parameters to DIVE object parser 400 can specify which semantic components are to be parsed into one or more ontologies. For example, the parameters can reflect user intent regarding whether or not private members, static objects, interfaces, and other software entities of code assembly 510 are parsed.

[0087] DIVE object parser 400 can recursively traverse object hierarchies of code assembly 510 using code reflection and expression trees. Using generalized, pre-defined rules, such as the generic and additional rules discussed above in the context of Figure 4, objects and other software entities can be parsed by DIVE object parser 400 into ontological components.

[0088] In scenario 500, DIVE object parser 400 outputs the ontological components in two formats: static ontology 520 corresponding to semantic components and relationships of code assembly 510 and dynamic ontology 520. Both static ontology 520 and dynamic ontology 530 can include an ontological definition that uses standardized ontology language. Dynamic ontology 530 can further include links into the object instance(s) of code assembly 510. For example, links between ontological components and object instances using delegate methods and lambda functions. Figure 5 shows the ontological components of ontologies 520 and 530 using circles, object instances of code assembly 510 linked to ontology 530 using rectangles, and links between ontological components and object instances in ontology 530 using solid grey lines between the two.

DIVE Scripting Techniques

[0089] DIVE supports the use of scripts to let users rapidly interact with the DIVE pipeline, plug-ins, data structures, and data. DIVE supports at least two basic types of scripting: plug-in scripting and μscripting (microscripting). DIVE can host components, including scripts, written in a number of computer languages. For example, in some embodiments, C# can be used as a scripting language.

[0090] Plug-in scripting is similar to existing analysis tools' scripting capabilities. Through the plug-in script interface, the user script can access the runtime environment, the DIVE system, and the specific plug-in. μscripting can provide direct programmatic control to experienced users and simple, intuitive controls to relatively-new users of DIVE.

[0091] 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 lambda function. Here's a schematic of a lambda function F1():

F1(datanode dn) => RHS;

[0092] The right-hand side RHS written by the user is inserted into the lambda function. The lambda function, including the user's right-hand-side code, is compiled at runtime. The client can provide any expression that evaluates to an appropriate return value. In general, plug-in scripting can be more powerful than μscripting, while μscripting can be simpler at first.

[0093] User scripts, such as plug-in scripts and μscripting-originated scripts, can be included into the DIVE system. For example, the user script can be incorporated into a larger, complete piece of code that can be compiled; *e.g.,* 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.

[0094] Table 1 below provides some μscripting examples.

Argument	Return Type	µscript	Comments
`datanode dn`	`double`	`3`	Basic constant numeric script
		`dn.X`	Basic per-datanode script
		`Math.Abs(dn.X)`	A µscript can call library functions
	`int`	`dn.X > 0 ? 1 : -1`	µscript syntax can be powerful.

`void`	`bool`	`{`
		`  int hour =`
		`  DateTime.Now.Hour;`
		`  return hour < 12;`
		`}`
			µscripts can include complex, multi-statement functions.

`datanode[]`	Dynamic Set	`from dn in dns`
		`group dn by`
		`  Math.Round(dn.X, 2)`
		`  into g`
		`select new`
			µscript for creating a histogram based on the datanode's "X" property.

		{
		bin = g.Key,
		population = g.Count()
		};	

		from dn in dns where dn.X > Math.Pi && dn.is_Superclass && dn.Func() = true select dn;	μscript for filtering datanodes on the basis of datanode properties, methods (e.g., Func()), and inherited type (e.g., is_Superclass).
		from dn1 in dnSet1 join dn2 in dnSet2 on dn1.X equals dn2.X select new (X = dn1.X, Y = dn2.Y);	μscript for using DIVE as OO database for joining multiple potentially disparate datasets.

Table 1

Appendix Table B.1 DIVE patent Table 1

Data Streaming Using DIVE

[0095] DIVE system 100 can support data streaming using an interactive SQL approach and a pass-through SQL approach. In some embodiments, database languages other than SQL can be utilized by either approach. Interactive SQL can be used for the immediate analysis of large, nonlocal datasets via impromptu, user-defined dynamic database queries using SQL by taking user input to build an SQL query.

The SQL query can include one or more data queries, as well as one or more functions for analysis of data received via the data queries. DIVE system 100 can send the SQL query to the SQL database and parse the resulting dataset. Depending on the query's size and complexity, this approach can result in user-controlled SQL analysis through the GUI at interactive rates. DIVE system 100 can facilitate interactive SQL by use of events generated at runtime; for example, DIVE events can be generated in response to mouse clicks or slider bar movements. Upon receiving these DIVE events, a DIVE component can construct the appropriate SQL query.

[0096] Figure 6 shows examples of interactive SQL streaming and pass-through SQL streaming, in accordance with an example embodiment. With respect to interactive SQL, Figure 6 shows an example SQL template 610 with tags for "time_step" and "atom". During interactive SQL streaming, the tags in SQL template 610 can be replaced with input from GUI elements, such as slider bar 620, 622 and atoms 624, where atoms 624 can be selected using GUI elements not shown in Figure 6.

[0097] An SQL query can use SQL template 610 to obtain and analyze data. In the example shown in Figure 6, the SQL query can obtain "coordinates" data for item "c1" and join data for item "c2" to become part of the "coordinates" data. Then, the time_step tagged data can be set to a "step" value of c1 and the atom tagged data can be set to an "atom_id" value of c1. Subsequently, when the step value of c1 equals a step value for c2 and the atom_id value of c1 equals an atom_id of c2, then the obtained data for c1 and c2 can be analyzed using a eucl_dist() function operating on "x", "y", and "z" values from both c1 and c2 to determine a resulting "distance" value.

[0098] The pass-through SQL approach can be used for interactive analysis of datasets larger than the client's local memory; *e.g.,* pass-through SQL can be used 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 data frames, such as the above-mentioned data frames 320 discussed at least in the context of Figure3.

[0099] A backing store can include a collection of one or more tables of instance data, where each table can contain one or more 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 then includes all the tables for the object instances occurring at the same point, or frame, in the streaming dimension.

[0100] Once a backing store has been created by DIVE system 100, copies of the backing-store structure can be generated with a unique identifier for each new frame. DIVE system 100 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. The use of backing stores also removes the overhead of allocating and de-allocating expensive objects by reusing the same object structures for each frame in the streaming dimension.

[0101] Pass-through SQL enables streaming through a buffered backing-store collection of backing stores representing frames over the streaming dimension. A backing-store collection is initially populated client-side for frames on either side of the frame of interest, where buffer regions are defined for each end of the backing-store collection. Frames whose data are stored in the backing-store collection 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; *e.g.,* by the pre-loader. If the client requests a frame outside the loaded set, a new backing-store collection can be loaded around the requested frame. Loaded backing stores no longer in the streaming collection can be deleted from memory to conserve the client's memory.

[0102] Figure 6 shows an example use of pass-through SQL streaming. On initial data frame request 630a, DIVE system 100 can construct a datanode hierarchy; *e.g.,* an ontology from object hierarchy 634 using DIVE object parser 400. Then, DIVE system 100 can generate backing stores 640 corresponding to the initial data frame that includes data retrieved from database(s) 632. Backing stores 640 can be arranged as one or more backing store collections.

[0103] On each subsequent data frame request 630b, DIVE system 100 can buffer data retrieved from database(s) 632 into backing stores 640 directly. In some embodiments, DIVE

system 100 can use multiple threads to buffer data into backing stores 640. DIVE system 100 can use pass-through SQL streaming to propagate large amounts of data through a DIVE pipeline using database(s) 632, object hierarchy 634, and backing stores 640 at interactive speeds; *i.e.,* by bypassing object-oriented parsing.

A DIVE Case Study: the Dynameomics Project

[0104] In a case study, DIVE is used by the Dynameomics project to provide molecular dynamics simulations for studying protein structure and dynamics. The Dynameomics project involves characterization of the dynamic behaviors and folding pathways of topological classes of all known protein structures.

[0105] An interesting facet of protein biology is that structure equals function; that is, what a protein does and how it does it is intrinsically tied to its 3D structure. During a molecular dynamics simulation, scientists simulate interatomic forces to predict motion among atoms of a molecule, such as a protein, and its environment to better understand the 3D structure of the molecule.

[0106] Figure 7 shows an example protein simulated using molecular dynamics, in accordance with an embodiment. Image 710 is an all-atom depiction of the example protein with a transparent surface. In most cases, the environment for a protein molecule is water molecules, although scientists can alter this to investigate different phenomena. For example, image 720 shows the protein depicted in image 710 solvated and shown in a water box.

[0107] The physical simulation is calculated using Newtonian physics; at specified time intervals, the simulation state is saved. This produces a trajectory or a series of structural snapshots reflecting the protein's natural behavior in an aqueous environment. Image 730 shows three structures selected from a trajectory containing more than 51,000 frames.

[0108] Molecular dynamics 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 molecular dynamics simulations to run in parallel, enabling virtual high-throughput experimentation. Third, molecular dynamics 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.

[0109] However, molecular dynamics simulations can produce datasets considerably larger than what most structural-biology tools can handle. So far, the Dynameomics project has generated hundreds of terabytes of data consisting of thousands of simulations and millions of structures, as well as their associated analyses, stored in a distributed SQL data warehouse. The data warehouse can hold at least four orders of magnitude more protein structures than the Protein Data Bank, which is the World's primary repository for experimentally characterized protein structures.

[0110] In particular, the Dynameomics project contains much more simulation data than many domain-specific tools are engineered to handle. One of the first Dynameomics tools built

on the DIVE platform was the Protein Dashboard. 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.

[0111] Figure 8 shows an example data flow using DIVE system 100 for the Dynameomics project, in accordance with an example embodiment. Using DIVE object parser 400, DIVE system 100 can integrate and use structures developed using a Dynameomics API (discussed after Figure 9) without changing DIVE's API. DIVE object parser 400 can then create 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 Dynameomics API can be translated into dataedges by DIVE object parser 400. The Dynameomics-related datanodes and dataedges generated by DIVE object parser 400 are available to the DIVE pipeline, indistinguishable from any other datanodes or dataedges.

[0112] The top of Figure 8 shows data sources for the data flow, including Dynameomics data stored a data warehouse in SQL format and the Protein Data Bank (PDB). Associated with the data sources are software object hierarchies for representing the data in software. In the example of the case study, the software object hierarchies are part of .NET assemblies. The software object hierarchies in the case study can be parsed using the DIVE object parser 400, as indicated by the middle portion of Figure 8. DIVE object parser 400 can generate datanodes and dataedges corresponding to the software object hierarchies.

[0113] The generated datanodes and dataedges, along with DIVE plug-ins, μscripts, plug-in scripts, DIVE tools, and/or other software entities, can be used together as a DIVE pipeline, as indicated a lower portion of Figure 8. The bottom portion of Figure 8 indicates that a user can interact with the DIVE pipeline via Protein Dashboard 800. Protein Dashboard 800 can allow access to multiple interactive simulations simultaneously.

[0114] Figure 9 shows an example view of Protein Dashboard 800, in accordance with an embodiment. The view of Protein Dashboard 800 is an example view generated by a graphical user interface for DIVE system 100. An upper portion of Protein Dashboard 800 includes pre-loader interface 910 to allow interaction with a DIVE pre-loader; *e.g.,* pre-loader 310. Pre-loader interface 910 provides user controls for loading and interacting with protein structures and molecular-dynamics trajectories.

[0115] At lower left of Figure 9, interactive rendering interface 920 shows an interactive 3D rendering of a protein molecule; using a cartoon representation of a backbone of the protein molecule, and a ball-and-stick representation of a subset of atoms in the protein molecule. The subset of atoms can be selected via interactive SQL interface 922, which includes a molecule selector to select a "1enh(678)" molecule, an indicator to show "Atom[s]" of the molecule, and script interface showing selection of data with the property "isHvy == YES", and an apply button. Once the apply button is selected, a selection made using interactive SQL interface 922 to Protein Dashboard 800 can be rendered and displayed using interactive rendering interface 920.

[0116] Chart region 930 shows one of many possible linked interactive charts for a "SASA1 Plot" related to "Residue SASA". The interactive charts can be generated using data streamed from the data sources mentioned in the context of Figure 8; *e.g.,* the Dynameomics data warehouse. In some embodiments and examples, Protein Dashboard 800 can provide more, fewer, and/or different windows, tabs, interfaces, buttons, and/or GUI elements than shown in Figures 8 and 9.

[0117] A tool implemented independently of DIVE and the Protein Dashboard is the Dynameomics API. The API can be used to establish an object hierarchy, provide high-throughput streaming of simulations from the Dynameomics data warehouse. The Dynameomics API includes domain-specific semantics and data structures and provides multiple domain-specific analyses. In some embodiments, the Dynameomics API can be user interface agnostic; then, the Dynameomics API can provide data handling and streaming support independently of how the user views and otherwise interacts with the data; *e.g.,* using the Protein Dashboard. In some embodiments, the API can be written in a particular computer language; *e.g.,* C#.

[0118] With the Dynameomics data and semantics available to the DIVE pipeline, a visual analytics approach can be applied to the Dynameomics data. Protein Dashboard 800 can be used to interact with and visualize the data. However, because the data flows through the Dynameomics API, wrapped by DIVE datanodes and dataedges, multiple protein structures from different sources can be loaded, including structures from the Protein Data Bank. Once loaded, the protein structures can be aligned and analyzed in different ways.

[0119] Furthermore, because Protein Dashboard 800 has access to additional data from the Dynameomics API via DIVE system 100, the utility of Protein Dashboard 800 increases. For instance, scientists can find utility in coloring protein structures on the basis of biophysical properties; *e.g.,* solvent-accessible surface area, deviation from a baseline structure. By streaming the data through the pipeline, these biophysical properties can be observed as they change over time. In some instances, some or all of the biophysical properties can be accessed through the data's inheritance hierarchy.

[0120] Applications built on DIVE system 100 have been used to accelerate biophysical analysis of Dynameomics and other data related to two specific proteins. 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.

[0121] The Y220C mutation of the p53 DNA binding domain is responsible for destabilizing the core, leading to about 75,000 new cancer cases annually according to Boeckler *et al*. The DIVE framework can analyze the structural and functional effects of the Y220C mutation using a DIVE module called ContactWalker. The ContactWalker module can identify amino acids' interatomic contacts disrupted significantly as a result of mutation. The contact pathways between disrupted residues can be identified identified using DIVE's underlying graph-based data representation.

[0122] Figures 10A and 10B show visualizations related to the respective p53 and SOD1 proteins provided by DIVE system 100, in accordance with an embodiment. Figure 10A 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 were isolated. Then, specific molecular dynamics time points and structures were selected for further investigation. For example, Figure 10A shows 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.

[0123] In particular, Figure 10A shows visualizations related to the p53 protein. In the top panel of Figure 10A, a ContactWalker summary of contact differences between wild-type and Y220C simulations is shown. The highlighted residues have contacts with $\geq$ 50% occupancy change. In the middle panel of Figure 10A, distances between P151 and L257 are outlined in black. In the bottom panel of Figure 10A, a visualization of the p53 protein is shown with ligand (stick figure at bottom) (Protein Data Bank code 4AGQ) in proximity to disrupted residues shown in black.

[0124] In another example, DIVE has been used in about 400 simulations of 106 disease-associated mutants of SOD1. Through extensive studies of A4V mutant SOD1, Schmidlin et al. previously noted the instability of two β-strands in the SOD1 Greek key β -barrel structure. 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.

[0125] DIVE system 100 was used to further explore the formation and persistence of the contacts and packing interactions in this region across multiple simulations of mutant proteins.

DIVE system 100 facilitates isolation of specific contacts, rapid plotting of selected data, and easy visualization of the relevant structures and geographic locations of specific mutations, while providing intuitive navigation from one view to another.

[0126] The top panel of Figure 10B maps secondary structure for different variants as an example of DIVE's charting tools. This chart can be quickly generated and contains results for 400 SOD1 mutant simulations. The chart 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. These data are in turn linked to protein structure modules, allowing interactive visualization of more than 60,000 structures from each of the 400 simulations, all streamed from the Dynameomics data warehouse. DIVE system 100 can simplify the transition between high-level protein views and atomic level details, facilitating rapid analysis of large amounts of data. DIVE system 100 can also show the context of the detailed results on other levels, such as worldwide disease incidence.

[0127] In particular, Figure 10B shows visualizations related to analysis of the SOD1 protein. In the top portion of Figure 10B, aggregated secondary structural data from mutant simulations is shown. The middle portion of Figure 10B is a plot of the Cα root-mean-squared (RMS) distances of the wild-type and A4V mutant simulations. In the bottom portion of Figure 10B, a visualization of molecular dynamics structures is shown.

Additional Example DIVE Pipelines

[0128] Example DIVE application pipelines are shown in Figure 11, in accordance with an embodiment. Figure 11 shows, at upper left and center, an example Gene Ontology/Mammal Taxonomy DIVE pipeline. This example shows a taxonomy of mammals built up from data from a static (non-streaming) Gene Ontology database for handling the concept of animal inheritance. In an example interaction with the Gene Ontology/Mammal Taxonomy DIVE pipeline, a user could ask for all mammals descended from tree shrews or all feline mammals. The DIVE Pipeline can be then be used to provide streaming data, such as camera feeds from mammalian research sources, as well as access to the Gene Ontology database. Then, if the user requests to "show all the streaming video data watching animals of subgenus platyrrhini" (e.g., New World monkeys), the Gene Ontology/Mammal Taxonomy DIVE pipeline can use and provide both the streaming data and the ontology together Once both data sources are available, a DIVE plug-in acting as a software agent can be added to the pipeline; e.g., to inform the user when an animal is in a frame of the streaming video data.

[0129] Figure 11 shows, at upper-right, an Animated Particle System DIVE pipeline. The DIVE pipeline renders the images based on an ontological representation of particles whose data is available in a data stream. Use of a particle ontology provides ready access for an application to query properties of various particles shown by the Animated Particle System DIVE pipeline. Another portion of the pipeline performs the simulation of particle interaction and, independently, the simulation is visualized. In some embodiments, the Animated Particle System DIVE pipeline can show how DIVE pipelines extend an existing library by added visualization and interaction components to a simple particle system.

[0130] Figure 11 shows, at center, an example baseball statistics DIVE pipeline. The incoming data source is stored using flat files. The baseball statistics DIVE pipeline illustrates that, even in a single-data-frame scenario, the remainder of the pipeline can remain the same. In other implementations, the flat files could be replaced by a tabular system where statistics are streamed; *e.g.,* streamed in real-time, on a per-year basis, on a per-player basis, or by some other basis.

[0131] The lower portion of Figure 11 shows an example real-time signal processing DIVE pipeline, processing data from a microphone. In this pipeline, the ontological data-graph is hooked back to a byte buffer, through which is streaming raw audio data. This pipeline illustrates the generality of pipeline processing of an ontological graph connected to *some* kind of dynamic data source. In other embodiments, multiple sensors could be connected to a related DIVE pipeline, where data from the sensors is represented via some ontology; *e.g.,* an ontology for medical sensors. Then, a user could request the pipeline to "alert when any sensor monitoring the cardio-pulmonary system downstream of the injection site registers a value outside of the specified safety thresholds." In this pipeline, the cardio-pulmonary specification would be derived from the overall ontology of sensors.

[0132] In another example, the user could request a continuous data stream based on location-related sensor data; *e.g.,* request data from "all deep-ocean current sensors within 100 miles of the up-to-the-minute GPS position of any Navy ship over 1000 tons and under the eventual command of Admiral Jones." In this case, the ontology graph would have to cover naval vessels, command hierarchies, and ocean sensor data. In this case, the subset of the

ontology can change in real time as the ships moves (and perhaps as command changes). Then, queries can be made against the larger ontological graph of naval vessels and undersea sensors using live data streams as part of the query to provide the requested continuous data stream. Many other example DIVE pipelines and uses of DIVE system 100 are possible as well.

Example Computing Network

[0133] Figure 12 is a block diagram of example computing network 1200 in accordance with an example embodiment. In Figure 12, servers 1208 and 1210 are configured to communicate, via a network 1206, with client devices 1204a, 1204b, and 1204c. As shown in Figure 12, client devices can include a personal computer 1204a, a laptop computer 1204b, and a smart-phone 1204c. More generally, client devices 1204a-1204c (or any additional client devices) can be any sort of computing device, such as a workstation, network terminal, desktop computer, laptop computer, wireless communication device (*e.g.,* a cell phone or smart phone), and so on.

[0134] The network 1206 can correspond to a local area network, a wide area network, a corporate intranet, the public Internet, combinations thereof, or any other type of network(s) configured to provide communication between networked computing devices. In some embodiments, part or all of the communication between networked computing devices can be secured.

[0135] Servers 1208 and 1210 can share content and/or provide content to client devices 1204a-1204c. As shown in Figure 12, servers 1208 and 1210 are not physically at the same location. Alternatively, servers 1208 and 1210 can be co-located, and/or can be accessible via a

network separate from network 1206. Although Figure 12 shows three client devices and two servers, network 1206 can service more or fewer than three client devices and/or more or fewer than two servers. In some embodiments, servers 1208, 1210 can perform some or all of the herein-described methods; *e.g.,* method 1400.

Example Computing Device

[0136] Figure 13A is a block diagram of an example computing device 1300 including user interface module 1301, network communication interface module 1302, one or more processors 1303, and data storage 1304, in accordance with an embodiment.

[0137] In particular, computing device 1300 shown in Figure 13A can be configured to perform one or more functions of DIVE system 100, data sources 110, pre-loader 310, data frames 320, data frame selection logic 330, data pins 332, data ontology 340, transform 350, data interactions 360, DIVE object parser 400, one or more DIVE pipelines, Protein Dashboard 800, client devices 1204a-1204c, network 1206, and/or servers 1208, 1210 and/or one or more functions of method 1400. Computing device 1300 may include a user interface module 1301, a network communication interface module 1302, one or more processors 1303, and data storage 1304, all of which may be linked together via a system bus, network, or other connection mechanism 1305.

[0138] Computing device 1300 can be a desktop computer, laptop or notebook computer, personal data assistant (PDA), mobile phone, embedded processor, touch-enabled device, or any similar device that is equipped with at least one processing unit capable of executing machine-

language instructions that implement at least part of the herein-described techniques and methods, including but not limited to method 1400 described with respect to Figure 14.

[0139] User interface 1301 can receive input and/or provide output, perhaps to a user. User interface 1301 can be configured to send and/or receive data to and/or from user input from input device(s), such as a keyboard, a keypad, a touch screen, a computer mouse, a track ball, a joystick, and/or other similar devices configured to receive input from a user of the computing device 1300.

[0140] User interface 1301 can be configured to provide output to output display devices, such as one or more cathode ray tubes (CRTs), liquid crystal displays (LCDs), light emitting diodes (LEDs), displays using digital light processing (DLP) technology, printers, light bulbs, and/or other similar devices capable of displaying graphical, textual, and/or numerical information to a user of computing device 1300. User interface module 1301 can also be configured to generate audible output(s), such as a speaker, speaker jack, audio output port, audio output device, earphones, and/or other similar devices configured to convey sound and/or audible information to a user of computing device 1300.

[0141] Network communication interface module 1302 can be configured to send and receive data over wireless interface 1307 and/or wired interface 1308 via a network, such as network 1206. Wireless interface 1307 if present, can utilize an air interface, such as a Bluetooth®, Wi-Fi®, ZigBee®, and/or WiMAX™ interface to a data network, such as a wide area network (WAN), a local area network (LAN), one or more public data networks (*e.g.,* the Internet), one or more private data networks, or any combination of public and private data

networks. Wired interface(s) 1308, if present, can comprise a wire, cable, fiber-optic link and/or similar physical connection(s) to a data network, such as a WAN, LAN, one or more public data networks, one or more private data networks, or any combination of such networks.

[0142] In some embodiments, network communication interface module 1302 can be configured to provide reliable, secured, and/or authenticated communications. For each communication described herein, information for ensuring reliable communications (*i.e.,* guaranteed message delivery) can be provided, perhaps as part of a message header and/or footer (*e.g.*, packet/message sequencing information, encapsulation header(s) and/or footer(s), size/time information, and transmission verification information such as CRC and/or parity check values). Communications can be made secure (*e.g.*, be encoded or encrypted) and/or decrypted/decoded using one or more cryptographic protocols and/or algorithms, such as, but not limited to, DES, AES, RSA, Diffie-Hellman, and/or DSA. Other cryptographic protocols and/or algorithms can be used as well as or in addition to those listed herein to secure (and then decrypt/decode) communications.

[0143] Processor(s) 1303 can include one or more central processing units, computer processors, mobile processors, digital signal processors (DSPs), graphics processing units (GPUs), microprocessors, computer chips, and/or other processing units configured to execute machine-language instructions and process data. Processor(s) 1303 can be configured to execute computer-readable program instructions 1306 that are contained in data storage 1304 and/or other instructions as described herein.

[0144] Data storage 1304 can include one or more physical and/or non-transitory storage devices, such as read-only memory (ROM), random access memory (RAM), removable-disk-drive memory, hard-disk memory, magnetic-tape memory, flash memory, and/or other storage devices. Data storage 1304 can include one or more physical and/or non-transitory storage devices with at least enough combined storage capacity to contain computer-readable program instructions 1306 and any associated/related data and data structures, including but not limited to, data frames, data pins, ontologies, DIVE data structures, software objects, software object hierarchies, code assemblies, data interactions, scripts (including μscripts).

[0145] Computer-readable program instructions 1306 and any data structures contained in data storage 1306 include computer-readable program instructions executable by processor(s) 1303 and any storage required, respectively, to perform at least part of herein-described methods, including, but not limited to method 1400 described with respect to Figure 14.

[0146] Figure 13B depicts a network 1206 of computing clusters 1009a, 1009b, 1009c arranged as a cloud-based server system in accordance with an example embodiment. Data and/or software for DIVE system 100 can be stored on one or more cloud-based devices that store program logic and/or data of cloud-based applications and/or services. In some embodiments, DIVE system 100 can be a single computing device residing in a single computing center. In other embodiments, DIVE system 100 can include multiple computing devices in a single computing center, or even multiple computing devices located in multiple computing centers located in diverse geographic locations.

[0147] In some embodiments, data and/or software for DIVE system 100 can be encoded as computer readable information stored in tangible computer readable media (or computer readable storage media) and accessible by client devices 1204a, 1204b, and 1204c, and/or other computing devices. In some embodiments, data and/or software for DIVE system 100 can be stored on a single disk drive or other tangible storage media, or can be implemented on multiple disk drives or other tangible storage media located at one or more diverse geographic locations.

[0148] Figure 13B depicts a cloud-based server system in accordance with an example embodiment. In Figure 13B, the functions of DIVE system 100 can be distributed among three computing clusters 1309a, 1309b, and 1308c. Computing cluster 1309a can include one or more computing devices 1300a, cluster storage arrays 1310a, and cluster routers 1311a connected by a local cluster network 1312a. Similarly, computing cluster 1309b can include one or more computing devices 1300b, cluster storage arrays 1310b, and cluster routers 1311b connected by a local cluster network 1312b. Likewise, computing cluster 1309c can include one or more computing devices 1300c, cluster storage arrays 1310c, and cluster routers 1311c connected by a local cluster network 1312c.

[0149] In some embodiments, each of the computing clusters 1309a, 1309b, and 1309c can have an equal number of computing devices, an equal number of cluster storage arrays, and an equal number of cluster routers. In other embodiments, however, each computing cluster can have different numbers of computing devices, different numbers of cluster storage arrays, and different numbers of cluster routers. The number of computing devices, cluster storage arrays,

and cluster routers in each computing cluster can depend on the computing task or tasks assigned to each computing cluster.

[0150] In computing cluster 1309a, for example, computing devices 1300a can be configured to perform various computing tasks of DIVE system 100. In one embodiment, the various functionalities of DIVE system 100 can be distributed among one or more of computing devices 1300a, 1300b, and 1300c. Computing devices 1300b and 1300c in computing clusters 1309b and 1309c can be configured similarly to computing devices 1300a in computing cluster 1309a. On the other hand, in some embodiments, computing devices 1300a, 1300b, and 1300c can be configured to perform different functions.

[0151] In some embodiments, computing tasks and stored data associated with DIVE system 100 can be distributed across computing devices 1300a, 1300b, and 1300c based at least in part on the processing requirements of DIVE system 100, the processing capabilities of computing devices 1300a, 1300b, and 1300c, the latency of the network links between the computing devices in each computing cluster and between the computing clusters themselves, and/or other factors that can contribute to the cost, speed, fault-tolerance, resiliency, efficiency, and/or other design goals of the overall system architecture.

[0152] The cluster storage arrays 1310a, 1310b, and 1310c of the computing clusters 1309a, 1309b, and 1309c can be data storage arrays that include disk array controllers configured to manage read and write access to groups of hard disk drives. The disk array controllers, alone or in conjunction with their respective computing devices, can also be configured to manage backup or redundant copies of the data stored in the cluster storage arrays to protect against disk drive or

other cluster storage array failures and/or network failures that prevent one or more computing devices from accessing one or more cluster storage arrays.

[0153] Similar to the manner in which the functions of DIVE system 100 can be distributed across computing devices 1300a, 1300b, and 1300c of computing clusters 1309a, 1309b, and 1309c, various active portions and/or backup portions of these components can be distributed across cluster storage arrays 1310a, 1310b, and 1310c. For example, some cluster storage arrays can be configured to store one portion of the data and/or software of DIVE system 100, while other cluster storage arrays can store a separate portion of the data and/or software of DIVE system 100. Additionally, some cluster storage arrays can be configured to store backup versions of data stored in other cluster storage arrays.

[0154] The cluster routers 1311a, 1311b, and 1311c in computing clusters 1309a, 1309b, and 1309c can include networking equipment configured to provide internal and external communications for the computing clusters. For example, the cluster routers 1311a in computing cluster 1309a can include one or more internet switching and routing devices configured to provide (i) local area network communications between the computing devices 1300a and the cluster storage arrays 1301a via the local cluster network 1312a, and (ii) wide area network communications between the computing cluster 1309a and the computing clusters 1309b and 1309c via the wide area network connection 1313a to network 1206. Cluster routers 1311b and 1311c can include network equipment similar to the cluster routers 1311a, and cluster routers 1311b and 1311c can perform similar networking functions for computing clusters 1309b and 1309b that cluster routers 1311a perform for computing cluster 1309a.

[0155] In some embodiments, the configuration of the cluster routers 1311a, 1311b, and 1311c can be based at least in part on the data communication requirements of the computing devices and cluster storage arrays, the data communications capabilities of the network equipment in the cluster routers 1311a, 1311b, and 1311c, the latency and throughput of local networks 1312a, 1312b, 1312c, the latency, throughput, and cost of wide area network links 1313a, 1313b, and 1313c, and/or other factors that can contribute to the cost, speed, fault-tolerance, resiliency, efficiency and/or other design goals of the moderation system architecture.

Example Methods of Operation

[0156] Figure 14 is a flow chart of an example method 1400. Method 1400 can be carried out by a computing device, such as computing device 1300 discussed above in the context of Figure 13A.

[0157] Method 1400 can begin at block 1410, where a computing device can receive data from one or more data sources, as discussed above in the context of at least Figures 1-3, 5, 6, and 8.

[0158] At block 1420, the computing device can generate a data frame based on the received data. The data frame can include a plurality of data items, as discussed above in the context of at least Figures 3, 5, and 6. In some embodiments, generating the data frame can include storing a subset of the received data in the data frame using a pre-loader, such as discussed above in the context of at least Figure 3.

[0159] At block 1430, the computing device can determine a data ontology. The data ontology can include a plurality of datanodes, as discussed above in the context of at least

Figures 3 and 5. In some embodiments, the data ontology can be related to a software object hierarchy, such as discussed above in the context of at least Figures 4-6 and 8. In other embodiments, the data ontology can be related to a chemical molecule, such as discussed above in the context of at least Figures 3 and 8.

[0160] At block 1440, the computing device can determine a plurality of data pins, as discussed above in the context of at least Figure 3. A first data pin of the plurality of data pins can include a first reference and a second reference. The first reference for the first data pin can refer to a first data item in the data frame and the second reference for the first data pin can refers to a first datanode of the plurality of datanodes. The first datanode can be related to the first data item.

[0161] At block 1450, the computing device can obtain data for the first data item at the first datanode of the data ontology via the first data pin, as discussed above in the context of at least Figure 3. In some embodiments, the second reference can refer to a datanode associated with a software object in the software object hierarchy. In other embodiments, determining the data ontology can include parsing the software object hierarchy, such as discussed above in the context of at least Figures 4 and 5. In still other embodiments, the plurality of pins can include a control pin, where the control pin indicates a control data item of the plurality of data items, such as discussed above in the context of at least Figure 3.

[0162] At block 1460, the computing device can provide a representation of the data ontology, such as discussed above in the context of at least Figures 3 and 8-11. In some

embodiments, the representation includes a visual representation, such as discussed above in the context of at least Figures 3 and 8-11.

[0163] In some embodiments, method 1400 can also include: receiving additional data from the one or more data sources; storing a subset of the additional data in a second data frame, where the second data frame includes the plurality of data items, and where the data in the second data frame differs from data in the first data frame, and changing the first reference of the first data pin to refer to the first data item in the second data frame, as discussed above in the context of at least Figure 3.

[0164] In other embodiments, method 1400 can also include: specifying a designated control for the control data item of the control pin, and after specifying the designated control, generating a data frame associated with the designated control, such as discussed above in the context of at least Figure 3. In particular embodiments, the designated control can be at least one control selected from the group consisting of a control based on a time, a control based on an identifier, and a control based on a location.

[0165] Unless the context clearly requires otherwise, throughout the description and the claims, the words ‘comprise’, ‘comprising’, and the like are to be construed in an inclusive sense as opposed to an exclusive or exhaustive sense; that is to say, in the sense of “including, but not limited to”. Words using the singular or plural number also include the plural or singular number, respectively. Additionally, the words “herein,” “above” and "below" and words of similar import, when used in this application, shall refer to this application as a whole and not to any particular portions of this application.

[0166] The above description provides specific details for a thorough understanding of, and enabling description for, embodiments of the disclosure. However, one skilled in the art will understand that the disclosure may be practiced without these details. In other instances, well-known structures and functions have not been shown or described in detail to avoid unnecessarily obscuring the description of the embodiments of the disclosure. The description of embodiments of the disclosure is not intended to be exhaustive or to limit the disclosure to the precise form disclosed. While specific embodiments of, and examples for, the disclosure are described herein for illustrative purposes, various equivalent modifications are possible within the scope of the disclosure, as those skilled in the relevant art will recognize.

[0167] All of the references cited herein are incorporated by reference. Aspects of the disclosure can be modified, if necessary, to employ the systems, functions and concepts of the above references and application to provide yet further embodiments of the disclosure. These and other changes can be made to the disclosure in light of the detailed description.

[0168] Specific elements of any of the foregoing embodiments can be combined or substituted for elements in other embodiments. Furthermore, while advantages associated with certain embodiments of the disclosure have been described in the context of these embodiments, other embodiments may also exhibit such advantages, and not all embodiments need necessarily exhibit such advantages to fall within the scope of the disclosure.

[0169] The above detailed description describes various features and functions of the disclosed systems, devices, and methods with reference to the accompanying figures. In the figures, similar symbols typically identify similar components, unless context dictates otherwise.

The illustrative embodiments described in the detailed description, figures, and claims are not meant to be limiting. Other embodiments can be utilized, and other changes can be made, without departing from the spirit or scope of the subject matter presented herein. It will be readily understood that the aspects of the present disclosure, as generally described herein, and illustrated in the figures, can be arranged, substituted, combined, separated, and designed in a wide variety of different configurations, all of which are explicitly contemplated herein.

[0170] With respect to any or all of the ladder diagrams, scenarios, and flow charts in the figures and as discussed herein, each block and/or communication may represent a processing of information and/or a transmission of information in accordance with example embodiments. Alternative embodiments are included within the scope of these example embodiments. In these alternative embodiments, for example, functions described as blocks, transmissions, communications, requests, responses, and/or messages may be executed out of order from that shown or discussed, including substantially concurrent or in reverse order, depending on the functionality involved. Further, more or fewer blocks and/or functions may be used with any of the ladder diagrams, scenarios, and flow charts discussed herein, and these ladder diagrams, scenarios, and flow charts may be combined with one another, in part or in whole.

[0171] A block that represents a processing of information may correspond to circuitry that can be configured to perform the specific logical functions of a herein-described method or technique. Alternatively or additionally, a block that represents a processing of information may correspond to a module, a segment, or a portion of program code (including related data). The program code may include one or more instructions executable by a processor for implementing

specific logical functions or actions in the method or technique. The program code and/or related data may be stored on any type of computer readable medium such as a storage device including a disk or hard drive or other storage medium.

[0172] The computer readable medium may also include non-transitory computer readable media such as computer-readable media that stores data for short periods of time like register memory, processor cache, and random access memory (RAM). The computer readable media may also include non-transitory computer readable media that stores program code and/or data for longer periods of time, such as secondary or persistent long term storage, like read only memory (ROM), optical or magnetic disks, compact-disc read only memory (CD-ROM), for example. The computer readable media may also be any other volatile or non-volatile storage systems. A computer readable medium may be considered a computer readable storage medium, for example, or a tangible storage device.

[0173] Moreover, a block that represents one or more information transmissions may correspond to information transmissions between software and/or hardware modules in the same physical device. However, other information transmissions may be between software modules and/or hardware modules in different physical devices.

[0174] Numerous modifications and variations of the present disclosure are possible in light of the above teachings.

CLAIMS

1. A method, comprising:

receiving data from one or more data sources at a computing device;

generating a data frame based on the received data using the computing device, the data frame comprising a plurality of data items;

determining a data ontology at the computing device, wherein the data ontology comprises a plurality of datanodes;

determining a plurality of data pins using the computing device, wherein a first data pin of the plurality of data pins comprises a first reference and a second reference, wherein the first reference for the first data pin refers to a first data item in the data frame, wherein the second reference for the first data pin refers to a first datanode of the plurality of datanodes, and wherein the first datanode is related to the first data item;

obtaining, at the computing device, data for the first data item at the first datanode of the data ontology via the first data pin; and

providing a representation of the data ontology using the computing device.

2. The method of claim 1, wherein the representation comprises a visual representation.

3. The method of claim 1 or claim 2, wherein generating the data frame comprises storing a subset of the received data in the data frame using a pre-loader.

4. The method of any one of claims 1-3, further comprising:

receiving additional data from the one or more data sources;

storing a subset of the additional data in a second data frame, wherein the second data frame comprises the plurality of data items, and wherein the data in the second data frame differs from data in the first data frame; and

changing the first reference of the first data pin to refer to the first data item in the second data frame.

5. The method of any one of claims 1-4, wherein the data ontology is related to a software object hierarchy.

6. The method of claim 5, wherein the second reference refers to a datanode associated with a software object in the software object hierarchy.

7. The method of claim 5 or claim 6, wherein determining the data ontology comprises parsing the software object hierarchy.

8. The method of any one of claims 1-7, wherein the plurality of pins comprise a control pin, and wherein the control pin indicates a control data item of the plurality of data items.

9. The method of claim 8, further comprising:

specifying a designated control for the control data item of the control pin; and

after specifying the designated control, generating a data frame associated with the designated control.

10. The method of claim 9, wherein the designated control is at least one control selected from the group consisting of a control based on a time, a control based on an identifier, and a control based on a location.

11. The method of any one of claims 1-10, wherein the data ontology relates to a chemical molecule.

12. A computing device, comprising:

a processor; and

a tangible computer readable medium configured to store at least executable instructions, wherein the executable instructions, when executed by the processor, cause the computing device to perform functions comprising the method of any one of claims 1-11.

13. The computing device of claim 12, wherein the tangible computer readable medium is a non-transitory tangible computer readable medium.

14. A tangible computer readable medium configured to store at least executable instructions, wherein the executable instructions, when executed by a processor of a computing device, cause the computing device to perform functions comprising the method of any one of claims 1-11.

15. The tangible computer readable medium of claim 14, wherein the tangible computer readable medium is a non-transitory tangible computer readable medium.

16. A device, comprising:

means for receiving data from one or more data sources;

means for generating a data frame based on the received data, the data frame comprising a plurality of data items;

means for determining a data ontology, wherein the data ontology comprises a plurality of datanodes;

means for determining a plurality of data pins, wherein a first data pin of the plurality of data pins comprises a first reference and a second reference, wherein the first reference for the first data pin refers to a first data item in the data frame, wherein the second reference for the first data pin refers to a first datanode of the plurality of datanodes, and wherein the first datanode is related to the first data item;

means for obtaining data for the first data item at the first datanode of the data ontology via the first data pin; and

means for providing a representation of the data ontology.

ABSTRACT

Methods and apparatus are provided for representing streamed and structured information. A computing device can receive data from data sources. The computing device can generate a data frame that includes data items based on the received data. The computing device can determine a data ontology, where the data ontology can include datanodes. The computing device can determine data pins which include a first data pin. The first data pin can include a first reference and a second reference. The first reference can refer to a first data item in the data frame and the second reference can refer to a first datanode of the plurality of datanodes. The first datanode can be related to the first data item. The computing device can obtain data for the first data item at the first datanode via the first data pin and then can provide a representation of the data ontology.

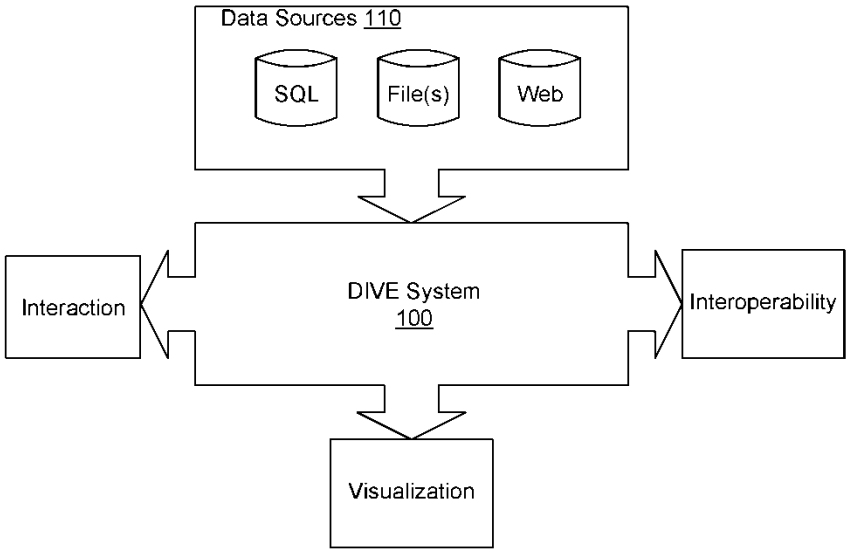


FIG. 1

Appendix Figure B.1 DIVE patent Figure 1

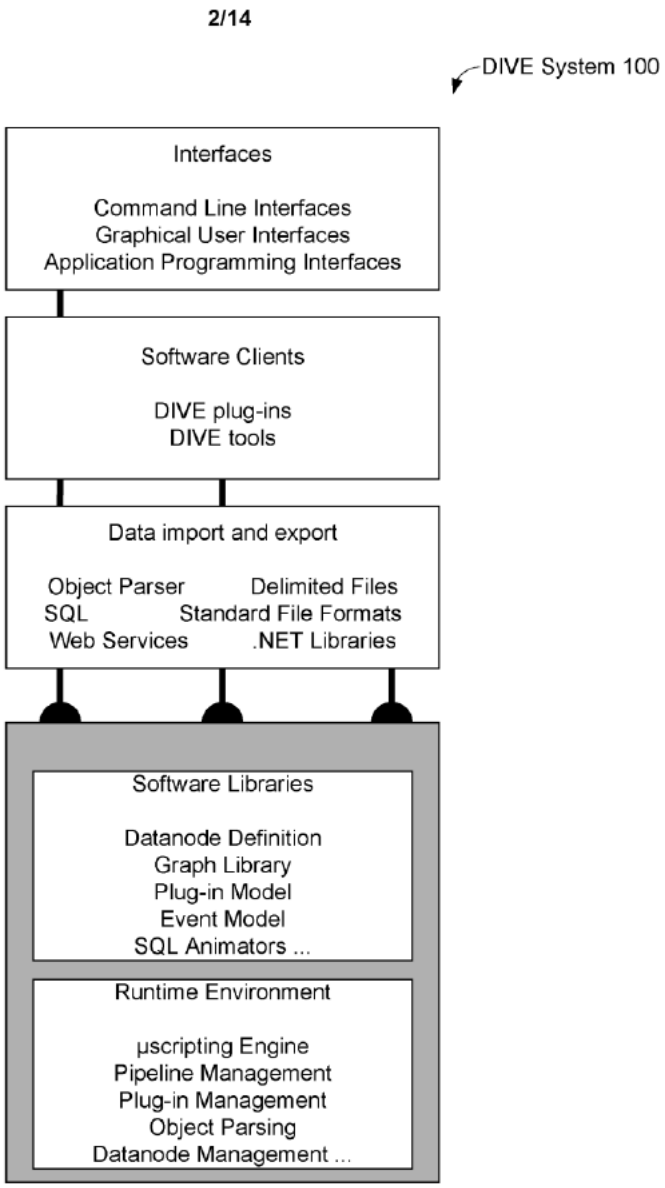


FIG. 2

Appendix Figure B.2 DIVE patent Figure 2

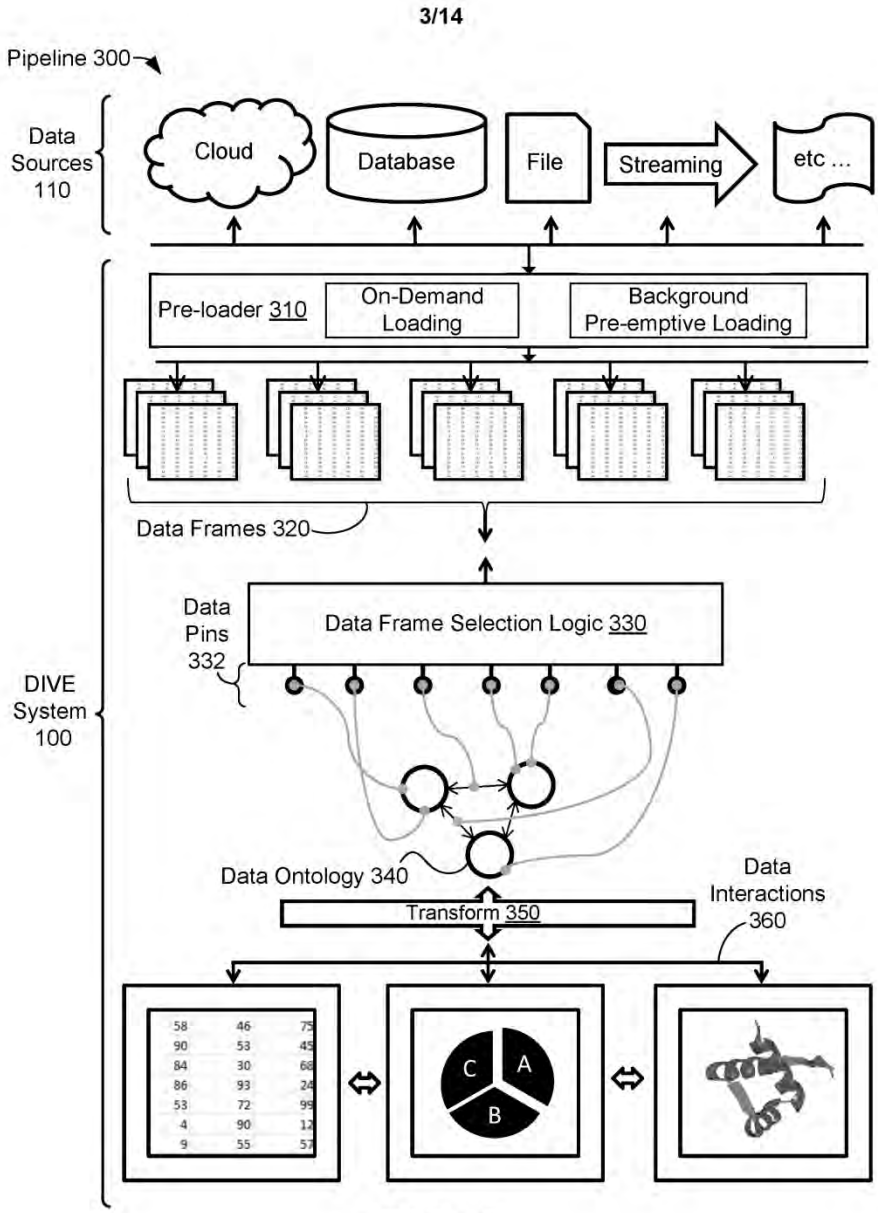


FIG. 3

Appendix Figure B.3 DIVE patent Figure 3

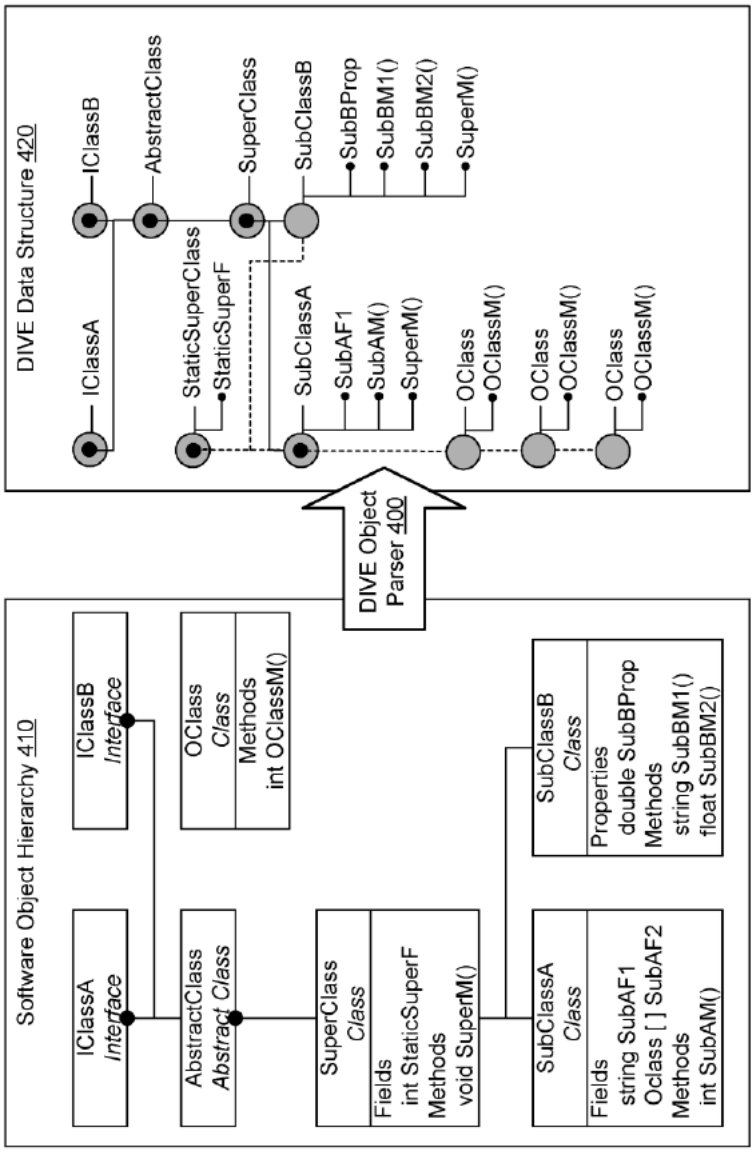


FIG. 4

Appendix Figure B.4 DIVE patent Figure 4

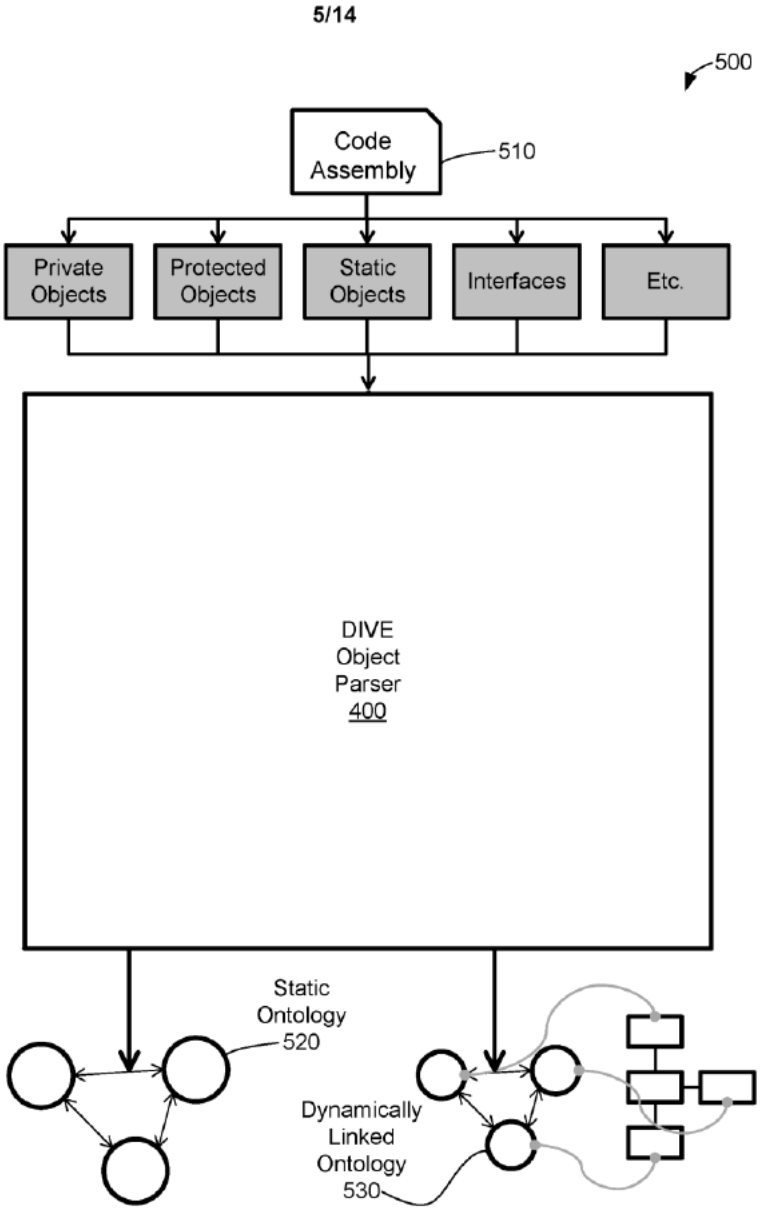


FIG. 5

Appendix Figure B.5 DIVE patent Figure 5

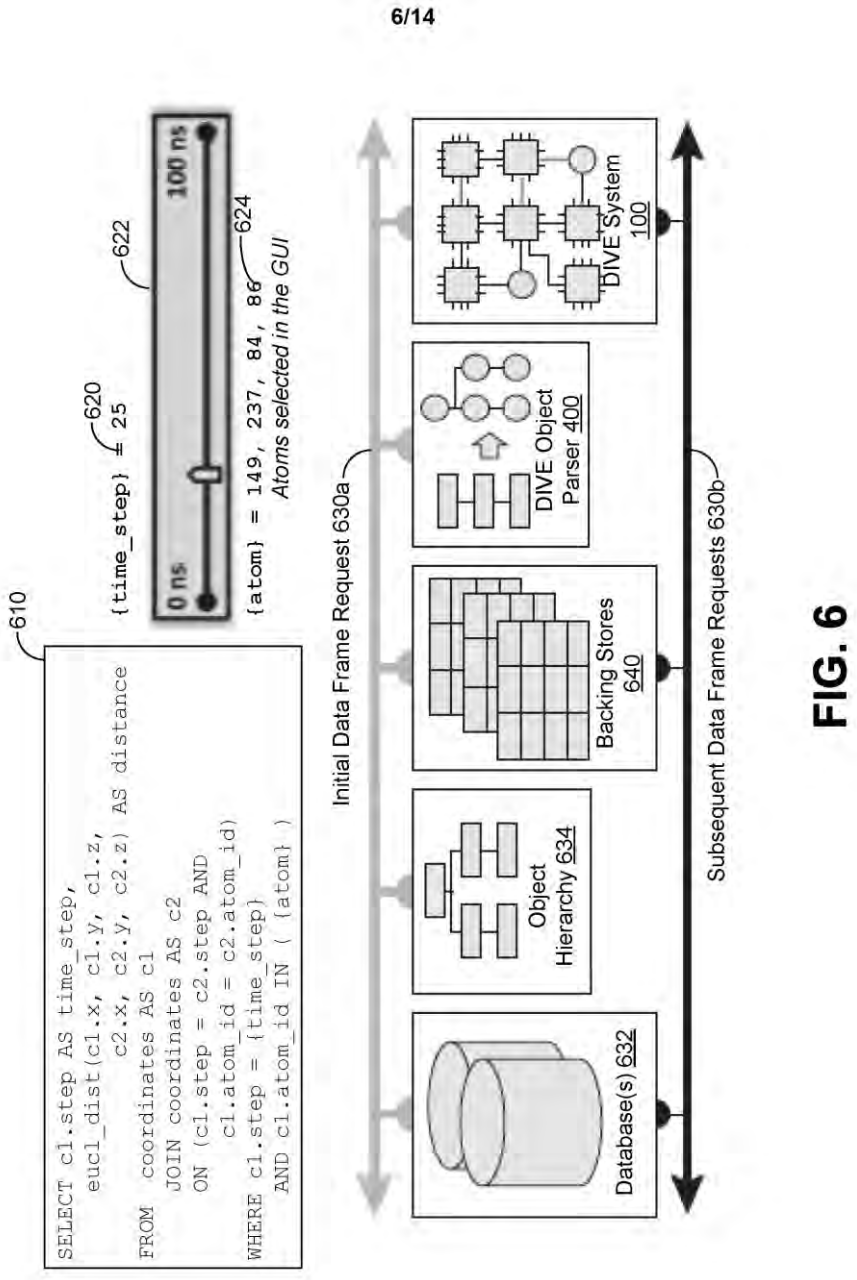


FIG. 6

Appendix Figure B.6 DIVE patent Figure 6

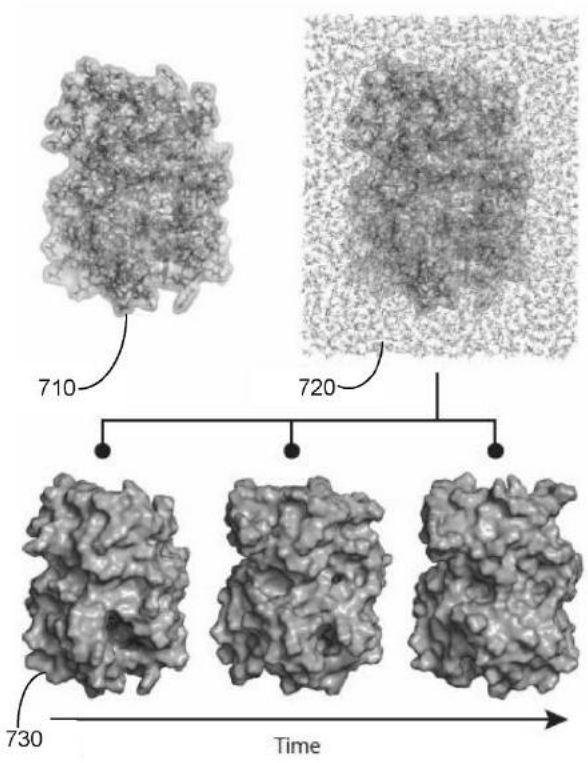


FIG. 7

Appendix Figure B.7 DIVE patent Figure 7

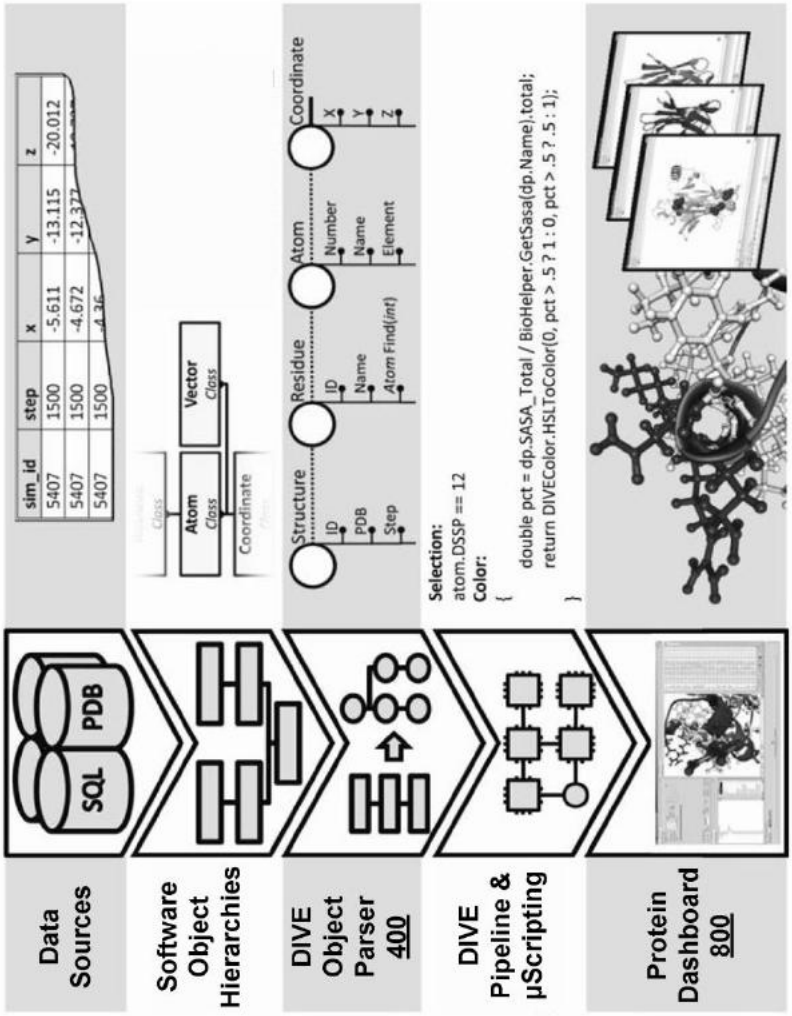


FIG. 8

Appendix Figure B.8 DIVE patent Figure 8

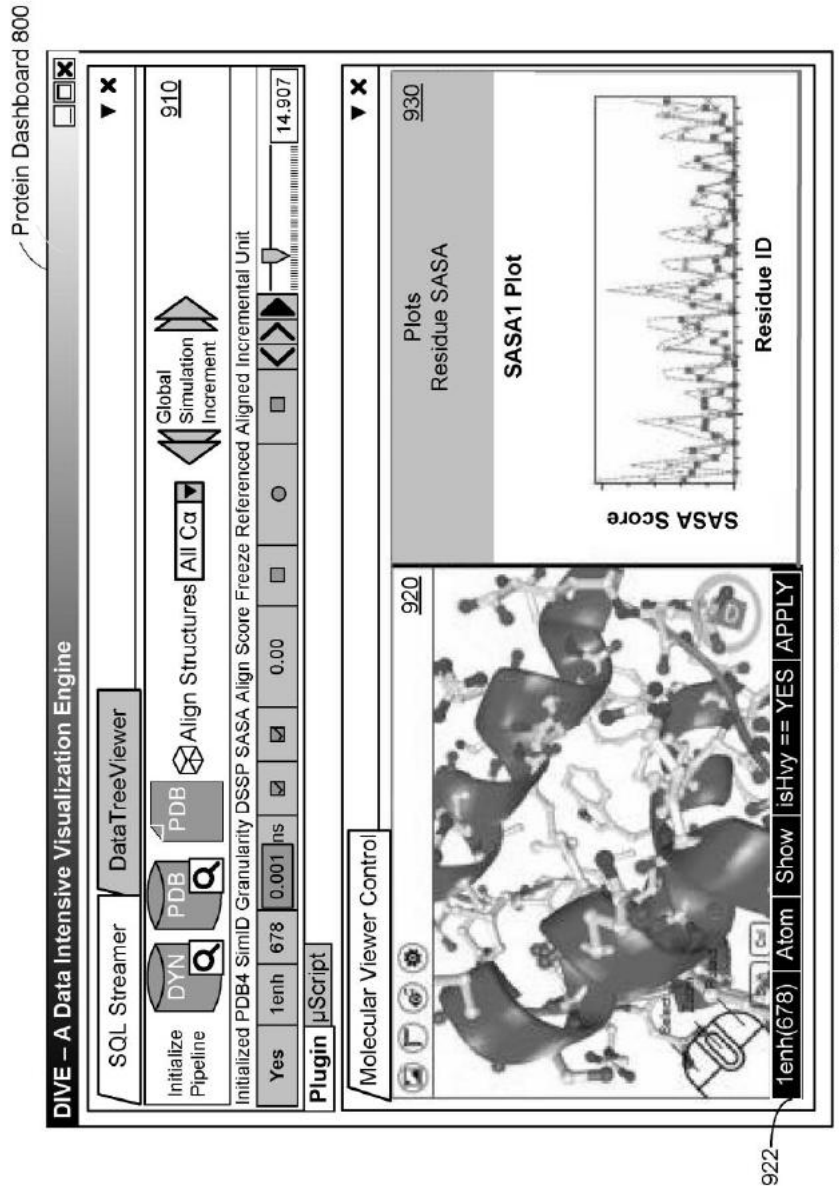


FIG. 9

Appendix Figure B.9 DIVE patent Figure 9

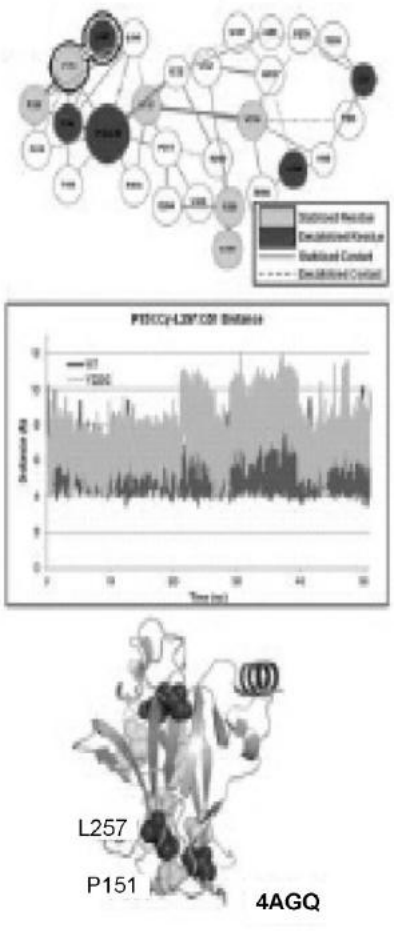


FIG. 10A

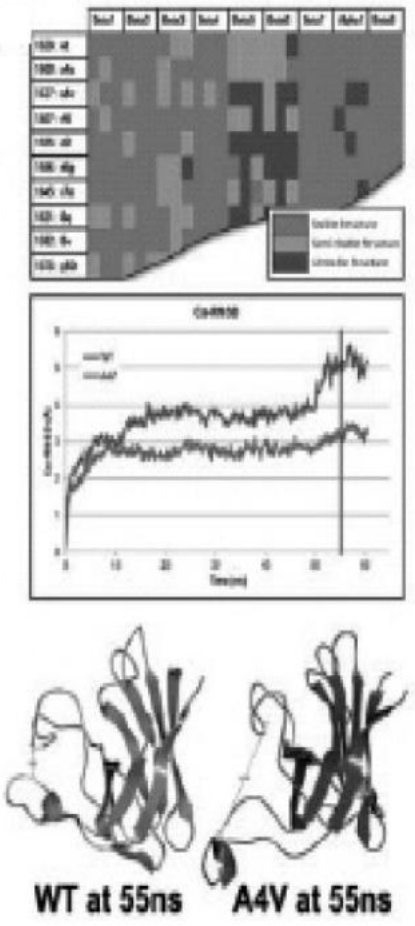


FIG. 10B

Appendix Figure B.10 DIVE patent Figure 10

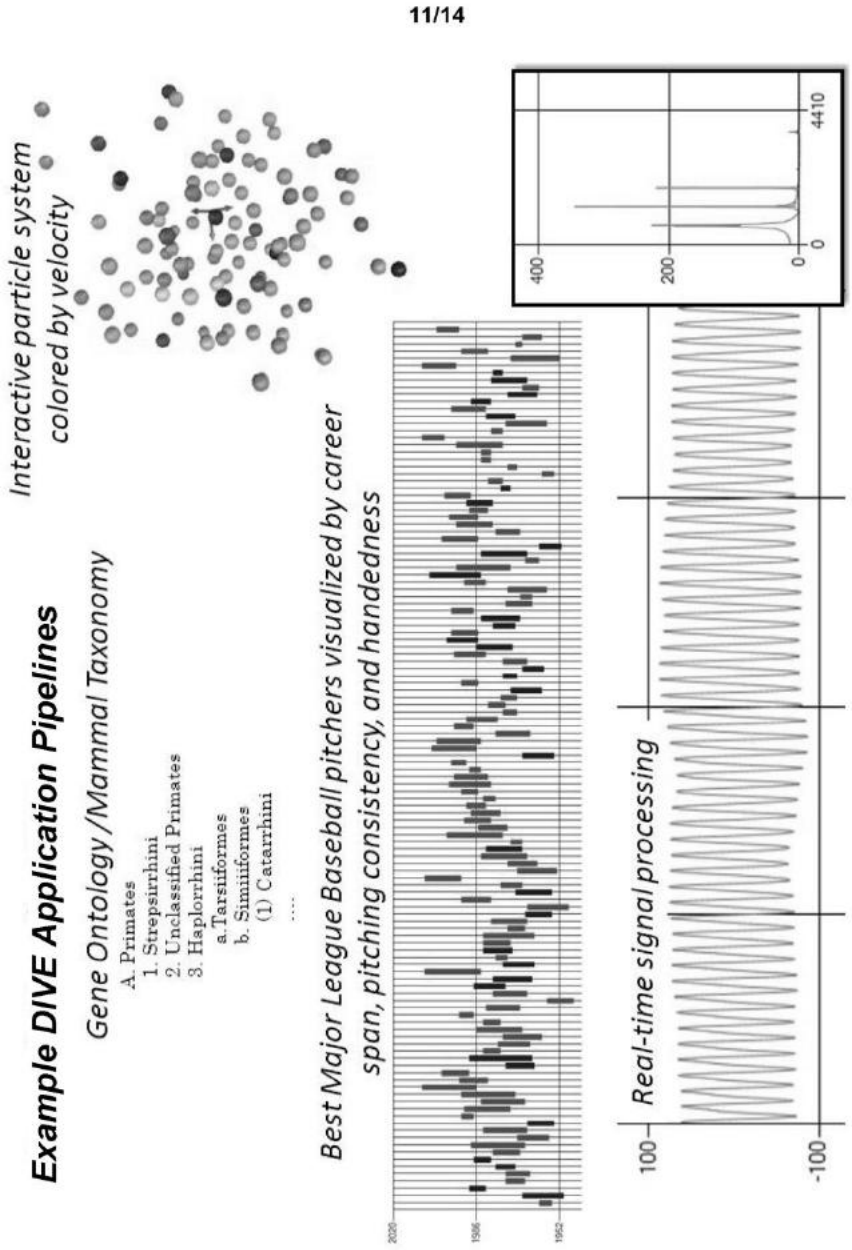


FIG. 11

Appendix Figure B.11 DIVE patent Figure11

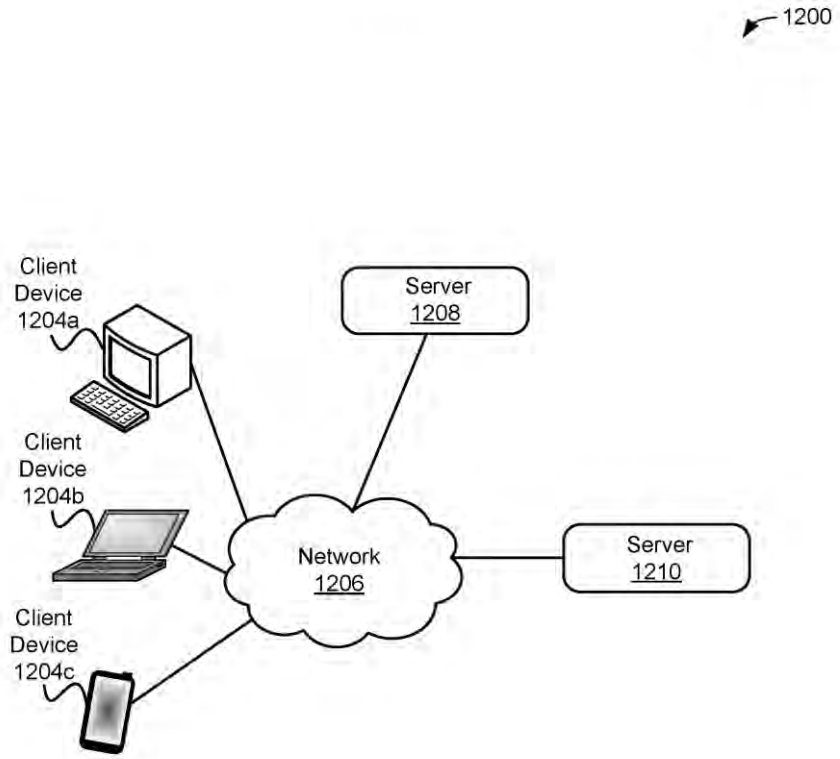


FIG. 12

Appendix Figure B.12 DIVE patent Figure 12

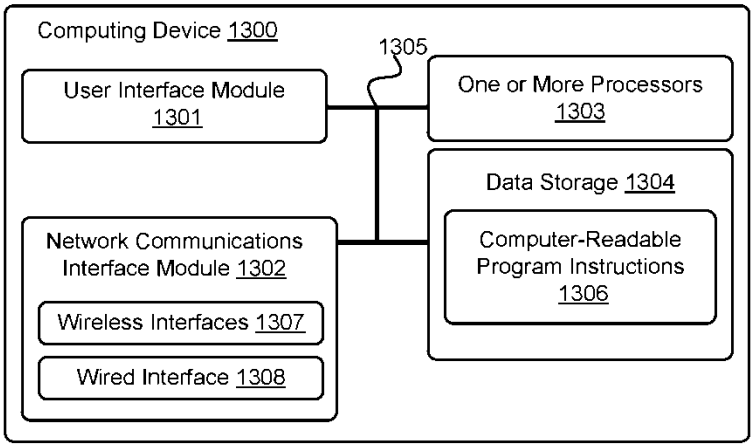


FIG. 13A

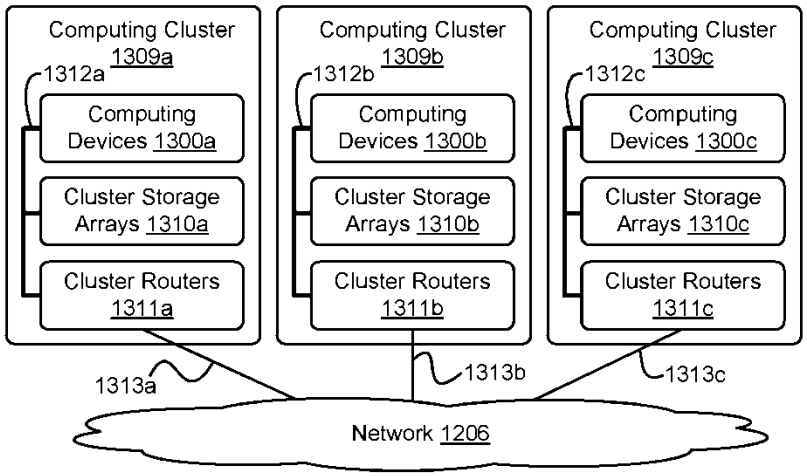


FIG. 13B

Appendix Figure B.13 DIVE patent Figure 13

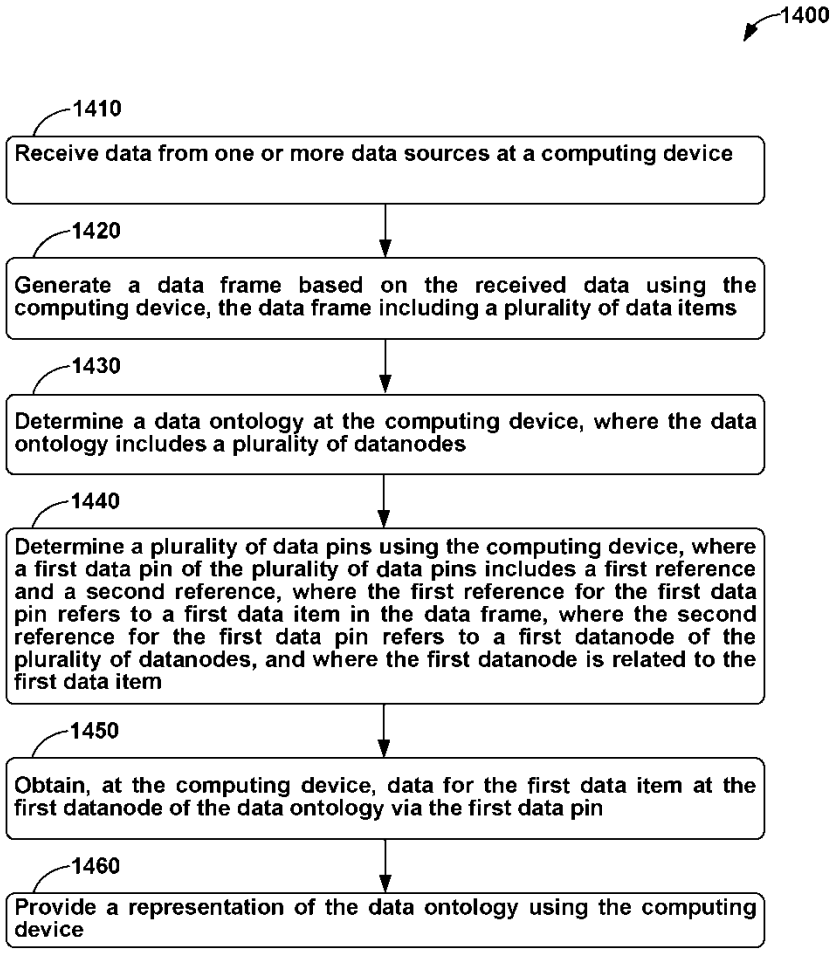


FIG. 14

Appendix Figure B.14 DIVE patent Figure 14

Appendix C

SUPPLEMENTARY MATERIALS FOR DIVE: A DATA INTENSIVE VISUALIZATION ENGINE

The contents of Chapter 3 and this appendix were previously published in the journal Bioinformatics (Bromley et al. 2014).

C.1 Protein Dashboard

The protein dashboard (Appendix Figure C.3) is a data exploration application that uses the DIVE framework and Dynameomics Application Programming Interface (Rysavy et al. 2014) 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 Appendix Figure C.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.

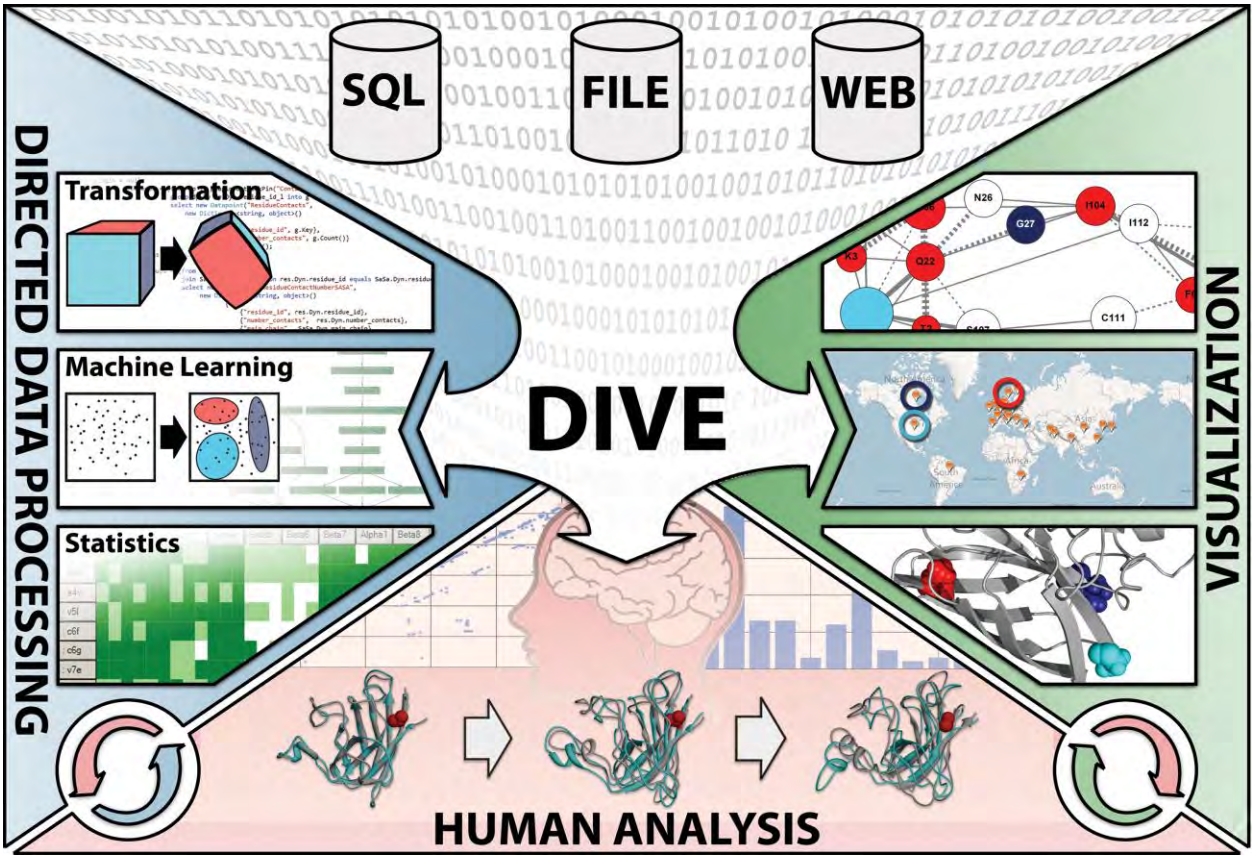
C.2 Gene Ontology

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 (Appendix Figure C.4). A more complex example that selects sections of the taxonomy in order to chart groups of interest is shown in Appendix Figure C.5. 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.

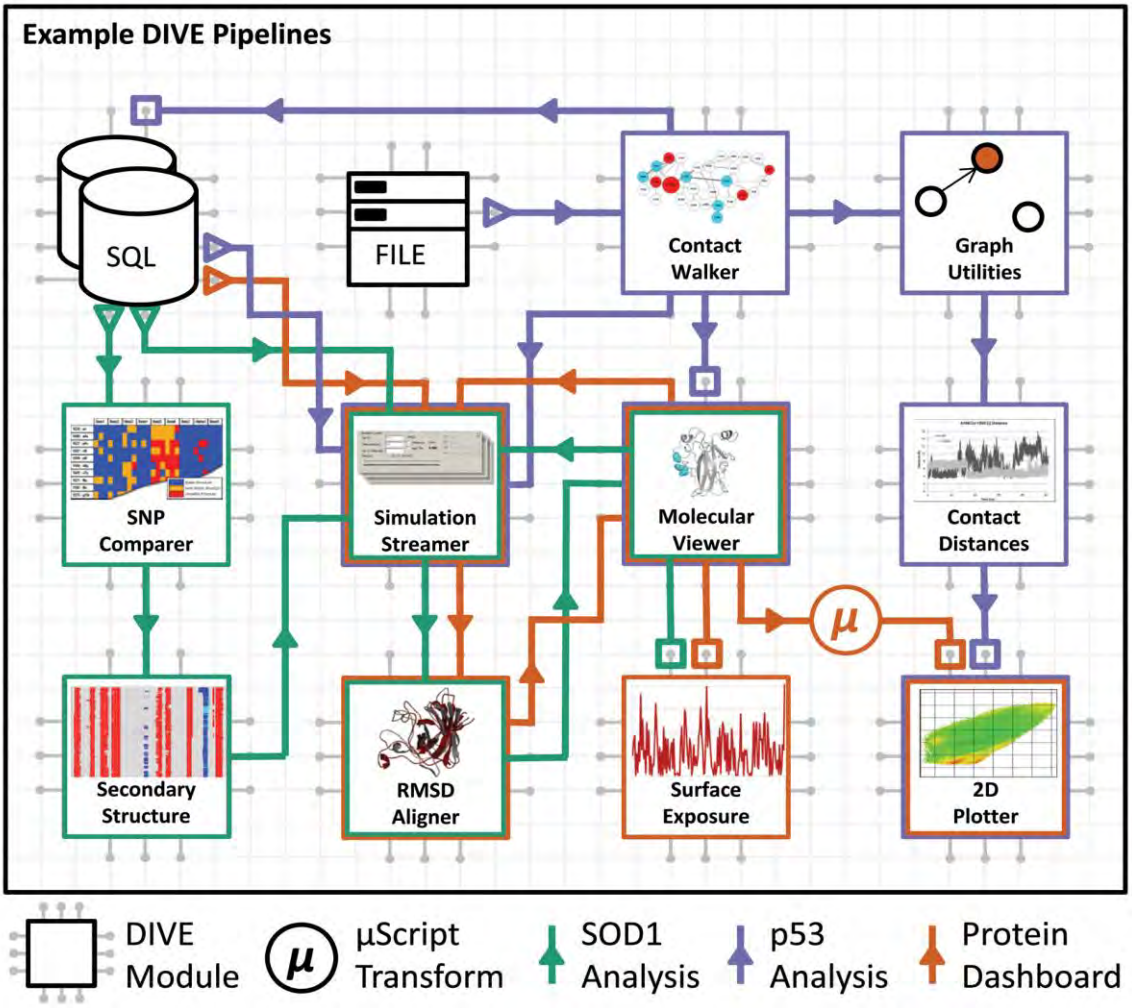
C.3 Professional Baseball Statistics

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 (Appendix Figure C.6). 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, Appendix Figure C.7), 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

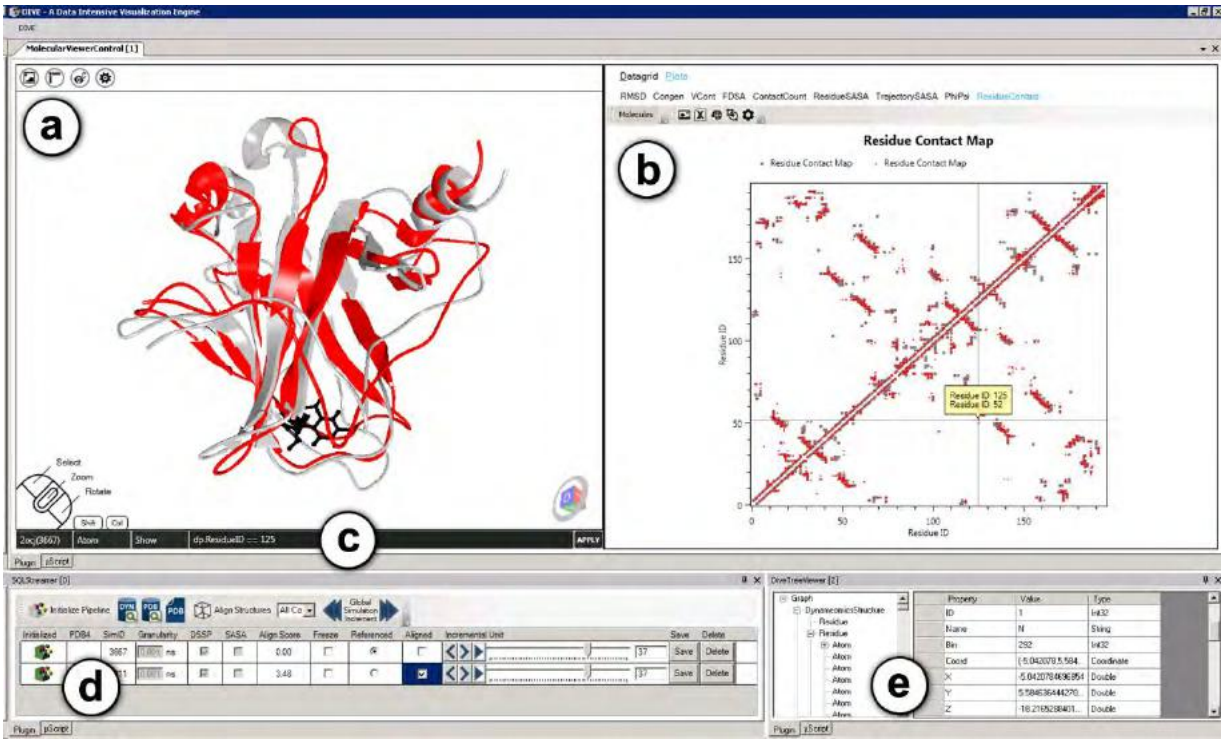
sources such as web searching (Appendix Figure C.8), data anomalies were easily explained. For example, Whitey Ford, one of the most consistent baseball pitchers, missed the 1951 season due to military service.



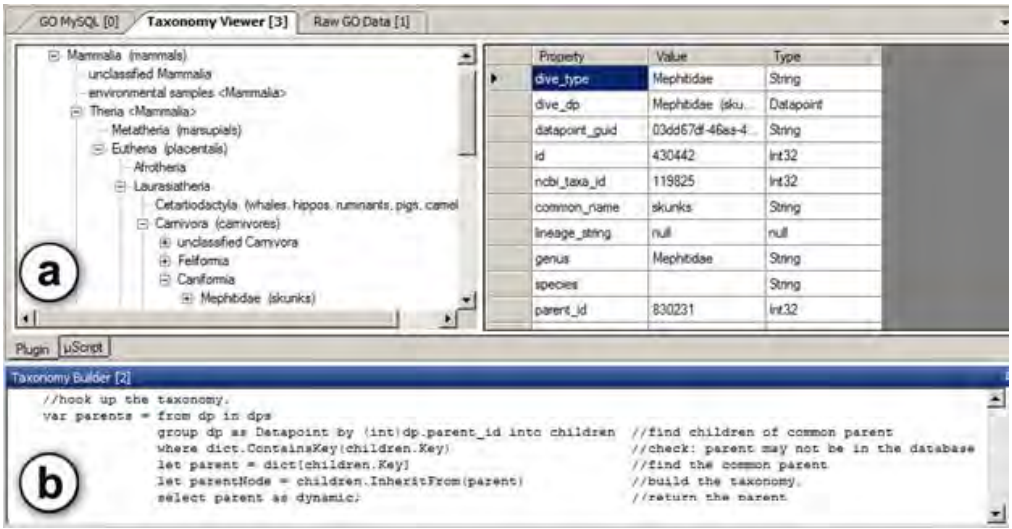
Appendix Figure C.1 Schematic of the data flow within DIVE.
Data enter DIVE from a variety of sources and are processed, analyzed and visualized by specific DIVE modules. The user interacts with these modules, iteratively refining the investigation as scientific insights develop. Analyses, visualizations, 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.



Appendix Figure C.2 Conceptual representation of DIVE modules and 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-squared 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.

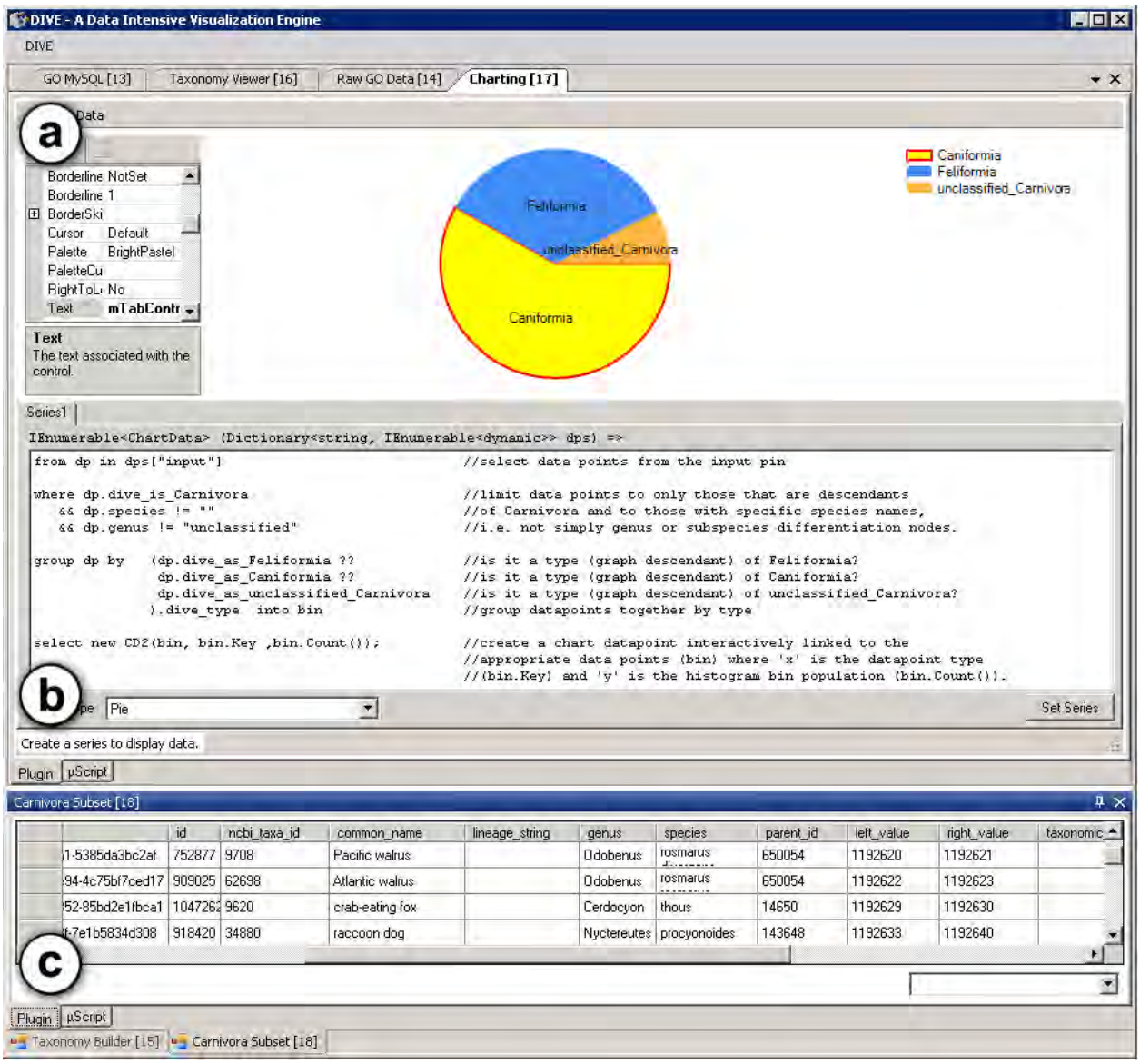


Appendix Figure C.3 Screenshot of the Protein Dashboard.
(a) 3D molecular visualization module depicting two aligned structures of p53 with the mutated residue highlighted in black. (b) Interactive residue-residue contact maps of the two p53 structures. The crossbar indicates a contact of the mutated 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.



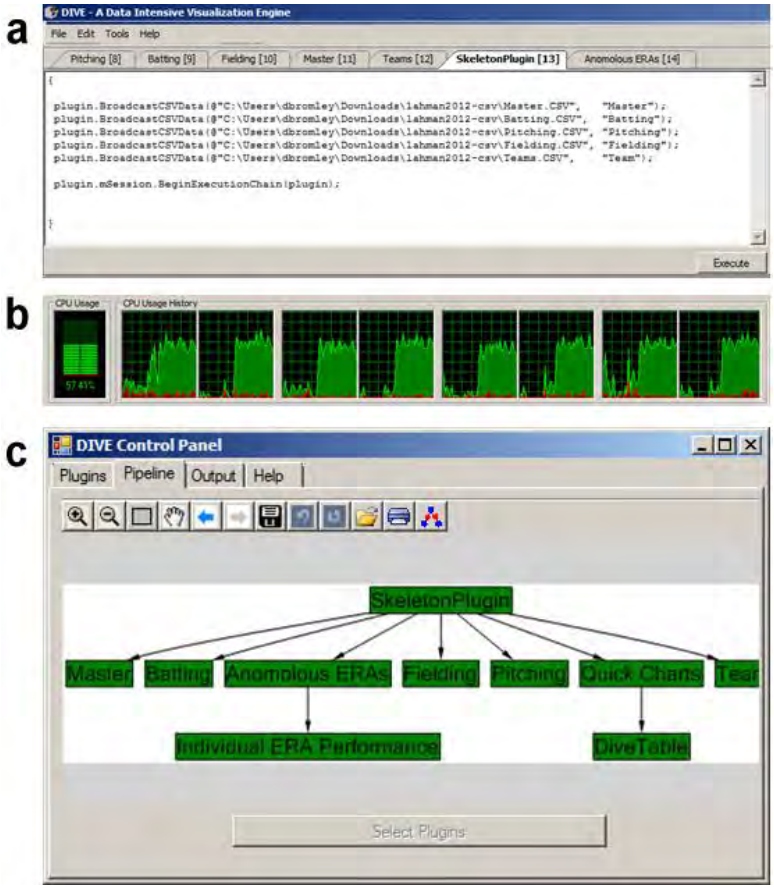
Appendix Figure C.4 Screenshot of DIVE displaying information from the Gene Ontology database.

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.



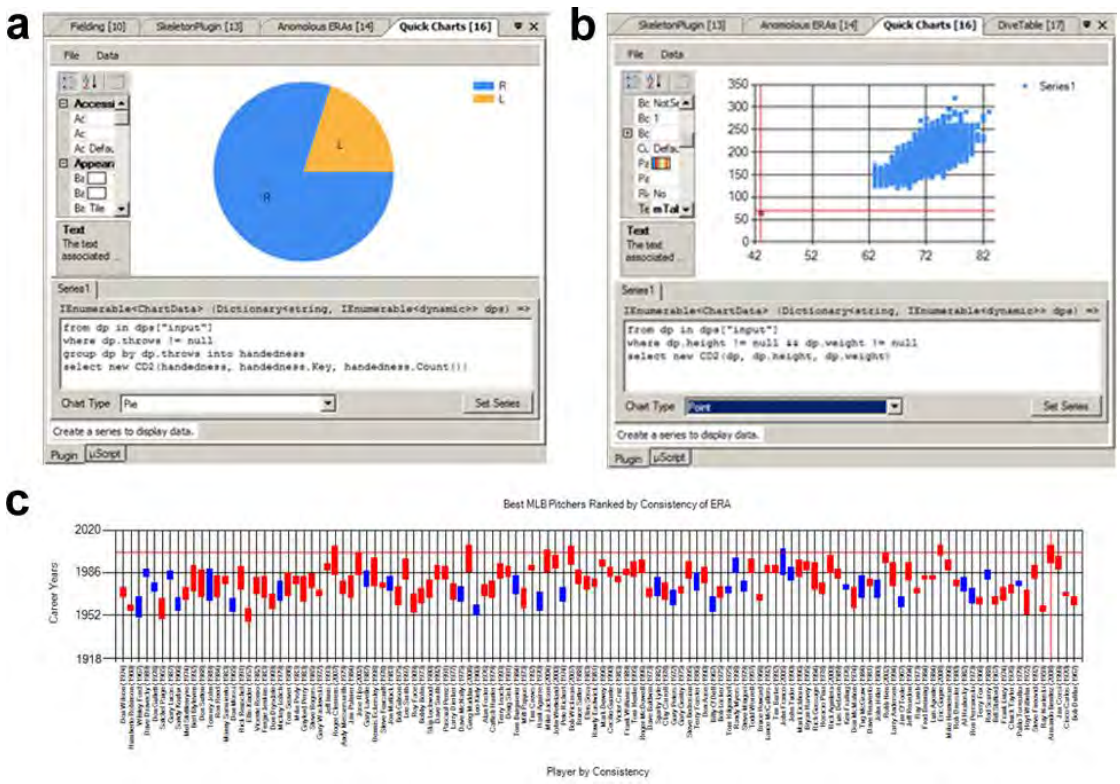
Appendix Figure C.5 Screenshot of DIVE showing investigation of the Gene Ontology species taxonomy

This demonstration is included in the downloadable DIVE software package. (a) Chart showing various types of carnivores. (b) Script to create histogram of carnivore sub-species 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).

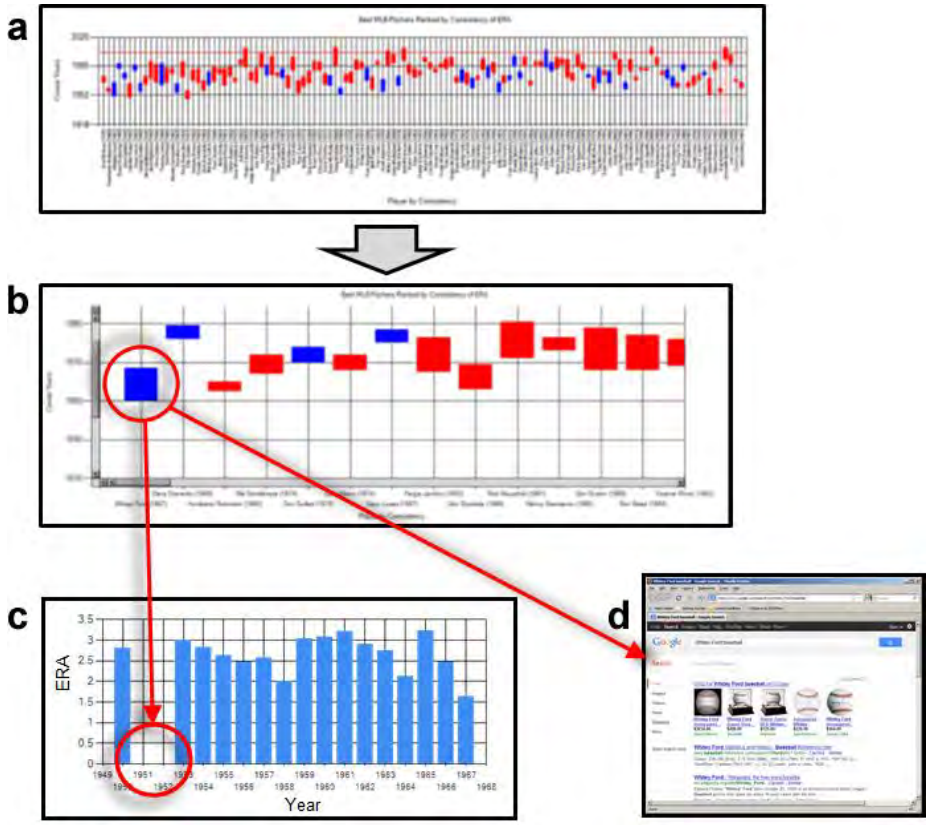


Appendix Figure C.6 Reusable DIVE components used to analyze professional baseball statistics

(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.

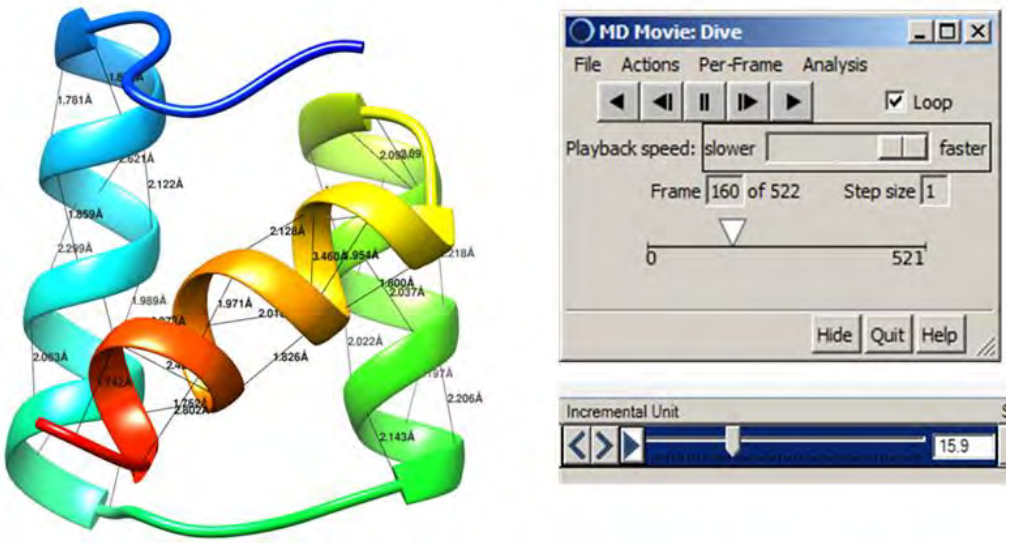


Appendix Figure C.7 Reusable DIVE charting plugins used for data exploration.
(a) Pie chart showing distribution of right-hand and left-hand pitchers. (b) Scatter plot of players' height and weight. Outlier identifies Edward Carl Gaedel, a major league player with dwarfism who played in 1951. (c) A more sophisticated DIVE chart illustrating relationships among career timespan, pitching handedness, and earned run average (ERA) consistency.



Appendix Figure C.8 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.



Appendix Figure C.9 DIVE using the Chimera molecular dynamics movie plugin
Chimera (Pettersen et al. 2004) is one of the major protein visualizers in use today. DIVE is able to incorporate tools like Chimera into a pipeline for scientific investigation.

Appendix D

SUPPLEMENTARY P53 MUTANT ANALYSES

These analyses are a supplement to Chapter 6.

Mutant Analyses

D.1 DNA-Contact Mutants

R248Q

The R248Q mutation is a DNA-contact mutation located in the L3 loop. Previous work (Bullock et al. 2000) has shown that this mutation destabilizes the p53 protein by 1.9 kcal/mol at 10° C and abolishes its DNA binding affinity at 20° C. Average Cα RMSD values of the simulations were 4.8 ± 0.2Å, 4.5 ± 0.1Å, and 4.7 ± 0.2Å. The average Cα RMSD values for the DNA-binding residues for the three simulations were 7.2 ± 0.4Å, 5.8 ± 0.2Å, and 5.6 ± 0.4Å. Cα RMSD analysis of the R248Q simulations showed that the main regions of structural deviation from starting structure were the loop-sheet-helix region, L2, the S7/S8 loop and, to a lesser degree, L3. Relative to the wild type, the R248Q mutant demonstrated larger-than-average Cα RMSD values in the loop-sheet-helix region (5Å), L2 (5Å), and the S6/S7 loop (4Å). Smaller-than-wild-type Cα RMSD values were observed in the S7/S8 loop (4Å), L3 (2Å), and the S9/S10 loop (2Å).

All three simulations showed a separation between L1 and H2 (~20Å) and wild type-like distance between L2 and L3. In one simulation, L2 swung away from S5 by 25Å. Secondary-structure analysis showed that one simulation lost H1 helical structure, one simulation gained α-

sheet structure between S1 and S3, and two simulations gained helical structure in L1. Cα RMSF values were mostly wild type-like with the exception of an increase (~2Å) in the loop-sheet-helix region, an increase (~2Å) in one simulation in the S7/S8 loop, and a small (< 1Å) decrease in fluctuation in the zinc-binding region of L2.

There were increases in contacts between L2 and L3 (notably, near the L2 zinc-binding residues), between S5 and S6, and within L2 near the H1 helix. There were decreases in contacts between H2 and L1, H2 and S2, and H2 and S2', consistent with the separation seen between H2 and the rest of the loop-sheet-helix region. Relative to wild type, across the three simulations, Q248 demonstrated increased contact gains with Q165 (L2), A276 (the S10/H2 loop), and T284 (H2), and both losses and gains with S240 (L3).

R273C

The R273C mutation is a DNA-contact mutation located in the S10 β strand. Average Cα RMSD values of the simulations were 4.6 ± 0.2Å, 3.6 ± 0.3Å, and 3.9 ± 0.1Å. The average DNA-contact average Cα RMSD values were 6.1 ± 0.3Å, 4.0 ± 0.1Å, 3.2 ± 0.4Å. Cα RMSD analysis of the three R273C simulations showed the β core typically wild type-like with the exception of S1 becoming slightly displaced in two simulations (~1.5Å Cα RMSD). All three simulations showed deviations from starting structure in the S7/S8 loop. One simulation showed large structural deviations in the L2 loop (~6Å average) and one simulation showed disruptions (~5Å average) in the loop-sheet-helix region.

Secondary-structure analysis showed a loss of H1 helical structure in one simulation, disrupted S1 β structure in all three simulations, and a gain of α-sheet structure between S3 and S8 in one simulation. One simulation showed a gain of helical content in L1 concurrent with a separation between L1 and H2, and one simulation showed a gain of helical content in L3. Cα RMSF values were basically wild type-like with small (< 2Å) increases in L1 and both increases and decreases (<2Å) relative to wild type in L2.

L1 separated from H2 in one simulation but the L2:L3 distances were within wild-type ranges. Relative to wild type, C273 lost contact occupancy with D281 and E285 (H2) in all three simulations. The contact occupancy between Y220 (the S7/S8 loop) and I232 (S8) was increased in every simulation, as was the overall contact between S5 and S7.

R273H

The R273H mutation is a DNA-contact mutation in the S10 β strand. Previous work (Bullock et al. 2000) has shown that this mutation destabilizes the p53 protein by 0.5 kcal/mol at 10° C and abolishes DNA binding affinity at 20° C relative to wild type. Average Cα RMSD values of the simulations were 3.2 ± 0.1Å, 3.7 ± 0.1Å, and 3.6 ± 0.2Å. The average DNA-contact average Cα RMSD values were 2.3 ± 0.2Å, 4.3 ± 0.1Å, and 4.1 ± 0.3Å. Cα RMSD analysis of the three R273H simulations indicated that the primary regions of structural deviation were L2, L3, and the S7/S8 loop. L2 and the S6/S7 loop demonstrated values as much a 5Å greater than wild type and the S7/S8 loop had values as much as 5Å less than wild-type values. The β-core was

relatively stable in all three simulations with average Cα RMSD values ranging between 1.6Å and 2.2Å.

Secondary-structure analysis showed that most structures were maintained for the majority of the simulation time; exceptions to this involved one simulation that showed an intermittent loss of H2 helical structure, one simulation that showed a gain of L3 helical structure, two simulations that showed an intermittent loss of S1 β content, and one simulation that showed an intermittent gain of α-sheet content between strands S1 and S3. Cα RMSF values were very consistent among the simulations and all values were within wild-type ranges.

Like the R273C mutant, the H273 mutation site consistently lost contact with D281 and E285 (H2), although, unlike R273C, R273H did not demonstrate a significant separation between L1 and H2, despite small but widespread Cα RMSD values in that region. L2 and L3 did not separate appreciably either, although two simulations showed L2 separating from S5 (16Å, 22Å). Contact between H1 and the C-terminal half of L2 increased, as did contacts between S5 and S6.

D.2 DNA-Region Mutants

F134L

The F134L mutation is located in S2' β sheet of the loop-sheet-helix region. Previous work (Bullock et al. 2000) has shown that this mutation destabilizes the wild type by 4.8 kcal/mol at 10° C and reduces the DNA binding affinity at 20° C by 66%. Average Cα RMSD values of the simulations were 4.8 ± 0.2Å, 4.7 ± 0.1Å, and 4.3 ± 0.1Å. The average Cα RMSD values for the DNA-contact residues were 6.3 ± 0.2Å, 4.9 ± 0.2Å, and 6.9 ± 0.3Å. Cα RMSD analysis of the simulations indicated that the β core and the overall protein were relatively stable. Most of the

structural deviation from the starting structure occurred in the loop regions; all three simulations exhibited deviation from wild-type structure in L2 (including H1), L3, the S7/S8 loop, and the loop-sheet-helix region. Some simulations showed larger-than-wild-type values (~5Å) in L1, the S6/S7 loop, and the S9/S10 loop. One simulation showed a large (24Å) separation between L1 and H2 and one simulation showed a large separation (15Å) between L2 and L3.

Secondary-structure analysis indicated that H1 helical structure was lost in two simulations and that one simulation showed a gain of β structure in the N-terminal loop region (residues 102-104) resulting in a small novel β strand that aligned with S10. Cα RMSF values were mostly within wild-type ranges with some small (~1Å) increased fluctuations near L1, the S3/S4 loop and the S9/S10 loop. A slightly larger (~2.5Å) increase in Cα RMSF was demonstrated by two simulations in the S7/S8 loop, and all three simulations showed a small (<1Å) decrease in fluctuation in the H1 region of L2. Contact losses were seen between S1 and S2', L1 and the S10/H2 loop, and L2 and S9, and contact gains were seen between L2 and the S6/S7 loop. L134 lost contact with E285 in all three simulations.

G245S

The G245S mutation is located in the L3 loop and is one of six "hot spot" mutations (Bullock et al. 2000). Previous work has shown that the mutation destabilizes the protein by 1.2 kcal/mol at 10° C and reduces its DNA-binding ability at 20° C by 37% (Bullock et al. 2000). Average Cα RMSD values of the simulations were 5.3 ± 0.2Å, 4.2 ± 0.2Å, and 4.6 ± 0.4Å. The average DNA-contact residue Cα RMSD values were 6.7 ± 0.2Å, 4.1 ± 0.4Å, and 3.3 ± 0.1Å. Cα RMSD

analysis of the G245S simulations showed consistent structural deviation in the loop-sheet-helix, S3/S4 loop, L2, S7/S8 loop, and L3 regions. Relative to wild type, there were increased Cα RMSD values in both the loop-sheet-helix region (< 5Å) and L2 (5-10Å), this last value a consequence of the C-terminus of L2 swinging away from S5 by 27Å and back toward the N-terminus of L2. L1 and H2 separated by approximately 10Å in two simulations and L2 and L3 separated between 10Å and 20Å in two simulations. The β core remained relatively stable in all simulations although one simulation exhibited some increased movement in the N-terminal residues of S8.

All three G245S simulations exhibited α sheet including one three-strand sheet among the S3, S8, and S5 regions, and one two-strand five-residue-long sheet between S1 and S3. Cα RMSF values were mostly wild type-like with small (<1Å) increases in the loop-sheet-helix region and larger (~2Å) increases in the S7/S8 loop and at the C-terminal end of L2. Almost all of the contact between L2 and S5 was abolished and the contact between the N-terminal half of L2 and the C-terminal half of L2 was also decreased Contact occupancy was lost between F134 (S2') and E285 (H2) and gained between A159 (S4) and H214 (S7).

H168R

The H168R mutation is located in the L2 loop region. Previous work (Joerger et al. 2005) indicates that this mutation destabilizes the protein by 3 kcal/mol at 10° C. Average Cα RMSD values of the simulations were 4.7 ± 0.2Å, 4.5 ± 0.1Å, and 4.3 ± 0.2Å and average DNA-contact residue Cα RMSD values were 4.6 ± 0.3Å, 6.2 ± 0.1Å, and 3.2 ± 0.1Å. Cα RMSD analysis of the

H168R simulations exhibited a consistent disruption in the L2, L3, and S7/S8 loops and the loop-sheet-helix region. Relative to wild type, increased Cα RMSD values (~5Å) were observed in S1, S3, S8, L2, and the S6/S7 loop, and a slight decrease (4Å) was observed in the S7/S8 loop; the rest were within wild-type ranges. One simulation showed a large (20Å) separation between L1 and H2 and one simulation showed a mild (10Å) separation between L1 and H2; all simulations retained wild-type-like separation between L2 and L3. Two simulations showed ~20Å separations between L2 and S5.

Secondary-structure analysis indicated a loss of β structure in S1, S5 and S6. Two simulations showed a gain of helical structure at the mutation site and two simulations exhibited α-sheet formation between S1 and S3. Cα RMSF values were mostly within wild-type ranges with the exception of slightly increased fluctuation in the L1 region (<2Å) and the S7/S8 loop region (<1Å), and slightly decreased fluctuation in the H1 region of L2 (<1Å). Contact was increased between the S3/S4 loop and the S7/S8 loop, between loops L2 and L3, and between strands S5 and S7.

R249S

The R249S mutation is one of four structural "hot spot" mutations and is located in L3. Previous work (Bullock et al. 2000) indicates that this mutation destabilizes the p53 protein by 1.9 kcal/mol at 10° C and completely abolishes its DNA-binding ability at 20° C. Average Cα RMSD values of the simulations were 4.9 ± 0.1Å, 3.9 ± 0.1Å, and 4.4 ± 0.1Å. The average Cα RMSD values for DNA-contact residues were 3.3 ± 0.1Å, 3.7 ± 0.1Å, and 5.3 ± 0.2Å. Cα RMSD analysis of the

R249S simulations showed consistent structural deviations from wild type in L2, L3, the S7/S8 loop, and the loop-sheet-helix region. Deviations in the S3/S4 loop were observed in one simulation. The β core remained stable and similar to starting structure with the exception of some disruption to S1 concurrent with the disruption to L1 in the loop-sheet-helix region. L1 moved apart from H2 in all three simulations (~20Å, ~7Å, ~7Å).

In general, secondary structures were maintained throughout the simulation. Exceptions to this were a loss of H1 helical content in one simulation, a gain of helical content in L1, a gain of helical content in L2, and a gain of α-sheet content between S1 and S3 in all three simulations. Increases in Cα RMSF relative to wild-type (~2Å) were observed in the loop-sheet-helix region, L2, and the S7/S8 loop. Small (<1Å) decreases in Cα RMSF were seen at E180 (H1) and N210 (S6/S7 loop).

The N-terminal loop decreased contact with both S1 and the S2/S2' loop and increased contact with S3. Contacts between L1 and the S10/H2 loop were not present. The S6/S7 loop increased contacts with both S4 and L2 while L2 in turn decreased contact with S9. S5 gained contact with L3 and the S5/S6 loop lost contact with S8. S249 lost contact with E171, Y163 lost contact with V172, the L114:T125 and L114:Y126 contacts were lost, and the H115:V122 contact occupancy was increased.

R282W

The R282W mutation is one of four structural "hot spot" mutations and is located in the H2 helix near L1. Previous work (Bullock et al. 2000) indicates that this mutation destabilizes the p53

protein by 3.3 kcal/mol at 10° C and reduces its DNA-binding affinity by 18% at 20° C. Cα RMSD values of the simulations were 4.9 ± 0.2Å, 5.0 ± 0.2Å, and 4.1 ± 0.2Å. The DNA-contact residue Cα RMSD values were 5.1 ± 0.5Å, 5.8 ± 0.3Å, and 4.4 ± 0.3Å and the loop-sheet-helix average Cα RMSD values were 6.0 ± 0.5Å, 6.6 ± 0.2Å, and 4.6 ± 0.6Å. Cα RMSD analysis of the R282W mutations showed consistent structural deviations from wild-type in the loop-sheet-helix region, L2, the S7/S8 loop and, to a lesser degree, L3. Some mild structural deviation was also seen in the S9/S10 loop. The β core remained relatively stable throughout the simulation and the secondary structures were generally maintained throughout the simulations. However, in one simulation, L1 gained helical content and H2 lost structure at the C-terminus, changing the structure of the loop-sheet-helix region. The same simulation also gained α-sheet structure in strands S1 and S3.

L1 separated from H2 to varying degrees across the three simulation s (~20Å, ~15Å, and ~7Å) and in two simulations L2 separated from S5 by ~20Å. Cα RMSF values were generally within wild-type limits with increases (<2Å) in the loop-sheet-helix region and the S7/S8 loop, and decreases (<1Å) in L2. Contacts were lost between L1 and the S2/S2' loop, between L1 and the S10/H2 loop, and between the S3/S4 loop and the S9/S10 loop. L114 lost contact with both T125 and Y126 in all three simulations.

D.3 ZINC-Region Mutants

C242S

The C242S mutation is a zinc-contact mutation that lies in L3. Previous work has shown that this mutation destabilizes the p53 protein by 3.1 kcal/mol at 10° C and abolishes its DNA-

affinity at 20° C (Bullock et al. 2000). Average Cα RMSD values of the simulation were 4.0 ± 0.2Å, 3.8 ± 0.2Å, and 3.5 ± 0.2Å. Average Cα RMSD values of the zinc-coordinating residues were 5.0 ± 0.5Å, 3.8 ± 0.1Å, and 3.6 ± 0.4Å. The three C242S simulations were quite similar, each demonstrating a relatively stable β core with consistent Cα RMSD values in L1, L2 (including H1), L3, and the S7/S8 loop. Deviations beyond wild-type ranges were few, limited to gains (<5Å) in the N-terminal loop and small portions of L2. The L1/H2 and L2/L3 distances did not exceed wild-type ranges.

In general, secondary structures were well-maintained throughout the simulations with the exception of S1 which lost much of its β structure. One simulation exhibited α-sheet content between S1 and S3 and in one simulation H1 elongated in the C-terminal direction by six residues. Contact gains were seen between the S3/S4 loop and the S7/S8 loop, between S4 and the S6/S7 loop, and between S5 and S7. Contact gains between S242 and C176 occurred in all three simulations.

M237I

The M237I mutation is a zinc-region mutation located at the N-terminus of L3 abutting the C-terminus of S8. Previous work has shown that this mutation destabilizes the p53 protein by 3.2 kcal/mol at 10° C and reduces its DNA-affinity by 85% at 20° C (Bullock et al. 2000). Average Cα RMSD values for the last 25ns of the simulations were 4.7 ± 0.1Å, 3.5 ± 0.1Å, and 4.2 ± 0.1Å; the average Cα RMSD values for the zinc-coordinating residues over the same time span were 3.1 ± 0.3Å, 3.1 ± 0.2Å, and 4.4 ± 0.3Å. Cα RMSD analysis showed the β core to be consistently

stable. The S7/S8 loop region was consistently the region most different from the starting structure; the other regions contributing to the structural deviation were L2, L3, and the loop-sheet-helix region. Most Cα RMSD values were within wild-type ranges with only a few increases (<5Å) in the N-terminal loop, L1, and the N-terminal end of L2.

Cα RMSF values were similarly wild-type-like with small increases (2Å) in the N-terminal loop and in L1. L1 separated from H2 between 10Å and 20Åin all three simulations and L2 separated from S5 between 15Å and 20Å in all three simulations. L2/L3 separation was within wild-type limits. DSSP analysis showed most secondary structures to be maintained throughout the simulations although two simulations lost H1 helical structure. Contacts were gained between the S3/S4 loop and the S7/S8 loop, between the S6/S7 loop and L2, between S5 and S6, and between S5 and S7.

R175H

R175H is a zinc-region structural "hot spot" mutation located in the L2 loop. Previous work (Bullock et al. 2000) indicates that this mutation destabilizes the p53 protein by 3.5 kcal/mol at 10° C and completely abolishes DNA-binding affinity at 20° C. Average Cα RMSD values of the simulations were 3.9 ± 0.1Å, 4.5 ± 0.2Å, and 3.8 ± 0.1Å and average Cα RMSD values for the zinc-coordinating residues were 4.1 ± 0.3Å, 4.7 ± 0.2Å, and 3.4 ± 0.1Å. Cα RMSD analysis of the three R175H simulations showed considerable deviation from the starting structure in the loop-sheet-helix region, the S3/S4 loop, L2, L3, and the S7/S8 loop. The β core remained relatively stable and wild type-like throughout the simulation, maintaining Cα RMSD values

typically < 2Å. Cα RMSD values were greater than wild type by approximately 5Å in the loop-sheet-helix region and less than wild type by approximately 3Å in the S7/S8 loop. L2 demonstrated Cα RMSD values both greater and less than wild-type ranges across the three simulations

With the exception of one simulation that lost helical structure in H1, secondary structure was well-maintained throughout the simulations, including the S1 region which was degraded in most other simulations. One simulation showed a gain of β-sheet structure between the N-terminal loop (residues 101-103) and strand S10; this same simulation showed a gain of α-sheet structure between strands S1 and S3. Cα RMSF values were mostly wild-type-like with small (1Å) increases in L1 and the S7/S8 loop and decreases near the C-terminus of H1. Relative to wild type, contacts increased between L2 and S7 and between H1 and L3. L1 and H2 separated in two simulations by 15Å-20Å and L2 separated from S5 by 15Å-20Å, slightly more than demonstrated by wild-type.

D.4 β-Sandwich Mutants

I195T

The I195T mutation is a β-sandwich mutation located at the N-terminal end of S5. Previous work (Bullock et al. 2000) indicates that this mutation destabilizes the p53 protein by 4.1 kcal/mol at 10° C relative to wild type and reduces its DNA-binding affinity by 64% at 20° C. Average Cα RMSD values of the simulations were 3.9 ± 0.2Å, 4.9 ± 0.2Å, and 4.6 ± 0.3Å; average values for the β core over the same time period were 2.2 ± 0.0Å, 2.3 ± 0.1Å, and 2.0 ± 0.2Å. Cα RMSD analysis of the I195T simulations showed consistent disruptions in L2 and the S7/S8 loop with

the S7/S8 loop showing values as much as 6Å larger than wild type. Two simulations showed deviations from starting structure in the loop-sheet-helix region as much as 5Å greater than wild type, and two simulations showed L3 disruptions ~2Å less than those in wild type.

Secondary-structure analysis showed that the S1 β structure became fragmented in all three simulations while the remaining secondary structures were mostly well-maintained. One simulation showed a gain of α-sheet structure between S1 and S3 and one simulation showed a gain of β-sheet structure between S10 and the N-terminal loop immediately adjacent to S1. Cα RMSF analysis showed a ~2Å increase above wild-type limits in the loop-sheet-helix region, particularly in L1, and a 1Å decrease in L2.

R196 showed increased contact with both Y205 (S6) and V216 (S7). Increased contact occupancy was also observed between L1 and S2', between the C-terminus of L2 and S6, and between S5 and both S6 and S7. All three simulations demonstrated separation between L1 and H2 with distances ranging between 10Å and 20Å and all three simulations demonstrated a 10Å-15Å separation between L2 and L3. In one simulation, L2 separated from S5 by 20Å.

I232T

The I232T mutation is located in the S8 strand. Previous work (Bullock et al. 2000) showed that this mutation destabilizes the p53 protein by 3.2 kcal/mol at 10° C relative to wild type and decreases DNA binding affinity by 39% at 20° C. Average Cα RMSD values of the simulations were 3.6 ± 0.2Å, 5.2 ± 0.2Å, and 4.0 ± 0.3Å; average β -core values over the same time period were 1.6 ± 0.1Å, 3.6 ± 0.1Å, and 1.9 ± 0.1Å. Cα RMSD analysis showed that L1, L2, L3, and

the S7/S8 loop were regions that deviated most from starting structure. The S6/S7 and S3/S4 loops also showed mild displacement from starting structure across all three simulations. Two simulations exhibited stable and wild type-like β-core displacements (<2Å) while one simulation showed a slightly larger β-core displacement (~4Å), with the majority of the displacement occurring in the smaller S1/S3/S8/S5 β sheet. Cα RMSD values 5Å larger than wild type were seen in L1 and the N-terminal half of L2; sub-wild-type values (<2Å) were observed in the region around S5, S6, and S7.

The largest departure from wild-type-like Cα RMSF values was in L1 where one simulation exceeded wild-type ranges by 4Å. Two simulations exhibited L1/H2 separation by 14Å, and L2 separation from L3 and S5 were both wild-type like. Secondary structure analysis indicated that most secondary structures remained intact. One simulation showed a loss of β structure across the entire S1/S3/S8/S5 β sheet as well as a gain of helical structure in L1. One simulation also adopted a four-residue α sheet between the N-terminal region of S1 and the C-terminal region of S3. Gains in contact occupancy were observed between F212 (S6/S7 loop) and M169 (L2) and between R175 (L2) and M237 (L3). There was also a loss of contact occupancy between the S5/S6 loop and the S8 strand.

L145Q

The L145Q mutation is located in the S3 β strand. Previous work has shown that it destabilizes the p53 protein by 3 kcal/mol at 10° C relative to wild type and decreases DNA binding affinity by 53% at 20° C (Bullock et al. 2000). Average Cα RMSD values of the simulations were 4.5 ±

0.1Å, 5.4 ± 0.3Å, and 5.5 ± 0.2Å; the β core values over the same time period were 1.8 ± 0.1Å, 2.3 ± 0.0Å, and 2.8 ± 0.1Å. Cα RMSD analysis of the three L145Q simulations showed that all simulations had structural disruptions in L2, L3 and the S7/S8 loop, and two simulations had consistent structural deviations in the loop-sheet-helix region. Cα RMSD values in the loop-sheet-helix region and L2 region were between 5Å-10Å greater than wild type. Conversely, Cα RMSD values in the S7/S8 loop region were 5Å-7Å less than wild type. Most Cα RMSF values were within wild-type ranges although H1 values were approximately 1Å less than wild type and one simulation exceeded wild-type values in the loop-sheet-helix region by approximately 2Å. Secondary structure analysis indicated that while the larger β sheet remained mostly intact, the smaller S1/S3/S8/S5 β sheet was disrupted across all simulations. One simulation gained α-sheet conformation between S1 and S3 but was otherwise intact. One simulation lost S1 and considerable S3 structure but gained β structure in the adjacent N-terminal loop (residues 101 – 104); these residues formed a small β sheet with S10. The remaining simulation lost structure across all four strands of the S1/S3/S8/S5 sheet and gained helical structure in residues 113-116, immediately C-terminal to S1. Two simulations exhibited 20Å-25Å separations between L1 and H2 in the loop-sheet-helix region. One simulation exhibited a 20Å separation between L2 and L3. There was a general gain of contact between the S3/S4 region and the S7/S8 region, between S5 and S6, and between L3 and both S9 and H2.

P151S

The P151S mutation is located in the middle of the S3/S4 loop. Previous work indicates that this mutation decreases DNA binding affinity by 57% at 20° C and destabilizes the p53 protein by 4.5 kcal/mol at 10° (Bullock et al. 2000). Average Cα RMSD values of the simulations were 4.5 ± 0.2Å, 4.1 ± 0.2Å, and 4.1 ± 0.1Å; average β-core values over the same time frame were 2.9 ± 0.1Å, 1.7 ± 0.1Å, and 1.9 ± 0.1Å. Cα RMSD analysis showed that most loop regions were disrupted to some degree with the largest disruptions occurring in the S3/S4 loop, L2, L3, and the S7/S8 loop. One simulation had large structural deviations in the loop-sheet-helix region. Most Cα RMSD values were within 1Å of wild-type ranges. Cα RMSF values were similarly wild type-like with the exception of one simulation with L1 values 3Å larger than wild type. Secondary structure analysis indicated that while most secondary structures were maintained, S1 lost structure in two simulations, S5 lost structure in one simulation, and α-sheet structure between S1 and S3 was adopted in two simulations. In one simulation, L1 separated from H2 by as much as 20Å and in two simulations, the C-terminus of L2 separated from S5 by 15Å-25Å. Across all three simulations, contact occupancy increased between S5 and S7 and between L3 and S9.

V143A

The V143A mutation lies in the S3 β strand. Previous work(Bullock et al. 2000) has shown that it destabilizes the p53 protein by 3.5 kcal/mol at 10° C and reduces its DNA binding affinity by 32% at 20° C relative to wild type. Average Cα RMSD values of the simulation were 4.0 ± 0.1Å, 4.9 ± 0.2Å, and 4.6 ± 0.1Å; β-core values over the same time period were 1.4 ± 0.1Å, 2.0

± 0.1Å, and 2.0 ± 0.1Å. Cα RMSD analysis showed that the primary structural disruptions occurred in the C-terminal region of L2, the S7/S8 loop, and L3. Smaller disruptions were seen in S1, L1, S3 and H2. Cα RMSD values exceeded wild-type values by 7Å in the C-terminus of H1and one simulation had values more than 5Å less than wild type in the S7/S8 loop. The largest departures from wild-type Cα RMSF values were 1Å-2Å increases in the C-terminal half of L2 and in the S7/S8 loop.

Most secondary structures were well maintained throughout the simulations with the exception of one simulation that lost structure in both S1 and H1 and two simulations that gained α-sheet structure, one between S1 and S3, and one between S3 and S8. One simulation showed a 20Å separation between L1 and H2. All three simulations demonstrated L2/S5 separation ranging from 15Å to 25Å. R175 lost contacts with Q192 and H193 in L2 and gained contacts with M237 and N239 in L3. Contacts between L1 and the S10/H2 loop were completely lost.

V157F

The V157F mutation lies in the S4 β strand. Previous work has determined that this mutation destabilizes the p53 protein by 3.9 kcal/mol at 10° C and reduces its DNA binding affinity by 28% at 20° C relative to wild type (Bullock et al. 2000). Average Cα RMSD values from the last 25 ns of the simulations were 5.1 ± 0.1Å, 4.7 ± 0.1Å, and 4.5 ± 0.1Å; average values for the β-core during the same time period were 2.9 ± 0.1Å, 2.1 ± 0.1Å, and 2.3 ± 0.1Å. Cα RMSD analysis showed the largest displacements in the S7/S8 loop followed by more wild type-like values in L2

and the loop-sheet-helix region. Most of these were within wild-type limits. Cα RMSF values were also generally within wild-type limits.

The V157F simulations lost H1 helical structure in two simulations and S1 β structure in two simulations. One simulation showed a gain of α-sheet content between S1 and S3 and one simulation showed a 20Å separation between L1 and H2. One simulation showed an L2/S5 separation of 20Å and two simulations showed a ~20Å separation of the S3/S4 and S7/S8 loops, measured between the Cα atoms of D148 and D228. Contact gains were seen between H115 and C124, R175 and M237, and R196 and Y205. Contact losses were seen between L114 and both T125 and Y126.

Y220C

The Y220C mutation lies at the N-terminal end of the S7/S8 loop. Previous work has shown that this mutation destabilizes the p53 protein by 4 kcal/mol at 10° C and reduces its DNA binding affinity by 55% at 20° C relative to wild type (Bullock et al. 2000). Average Cα RMSD values from the last 25 ns of the simulations were 4.0 ± 0.2Å, 4.2 ± 0.2Å, and 4.5 ± 0.3Å; the β-core values from that same time period were 1.5 ± 0.1Å, 2.5 ± 0.1Å, and 2.8 ± 0.2Å. The three Y220C simulations were varied in their deviations from the starting structure. One simulation was primarily disrupted in the L1 region with some wild type-like disruptions in the S7/S8 loop. The other two simulations had more widespread disruptions across the loop-sheet-helix region, the S3/S5 loop, L2, L3, and the S7/S8 loop. Cα RMSD values exceeded wild-type values by as

much as 5Å in the loop-sheet-helix region, the S3/S4 loop, and L2. Cα RMSF values were within 1Å of wild type with slightly increased (<2Å) values in the S7/S8 loop and H2.

Secondary structures were well-maintained throughout the simulations with the exception of one simulation that lost β structure in S1 and gained β structure in the adjacent N-terminal loop, one simulation that gained helical structure in L1, and one simulation that gained α-sheet content between S1 and S3. One simulation exhibited a 25Å separation between L1and H2 and one simulation demonstrated a 25Å separation of L2 and S5. Contact gains were seen between S4 and the S6/S7 loop, between L2 and the S6/S7 loop, and between H1 and the C-terminal half of L2.

Y220H

The Y220H mutation lies at the N-terminal end of the S7/S8 loop. Average Cα RMSD values of the simulations were 4.6 ± 0.2Å, 3.7 ± 0.3Å, and 4.0 ± 0.1Å. Cα RMSD analysis showed that most of the deviation from the starting structure occurred in the loop-sheet-helix region, the S7/S8 loop, L2 and L3, although the degree of deviation varied among the three simulations. Cα RMSD values in the loop-sheet-helix and L2 regions were as much as 5Å larger than wild type; one simulation demonstrated sub-wild-type values in the S7/S8 loop. Cα RMSF values were ~1Å larger than wild type in the loop-sheet-helix, L2, and S7/S8 loop regions.

One simulation maintained all secondary structures, one simulation lost most of S1 β structure and gained some α-sheet structure between S1 and S3, and one simulation lost structure in both

S1and S5. Two simulations demonstrated separations of L1 and H2 between 7Å-15Å. An increase in contact occupancy was observed between S5 and S6.

Tables and Figures

Residue	**Residue**
171 (L2)	249 (L3)
114 (L1)	126 (S2)
163 (S4)	172 (L2)
108 (NT/S1)	147 (S3/S4)
104 (NT/S1)	109 (NT/S1)
104 (NT/S1)	107 (NT/S1)
254 (S9)	268 (S10)
117 (L1)	122 (L1)
165 (L2)	168 (L2)
104 (NT/S1)	108 (NT/S1)
253 (S9)	268 (S10)
104 (NT/S1)	268 (S10)
110 (S1)	145 (S3)
163 (S4)	171 (L2)
207 (S6)	213 (S6/S7)
118 (L1)	282 (H2)
158 (S4)	206 (S6)
173 (L2)	251 (S9)
250 (L3)	273 (S10)
115 (L1)	144 (S3)
115 (L1)	128 (S2/S2')
160 (S4)	193 (L2)
106 (NT/S1)	149 (S3/S4)
114 (L1)	142 (S3)
100 (NT/S1)	252 (S9)
133 (S2p)	143 (S3)
104 (NT/S1)	266 (S10)
269 (S10)	271 (S10)
100 (NT/S1)	269 (S10)
111 (S1)	133 (S2p)
108 (NT/S1)	148 (S3/S4)
165 (L2)	169 (L2)
200 (S5/S6)	231 (S8)
165 (L2)	249 (L3)
111 (S1)	126 (S2)
161 (S4)	254 (S9)
107 (NT/S1)	148 (S3/S4)
205 (S6)	217 (S7)
118 (L1)	279 (H2)
159 (S4)	193 (L2)

Appendix Table D.1 Contacts lost in the pseudo-wild type relative to the wild type

Residue	Residue
163 (S4)	166 (L2)
285 (H2)	289 (H2/CT)
132 (S2p)	272 (S10)
197 (S5)	218 (S7)
221 (S7/S8)	229 (S7/S8)
236 (S8)	270 (S10)
140 (S2/S3)	198 (S5/S6)
198 (S5/S6)	233 (S8)
123 (L1)	141 (S3)
114 (L1)	123 (L1)
286 (H2)	289 (H2/CT)
174 (L2)	249 (L3)
165 (L2)	250 (L3)
168 (L2)	249 (L3)
240 (L3)	250 (L3)
194 (L2)	239 (L3)
238 (L3)	272 (S10)
236 (S8)	274 (S10)
122 (L1)	279 (H2)
193 (L2)	205 (S6)
240 (L3)	249 (L3)
120 (L1)	278 (H2)
133 (S2p)	236 (S8)
172 (L2)	209 (S6/S7)
238 (L3)	251 (S9)
175 (L2)	180 (H1)
120 (L1)	280 (H2)
200 (S5/S6)	218 (S7)
120 (L1)	277 (S10/H2)
154 (S3/S4)	219 (S7)
255 (S9)	269 (S10)
163 (S4)	168 (L2)
193 (L2)	206 (S6)
239 (L3)	274 (S10)
195 (S5)	205 (S6)
197 (S5)	205 (S6)
193 (L2)	207 (S6)
243 (L3)	248 (L3)
196 (S5)	205 (S6)
238 (L3)	274 (S10)
243 (L3)	247 (L3)

Appendix Table D.2 Contacts gained in the pseudo-wild type relative to the wild type

Residue†	Residue
114 (L1)	126 (S2)
125 (S2)	282 (H2)
127 (S2)	282 (H2)
130 (S2/S2')	285 (H2)
273 (S10)	285 (H2)
248 (L3)	285 (H2)
130 (S2/S2')	286 (H2)
122 (L1)	278 (H2)
123 (L1)	141 (S3)
114 (L1)	125 (S2)
117 (L1)	122 (L1)
126 (S2)	131 (S2/S2')
125 (S2)	278 (H2)
118 (L1)	283 (H2)
116 (L1)	125 (S2)
117 (L1)	125 (S2)
115 (L1)	125 (S2)
121 (L1)	278 (H2)
115 (L1)	282 (H2)
127 (S2)	286 (H2)
117 (L1)	282 (H2)
134 (S2p)	285 (H2)
111 (S1)	126 (S2)
133 (S2p)	143 (S3)
135 (S2p)	278 (H2)
119 (L1)	279 (H2)
118 (L1)	279 (H2)
114 (L1)	143 (S3)
114 (L1)	144 (S3)
120 (L1)	279 (H2)
116 (L1)	122 (L1)
114 (L1)	124 (S2)
123 (L1)	142 (S3)
115 (L1)	126 (S2)
123 (L1)	278 (H2)
117 (L1)	279 (H2)
121 (L1)	136 (S2'/S3)
118 (L1)	282 (H2)
111 (S1)	114 (L1)
119 (L1)	280 (H2)
126 (S2)	282 (H2)
115 (L1)	128 (S2/S2')
111 (S1)	133 (S2p)
128 (S2/S2')	282 (H2)
121 (L1)	277 (S10/H2)
117 (L1)	120 (L1)
113 (L1)	*143 (S3)*

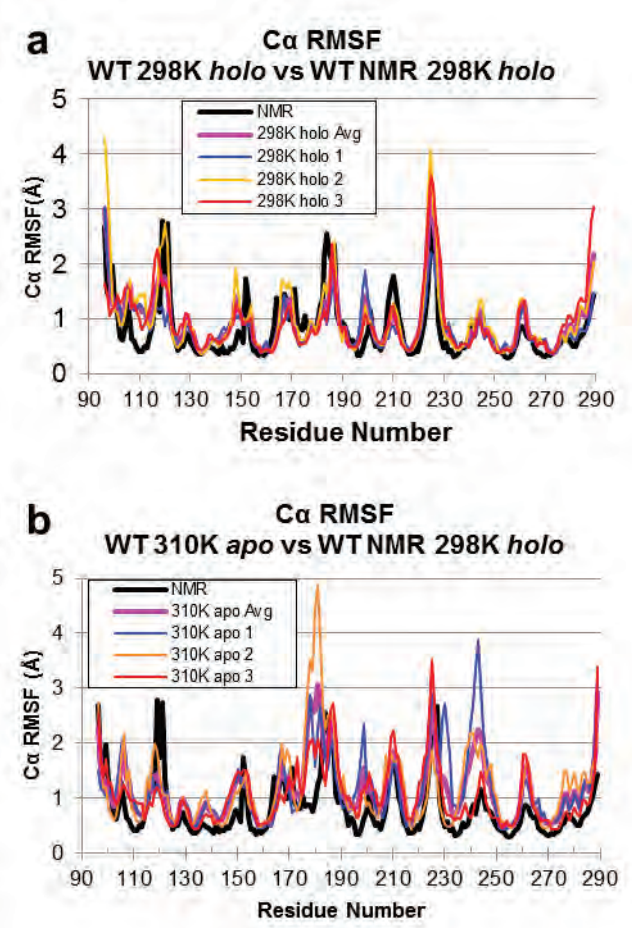
Appendix Table D.3 Contacts common to L1/H2 separation conformations

†Italic indicates stabilized relative to wild type, non-italic indicates destabilized relative to wild type. There was only one contact that was commonly stabilized relative to wild type.

Residue†	Residue
163 (S4)	172 (L2)
165 (L2)	169 (L2)
165 (L2)	170 (L2)
173 (L2)	193 (L2)
175 (L2)	180 (H1)
175 (L2)	192 (L2)
176 (L2)	179 (H1)
176 (L2)	180 (H1)
177 (H1)	180 (H1)
165 (L2)	171 (L2)
190 (L2)	193 (L2)
161 (S4)	173 (L2)
175 (L2)	184 (L2)
171 (L2)	249 (L3)
177 (H1)	181 (L2)
175 (L2)	191 (L2)
174 (L2)	192 (L2)
184 (L2)	196 (S5)
162 (S4)	172 (L2)
173 (L2)	192 (L2)
191 (L2)	205 (S6)
175 (L2)	194 (L2)
175 (L2)	193 (L2)
165 (L2)	168 (L2)
179 (H1)	184 (L2)
190 (L2)	196 (S5)
175 (L2)	185 (L2)
175 (L2)	190 (L2)
173 (L2)	194 (L2)
175 (L2)	189 (L2)
97 (NT/S1)	169 (L2)
162 (S4)	171 (L2)
163 (S4)	174 (L2)
172 (L2)	214 (S7)
163 (S4)	194 (L2)
190 (L2)	205 (S6)
180 (H1)	184 (L2)
175 (L2)	196 (S5)
180 (H1)	185 (L2)
160 (S4)	172 (L2)
174 (L2)	194 (L2)
173 (L2)	251 (S9)
162 (S4)	169 (L2)
172 (L2)	212 (S6/S7)
192 (L2)	214 (S7)
194 (L2)	246 (L3)
179 (H1)	239 (L3)
161 (S4)	172 (L2)
196 (S5)	*237 (L3)*

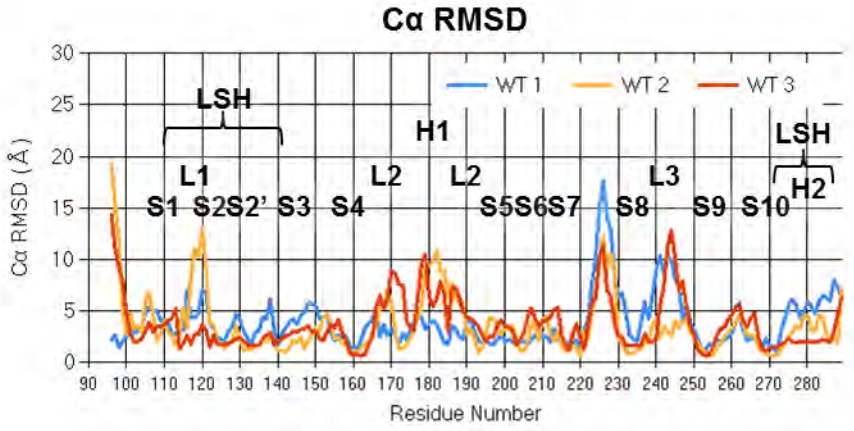
Appendix Table D.4 Contacts common to L2/S5 separation conformations

†Italic indicates stabilized relative to wild type, non-italic indicates destabilized relative to wild type. There was only one contact that was commonly stabilized relative to wild type.

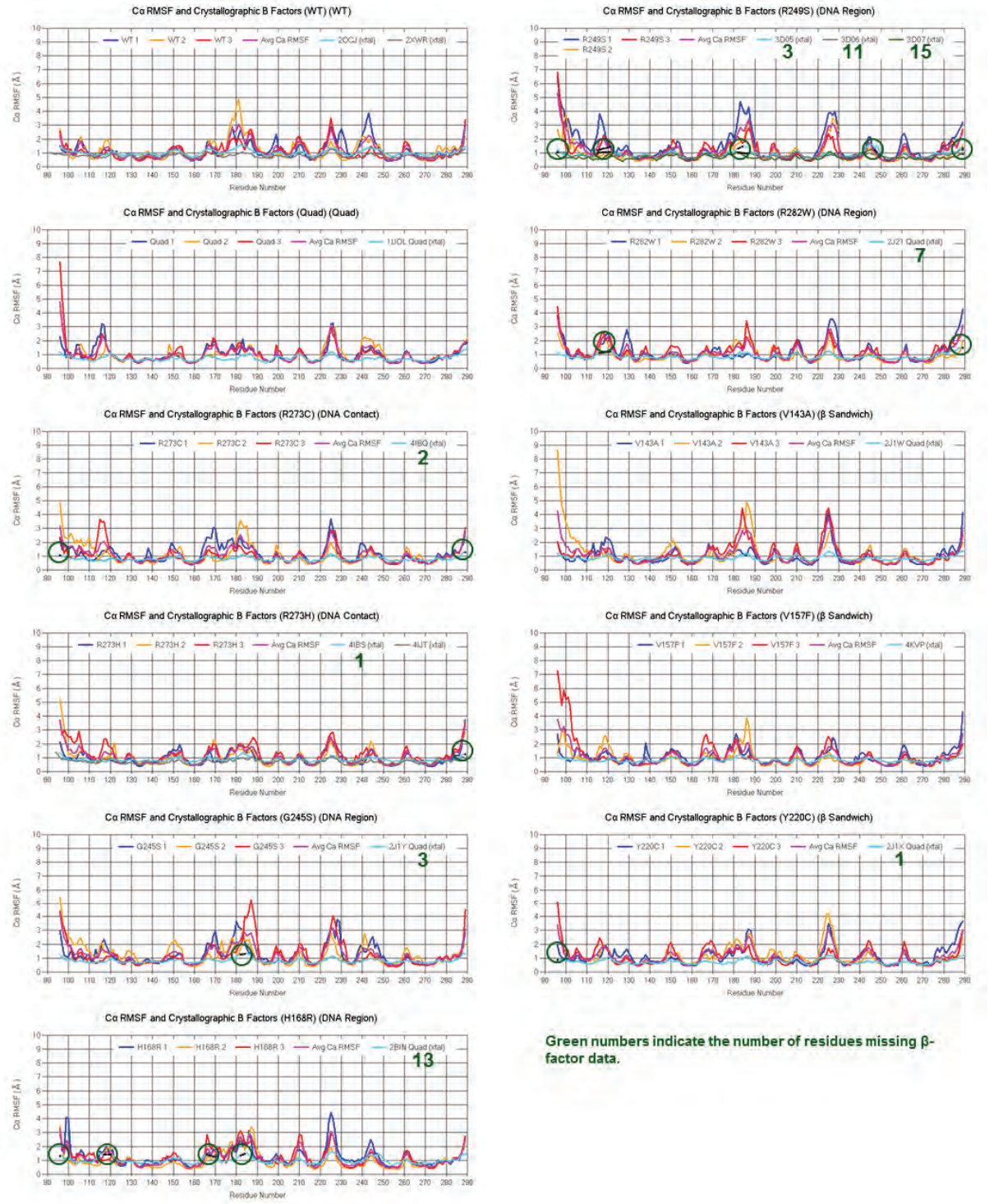


Appendix Figure D.1 Wild-type simulation Cα RMSF compared to experimental NMR Cα RMSF

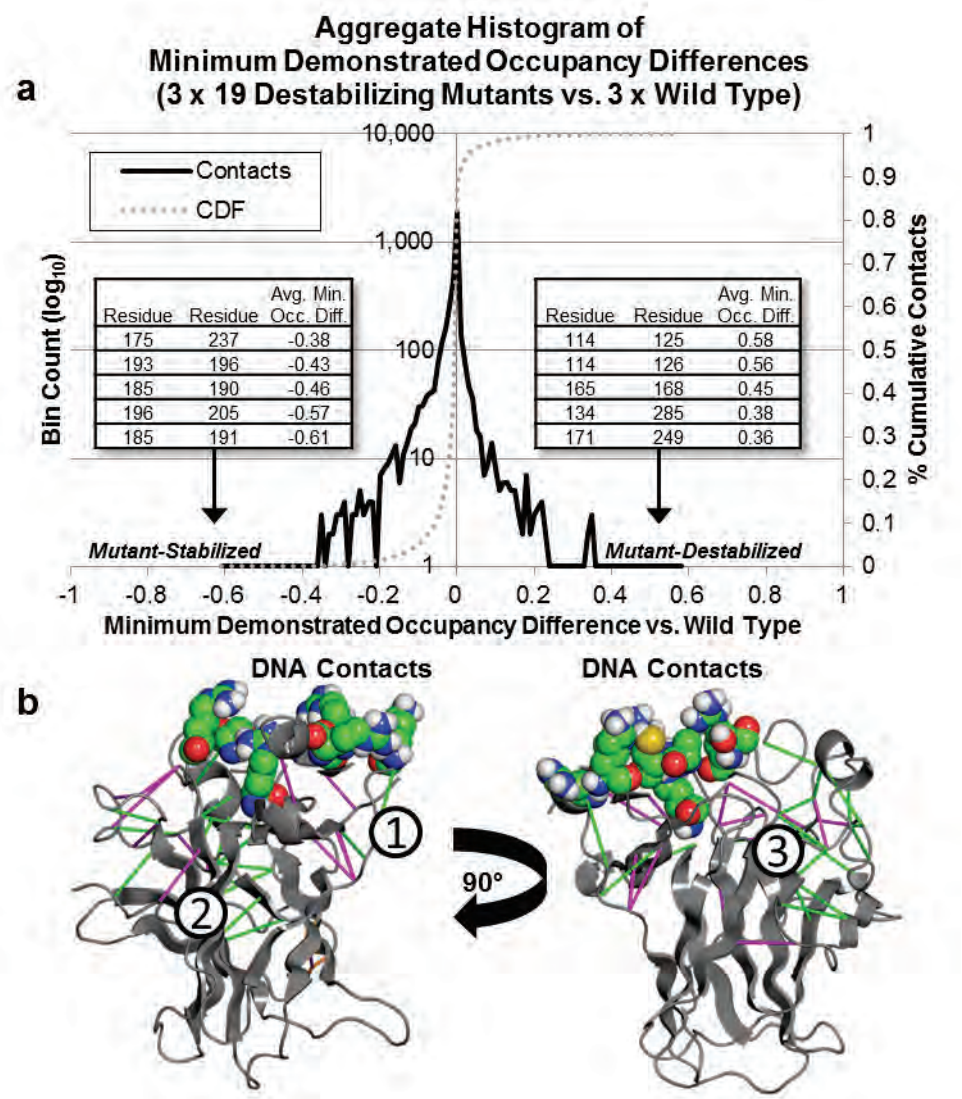
(a) 298K *holo* MD vs. 298K *holo* NMR (b) 310K *apo* MD vs. 298K *holo* NMR



Appendix Figure D.2 Wild-type average Cα RMSD values.
The loop-sheet-helix (LSH) region is labeled.



Appendix Figure D.3 Simulation Cα RMSF values vs. available B-factor data



Appendix Figure D.4 Aggregate contact occupancy-change for destabilizing mutants
(a) Histogram (left/center axis, $\log_{10}$ scale) and cumulative distribution function (CDF) (right axis) for the minimum demonstrated contact occupancy changes relative to wild type for all 19 destabilizing mutants. The contacts with the five lowest (stabilized) average contact changes and five highest (destabilized) contact changes are shown inset with arrows indicating their location in the distribution. (b) p53 starting structure, DNA-contact residues are rendered as spheres for orientation. Mutant-stabilized (green) and mutant-destabilized (magenta) contacts are shown. Shown are 31 contacts whose occupancy change was greater than 0.3 or less than -0.3 in the histogram in (a). Three separable regions were identified: (1) the loop-sheet-helix region (8 contacts), (2) NT/S9 (1 contact), (3) the L2/S8/S5/S6/S7 region (22 contacts).

Appendix E

ADDITIONAL BIOINFORMATICS TOOLS

Several software tools were developed over the course of my thesis work. Some, like DIVE (Chapters 2 and 3), were large-scale, general-purpose software endeavors. Others were smaller-scale tools that directly solved an immediate problem. In this appendix, I describe several of these smaller-scale tools, many of which are tools built using the DIVE application programming interface (API) while others are actually DIVE pipelines that required no additional programming.

E.1 Interactive Contact Maps

Inter-residue contact analysis, as discussed in Chapters 4, 5, and 6, is an effective method for quantifying and analyzing protein structure. In many instances, only contact information from a single point in time is needed; in these cases, non-interactive analysis of contact data may be sufficient. However, in some cases, such as protein unfolding, it is desirable to watch the inter-residue contacts change over time. For these scenarios, I developed an interactive contact map tool capable of displaying per-residue contacts at arbitrary time resolutions. The data points – one per contact – can be colored by an arbitrary property and can be queried for specific details. A screen shot of the interactive contact map tool is shown in Appendix Figure E.1.

Initial contact data is loaded from a comma-separated-value (CSV) file on disk. Data is loaded from disk rather than the Dynameomics data warehouse because 1) contact calculations cannot be done at interactive speeds and are thus calculated offline and 2) much of the work in

our lab is done using flat files and avoiding a dependency on the Dynameomics data warehouse increased access to the tool.

The contact data, including arbitrary property data, is then loaded and pre-processed for visualization. One or more sets of contact can be loaded at once and these data can be displayed separately or together. The data are displayed in the same format as a standard contact map with the data point (x,y) coordinate set to the (residue number, residue number) of the contacting residues. The data points are then colored by the property included in the input file. Property coloring is calculated by scaling the property values to the minimum and maximum property values read in from file. These 0-100% values are then mapped from blue → red. Because viewing the complete color spectrum can result in a noisy visual field, the range of the color spectrum can be interactively expanded or contracted to better highlight contacts of interest.

E.2 High-Resolution Property Plots

Time-series properties such as Cα RMSD or SASA are useful, but they are often viewed as whole-protein aggregates which can hide many of the important details. To address this, I used DIVE to build per-residue, per-picosecond plots of different properties such as Cα RMSD and SASA. An example of a Cα RMSD plot is shown in Appendix Figure E.2. These plots break down the property by residue over time with the individual data points colored by scaled property values (0-100% maps to blue → red). Property plots of aggregated subsets of residues are shown at the bottom. The horizontal axis is time, the left-side axis is residue number, and the right-side axis is property-dependent, in this case angstroms. The middle of the plot shows

various aggregated statistics for the residue subsets. Secondary-structure indicators can be placed behind the property data to aid in analysis.

E.3 Multi-Protein Plots

It is often useful to plot data from multiple proteins together. While this can be done using many different tools, I chose to use DIVE because of the built-in data-handling capabilities. An example of a multi-protein plot is shown in Appendix Figure E.3. This example shows per-residue average Cα RMSD data for all three runs of all 21 p53 proteins (Chapters 5 and 6) plotted next to each other. Also, this figure shows the demonstrated differences between the individual mutants and the wild type. To illustrate, the full wild type data is shown in the upper left. For each mutant, if the average Cα RMSD value for a residue is greater than the highest demonstrated wild-type value from any wild-type simulation, the difference (a positive value) between the mutant value and the highest wild-type value is plotted for that mutant residue. Similarly, if the average Cα RMSD value for a residue is lower than the lowest demonstrated wild-type value from any wild-type simulation, the difference (a negative value) between the mutant value and the lowest wild-type value is plotted for that mutant residue. Mutant values that fall within the demonstrated wild-type ranges are displayed as zero. The resulting plots then clearly illustrate where and by how much each simulation of each mutant protein has differed from the wild type. Once the software framework for generating plots was established, it became very easy to data-mine the simulations by generating multiple different kinds of plots. While some of these were useful and others were not, the speed with which novel charts and plots could be tried and tested resulted in many useful insights into the data.

E.4 Interactive Ramachandran Diagrams

Ramachandran diagrams are a way of viewing protein structure by essentially making a scatterplot of two angles - Phi (ϕ) and Psi (ψ) - that describe much of the protein's backbone structure. As the protein backbone changes conformation, these angles change as well. A Ramachandran diagram can be divided into specific regions that correlate to specific protein structures such as α helices and β sheets.

In our simulations, we calculate the ϕ and ψ angles and use them to analyze protein structure. To make this analysis more efficient, I used DIVE to build an interactive Ramachandran diagram. This diagram can plot per-picosecond ϕ/ψ angles at interactive speeds and color the data by an arbitrary property such as SASA (each data point represents an amino acid). Because it is interactive, it can also give feedback about individual data points and, if desired, send specific data points down the DIVE pipeline for further analysis.

Appendix Figure E.4 shows a screenshot of this DIVE tool. It is composed of three specific parts – a generic DIVE plotting plugin, a generic DIVE SQL plugin, and a DIVE interactive SQL plugin. The user first creates an interactive slider named 'ps' (picosecond). Movement of this slider sends a DIVE event. This event is caught by the SQL plugin which inserts the slider value into the SQL query (Appendix Figure E.5a, see 'Interactive SQL' in Chapter 3). This SQL query, which joins several SQL tables to integrate ϕ/ψ data with both SASA and structural data, is then executed and the resulting data points are sent down the DIVE pipeline to the plotting plugin. The plotting plugin pulls the data off of the sink pin "input", uses the DIVE.Bio helper library to calculate colors based on SASA, sets custom tooltip text to aid the user, and plots the data. This entire sequence executes at interactive speeds. Moreover, the

sequence is completely generic – the SQL code and plotting code can be replaced with any query for any SQL database.

E.5 Per-Residue Secondary-Structure Propensity

As discussed in the multi-protein plots above, it can be useful to see the entire set of simulated proteins at once. During the p53 analysis described in Chapter 6, I wanted to quantify the propensity of each residue in each simulation to adopt specific secondary structures such as α-sheet. I further wanted to understand if there were patterns to these propensities among the simulations. To do this, for each secondary-structure type, I calculated the percent time that each residue of each simulation spent in that DSSP-assigned secondary structure. I then output a comma-separated-value (CSV) file containing a grid of simulation X residue with each grid cell containing the calculated percentage. Visualization of these data provided multiple insights including identification of regular and correlated α-sheet propensities across almost all p53 proteins.

E.6 Secondary-Structure Contacts

While inter-residue contact analysis is useful, it is often more intuitive to think about protein structure in terms of secondary structures and secondary-structure contacts. To address this, I built a DIVE tool that analyzes per-residue contact information and outputs inter-secondary-structure contact-occupancy information in a comma-separated-value (CSV) file. The contact-occupancy information is the sum of all contact-occupancies between any two secondary structures. This information can then be loaded back into DIVE or into many other data tools

and plotting programs. To facilitate adoption in the lab, I designed an Excel spreadsheet that loads the data, calculates inter-secondary-structure contact-occupancy average ± stdev ranges, and indicates if mutant ranges overlap with wild-type ranges.

E.7 Interactive Dihedral-Angle Analysis

Similar to the interactive Ramachandran diagrams, interactive analysis of protein dihedral angles (the angles that describe the geometry of the protein residues, of which ϕ and ψ are two) was also useful. Again, this was a DIVE tool that required no additional programming. Like the Ramachandran diagram, this tool used interactive SQL and DIVE plotting. First, an interactive drop-down list allowed me to select a residue to analyze. This triggered a DIVE event which inserted the residue number into a SQL query. The SQL query was then executed and dihedral-angle data were retrieved from the Dynameomics data warehouse. These data were then binned into an aggregate histogram and presented to the user (Appendix Figure E.6).

E.8 NOE Analysis

Nuclear Overhauser Effect (NOE) analysis is used to validate that an ensemble of simulation structures has similar structural characteristics to experimental NMR structures. It does this by measuring the average distance between specific protons in a protein simulation and verifying that that average distance is less or equal to the average distance measured by NMR. If the average distance meets this criterion, the NOE is 'satisfied'. Otherwise, the NOE is 'violated'.

The NOE analysis described in Chapter 3 is performed by the *il*mm molecular dynamic software library (Beck et al. 2000-2014). Here I discuss several SQL tools that were designed to

analyze these data. The first tool simply loads the analysis data into a SQL database. The SQL data tables include basic information such as the average distance between protons and the distance cutoff required to satisfy the NOE. Additional structural data such as the residue number, residue type, secondary-structure type of the residue containing each proton, and Dynameomics simulation identifier are also included. A unique NOE identification number is also assigned to each NOE.

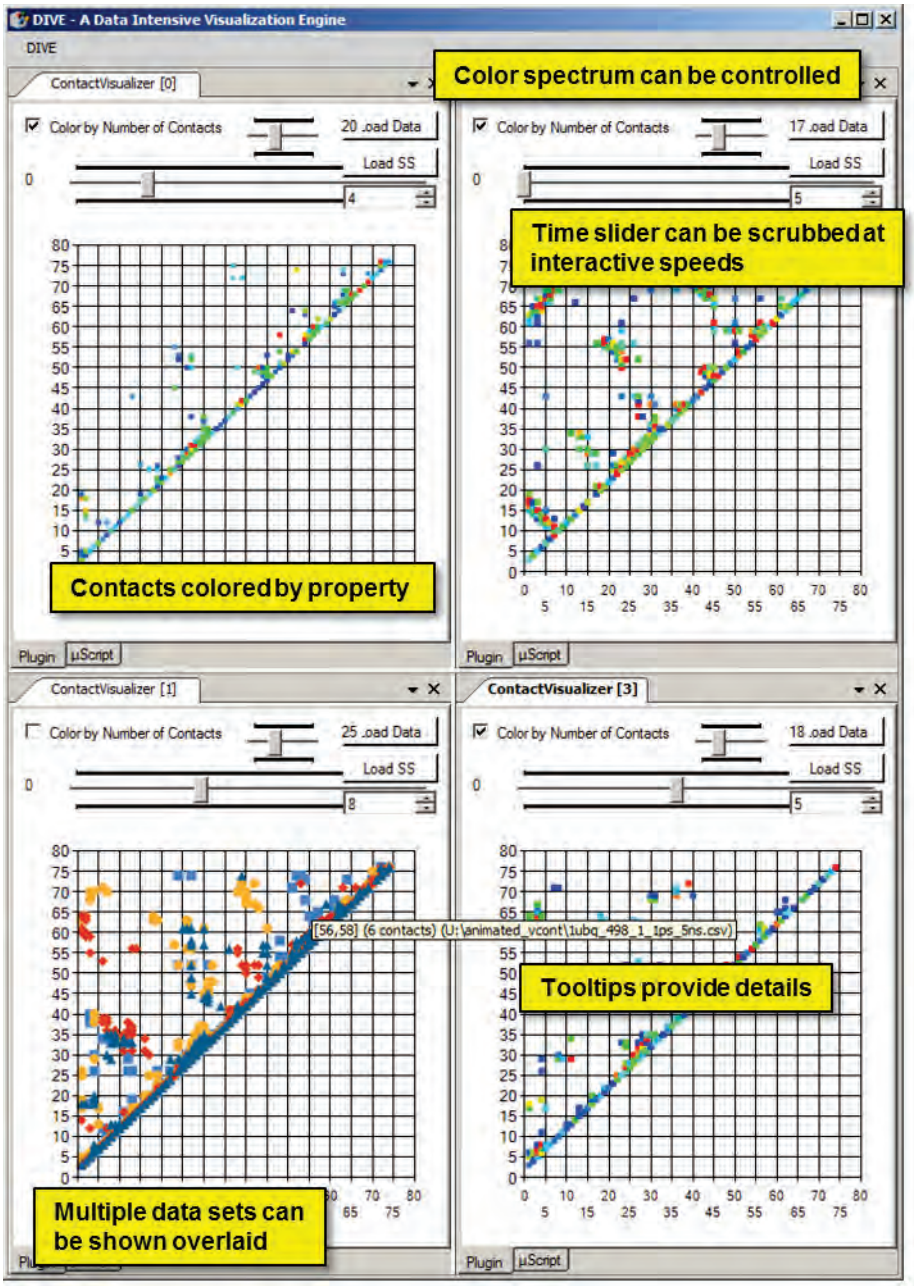
From this, a DIVE software tool queries the NOE tables and generates a summary analysis file including overall NOE satisfaction (the percentage of NOEs that were satisfied) as well as NOE satisfaction groups broken down by loop:non-loop and main-chain:side-chain classifications and binned by distance in physical space as well as distance in sequence space. This analysis can also include an NOE satisfaction baseline for comparison; for the p53 analysis, I used the simulation starting crystal structure as my NOE baseline.

From here, there are three SQL queries that are used to gain further insight into the NOE violations. The first is used to gain some estimation of the severity of an NOE. To some degree this is achieved by the summary analysis described above; for example, analysis of the NOE 298K *holo* NOE violations described in Chapters 5 and 6 showed that only 12% of the violations occurred between main-chain atoms, indicating that while the side-chains may have moved, the overall structure of the protein backbone was still very native-like. Here, the SQL query estimates violation severity by calculating the number of NOE violations whose containing residue:residue pair also contains an NOE satisfaction. Thus, as discussed in Chapter 6, if 45% of the residue pairs containing an NOE violation also contained an NOE satisfaction, we can

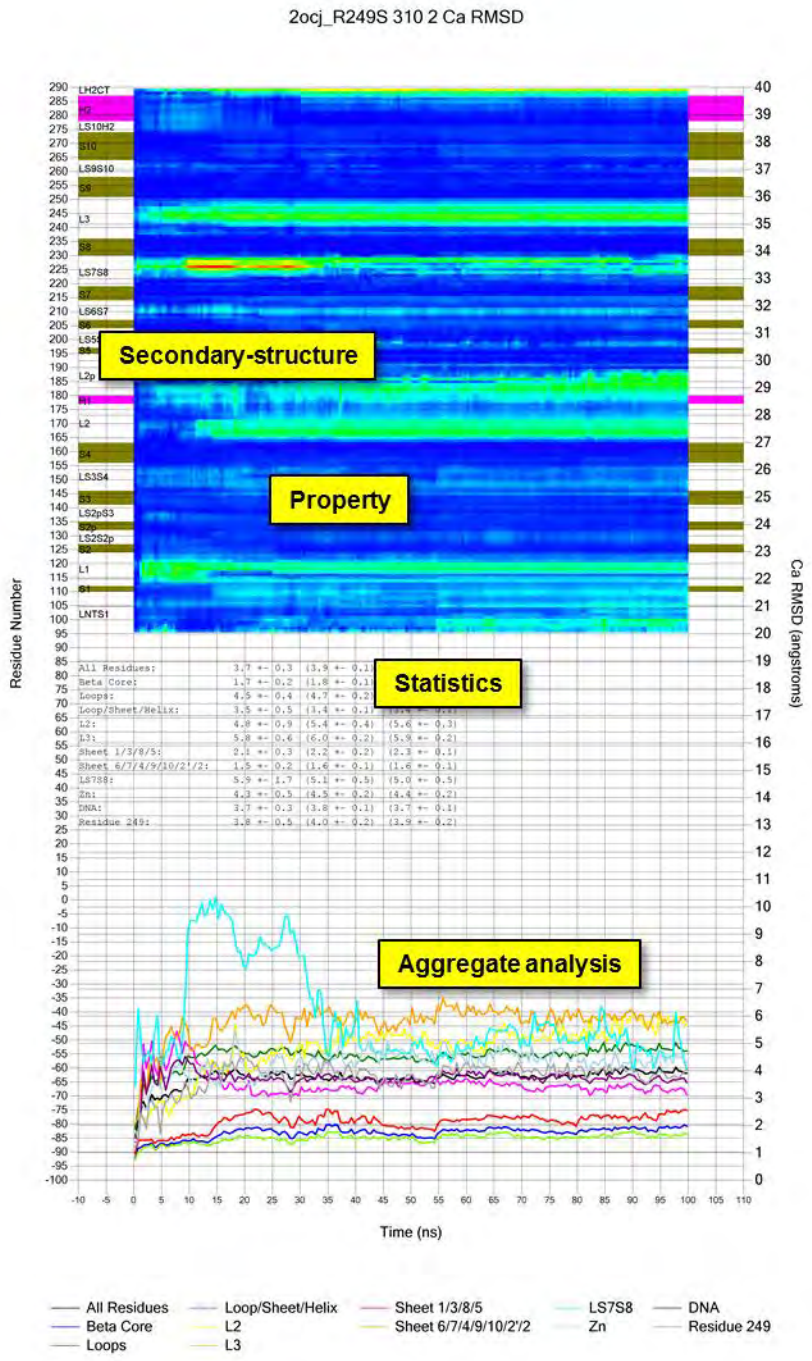
ascertain that almost half of the NOE violations were not associated with significant structural changes such as unfolding.

The next SQL query indicates which residue pairs are associated with the largest violations. For each residue:residue pair in the set of NOE violations, the query calculates 1) the average violation distance for all violations between those residues and 2) the average violation distance of all remaining violations, absent the residue pair in question. The results are then ordered by the latter. This let me understand very quickly which parts of the protein were causing the largest violations. It also isolated those residues associated with the largest violations.

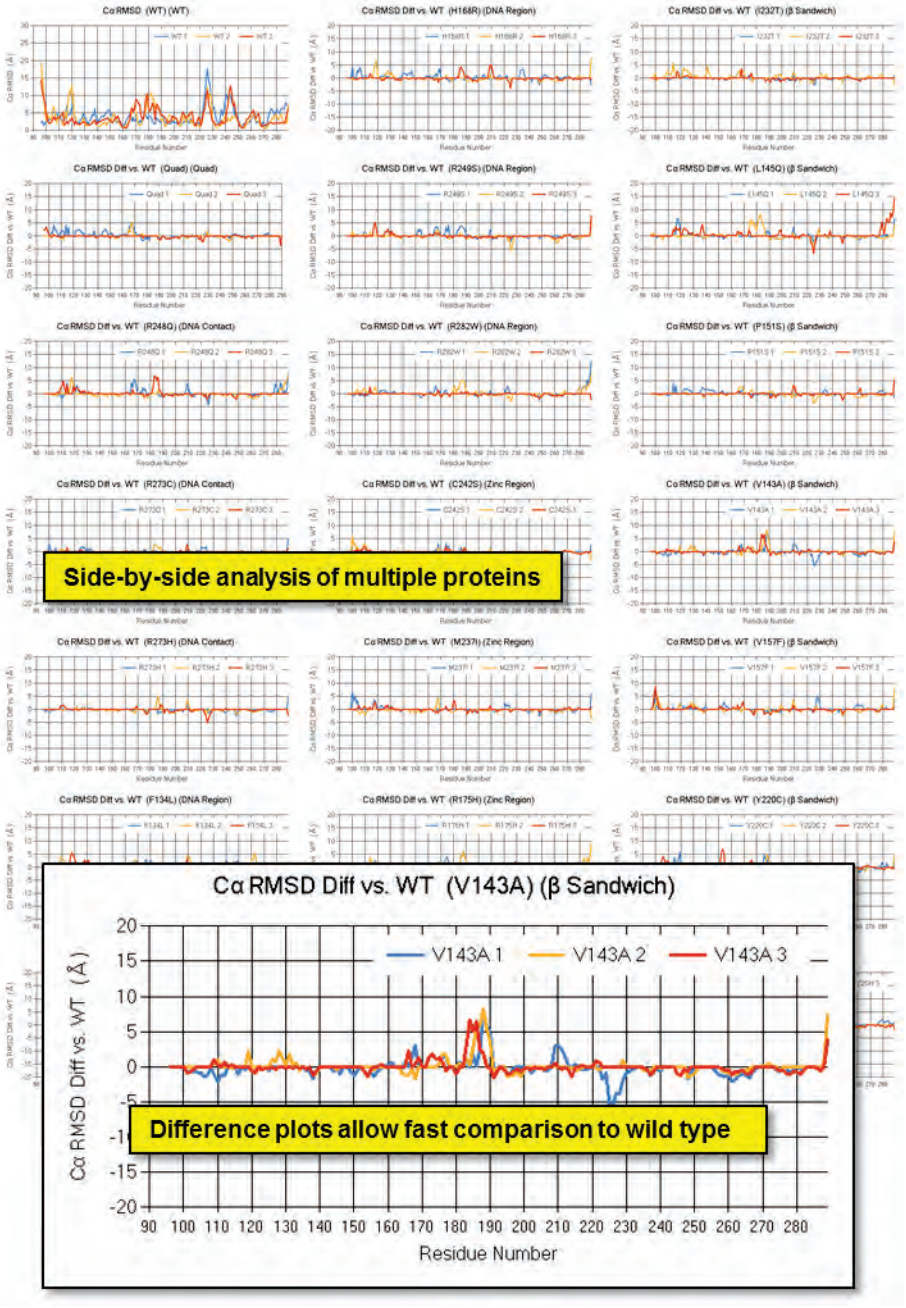
Once the residues associated with the largest violations have been identified, a similar SQL query calculates 1) the average violation distance of NOE violations associated with those residues and 2) the average violation distance of NOE violations not associated with those residues. Again, this helped me isolate the regions of the protein that were most in violation.



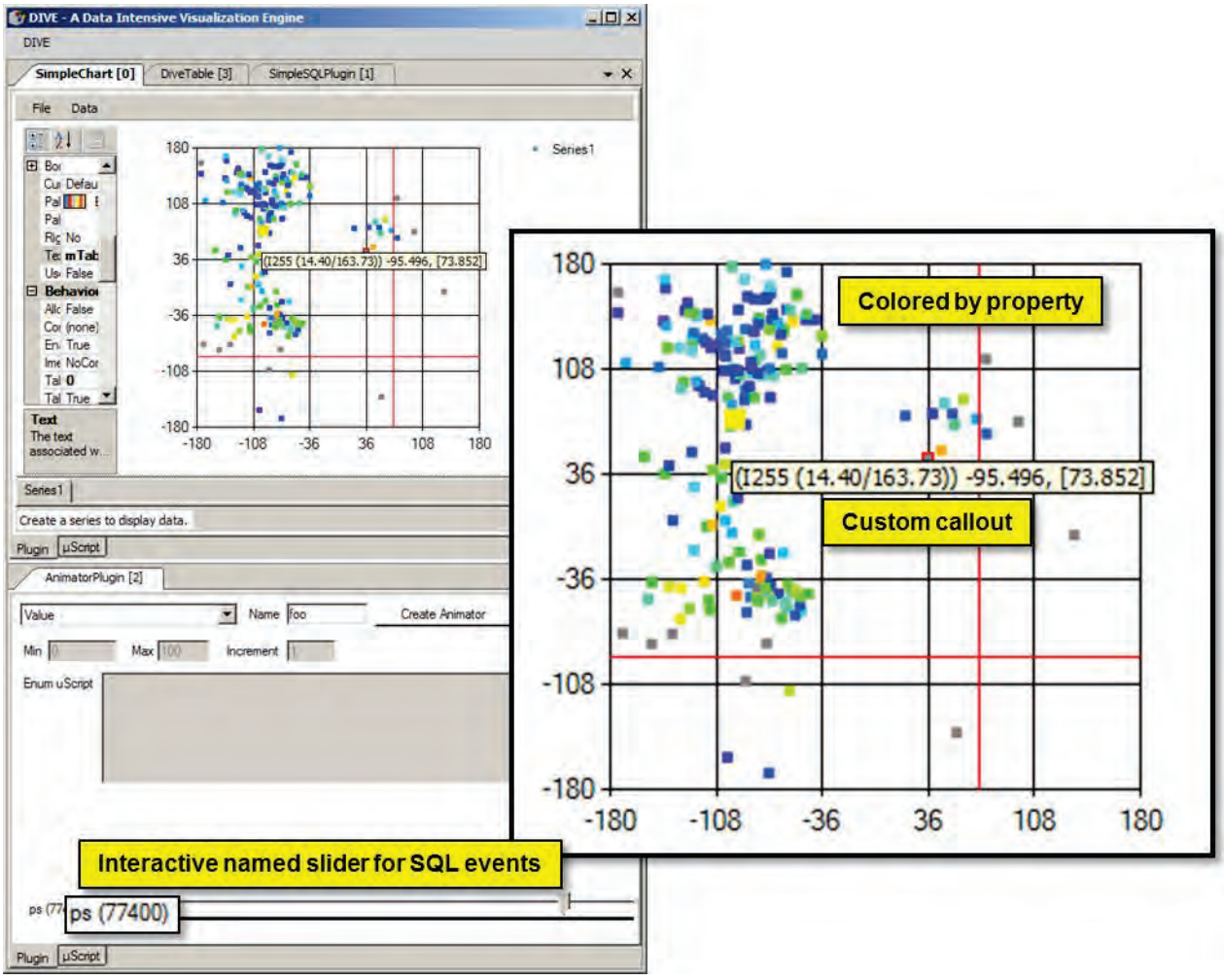
Appendix Figure E.1 Interactive contact maps



Appendix Figure E.2 High-resolution property plots



Appendix Figure E.3 Multi-Protein Plot



Appendix Figure E.4 Screenshot of DIVE interactive Ramachandran diagram

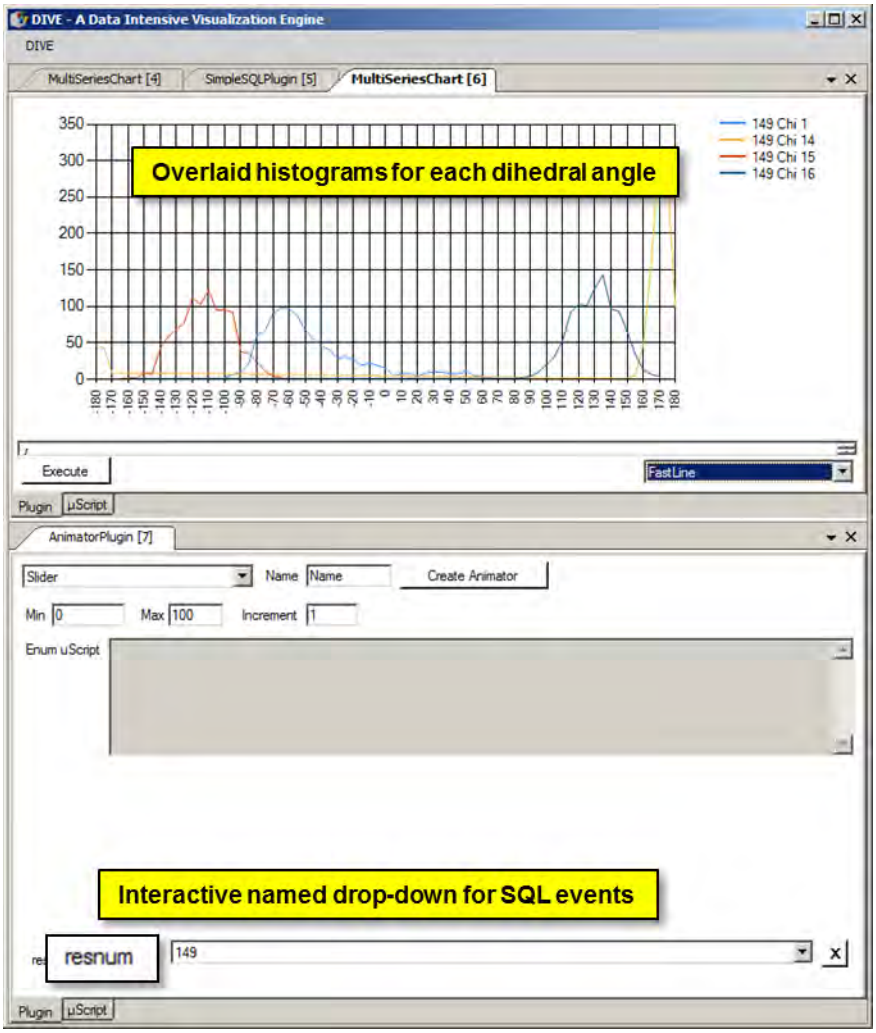
a

```
SELECT pp.residue_id,
       phi, psi,
       total, side_chain,
       id.residue

FROM PhiPsi_3667  as pp

JOIN SASA_3667    as sasa
ON( pp.step       = sasa.step
AND pp.residue_id = sasa.residue_id)

JOIN ID           as id
ON (pp.struct_id  = id.struct_id
AND pp.residue_id = id.residue_id
AND id.atom_name  = 'CA')

WHERE pp.step = {ps}*500
```

b

```
from dp in dps["input"]

let sasa = dp.side_chain

let max_sasa = BioHelper.GetSasa(dp.residue).side_chain

let sasa_pct = sasa/max_sasa

let r = string.Format("{0}{1} ({2:0.00}/{3:0.00})",
BioHelper.NormalizeResName(dp.residue),
dp.residue_id + 95,
sasa,
max_sasa)

select (new CD2(dp, dp.phi, dp.psi))
.Alter(x => x.mLabel = r)
.Alter(x => x.mColor = DIVEColor.BlueToRedDrawingColor(sasa_pct));
```

Appendix Figure E.5 User code for DIVE interactive Ramachandran diagram
(a) Interactive SQL code. Insertion point for interactive SQL is highlighted in yellow.
(b) LINQ μScript for plotting individual data points.



Appendix Figure E.6 Interactive dihedral angle analysis

VITA

Dennis Nathan Bromley received his Bachelor of Arts degree in Computer Science from Harvard University in 1996. He received his PhD in Biomedical and Health Informatics from the University of Washington in 2014.