Mining Mountains of Data: Organizing All Atom Molecular Dynamics Protein Simulation Data into SQL and OLAP Cubes

Andrew M. Simms

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2011

Program Authorized to Offer Degree:
Medical Education and Biomedical Informatics

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Andrew M. Simms

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of the Supervisory Committee:

Valerie Daggett

Reading Committee:

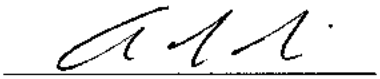
Valerie Daggett

Ira Kalet

Peter Myler

Date: 2-May-2011

In presenting this dissertation in partial fulfillment of the requirements for the doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of the dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to ProQuest Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, 1-800-521-0600, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature 

Date May 2, 2011

University of Washington

Abstract

Mining Mountains of Data: Organizing All Atom Molecular Dynamics Protein Simulation Data into SQL and OLAP Cubes

Andrew M. Simms

Chair of the Supervisory Committee:
Professor Valerie Daggett
Bioengineering

Across scientific disciplines, the ability to generate, collect, and store data has outpaced the ability to make sense of them. Methods and technology exist today for working with extremely large data sets, yet the most common data organization paradigm in science is to create files, in some cases millions of files, and store them in file systems. Despite the best intentions, these repositories quickly become disorganized, fragile, and difficult to manage; hindering mining and exploitation of the data they contain. This is fundamentally an informatics problem, and here I present the design of a very large scale repository to organize and mine molecular dynamics simulation data.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to give thanks to my wife, Merianne White, for all her encouragement and support that has made this journey possible.

Chapter 1: Introduction

Across scientific disciplines, the ability to generate, collect, and store data has outpaced the ability to make sense of them. Methods and technology exist today for working with extremely large data sets, yet the most common data organization paradigm in science is to create files, in some cases millions of files, and store them in file systems. Despite the best intentions, these repositories quickly become disorganized, fragile, and difficult to manage; hindering mining and exploitation of the data they contain. This is fundamentally an informatics problem, and the following chapters detail the design of a very large scale repository to organize and mine molecular dynamics simulation data.

The design of large-scale informatics infrastructure begins with a thorough analysis of the underlying data. The goal of this analysis is to discover and establish the interrelationships and boundaries of the data being captured or generated, as the data do not organize themselves. This process is by no means static, and will evolve as hypotheses are generated, tested, and refined. The urge to rush toward implementing persistence for internal data structures of specific algorithms must be avoided, as this will only result in needless data transformation and code refactoring as new algorithms are developed. Instead, data storage should be designed around conceptual structure of the data and intended paths of analysis. It should then be implemented using the primary objects of the chosen storage engine.

Dimensional modeling is an approach to database design that focuses analysis as a primary consideration. Originally pioneered as a method to organize large volumes of financial data, it is well suited to scientific data. Chapter 2 describes the dimensional model for protein simulation data and its implementation using a relational database.

On-line Analysis Processing (OLAP) is type of database developed specifically to address the needs of data analysis as opposed to managing transactions. In contrast to relational databases, OLAP databases fundamentally store and operate on multi-dimensional data. Chapter 3 explores OLAP and details the implementation of the Dynameomics data model using the

multi-dimensional OLAP feature of SQL Server Analysis Services.

Relational databases are general purpose tools and are largely agnostic to the semantics of the data they house. By coding inherent data features into relational primitives, huge performance gains are possible. Chapter 4 describes spatial indexing, where three-dimensional coordinates are placed into a 1-dimensional index and implemented as a simple foreign key, enabling rapid calculation of contact distances.

A single data model is unlikely to be effective for all potential applications. Large repositories can contain many interesting subsets of data, each with specific organizational semantics. Chapter 5 describes the Consensus Domain Dictionary, a unification of protein fold classification systems with a relational database linked to the primary data warehouse at its core.

As repositories grow, even the relatively small amount of information used by the tools that generate the data can grow to the point of being unmanageable. Molecular dynamics simulations must manage thousands of force field parameters that are associated with each atom in simulation. Chapter 6 describes the Molecular Mechanics Parameter markup Language (MMPL), an XML language for managing and sharing force-field parameters.

There are four fundamental problems in building large data repositories. These are: the design of the data model, the implementation and operation of the model, and the mining of the data, and sharing of the data. The previous chapters have described methods for the addressing the first three in the context of protein dynamics, and can clearly be applied to other scientific domains. The fourth problem, sharing, remains an issue even with a well-organized repository. Chapter 7 discusses future directions and ideas to solve sharing large volumes of scientific data.

Chapter 2: Protein Simulation Data in the Relational Model

High performance computing is leading to unprecedented volumes of data. Relational databases offer a robust and scalable model for storing and analyzing scientific data. However, these features do not come without a cost—significant design effort is required to build a functional and efficient repository. Modeling protein simulation data in a relational database presents several challenges: the data captured from individual simulations are large, multi-dimensional, and must integrate with both simulation software and external data sites. Here we present the dimensional design and relational implementation of a comprehensive data warehouse for storing and analyzing molecular dynamics simulations using SQL Server.

Introduction

Increasing processor power and access to supercomputer facilities have created an unprecedented amount of data in a variety of scientific disciplines. As the volume of data increases, the problem is no longer one of performing calculations utilizing high performance computing resources. Instead the challenge has become how to manage, organize, mine, and exploit the data. As such, this has become an informatics problem, one created by high performance computing. Such large datasets become intractable to efficiently manage and exploit on traditional file systems. However, they are well served, on many levels, by well-designed databases.

There are two schools of design for building systems with relational databases: relational modeling, which is used with transactional systems; and dimensional modeling, which is used in data warehousing applications. Both can be traced to E. F. Codd, who created the relational model (Codd, 1970) and proposed the on-line analytical processing (OLAP) model (Codd, 1993). Relational design is the organization of data into collections of sets known as relations. The process begins with a requirements analysis, which identifies all the attributes to be modeled and their functional dependencies. The Cartesian product of all attributes in the system, called the universal relation, can be conceptualized as a table where columns correspond

to attributes and the rows contain specific data items. Functional dependencies identify sets of attributes whose values are wholly determined by other attributes. The universal relation is broken up into smaller relations following a design pattern known as a loss-less join decomposition. The goal of decomposition is to significantly reduce or eliminate duplicate information. Although it is possible to automatically calculate decompositions that minimize duplicated data using functional dependencies, the process is typically driven by a designer. The designer will consider other constraints, such transactional and query performance of the application as well as the target database platform.

In contrast, dimensional modeling is driven almost entirely by both the innate structure of the data being modeled and reporting requirements. Dimensional modeling involves classifying data into two categories: facts and dimensions. Facts are continuous numerical quantities, dimensions are discrete classification values. Although space efficiency is important, it is not a central design goal. Instead, the primary goal of dimensional design is to yield a structure that is both easy and efficient to query. Dimensional models assume that data are primarily read-only, which allows liberal use of indexes to achieve query performance.

Dimensional models can be implemented in a relational database. Fact data are organ-

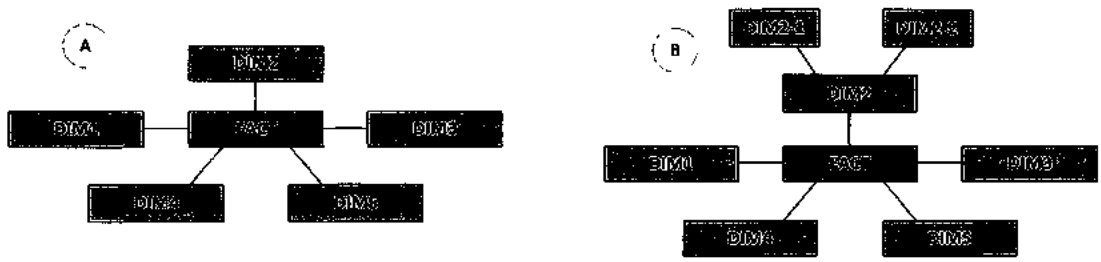


Figure 1. Star and Snowflake Schemas. A star schema (A) is distinguished by a central fact table and a set of dimensional tables surrounding it. Each dimensional row is associated with one or more fact table rows. A snowflake schema is a star schema with the addition of secondary dimensions (DIM2-1 and DIM2-2) that are related to a dimension and thus only indirectly related to the fact table.

ized into fact tables; dimensional data are placed in dimension tables that are linked via foreign key relationships. When visualized using UML or an ER diagram, fact tables appear as the center of a cluster of dimensions, forming a star shape. If dimensions relate to facts indirectly

through other dimensions, a snow-flake shape is observed. These are referred to as a star and snow-flake schemas, respectively.

Molecular dynamics (MD) simulation data can be described using a dimensional model. Fact data, at a high level, are sets of three-dimensional Cartesian coordinates for all simulated atoms. Secondary analyses are either related directly to atom coordinates, or aggregated at the residue, molecule, or simulation level. Dimensional data organize these facts by chemical structure, simulation time, and into groups of simulations and structures. The following sections detail the dimensional model, its translation to a relational model, and its implementation in SQL Server.

A Dimensional Model for MD Simulation Data

We have developed a four-dimensional model for representing MD simulation data. The primary dimensions: (A) simulations, (B) structures, (C) simulation groups, and (D) structure groups; are illustrated in Figure 2. The structure and simulation dimensions are organized hierarchically and are used to identify specific facts. The remaining dimensions are used to organize one or more simulations or structures into specific curated sets for analysis.

Structure and Structure Group Dimensions

The structure dimension provides the semantic context for interpreting and mining coordinate and analysis data from simulations. Attributes of this dimension are organized into a five level hierarchy as shown in Figure 2B, with structure type (Type) as the highest and atoms (Atom) as the lowest level. The structure dimension contains the attributes that describe structures being simulated and as well as links to the Protein Databank (PDB) for initial structures (Berman, 2000), the Chemical Component Dictionary for standard atom and residue names (Henrick, 2008), the Parameter Library, and Simulations as shown in Figure 3.

The Type level classifies structures (molecules) by creation method; current types are

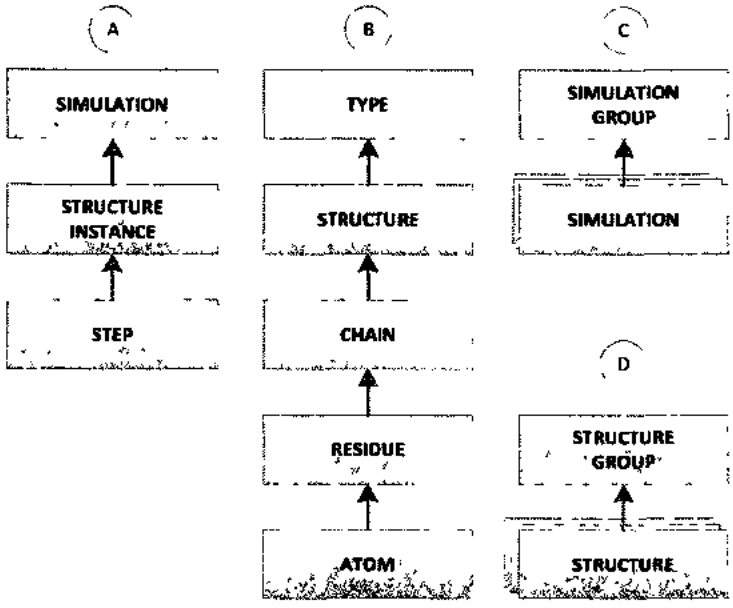


Figure 2. Dimensional Hiearchies and Groups. The simulation hierarchy (A) links simulation time (step) through structure to simulation parameters. The structure hierarchy describes chemical structure starting from individual atoms. The simulation group (C) and structure group (D) dimensions allow simulations and structures to be placed in curated groups for analysis.

X-Ray, NMR, Homology Model, or Engineered Rotomer. The structure level includes identifying information such as the structure identifier (struct_id), structure, PDB code, name, and additional attributes that apply to an entire structure.

Organization within a structure begins at the chain level of the hierarchy. A single PDB entry may contain multiple polymers, each are assigned a unique chain identifier. A polymer is composed of a sequence of residues. A residue is a logical grouping of atoms, usually corresponding to an amino acid, but it can also be used for non-polymers such small molecules, ions, and ligands. Non-polymers will be assigned the same chain identifier as the polymer with which they are associated.

Residue attributes include a residue name and abbreviation, description and general properties. Also included are residue number, and insertion code (ICode). When combined with the PDB code and chain identifier, the residue number and insertion code provide a direct link back to the original PDB entry. Residue numbers are sequential integers and are applied

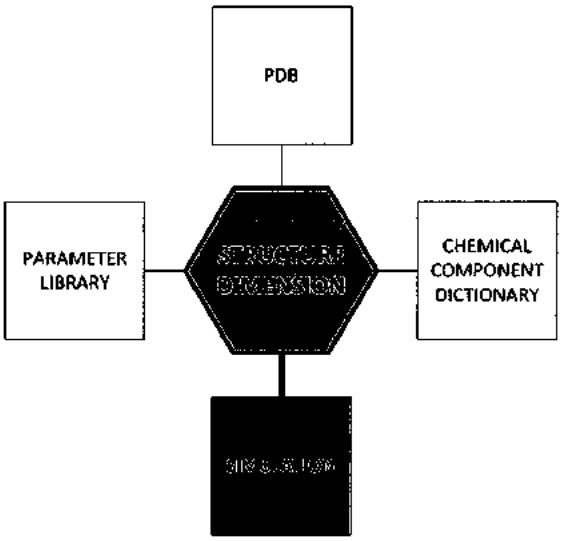


Figure 3. Structure Dimension Links. The structure dimension links simulations to the parameter library, the Protein Databank, and also uses standard atom and residue names from the Chemical Component Dictionary.

within a chain, but the sequence may include gaps (missing residues) or insertions (residues added with the same residue number). Gaps are not stored in the dimensional model. An insertion code will be set for each residue added at the same residue number; the sequence is typically "A, B, C ..." etc.

The lowest level of the structure hierarchy is Atom. Atom attributes include a name, type, and a sequence number. Following the PDB convention, atoms are numbered sequentially within structures using positive integers. The atom number and structure identifier uniquely identify members and thus serve as the key of the dimension.

The Simulation Dimension

Molecular dynamics (MD) is a technique from theoretical physics to simulate the interaction and motion of a system of particles. The simulation dimension models starting parameters, the set of molecules being simulated, and time. The simulation attribute hierarchy reflects this organization and includes levels for simulation, system, and step.

The simulation level holds simulation starting parameters, including the set of parameters that

Table 1. Unique Simulation Attributes. These dimension attributes are the set of starting parameters that uniquely identify a simulation. Each combination of these values is assigned a single integer simulation identifier (sim_id), which is then used throughout the warehouse. Managing simulations based on these attributes allows for a clean separation of physical storage and simulation definition.

Attribute	Description
structures	The set of structures included in the simulation system
minimized structures	The set of minimized structures used as starting structures
temp	Simulation temperature (K)
run	A locally assigned positive integer used to differentiate multiple executions
pH	Qualitative definition of acidity/basicity of the simulation environment (high, medium, low)
density	Solvent density (g/ml)
random seed	Random number seed used for initial random assignment of velocities
time step	Conversion factor for calculating time in picoseconds from a step (ps), typical value is 0.002 ps
initial box size	Dimensions (x, y, z) of periodic box (Å)
c scale	Charge scalaing factor for electrostatic potential
a scale	Scaling factor for 12/6 attractive and 12/6 repulsive terms of the Lennard Jones potential
cutoff range	Maximum distance between two atoms to include electrostatic interactions (Å)
h3d sync	Number of steps to reuse the non-bonded interaction pair list
simulation engine	Simulation software used to run the simulation

uniquely identify a simulation (Table 1). A simulation identifier and some annotation attributes are also part of the simulation level of the dimension. A simulation will contain one or more structures, and are referenced by structure instance in the system level of the hierarchy.

The lowest level of the simulation dimension hierarchy is step. At the core of simulation engine is a potential function, which is an equation used to calculate the energy of a system based on the relative locations of participating particles. In all-atom protein simulations, the number of particles being tracked is large, and the classical equations of motion must be solved numeri-

cally. This is accomplished by employing the assumption that for sufficiently small periods of time, positions for participating particles can be calculated based solely on their location relative to other particles. The implication is that the primary simulation output, coordinates, will be output at regular intervals referred to as steps or frames. A step, structure instance, and simulation identifier form the key of the simulation dimension.

The Structure and Simulation Group Dimensions

The structure group dimension allows structures of any type to be placed into curated sets, which can be referenced easily in queries, used in aggregates, and annotated using detailed description attributes. A structure may participate in zero or more structure groups. The simulation group dimension performs a similar function—it allows simulations to be placed into curated sets, and similar to structure groups allows sets of simulations to be referenced easily in queries.

Relational Design and Implementation

A dimensional model must be mapped to tables in order to be implemented in a relational database. In addition to tables required for dimensional attributes, tables must be created to hold fact data and to manage identifiers. An initial design was described by Simms et al. (Simms, 2008), but it has changed significantly since the first implementation. Major changes include extensions to support multiple MD simulation packages, better integration with the PDB, structure groups, the 2009 Consensus Domain Dictionary (CDD) (Schaeffer, 2011), Molecular Mechanics Parameter markup Language (MMPL) (Simms, 2011), in lucem molecular mechanics v2009 (Beck, 2000-2011), spatial indexing (Toofanny, 2011), and standardizing on step to represent simulation time. The following sections discuss the relational design and database platform-independent implementation.

Directory and Simulation Databases

MD simulations are fundamentally very large sets of three-dimensional spatial coordinates, ordered by time. Analyses are derived from coordinates by calculating various statistics, which can be associated with any level of the structural hierarchy. Simulations and analyses are facts in the dimensional model. The raw coordinates and analyses cannot be interpreted without being tightly integrated with structural information, and coordinates from two simulations of the same structure are independent. Thus, a natural organization is to store each simulation and associated analyses in separate relational tables. To avoid having thousands of tables in a single database, simulations and analyses are grouped by project and structure into multiple simulation databases. A single database, called the Directory database, is used to house structure dimensions, manage identifiers, and record the physical location of simulation databases. This model facilitates the distribution of simulation data across multiple servers.

The schema of the Directory database is illustrated in Figure 4. It includes tables related to the structure, simulation, structure group, and simulation group dimensions; mechanism for managing structure identifiers, simulation identifiers; dimensions for analyses; and tables to support MMPL. Tables that are part of the dimensional model, provide identifier support, or used by front-end applications for navigating the model are named with "Master" as a prefix.

Molecular Structure

The structure and structure group dimensions are implemented using the set of tables shown in Figure 4 (C, D). The primary structure dimension tables are Master_Structure and Master_ID, which are the store of record for structure attributes. Two additional tables, Master_ProteinMap and Master_StructureMap, and stored procedures implement the allocation of new structures. The Master_MinStructure, Master_MinID, Master_MinStructureMap, and Master_MinIDMap tables mirror their counterparts for the management of minimized structure attributes; however, these are currently used only for structure allocation are not part of the di-

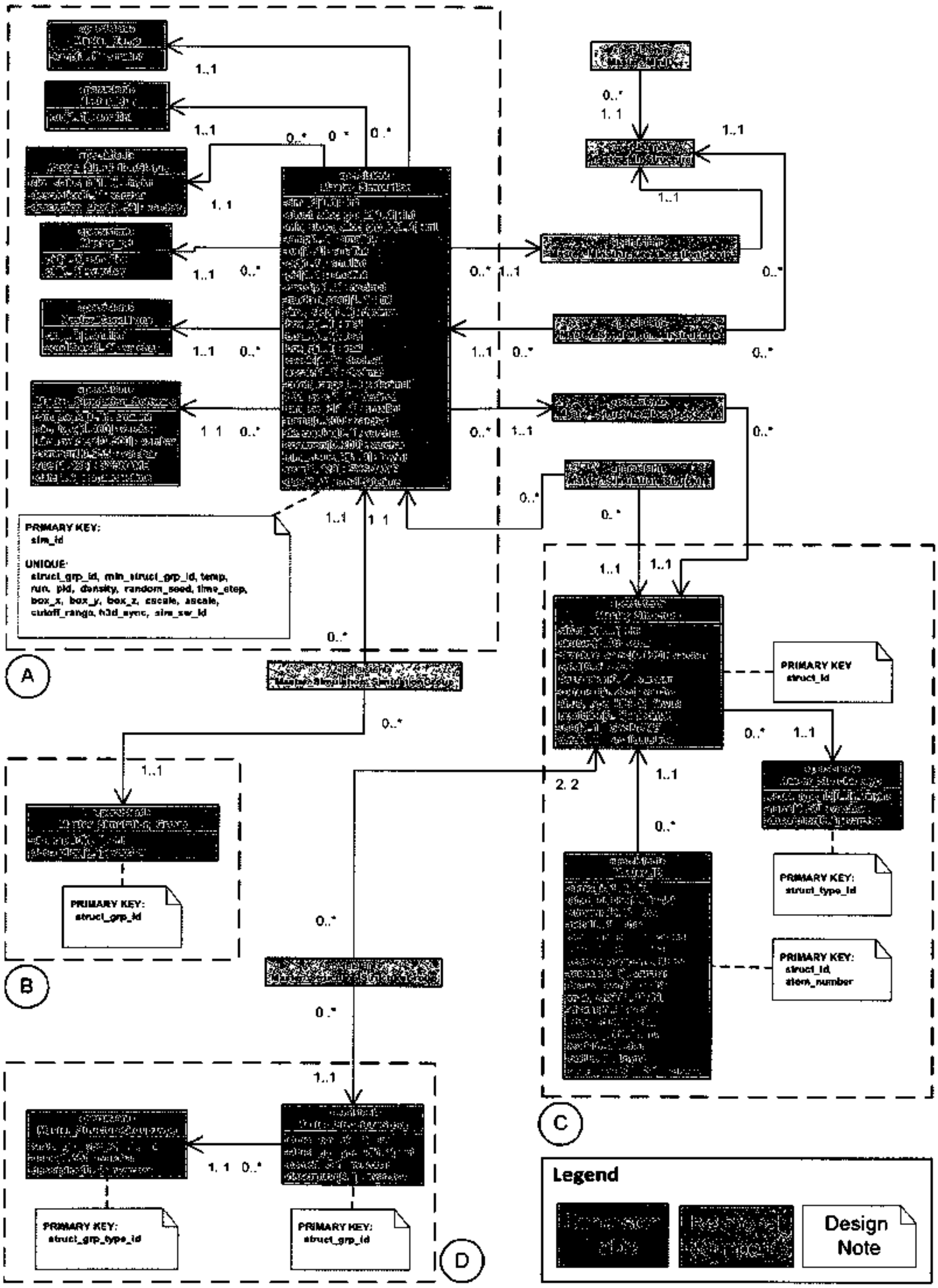


Figure 4. Directory Schema Diagram. The Directory database contains a relational implementation of the four primary dimensions: Simulation (A), Simulation Group (B), Structure (C) and Structure Group (D).

mensional model. Following the dimensional model, the Master_ID table is keyed on struct_id and atom_number. Since the Master_Structure table does not contain atom attributes, it is keyed only on struct_id, and a foreign key constraint insures that all rows of Master_ID are associated with a structure.

Master_Structure also manages a second key, called structure. This identifier was introduced because although it is common practice to refer to simulated proteins by their PDB code (a four character identifier assigned by the Protein Databank), there are several issues with attempting to use these codes directly as identifiers. First, PDB structures are routinely modified in order to prepare them for simulation. This process can involve selecting a specific chain, adding hydrogens, excising residues, mutating residues, building in missing residues, and many other transformations. The result of any of these transformations is a new structure, which although derived from a PDB structure, is a unique entity. A second issue involves the simulation of small molecule cofactors that are included in the PDB structure. It is common to simulate the protein by itself (apo) and with the cofactor present (holo). These are different structures from the standpoint of simulation. Lastly, there are many structures that do not have a PDB code. Some examples include synthetic proteins and homology models. The structure field addresses these shortcomings by combining a character prefix called a structure base (e.g., a PDB code) and numeric suffix.

A stored procedure manages the creation of both the structure and struct_id identifiers. It performs a residue sequence structural comparison when determining whether or not to allocate a new structure identifier. This comparison considers only at the supplied structure base and the residue sequence. If the structure base and residue sequence exactly match an existing

Table 2. Structure Group Type. Structure groups are classified by a type value stored in the Master_StructureGroupType table. The current types are shown below and can be expanded by adding new rows to this table.

ID	Name	Description
1	simulation modification	structure changes required for simulation (e.g. protenation).
2	SNP mutation	single nucleotide polymorphism
3	holo/apo	indicates structure was formed from holo structure

structure, the existing structure will be used. If there is any deviation, a new structure will be allocated. Minimized structures (Master_MinStructure, Master_MinID) are handled similarly, but are currently only used for simulation allocation, which is discussed in the next section.

Structure groups allow for a simple two-level hierarchical organization of related structures. One structure serves as the parent, and one or more related structures as children. This concept was introduced to accommodate accurate counting and tracking of structures that are derived by modifying a base structure. There are currently three types of structure groups in

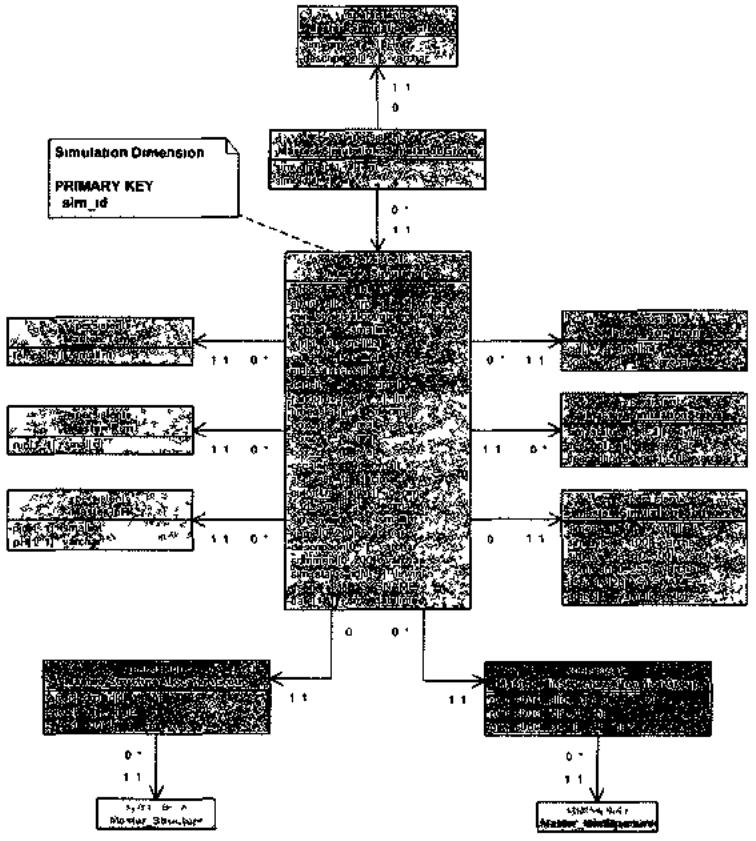


Figure 5. Simulation and Simulation Group Dimension Tables. Relationships for the simulation dimension and associated snowflake dimensions.

Master_StructureGroupType, as shown in Table 2, and more can be defined as needed.

The Master_StructureGroup table stores identifiers, names, and a description. The Master_Structure_StructureGroup table links a child structure to a parent structure. The optional

Table 3. Simulation Dimension Attributes and Relational Columns. The attributes of the simulation dimension are mapped to SQL Server data types and stored in the Master_Simulation table. The fixed exact size type DECIMAL (9,5) (a 5 byte floating point value) is used for floating point quantities because these columns will be included in a unique index. Structures (and minimized structures) are mapped to a single integer identifier; other integer values are represented directly.

Attribute	Relational Column(s)	SQL Datatype
structures	struct_alloc_grp_id	INT
minimized structures	minstruct_alloc_grp_id	INT
temp	temp	SMALLINT
run	run	SMALLINT
pH	pH	SMALLINT
density	density	DECIMAL(9,5)
random seed	random_seed	INT
time step	time_step	DECIMAL(9,5)
box dimensions	box_x, box_y, box_z	DECIMAL(9,5)
c scale	cscale	DECIMAL(9,5)
a scale	ascale	DECIMAL(9,5)
cutoff range	cutoff_range	DECIMAL(9,5)
h3d sync	h3d_sync	INT
simulation engine	sim_id_type	SMALLINT

relationship_tag field is used to annotate a specific parent-child relationship, for example this field is used with single nucleotide polymorphisms (SNPs) to record the residue number and mutation.

Simulation Parameters

Simulation and simulation group dimensions attributes are stored in the set of tables illustrated in Figure 5. Simulations are assigned unique integers based on the attributes listed in Table 1, which are mapped to columns as shown in Table 3. It is a requirement that the structures being simulated be previously allocated. Since a simulation may contain multiple structures (or even multiple copies of the same structure), the Master_StructureAllocationGroup table is used to assign a single integer id to sets of structures, struct_alloc_grp_id. Sets of minimized structures are also assigned a single integer id, min_struct_alloc_grp_id, and stored in the Master_MinStructureAllocationGroup table. An important consequence of this approach is

that the order structures are added to a simulation is not considered when determining if a simulation has been previously allocated.

Once structure allocation group identifiers have been assigned, a stored procedure uses a mapping table, Master_SimulationMap, to generate a new simulation identifier (**sim_id**) or to find an existing id. Similar to the structure dimension tables, restricted data types and check constraints are employed to prevent invalid values from being entered manually or by software failures. Check constraints for secondary dimension attributes, such as pH, are defined on the associated table, and enforced via foreign key constraints.

The simulation dimension hierarchy contains two more levels: system and step. The system level accounts for the structures included in the simulation, and step is a proxy for time. Multiple structures can be associated with a simulation, and more than one copy of a structure may be present. Each structure is assigned a structure instance identifier (struct_inst), which is scoped to that simulation. Because each struct_id, struct_inst, and step are stored in the fact table, there is no need to create an additional relational table with these values.

The simulation group dimension enables simulations to be organized into groups. The dimension consists of the Master_SimulationGroup table and linking table, Master_Simulation_SimulationGroup, which implements a many-to-many relationship between the group definitions and simulations. Simulation groups are assigned an identifier (sim_grp_id) as well as a name (sim_group_name), description, and a curation status (curated). The curated flag, when set, indicates that the simulations associated with the group are final. Simulation group membership cannot be altered while the curated flag is set.

Facts

Fact tables store continuous measurements and are linked to dimensions through key attributes. In a relational implementation, the key of the fact table is the set of dimension attributes for a row. The warehouse currently supports 18 distinct fact types, which are listed in Table 4. Each fact type is linked to a level in the attribute hierarchy in one or more dimensions.

Table 4. Supported and Planned Fact Types. Fact data are stored as tables named using a type abbreviation, an underscore ("_"), and a simulation identifier. Coordinate trajectories are produced during simulation and stored in Coord tables (centered and aligned, suitable for viewing) and GCoord tables (untranslated). The remaining fact types are used to store analysis data derived from the coordinates.

Abbreviation	Description
astrand	Alpha Sheet Residue[1]
Box	Periodic Box Size[2]
Bins	Hash 3D Spatial Index of Neighbors
Congen	Conformational Geneology[2]
Contact	Native Contacts By Time[2]
Coord	Coordinate Trajectory[2]
Dihed	Dihedral Angles[2]
DSSP	Dictionary Secondary Structure Prediction[2]
FContact	Full Heavy Atom Contact Distance By Time
FContactSolv	Full Heavy Atom Contact Distance by Time with Solvent
FDSASum	Fine Detail Structure Analysis Summary By Time
Flex	Flexibility (Per Atom)
Forces	Instantaneous forces
ForcesSolv	Instantaneous forces with solvent
ForVel	Per Atom Force and Velocity
GCoord	Global Coordinate Trajectory
GCoordSolv	Global Coordinate Trajectory with Solvent
NOE	Nuclear Overhauser Effect[1]
period	Periodic Contacts[1]
PhiPsi	Phi Psi Angles[2]
Radgee	Radius of Gyration[2]
RMSD	Root Mean Square Distance from Starting Structure[2]
RMSF	Root Mean Square Fluctuation
SASA	Solvent Accessible Surface Area[2]
VCont	Verbose Contacts Summary

[1]Reserved for future implmentation; [2]Original 2007 release

When the linking attribute corresponds to primary key in a dimension table, a formal foreign-key relationship is created and enforced via a constraint. In other cases, the relationship is implied. As mentioned previously, simulations are distributed to multiple databases to avoid large numbers of tables in a single database. Because referential constraints only apply within a database, dimensional data from the Directory database must be replicated to individual databases in order to create and enforce explicit foreign key constraints. However, since each database contains only a subset of the entire set of structures and simulations, only dimension data related to the subset are required.

Table 5. Shared Identifiers. The data warehouse andthe ilmm simulation engine share semantics for these identifiers, allowing interoperability between the warehouse and simulations. In general, the warehouse is responsible for allocating identifiers.

Identifier	Type	Description	Valid Ranges	Notes
step	Int32	Simulation step (frame)	[0, +2billion)	
struct_inst	Int32	Structure Instance	[0, +2billion)[a]	ilmm Molecule Number + 1
struct_id	Int32	Structure Identifier	[0, +2billion)[a]	Stored in system_mmpl.xml after allocation
atom_number	Int32	Atom Number	[0, +2billion)[a]	Defined by MMPL
residue_id	Int32	Residue Identifier	[0, +2billion)[a]	Proxy for PDB residue number, chain and icode

[a]0 is a reserved value.

General Simulation Engine and PDB Integration

Key goals for the Dynameomics data warehouse after 2007 were to achieve deep integration with the lab's in-house simulation package, ilmm v2009; the Protein Databank (PDB); and to support simulations created by other simulation packages. Achieving tight ilmm integration required that there be a fundamental alignment of data types and recognition of responsibilities for managing data between ilmm and the warehouse. This alignment consists of two parts, first there are shared identifiers which are to be supported natively by ilmm and the warehouse; second is an accepted definition of a set of attributes, other than file system location, that uniquely identifies a simulation. The shared identifiers are listed in Table 5.

PDB integration was determined to be a critical requirement for all simulations using PDB based structures. Earlier versions of *il*mm systematically pruned PDB residue number information out of structure data, replacing it with more computationally convenient zero based identifier, which the warehouse would store as well. Because PDB structures can contain missing residues (gaps), negative residue numbers, and can even contain duplicate residue numbers (which are differentiated by insertion codes), both the warehouse and *il*mm were modified to preserve and support the original PDB residue numbering.

Supporting other simulation packages involved identifying the key set of starting parameters and then storing these values for each loaded simulation. The canonical list of simulation attributes was shown earlier in Table 1 and accommodate both ilmm and ENCAD (Levitt, 1983)(Levitt, 1995)[11] style simulation engines. Supporting other engines involves defining a simulation engine and mapping additional attributes unique to that engine into the conditions text field.

SQL Server Implementation

SQL Server is a relational database platform from Microsoft (Microsoft Corporation, 2007). The latest versions include many features defined in the SQL99 (International Organization for Standardization, 2001) specification in addition to proprietary features. This database platform was chosen based on prior experience and support from Microsoft Research. In order to understand the implementation approach of the data warehouse, it is important to know about the physical data model of SQL Server and to consider the configuration of servers. In this section the decisions made to produce an optimal SQL Server implementation are detailed; however, many of the choices can be adapted to any vendor's implementation.

SQL Server Architecture

SQL Server is available in several editions that vary widely in cost and features. This

project uses SQL Server 2008 Enterprise Edition R2 x64 (Microsoft Corporation, 2007) installed on Windows 2008 Server R2 Enterprise Edition x64; the database engine, critical database services, and the Windows Server operating system are all native 64 bit binaries running in a 64 bit environment. The enterprise edition of Windows 2008 x64 was chosen as the base operating system primarily because it can support a maximum of 2TB of RAM (the standard edition is limited to 32GB of RAM). SQL 2008 Enterprise edition R2 was chosen for its support of partitioning, data compression, and large memory support (2TB maximum). The project currently does not utilize failover clustering. SQL Server supports a concept of instances, which are independent environments that contain databases. Currently, a single instance (referred to as the default instance) is configured on each server in the warehouse.

Databases

The fundamental unit of organization within an instance is the database. Databases consist of sets of data and transaction log files, and each type is managed differently. Multiple data files are used to manage space and to distribute I/O activity to multiple disks and/or disk controllers. In contrast, only a single log file is active at a time and thus multiple files are used only to manage growth. By default, when a database is created it will consist of a single data file (MDF) and a single log data file (LDF). Storage for tables is allocated inside both the MDF and LDF during loading, and moves entirely to the MDF file once transactions are committed and the log file is truncated. Data files contain data structures called pages, which are 8KB in size and are read and written to disk in groups of 8 called extents (64KB). LDF files contain transactional log information, effectively recording changes to pages in the MDF.

Tables and Indexes

Within a database, the primary objects are tables and indexes and the data for each are

stored in pages. Tables are classified into two types based on storage—heap mode (no clustered index) and index mode (clustered index present). Heap mode tables are unordered collections of pages; Index tables contain pages sequenced in the order of the clustered index. Indexes on a table, including clustered indexes, are implemented as Balanced Trees (BTrees) for efficient searching. In the non-clustered case, leaf nodes contain pointers to the data pages for the table. For clustered indexes the leaf nodes of the index are the data pages for that table, thus tables can have only one clustered index.

The lowest level of data organization in SQL Server is the row, which contains the individual data items for each column of a table. Rows are stored in pages, sequentially. The number of rows that can be stored in a page depends on the data types chosen for the columns. However, a fundamental rule is that rows cannot span page boundaries, which constrains the total size of a row to 8060 bytes. There are some exceptions for specific data types, variable length text fields will be moved automatically to special overflow data pages if they would cause a row to exceed the limit. Large object types store only a pointer in the data row, and the actual column content is stored in a page type reserved for binary large object (BLOB) type data. Additional details on how tables are mapped to pages can be found in SQL Server Books Online (Microsoft Corporation, 2010) and Fritchey and Dam (Fritchey, 2009).

Performance Optimization

Fundamentally, all performance tuning of a SQL database comes down to minimizing I/O operations. When a query is executed on a heap mode table, the data engine reads all the extents associated with that table, literally traversing every row looking for data to satisfy the query in a costly operation known as a table scan. When a table with indexes is queried, the query optimizer will attempt to use the indexes to limit reads to fewest extents as possible to satisfy the query. In contrast, the fastest write (insert) operations occur on heap-mode tables because the server can add pages without regard for order. This makes indexes highly desirable

Table 6. Common SQL Server Data Types. SQL Server supports a variety of data types. For numeric dimensional columns, the smallest fixed size exact numeric types that can accomodate the data are preferred. Fixed characters can be used but it usually preferable to code categorical string values to fixed numerics.

Size	Type	Domain	Name	Min Size[1]	Max Size[1]
Fixed	Exact Numerics	Integer	BIGINT	8	8
			INT	4	4
			SMALLINT	2	2
			TINYINT	1	1
		Real	DECIMAL	5	17
			MONEY	8	8
			SMALLMONEY	4	4
	Approximate Numeric	Real	FLOAT	4	8
			REAL	4	4
	Strings	Characters	CHAR	1	8000
			NCHAR	2	8000
	Binary	Binary	BINARY	1	8000
Variable	Strings	Characters	VARCHAR[2]	1	8000
			NVARCHAR[2]	2	8000
	Binary	Binary	VARBINARY[2]	1	8000

[1]Size in bytes [2]Supports large object extension MAX, resulting in off-page storage

for read operations but a severe burden on write operations. In a data warehouse, data are primarily read-only and thus indexes are used extensively to limit I/O operations for queries. In this project, fact tables are created as heap-mode tables, loaded using fast bulk load primitives, and then indexed afterwards. A SQL Server feature, used for coordinate tables only, builds an empty table with a clustered primary key and the loads the data in clustered key order. Remaining indexes and constraints are added after loading.

Design Considerations for Fact Tables

Fact tables will contain columns for measures and for a set of dimensional keys that link the measures to the dimensional hierarchy. The set of dimensional keys columns are a candidate key of the table, meaning they uniquely identify a row and are not null-able. Beyond meeting the requirements of the dimensional model, there are three primary considerations in designing fact tables: total row size, indexes, and check constraints. Although these considerations apply to any relational design, they are especially important for fact tables as they house the majority of data in a warehouse.

Row Size

SQL Server supports a variety of data types for columns, which are classified into three major categories: native types, native large object types, and Common Language Runtime (CLR) user defined types. A subset of native data types used for fact and dimensional quantities as are listed in Table 6. The implementation of these data types is highly optimized for search and storage. Native types are subdivided into five subgroups: fixed length numeric, fixed length character, fixed length binary, variable length character, and variable length binary. Numeric data types include approximate floating point types based on the IEEE 754 standard (IEEE Computer Society Standards Committee, 1985), integers, and a set of exact numeric types. Native large object types are used specifically to work with binary or text data that are too large to be stored in an individual data page. These were originally vendor extensions, and have been largely subsumed by variable length native types. SQL Server also supports common language runtime (CLR) user-defined data types, used for object-relational applications. The use of various data types in fact tables are discussed in the following sections.

It is always preferable to implement fact tables using the smallest native fixed size data types that will accommodate the data. Variable length fields cause row sizes to vary within a page, and if the actual field length plus the size of other columns exceeds 8060 bytes, data are moved into one or more overflow pages. Variable length columns require additional bookkeeping overhead to track field length. Overflow pages and bookkeeping overhead reduce the number of rows that can be stored per page, increasing overall table size and decreasing efficiency. In contrast, the size of a row containing only fixed length data types is determined by equation (2.1). Although there is some overhead for tracking column null-ability, the primary row size contribution is the fundamental size of the data type (see Table 6). The net results of using only

$$4+\left(2+\frac{(columns+7)}{8}\right)+\sum_{i=1}^{columns} datatypesize(i) \quad (2.1)$$

fixed data types are a consistent and minimal row size.

Index Design

Indexes are used to limit I/O operations during queries, and to enforce primary key and unique constraints. Indexes in SQL Server are implemented as balanced trees (BTrees) and are stored in page structures similar to data. Index rows contain the nodes of the BTree. Each node, starting at the root, contains lowest value of and a pointer to each subtree. The leaf nodes of a clustered index are the data pages of the table, the leaf nodes of a non-clustered index contain the primary key columns if the table has a clustered primary key or a row identifier pointer otherwise. This means that indexes benefit from using narrow fixed length data types, to enable the greatest number of sub-trees per node. The rows of an index are ordered by the contents of the index's columns. Indexes can be built on any column data types with the exception of the large object types; however, there are special issues for some of the remaining column types. For character and native variable length columns, the index can only include data the data that fits in the standard index page—characters outside this range will not be included. This is a second reason not use variable length columns in a fact table. Approximate floating point data types should be avoided in index columns—these types use an efficient but non-unique bit representation of values (meaning that more than one real number is mapped to the same bit pattern). This makes indexes built on approximate types unpredictable. CLR data types can be included in indexes, but are treated as binary values. Four final special cases are the native fixed size integer types, TINYINT, SMALLINT, INT and BIGINT. These values can be directly loaded, tested and manipulated in integer registers found on x64 architecture microprocessors, and are the most frequently used key types in star schemas.

In order for an index to be used in the processing of a query, the query must contain a sargable predicate. The term sargable predicate, which is a contraction of "search argument able," refers to an expression in the where clause of a query containing tests of equality (=),

less than (<), greater than (>), less than or equal (<=), greater than or equal (>=), BETWEEN, or LIKE using a prefix search (Fritchey, 2009). This is the direct result of the underlying data type's or types' support for comparison operations based on mathematical inequality (less than, greater than), or equality (equal to). All integer and exact numeric types support less than, greater than, or equal to operations and thus when indexed can be searched with sargable predicates. This makes these types useful for fact tables. Character types (fixed and variable) can be as well, but row and index size considerations discussed earlier make these poor choices for fact columns. An interesting corner case is the fixed size uniqueidentifer (uid, also called a globally unique identifier or guid). This data type supports equality and inequality comparisons, but does not support any mathematical operations. In this sense, a sargable predicate can be used with a uid. However, since uids have no intuitive data ordering, they are really only useful for decentralizing identity assignment. Uids cannot be used as a partitioning scheme, and their 16 byte size adds significant row size overhead both in a data page and any index pages.

Check Constraints

Check constraints are used to block incorrect data from either being inserted into a table or existing data being incorrectly modified. Check constraints are declared at the table level in the form of a predicate expression that can reference columns and constants. The expression is evaluated as data are modified or added, and if the new or modified data does not satisfy the check constraint expression, an error is thrown and the row is rejected. In SQL server, check constraints are also used by the query optimizer in selecting rows from views, unions, and individual tables. For an individual table with a column sim_id, and a constraint limiting the value of this column to 123, a query against that table asking for sim_id 234 will immediately return with no results. When a view or a set of tables combined using UNION are queried, and the query predicate references a column with a constraint, and the data requested is outside the range of the constraint for some tables, the query optimizer will drop those tables from consid-

eration.

Coordinate Fact Table Design

For MD simulations, coordinates make up most of the data being stored. Even when simulations are stored as individual tables, they may contain as many as a billion rows of information. This makes the choice of data types and design of indexes extremely important as it will determine how efficiently data and index rows can be mapped to pages, which in turn dictates table size, and ultimately query performance. For a coordinate fact table, there are nine columns, four columns for the three-dimensional atomic coordinate and bin index, and 5 dimensional columns that relate the coordinate back to a structure. The range of each coord_x, coord_y, and coord_z value is limited by the box size of a simulation, and are well within a range of -500.0 Å and 500.0 Å. Because coordinates do not participate in an index, the 4 byte REAL approximate type is used for these columns. The bin column is used to store a non-negative integer quantity, which is also limited by box size and will not exceed 100,000, allowing a 4 byte signed integer (INT) column to be used. At the current resolution of 0.002 ps per step, an INT can accommodate a simulation of up to 4 μs in length. The remaining dimensional columns of struct_id, struct_inst, and atom_number are all implemented as 4 byte INTs. Recall that after overhead, 8060 bytes are available for row storage. All nine coordinate table columns are 4 byte fields, 3 are type REAL and the remaining are INT. The data storage per row consumed by this structure is 36 bytes, three bytes of null tracking overhead, and a 4 byte row header, which means a single data page can accommodate 187 coordinate rows.

It is critical to allow coordinate rows to be efficiently located. A candidate key in a relational table includes a set of columns that uniquely identify a row and which cannot take on null values. In a dimensional model, the set of dimension foreign keys constitute a candidate key. One candidate key is typically chosen as the primary key, which usually only includes only the minimum set of columns that uniquely identify a row. Although column order is not

a consideration for key purposes, the primary key is most often implemented in tandem with a clustered index in which column order is essential. Looking again at the coordinate fact table, a minimal data column footprint has been determined by choosing 4 byte data types for columns. The columns specified and the order they appear in the key should follow the most common pattern of usage. For coordinates this pattern is to locate frames and then atoms within frames. However, there are two opportunities for optimization. First, since simulations are placed in separate tables, the sim_id should not be included in the clustered index. Structure identifiers (struct_id) should also not be included, as this column is always determined by struct_inst. The second opportunity is to not even include struct_inst, when there is only one structure in a simulation. These two changes reduce the index row size by 12 bytes for single structure simulations, a significant savings over simply building an index on all dimension columns. The minimal clustered primary key also benefits to two additional non-clustered indexes for spatial index queries and an index for fast coordinate retrieval by atom_number. Non-clustered index leaf nodes store the primary key columns of the target data table, so reducing the size of a primary key will also reduce the size of non-clustered indexes.

Coordinate fact tables use CHECK constraints to both protect against bad data and to optimize queries where fact tables are grouped in views joined using UNION. The sim_id column is always constrained to single value and is not included in the clustered primary key. If the simulation contains only one structure, the struct_inst column is limited to a value of 1 and the struct_id column is limited to one value. If the simulation contains more than one structure, struct_id is constrained to a set of values, and struct_inst is constrained to a range of values.

Analysis Fact Table Design

Analysis fact tables contain data that are derived from coordinates, but can have different dimensionality. Coordinates are linked to the lowest level of the simulation and structure hierarchies, and thus establish the primary dimension keys for simulation (sim_id, step) and struc-

ture (struct_id, atom_number). Analyses that contain per atom and per step quantities, such as instantaneous forces, use the same dimension keys as coordinates. Other physical properties are associated with different levels of the simulation and structure hierarchies through many-to-one relationships. For example, Cα root-mean-squared-deviation root from starting structure (RMSD) is linked to structure at the residue level, and to simulation at the step level. Relationships between all analysis fact tables and dimensions are summarized in Table 7. Like coordinates, analysis tables never include the sim_id column in the primary key and only include struct_inst for multi-structure simulations. Check constraints are also used to ensure that the sim_id column is a constant, struct_id is either a constant or a limited range of values, and other columns limited as appropriate.

Some analyses include multiple distinct quantities that are associated with the same structure and step, or that contain categorical names. These are modeled through the use of an additional dimension, which is unique to the analysis. One example is the dihedral analysis, which contains a variable number of rows that are associated with a structure at the residue level and a simulation at the step level. Each row contains a dihedral angle, which is a measurement rotation about specific named bond inside the residue or along the main chain at the Cα where the residue is attached. The number of rows depends on the number of carbon-carbon bonds present in the residue, as each angle is associated with a specific named bond. Dihedral angle names and abbreviations are broken out to a small dimension table called Dihedral_Angle (Table 8), allowing the Dihedral fact table to use a single byte identifier (dh_id) as a link to the angle name. The Dictionary of Secondary Structure Prediction (DSSP) analysis follows a similar pattern, using the dimension table Secondary_Structure (Table 8) to define secondary structure types under a single byte identifier (ss_id). The PhiPsi analysis includes only one set of values per residue, but includes an assignment to secondary structure state categories shown in Table 8. Here a small dimension table is used to avoid placing character data in the PhiPsi fact tables, saving space.

Table 7. Dimensional Key Column Usage. A consistent set of column names are used throughout the warehouse to refer to dimension table keys. Where possible, these relationships are enforced explicitly through the use of primary key/foreign key constraints.

	Columns									
Fact Table Type	**step**	**struct_inst**	**struct_id**	**residue_id**	**atom_number**	**dh_id**	**ss_id**	**stid**	**hash3d_index**	**hash3d_index_neighbor**
Box	PK									
Congen	PK	O	FK[1]							
Contact	PK	O	FK[1]							
FDSASum	PK	O	FK[1]							
Radgee	PK	O	FK[1]							
RMSD	PK	O	FK[1]							
Vcont	PK	O	FK[1]							
Flex	PK	O	FK[1]	PK						
Dihed	PK	O	FK[1]	PK		FK[3]				
DSSP	PK	O	FK[1]	PK			PK,FK[4]			
PhiPsi	PK	O	FK[1]	PK				PK,FK[5]		
SASA	PK	O	FK[1]	PK						
Coord	PK	O	FK[1,2]		PK,FK[2]					
Forces	PK	O	FK[1,2]		PK,FK[2]					
ForVel	PK	O	FK[1,2]		PK,FK[2]					
S2	PK	O	FK[1,2]		PK,FK[2]					
RMSF		O	FK[1]	PK						
Bins									PK	PK

PK = Primary Key Column, O = Optional Primary Key Column, FK = Primary Key Column referencing dimensional table: 1. Structure. table 2. ID table. 3. DihedralAngle table. 4. SecondaryStructure table. 5. State table. 5.State table.

Table 8. Secondary Dimensions for dihedral angles, secondary structure, and Φ/Ψ state. The dihedral analysis calculates multiple bond angle values per residue at each time step. Each value is associated with a specific named bond, which are assigned to an id defined in this dimension table. This identifier is then used in the fact table in place of an explicit string constant. Secondary Structure is produced by the DSSP analysis. It produces multiple values per residue and step, and similar to dihedral analysis, an id is defined for each character and structure definition. The Φ/Ψ analysis produces only one value per residue, the calculation also includes a structure state prediction. The state labels are assigned ids defined in this table and then used in the fact table in place of string labels. These dimensions can be extended and additional dimensions added at any time to support new analysis.

Dihedral Angles			Secondary Structure			Φ/Ψ	
id	name	angle	id	character	description	id	state
1	chi1	X1	1	a	alpha strand, parallel	1	beta
2	chi2	X2	2	A	alpha strand, anti-parallel	2	other
3	chi21	X21	3	b	beta strand, parallel	3	extended
4	chi22	X22	4	B	beta strand, anti-parallel	4	helix
5	chi3	X3	5	c	mixed alpha/beta strand, parallel		
6	chi31	X31	6	C	mixed alpha/beta strand, anti-parallel		
7	chi32	X32	7	r	alpha bridge, parallel		
8	chi4	X4	8	R	alpha bridge, anti-parallel		
9	chi5	X5	9	s	beta bridge, parallel		
10	chi6	X6	10	S	beta bridge, anti-parallel		
11	chi61	X61	11	G	3-10 helix (3 residues per turn)		
12	chi62	X62	12	H	alpha helix (4 residues per turn)		
13	cis	cis	13	I	pi helix (5 residues per turn)		
14	omega	Ω	14	-	loop, or no assigned structure		
15	phi	Φ					
16	psi	Ψ					
17	theta	Θ					

New fact tables can be added to the warehouse as new analyses are developed. The process requires the selection of a short name, which will become the prefix of the tables created; the determination of dimensions, and the selection column data types. The short name must follow the naming conventions listed in Table 9 to avoid conflicts and to maintain consistency across the warehouse. This name is combined with a single underscore character ("_") and simulation identifier to form the final table name. All tables associated with a simulation are tracked through property views available in individual simulation databases and in aggregate in the Directory database Master_Property_v view.

Table 9. Naming rules for coordinate and analysis tables. Table names conform to a simple naming standard to avoid conflicts and to maintain a consistent interface for users.

#	Rule
1	Length: 7 Character max on the main name, note that a clarifying suffix may be added such as "Sum" or "PerAtom" does not count towards the total.
2	Characters: No spaces, characters from this set [a-zA-Z1-9_] only.
3	Capitalization: Words capped and abbreviations ALL CAPS.
4	Names and definitions must be assigned in both the Simulation and Directory databases

Conclusions and Future Directions

We have presented a detailed model for storing and analyzing data from MD simulations and its implementation in a relational database. The dimensional approach of organizing data into continuous facts and discreet dimensions is well suited to MD simulation data and could be used in many scientific applications. The implementation of this model in a relational database required careful design to overcome challenges inherent in a 100 TB data set. A directory database centralizes management of identifiers and data location, facilitating the distribution of data to multiple databases and servers. Within databases tables are highly optimized by carefully choosing column data types, building efficient clustered indexes, and using check constraints for query efficiency and data quality.

Initial work on the data model described here began in 2005 and was first released in 2007. Since the beginning, both the model and relational implementation have been in continuous development, adding new analyses, extending the relational schema, improving performance, adding more (and larger) servers, upgrading through two operating system releases and three SQL Server releases. Overall capacity has increased by nearly an order of magnitude to 150TB since the first two servers were purchased, and trajectories and analyses for over 11,000 simulations are available in the warehouse.

Chapter 3: Augmenting the Relational Model using Online Analytical Processing

A relational database provides great flexibility for describing data and building interactive applications. With nearly 40 years of development and improvement, this technology can easily manage extremely large volumes of data as well as service multiple users simultaneously. Unfortunately, the fundamental unit of storage, the two dimensional table, is not ideal for efficiently storing and analyzing multidimensional data. On-line Analytical Processing was proposed early on as an alternative to the relational model, specifically to address this and other shortcomings. The market need to store and analyze an increasing volume of financial information spawned an entire software industry segment focused on development of analysis centric tools. Despite wide spread adoption for commercial applications, using these tools to manage and analyze scientific data is far less common. In this chapter I demonstrate how this model can be applied to protein simulation data, describe an implementation using a commercial OLAP product, and present results of a storage and query performance analysis.

Introduction

Relational databases were first described in 1970 by Codd (Codd, 1970) and commercial implementations quickly established dominance in a rapidly growing and very fluid database market. Relational databases provide a rich and diverse ecosystem of general purpose features. However, much of the relational database's success can be attributed to just three: SQL, transactions, and constraints. SQL is described as "intergalactic dataspeak," (Stonebraker, 1990) and is the most common language used for expressing queries today. Although vendors implement their own dialects and special features, an analyst who knows SQL can quickly be productive accessing and manipulating data if the underlying database server supports it. Transactions enable multiple agents to safely access and modify data, eliminating the potential for data corruption and inconsistency when multiple writers and readers attempt to access the same data. In addition to supporting concurrency, transactions also protect hardware failures by in-

suring that changes are either written in their entirety, or rolled back in their entirety in the event of a failure. Finally, constraints insure data integrity by expressing business logic as declarative statements that are enforced at the server level, relieving client programs from having to enforce the same checks.

Ironically, two of the three features that make relational databases such a powerful solution are actually a detriment for data warehousing. As data in a warehouse are primarily read only (and typically modified only during bulk import from primary sources), the overhead of transaction support is a significant performance burden. Constraints, specifically foreign key constraints, introduce significant storage overhead when every fact row contains several columns referencing dimensional tables. Codd recognized the requirements of a database to support analysis are fundamentally different than the requirements for a transactional database and coined the term Online Analysis Processing (OLAP) in 1993 (Codd, 1993). This report outlines a set of principles that should be supported by an analysis centric database. Unlike Codd's papers describing the relational model, this document was written under commission for a software company and not published in a peer-reviewed journal. More importantly, it did not contain a mathematical description of the concepts, leaving this as an implementation detail for future developers. Nonetheless, the report established the importance of OLAP databases in general and specifically called for the support of sparse multi-dimensional matrices as the fundamental unit of storage. The report also emphasized the complementary approach of OLAP to other database technologies and highlighted the concept that OLAP could mediate between other data sources to present a consistent model for analysis.

Multi-dimensional matrices are the fundamental unit of analysis in OLAP systems. Although vendor products vary widely in their fundamental storage engine implementation, they all are designed to efficiently locate fact data by filtering on dimensions (slicing), to pre-calculate aggregate functions along dimensional hierarchies (aggregation), and to efficiently accommodate missing data (sparsity). Multi-dimensional matrices are referred to as hypercubes, and

the term hypercube is typically shortened to just "cube." OLAP systems typically do not support transactions, data modification after initial loading, nor constraint frameworks to protect against malformed data. Although some error checking is available during import, OLAP systems tend to assume that data being loaded has already been checked for errors (scrubbed). Thus OLAP systems are not used as the primary data store or "store-of-record" and depend on other systems, typically relational databases, to perform this function. Extraction, transformation, and loading (ETL) is the process of regularly importing data from existing data sources into the data warehouse. An additional step, called "processing" is often called after ETL to compile imported data into one or more cubes.

SQL Server Analysis Services

As described in Simms and Daggett (Simms, 2011), multidimensional data can implemented in a relational database by translating the dimensional model into a set of fact and dimension tables. Fact tables use columns to represent fact data (also known as measures) and additional columns that link each fact row back to one or more dimension tables. The links between facts and dimensions are implemented explicitly as primary keys in dimension tables and foreign key constraints in fact tables. Optimal performance on specific hardware is achieved by designing tables, indexes, and constraints. In an OLAP system, the translation of the multidimensional model to fundamental storage varies widely. For example, Oracle, Microsoft, and many others provide customized versions of their relational database software preinstalled and tuned for specific server hardware configurations and are sold together as an information appliance. Other vendors, such as Neteeza, provide a relational database engine stripped of transactional functions and that is tightly integrated with massively parallel proprietary hardware. For this project I focused on Microsoft SQL Server Analysis Services 2008 R2 (SSAS)(Microsoft Corporation, 2007), a non-relational OLAP product that is bundled with SQL Server 2008 R2.

SQL Server Analysis Services (SSAS) uses a proprietary multi-dimensional storage engine that runs on general purpose hardware, and there is no concept of tables at all. The top level of organization in SSAS is a database, which is illustrated in Figure 6. Data sources are typically relational databases, but can also include files and other database formats. A data source view is used to capture the set tables (files) from the data sources and to define their interrelationships. These relationships, taken together, are mapped to dimensions and eventually to cubes. Cubes can then be queried using the Multi-dimensional Expressions (MDX) query language (Microsoft Corporation, 1997). Each of these components is described in the following sections. Additional details on SSAS and MDX can be found in Gorbach et al. (Gorbach, 2009), Webb et al. (Webb, 2009), and Whitehorn et al. (Whitehorn, 2006)

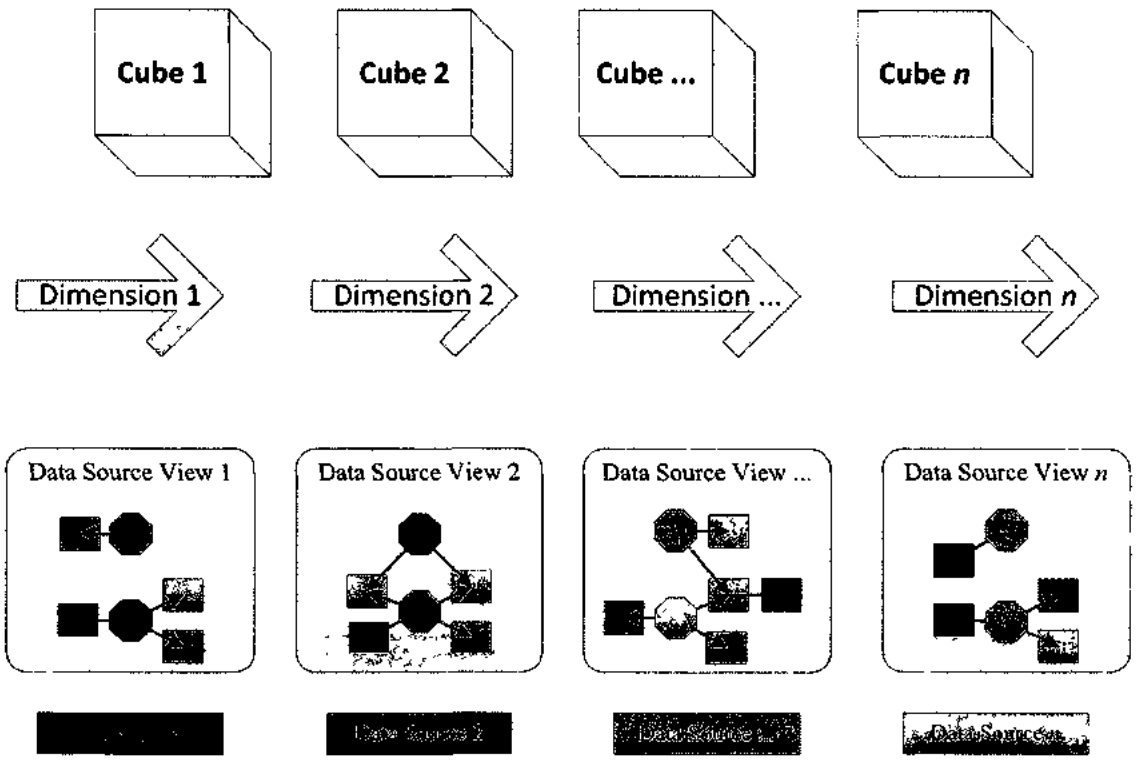


Figure 6. High level view of an Analysis Services Database. Analysis Services databases hold data sources, data source views, dimensions, and cube structures.

Data Sources

A data source is a description of the resource containing data to be imported. Data sources allow SSAS databases to include data from remote servers, and SSAS databases can contain multiple data sources. Each data source has a name and a connection string that describes the location of the data, drivers required, and authentication credentials. Simple data sources, such as files, only require the location of the files in an accessible file system. Other database sources, such as SQL Server instances, require a server name and a database name.

Data Source Views

Data Source Views (DSVs) are an abstraction layer between data sources and the Analysis Services database. The DSV maps the data available in each data source to a set of data tables and relationships. The DSV can use existing tables from the data source, extend existing tables with computed columns, or create virtual tables constructed from SQL queries. DSV relationships can be established between any two data table objects, and these relationships are not restricted to primary key columns and foreign key columns. This extension makes it possible to define explicit partial foreign-key relationships between arbitrary columns. The importance of virtual tables and partial foreign-key relationships will be shown later.

Dimensions

Dimensions are the fundamental data structures used to organize and locate fact data. They are represented explicitly in SSAS as first class objects in contrast to a relational database, where a dimension is only implied for a given table by the existence of foreign keys. Dimensions contain two types of information: attributes and attribute relationships. Attributes are discreet values that are associated with fact data and are similar to the columns in a relational dimension table. These values are called members, and members contain Attributes are bound to columns in DSV data table objects and to the relationships defined on those objects. In addition, attributes can be related to each other using attribute relationships. One attribute

will be used as the key of the dimension, meaning that component columns from the original data source can uniquely identify any other attribute in the dimension. The key attribute is also called the granularity attribute, as it is the fundamental unit of aggregation in cubes. The key attribute and attribute relationships allow attributes to be organized into hierarchies that can later be used in queries. An important aspect of dimension attributes and associated hierarchies is that these data are ordered. This is a fundamental difference between SSAS and relational databases and is a key to understanding MDX.

Cubes

A cube is an object that contains dimensions, measure groups, and fact data. Cubes are created within SSAS databases, and a database can contain multiple cubes. Cube dimensions are references to any of the dimensions defined in the database, and can also include multiple references (uniquely named) to the same database dimension. Fact data are stored in a data structures called partitions, each measure group can have one or more partitions. Cubes are typically used to facilitate analysis of specific sets of fact data and are largely self-contained, although there are facilities for linking to other cubes. Cubes are queried using the MDX query language, described in the next section.

Cubes contain cells, one cell for the Cartesian product of all cube dimensions key attributes. Inside each cell are the measure data taken from each measure group partition. Cubes are in effect, very large multi-dimensional arrays. However, instead of allocating space for every possible value from all dimensions, SSAS implements an extremely efficient multidimensional storage engine that reduces dimension data to bit vector indexes. In addition, the engine supports fact data sparsity, meaning that space is allocated only when fact data are actually present for a given cell. This loading and compaction of data, referred to as processing, makes cube data structures effectively read only, as fact and dimension data are compiled down to minimal representations. Once a cube's dimension structure is processed, the only ways to

modify fact data are to remove partitions or add new partitions. Dimension data can be added but dimension data that references existing fact data cannot be removed.

The set of dimensions form an addressing mechanism to find any given cell in the cube. This makes dimensions similar to numbers on a number line, but tick marks are not limited to integers. A unique address in a dimension is called a member, and as mentioned previously, attributes and hence members can be organized into hierarchies. A set of members from all cube dimensions is called a tuple, and a tuple uniquely identifies a cell in the cube. Tuples can be grouped into sets called tuple sets or simply sets. When a tuple does not contain a member from one or more dimensions, the result is a set of cells called a slice.

A measure group is created from a data table in the DSV, and contains a set of measures, i.e. facts, which are stored in cells at the intersection of dimensions. The underlying DSV data table defines the data type of each measure (fact) and dimensional column. A measure is a column present in the DSV data table and explicitly linked to at least one dimension through the dimensional columns present in the fact table. The measure is also assigned an aggregation function, which is applied when a measure is projected from multiple source cells into a single result cell during an MDX query. When a cube is processed, one or more partitions matching the shape of the DSV data table are loaded into the measure group populating it with data. It is important to note that fact data are only loaded into a cell when a dimensional tuple exists defining that cell, otherwise the data are skipped.

MDX

MDX is a query language designed specifically for multi-dimensional data, and is available on several vendors' OLAP platforms. At first glance it appears similar to SQL because the main statement of the language, **SELECT**, uses some of the same keywords; however, the languages are completely different. The purpose of MDX is to operate on and to produce multidimensional result sets, i.e. sub-cubes, by selecting a set of data from a cube, applying calcu-

lations, and returning a result projected to multiple axes.

The primary statement used to retrieve data is the **SELECT** statement, an example is shown in Figure 7. MDX employs a two-pass query process. The first step, slicing, selects the set of data to be analyzed and produces a logical sub-cube that is used for the rest of the query. Slicing is controlled by the **WHERE** clause of the query, using a tuple set expression. The second pass, dicing, projects the desired results onto one or more axes. After both passes, the result cube is returned to the caller. MDX makes no distinction between result axes, making it possible to build a variety of result sets. Among the most useful of these options are matrices, which would involve complex **UNION** and/or **PIVOT** statements to achieve in SQL. It should be noted that the most common tool for viewing MDX results, SQL Server Management Studio, can only display two dimensional results.

```
WITH
      SET mystatoms AS SUBSET(FILTER(
            DESCENDANTS( [Structure].[2adr-3]
                  , [Structure].[Structure Hierarchy].[Atom] )
                  , [Measures].[distance] > 0),0,10)

      SET mystlatoms AS SUBSET(FILTER(
            DESCENDANTS( [Structure1].[2adr-3]
                  , [Structure1].[Structure Hierarchy].[Atom] )
                  , [Measures].[distance] > 0 ),0,10)

SELECT mystatoms on columns
      , mystlatoms on rows

FROM [Spatial Index]
WHERE ( Time.Step.&[5438]&[0]
        , Time1.Step.&[5438]&[0]
        , [Measures].[distance])
```

Figure 7. Example MDX Statement. This slices the cube called [Spatial Index] to include only step zero of simulation 5438. The first 10 atoms of the structure 2adr-3 are projected onto the columns and rows of the result. Listing a single measure, in this case distance, in the WHERE clause causes it to be projected into the result at the intersection of the defined rows and columns.

In addition to projecting existing attribute or fact data onto axes, MDX also supports a variety of scalar and set calculations. The results of these calculations fall into two categories, named sets and calculated members. Named sets facilitate additional calculation by creating

additional cells; calculated members provide a mechanism for creating new measures. Measures and calculated members are always evaluated in an aggregation context. For example, if a single measure from multiple cells is projected into a result cube in a single cell, the values of the measures will be aggregated according to their defined aggregation function, typically summation. A small variety of other aggregation functions are available.

Dynameomics OLAP Database Design and Implementation

The Dynameomics OLAP database is based on the dimensional model described by Simms and Daggett (Simms, 2011) and includes an implementation of spatial indexing as described in Toofanny et al. (Toofanny, 2011). The entire database structure is built using two C# command line utilities that communicate with an SSAS server using the Analysis Management Object API (AMO). The buildssascube (bcube) creates a new database on an existing SSAS server, populating it with three data source views for interacting with the primary data warehouse, a set of data sources, 9 OLAP dimensions, and one OLAP cube definition supporting 17 measure groups. The second utility, addpartitions (addp), creates data table objects in the data source view and then adds data partitions to measure groups. The entire process is driven by a simulation group identifier, which defines a group of simulations and analyses in the primary data warehouse.

Data Sources

Data in the primary data warehouse are distributed across multiple servers. The bcube utility creates three data source views to hold data table objects: DimensionDSV, WarehouseFactData, and DerivedWarehouseFactData. SQL queries are executed against the Directory database to determine the servers, databases, and table names for all data associated with user supplied simulation id. The bcube utility then adds data sources for each required server and database. Next bcube then assembles shape fact and dimension data table objects in the DimensionDSV data source view to facilitate the creation of dimensions. Small dimension tables

that do not contain simulation or structure identifiers, such as the list of dihedral angle types are copied directly into the data source view. Other dimensional tables that can be limited based on the simulation group id are built as named queries. The time dimension is a special case, because in the main warehouse time is not broken out as its own dimensional table. Instead of using a table, the named query facility is used to create a view based on a SQL SELECT statement. The FROM clause does not reference a table, instead it uses a SQL CLR user defined table function that generates step values at up to a specified simulation length and granularity, defaulting to 65 ns at 0.1ps granularity. Finally, the bcube utility builds foreign key and partial foreign key relationships between fact and dimension objects in the DSV. These relationships are required for dimensions to be created and built.

Dimensions

After the primary data source view is created and populated with shape tables, named queries, and relationships, dimension structures are created. The primary dimensions of the model are Simulation, SimulationStructures, Structures, and Time. Analysis specific dimensions are DihedralAngle, SecondaryStructure, and PhiPsiStructureState. Also included are the SpatialIndex and SpatialIndexNeighbor dimensions, which are used to implement spatial hashing. Dimensions and hierarchies are illustrated in Figure 8.

The Simulation and SimulationStructures dimensions model parameters that do not vary over the course of the simulation. The Simulation dimension contains multiple hierarchies that facilitate easy assembly of simulation sets based on temperature, run, and other attributes. The Time dimension models simulation time and contains a single hierarchy allowing selection of time by step or picosecond. SimulationStructures uses a single hierarchy, but the level structure is inverted—simulation is at the top. Three additional dimensions, DihedralAngle, SecondaryStructure, and PhiPsiStructure support analysis specific attributes specific to the dihedral, DSSP, PhiPsi analyses, respectively. Two additional dimensions are included specifically to support

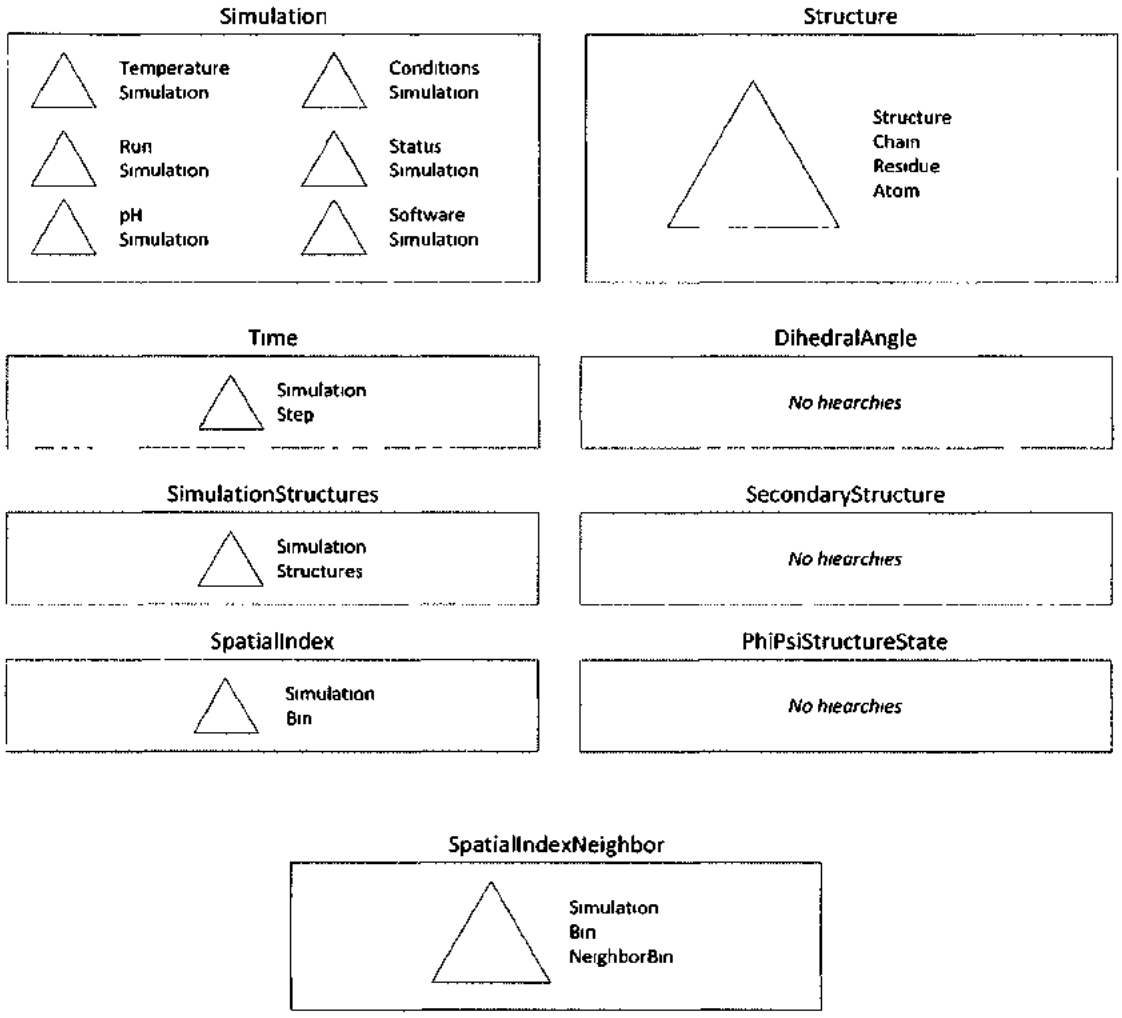


Figure 8. Dimensions and hierarchies. Discreet data in an OLAP database are organized into dimensions as attributes. Attribute values are called members, and attributes can be organized as hierarchies. For example, attributes in structure are organized into PDB structure, chain, residue and atom.

spatial indexing: SpatialIndex and SpatialIndexNeighbor. The complete list of all dimension and their attributes are in Table 10.

Cube Design

The cube design consists of 17 measure groups and 78 distinct measures. Measure groups and measures are detailed in Table 11. Source data for dimensions are taken from the primary data source view DimensionDSV. Data for measure groups are loaded from partitions,

Table 10. Dimensions and attributes. The attributes associated with each of the 9 database dimension are listed along with their definition. Attributes marked with "(key)" are the key attribute, defining the granularity of the dimension.

Simulation:	
Simulation (key)	Simulation identifier from the Dynameomics Data Warehouse
Simulation Name	Simulation name
Simulation Comment	Optional comment
Time Step	Conversion factor to calculate time in picoseconds from a simulation step (units = ps)
Random Seed	Integer value used to seed random number generator prior for initial velocity assignment
Density	Solvent density (g/ml)
Initial Box x	Initial box x dimension (Å)
Initial Box y	Initial box y dimension (Å)
Initial Box z	Initial box z dimension (Å)
C Scale	Charge scaling factor for electrostatic potential
A Scale	Scaling factor for 12/6 attractive and 12/6 repulsive terms of the Lennard Jones potential
Simulation Temperature	Simulation temperature (Kelvin)
Run Number	Simulation run number
Simulation pH	Simulation acidity environment (low, medium, high)
Simulation Software	Molecular dynamics simulation engine used for this simulation
Software Description	Full name of the simulation software used for this simulation
Software Comment	Optional comment for the simulation software used to create this simulation
Simulation Status	Simulation status
Status Description	Status description
Simulation Conditions	Other simulation conditions
Time:	
Step (key)	Simulation step
Time	Simulation time (picoseconds)
Simulation	Simulation identifier
Simulation Name	Simulation name
SimulationStructures:	
Structure Instance (key)	Instance identifier for structure within this simulation
Structure	Structure identifier
Simulation	Simulation identifier
SpatialIndex:	
Bin (key)	1 dimensional bin index within periodic box
Simulation	Simulation identifier
SpatialIndexNeighbors:	
NeighborBin (key)	1 dimensional bin index of adjacent bin within periodic box
Bin	1 dimensional bin index within periodic box
Simulation	Simulation identifier

Table 10, continued

Structure:	
Atom (key)	Atom within a structure
Atom Number	Identifier within structure for this atom
Atom Type	Atom type code from the Chemical Component Dictionary
Main Chain	Atom is not part of a side chain
Heavy Atom	Atom is not hydrogen
Residue	Protein Data Bank residue abbreviation
Residue ID	Database residue id
Residue Number	Protein Data Bank residue number
ICode	Protein Data Bank insertion code
Built	This residue was added to the original PDB structure
MMPL Name	The official MMPL name of this residue
Residue Name	Chemical name for residue
Residue Abbreviation	The official single character abbreviation for this residue
Polar	Polar charge
Non-Polar	Non-polar charge
Acidic	Acidic
Basic	Basic
Chain	Protein Data Bank chain identifier
Structure	A collection of residues and atoms
PDB4	Protein Data Bank PDB code
Structure Description	Optional description for this structure
Resolution	Optional resolution for X-Ray crystallographic structures
Structure Type	Structure determination method
Structure Determination Method	Structure determination method, detailed
DihedralAngle:	
Angle (key)	Dihedral angle name
Angle Symbol	Dihedral angle symbol
SecondaryStructure:	
SS Char (key)	Secondary structure abbreviation
SS Description	Secondary structure definition
SS Comment	Secondary structure comment
PhiPsiStructureState:	
Structure State (key)	Phi Psi analysis structure state prediction

created by the addpartitions utility (addp). After the initial empty cube structure is created, addp is run to locate all fact tables associated with the specified simulation group identifier. Once located, addp builds data table objects by measure group, adding appropriate data tables

in the WarehouseFactData DSV for pre-calculated measures and named queries into the DerivedWarehouseFactData DSV for on-the-fly analyses. As each data table object is created it is added to the partition group collection of its measure group.

The cube contains all 9 database dimensions, and creates an additional 3 role-playing dimensions (also known as shadow dimensions) on Time, Structure, and SimulationStructures. Measure groups are linked to all 12 defined cube dimensions, as detailed in Table 12. The majority of measure group relationships are regular, meaning that fact tables participate in a many-to-one relationship with their associated dimensions. The one exception is the Coordinate measure group, which participates in a reflexive many-to-many relationship used to implement spatial indexing for the efficient calculation of contact distances. The SpatialIndex and Spa-

Table 11. Measure group definitions. Measure groups hold fact data in partitions. The complete set of measure groups and their associated fact data columns in the data source view are defined here. The fact tables are placeholders for the actual fact data which comes from partitions.

Box (Size of periodic box, 4 measures)		
box_count	Box row count	FactBox.x_size
x_size	x component of periodic box size (Å)	FactBox.x_size
y_size	y component of periodic box size (Å)	FactBox.y_size
z_size	z component of periodic box size (Å)	FactBox.z_size
Congen (CONGENEAL structural dissimilarity score, 2 measures)		
congen_count	Congen row count	FactCongen.dissimilarity_score
dissimlarity_score	CONGENEAL structural dissimilarity score (http://dx.doi.org/10.1002/pro.5560020603)	FactCongen.dissimilarity_score
Contact (Periodic contact summary, 4 measures)		
contact_count	Contact row count	FactContact.total
total_contacts	Total of native and nonnative contacts	FactContact.total
native_contacts	Number of contacts between residues that are three or more residues apart in sequence	FactContact.native
non_native_contacts	Number of contacts between adjacent residues	FactContact.non_native
Dihed (Dihedral angle measurements, 2 measures)		
dihed_count	Coordinate row count	FactDihed.dh_angle
dh_angle	Dihedral angle (degrees)	FactDihed.dh_angle
DSSP (Dictionary of Secondary Structure Prediction (http://dx.doi.org/10.1002/bip.360221211), 2 measures)		
dssp_count	DSSP row count	FactDSSP.ss_id
ss_id	Secondary Structure Code	FactDSSP.ss_id

Table 11continued.

FDSASum (Fine Detail Structure Analysis Summary, 7 measures)		
fdsasum_count	FDSA Summary row count	FactFDSASum.intra_hbonds
intra_hbonds	Intra-molecular hydrogen bond: max separation is 2.6 Å D-H <- 2.6 Å -> A where D is donor and A is acceptor 45.0 deg. > theta > 135.0 deg. is the angular range theta is angle D-H-A where D is donor and A is acceptor	FactFDSASum.intra_hbonds
intra_hphobs	Intra-molecular hydrophobic contacts: max separation is 5.4 Å CHx <-5.4 Å-> CHx where x <= 1	FactFDSASum.intra_hphobs
intra_others	Intra-molecular other contacts: 4.6 Å is the max distance between heavy atoms must not be a valid hphob or valid hbond	FactFDSASum.intra_others
inter_hbonds	Inter-molecular hydrogen bond: max separation is 2.6 Å D-H <-2.6 Å-> A where D is donor and A is acceptor 45.0 deg. > theta > 135.0 deg. is the angular range theta is angle D-H-A where D is donor and A is acceptor	FactFDSASum.inter_hbonds
inter_hphobs	Inter-molecular hydrophobic contacts: max separation is 5.4 Å CHx <-5.4 Å-> CHx where x <= 1	FactFDSASum.inter_hphobs
inter_others	Inter-molecular other contacts: 4.6 Å is the max distance between heavy atoms must not be a valid hphob or valid hbond	FactFDSASum.inter_others
Forces (Instaneous forces, 4 measures)		
forces_count	Forces row count	FactForces.x_force
x_force	Instantaneous force, x component (amu · Å/s^2)	FactForces.x_force
y_force	Instantaneous force, y component (amu · Å/s^2)	FactForces.x_force
z_force	Instantaneous force, z component (amu · Å/s^2)	FactForces.z_force
PhiPSi (Phi/Psi angles (see Dihedral), 4 measures)		
phpsi_count	PhiPsi row count	FactPhiPsi.phi
phi	Phi angle (degrees)	FactPhiPsi.phi
psi	Psi angle (degrees)	FactPhiPsi.psi
jcoup	Instantaneous Jcoupling constant from Karplus relation	FactPhiPsi.Jcoup
Radgee (Radius of Gyration, 3 measures)		
radgee_count	radgee row count	FactRadgee.radgyr
radgyr	Radius of Gyration (Å), aggregation function: Max	FactRadgee.radgyr
end2end	End to end distance (Å), aggregation function: Max	FactRadgee.end2end

Table 11continued.

RMSD (Root-means-square deviation from starting structure, 3 measures)		
rmsd_count	radgee row count	FactRMSD.rmsd
rmsd	Root-means-square distance from starting structure, aggregation function: AverageOfChildren	FactRMSD.rmsd
rmsd100	Normalized root-means-square distance from starting structure, aggregation function: AverageOfChildren	FactRMSD.rmsd100
RMSF (Root-means-square fluctuation, 4 measures)		
rmsf_count	radgee row count	FactRMSF.rmsfwavg
rmsfwavg	Root-means-square fluctuation, windowed average	FactRMSF.rmsfwavg
rmsfwstd	Root-means-square fluctuation, windowed standard deviation	FactRMSF.rmsfwstd
rmsf	Root-means-square fluctuation	FactRMSF.rmsf
S2 (S2 Order parameters, 4 measures)		
s2_count	S2 row count	FactS2.x
s2_x	S2 parameter x	FactS2.x
s2_y	S2 parameter y	FactS2.y
s2_z	S2 parameter z	FactS2.z
SASA (Solvent Accessible Surface Area, 10 measures)		
sasa_count	SASA row count	FactSASA.main_chain
main_chain	Solvent accessible surface area (main chain)	FactSASA.main_chain
side_chain	Solvent accessible surface area (side chain)	FactSASA.side_chain
polar	Solvent accessible surface area (polar residues)	FactSASA.polar
non_polar	Solvent accessible surface area (nonpolar residues)	FactSASA.non_polar
mc_polar	Solvent accessible surface area (main chain polar residues)	FactSASA.mc_polar
mc_non_polar	Solvent accessible surface area (main chain nonpolar residues)	FactSASA.mc_non_polar
sc_polar	Solvent accessible surface area (side chain polar residues)	FactSASA.sc_polar
sc_non_polar	Solvent accessible surface area (side chain nonpolar residues)	FactSASA.sc_non_polar
total	Solvent accessible surface area (total)	FactSASA.total
VCont (Verbose contacts summary, 18 measures)		
vcont_count	VCont row count	FactVCont.nat_atom_mc_mc
nat_atom_mc_mc	Native atom-atom main chain-main chain contacts	FactVCont.nat_atom_mc_mc
nat_atom_mc_sc	Native atom-atom main chain-side chain contacts	FactVCont.nat_atom_mc_sc
nat_atom_total	Native atom-atom contacts, total	FactVCont.nat_atom_total
nnat_atom_mc_mc	Non-native atom-atom main chain-main chain contacts	FactVCont.nnat_atom_mc_mc
nnat_atom_mc_sc	Non-native atom-atom main chain-side chain contacts	FactVCont.nnat_atom_mc_sc
nnat_atom_sc_sc	Non-native atom-atom side chain-side chain chain contacts	FactVCont.nnat_atom_sc_sc

Table 11continued.

VCont (Verbose contacts summary, 18 measures, continued)		
nnat_atom_total	Native atom-atom contacts, total	FactVCont.nnat_atom_total
nat_res_mc_mc	Native residue-residue main chain-main chain contacts, aggregation function: Sum	FactVCont.nat_res_mc_mc
nat_res_mc_sc	Native residue-residue main chain-side chain contacts, aggregation function: Sum	FactVCont.nat_res_mc_sc
nat_res_sc_sc	Native residue-residue side chain-side chain contacts, aggregation function: Sum	FactVCont.nat_res_sc_sc
nat_res_total	Native residue-residue contacts, total, aggregation function: Sum	FactVCont.nat_res_total
nnat_res_mc_mc	Non-native residue-residue main chain-main chain contacts	FactVCont.nnat_atom_mc_mc
nnat_res_mc_sc	Non-native residue-residue main chain-side chain contacts	FactVCont.nnat_res_mc_sc
nnat_res_sc_sc	Non-native residue-residue side chain-side chain contacts	FactVCont.nnat_res_sc_sc
nnat_res_total	Non-native residue-residue contacts, total	FactVCont.nnat_res_total
tot_atm	Total atom contacts	FactVCont.tot_atm
tot_res	Total residue contacts	FactVCont.tot_res
Bins (Bridge measure group for Spatial Index Support, 1 measures)		
neighbors	Bridge table, aggregation function: Count	FactBins.hash3d_index_neighbor
Coord (Atomic coordinates, 4 measures)		
coord_count	Coordinate row count	FactCoord.x_coord
x	x coordinate	FactCoord.x_coord
y	y coordinate	FactCoord.y_coord
z	z coordinate	FactCoord.z_coord
ContactDistance (Heavy atom contact distance, 2 measures)		
contactdistance_count	ContactDistance row count	FactContactDistance.distance
distance	Distance (Å)	FactContactDistance.distance

tialIndexNeighbor dimensions are built from a single fact table structure in the primary data warehouse. To avoid duplicating data, the named query facility of the data source view is used to produce views that serve as the two tables required by SSAS to make a many-to-many relationship (DimSpatialIndex, DimSpatialIndexNeighbor). The Bins measure group is linked to SpatialIndex and SpatialIndexNeighbors using regular relationships, similarly the Coord measure group is linked to the SpatialIndex dimension. The Coord measure group is then linked to

Table 12. Measure groups and relationships to cube dimensions. Regular many-to-one relationships are denoted by R, N indicates no relationship. M indicates a many-to-many relationship.

	Box	Congen	Contact	Dihed	DSSP	FDSA Sum	Forces	PhiPSi	Radgee	RMSD	RMSF	S2	SASA	VCont	Bins	Coord	ContactDistance
Simulation	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
Time	R	R	R	R	R	R	R	R	R	R	N	R	R	R	N	R	R
Time1	R	R	R	R	R	R	R	R	R	R	N	R	R	R	N	R	R
SimulationStructures	N	R	R	R	R	R	R	R	R	R	R	R	R	R	N	R	R
SimulationStructures1	N	R	R	R	R	R	R	R	R	R	R	R	R	R	N	R	R
Structure	N	R	R	R	R	R	R	R	R	R	R	R	R	R	N	R	R
Structure1	N	R	R	R	R	R	R	R	R	R	R	R	R	R	N	R	R
DihedralAngle	N	N	N	R	N	N	N	N	N	N	N	N	N	N	N	N	N
PhiPsiStructureState	N	N	N	N	N	N	N	R	N	N	N	N	N	N	N	N	N
SecondaryStructure	N	N	N	N	R	N	N	N	N	N	N	N	N	N	N	N	N
SpatialIndex	N	N	N	N	N	N	N	N	N	N	N	N	N	N	R	R	N
SpatialIndexNeighbors	N	N	N	N	N	N	N	N	N	N	N	N	N	N	R	M	N

SpatialIndexNeighbor through the Bins measure group, forming a many-to-many relationship.

Storage and Calculation Performance Analysis

Four SSAS databases, each containing a single cube, were created and processed on a Dell server as described in Table 13 using the bcube and addp utilities described earlier. Each cube structure includes a different combination of spatial indexing dimensions and pre-computed heavy atom contact distances. All four cubes were loaded with 51ns coordinate trajectories and analyses for the set of proteins described by Toofanny et al. (Toofanny, 2011). The

Table 13. Test server configuration.

Hardware	Description
Server	Dell R710
Processors	Dual Intel Xeon X5650s (x64 Hex Core)
Memory	48 GB
Storage	H700 Integrated RAID SAS Disk Controller
System Disks	136 GB on two 15K RPM 150GB SAS disks , RAID 1 (Mirrored)
Data Disks	7,450 GB on six 7200 RPM 2TB SAS disks, RAID 0 (Striped)

bcube and addp utilities together take around 2 minutes to run, creating the complete database structure including data and data source views, dimensions, and cubes. After the structure is created, the database must be processed in order to load data into measure group partitions. Although processing time is dependent on the load of source servers providing data, complete processing time not including the ContactDistance measure group was consistently 45 minutes. When the ContactDistance measure group is included, processing time jumps to just over 6.5 hours to complete.

After processing was completed, the disk space used by each cube and its measure group partitions were measured. These results, as well as the size of the original SQL data, are summarized in Table 14. SSAS data structures are significantly more efficient than uncompressed SQL tables, showing an 80% reduction in space required for cubes built without the ContactDistance measure group. It is interesting that even when ContactDistance data are added, the total cube size is still 20% smaller than the raw SQL tables which do not contain contact distances.

After all cubes were processed, a set of timing tests were performed on each cube variant to determine how quickly heavy atom contact distances could be determined. An initial attempt was made to duplicate timing results described by Toofanny using a lookup query and a calculation query. The same limitations of 1,000 (1ns) and 51,000 frames (51ns) were attempted, but neither returned successfully. The raw result set was either too large to processed by the client library or the server failed with an out of memory exception preparing the result set.

Table 14. Storage analysis. Four SSAS databases each containing a single cube were created and processed. When processing completed, storage space was measured and is summarized below. It is interesting to note

		Cube Storage (MB)				
Measure Group	**Rows**	**+SI +CD**	**+SI -CD**	**-SI -CD**	**-SI +CD**	**SQL Storage (MB)**
Bins	399,033	4.92	4.922			8.94
Box	574,408	30.14	30.137	30.137	30.137	16.83
Congen	574,408	24.92	24.918	24.918	24.918	18.39
Contact	574,408	25.41	25.412	25.412	25.412	22.91
ContactDistance		49,801.19		49,935.87		
Coord	1,649,285,467	34,439.45	34,439.45	31,691.52	31,691.52	197414.92
Dihed	614,699,496	5,370.12	5,370.12	5,370.12	5,370.12	20226.15
DSSP	105,263,035	835.61	835.36	836.22	832.915	3066.70
FDSASum	131,995	5.69	5.688	5.688	5.688	6.52
PhiPsi	104,006,194	1,292.96	1,292.96	1,292.96	1,292.96	5537.63
Radgee	574,408	25.62	25.62	25.621	25.621	20.70
RMSD	574,408	25.52	25.52	25.52	25.52	20.72
RMSF	2,033	0.03	0.03	0.03	0.03	0.22
SASA	105,155,010	2,598.43	2,598.43	2,598.43	2,598.43	6751.23
VCont	574,397	34.06	34.06	34.06	34.06	57.95
Cube Dimension Data		1,664.99	1,664.68	1,619.00	1,619.68	
TOTAL		96,179.05	46,377.31	93,516.52	43,577.01	233169.81
Space Savings vs. SQL			80%		81%	100%

The lookup query was limited to include only 100 simulation frames and was run against the two cubes containing the ContactDistance measure group. Three runs for each protein were completed, and the results are shown in Figure 9. Then a set of contact calculations were attempted, using cubes that specifically excluded the ContactDistance measure group and either using or excluding the SpatialIndex dimensions. These queries had to be repeatedly reduced in size, finally stopping at a mere 5 frames in order to avoid memory errors in the client library. The partial results of these timings are shown in Figure 10. Although it is clear that finding desired data is faster than attempting to do the calculation on-demand, the behavior of the calculation query is unexplained. First query plan appears to change for proteins larger than 1okt (85 residues) and again beyond 1hgu (189 residues). However, the bigger issue is highlighted by the blue lines shown both Figures 9 and 10. The lower line is the slowest execution time taken by a 1,000 frame contact calculation using spatial indexing for the largest protein in the

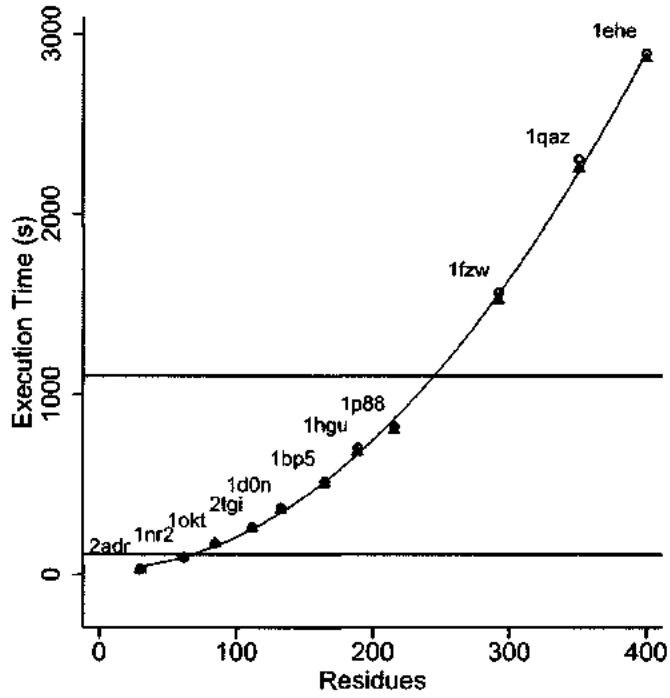


Figure 9. MDX lookup query execution times. Execution time of a lookup of data from the ContactDistance measure, limited to 100 frames. The lower blue line is the slowest recorded time for SQL Server to produce 1000 frames of contact data for 1ehe, the largest protein in the test set using spatial indexing. The upper blue line is the for the SQL execution time for 1ehe not using spatial indexing.

test set (1ehe, 400 residues) on SQL Server using the same host machine. The upper blue line is the slowest time for 1ehe not using spatial indexing.

Discussion

The ability to store large, multi-dimensional scientific data sets efficiently and without have to translate them into two-dimensional tables certainly has great appeal. Interacting with those result sets using a multi-dimensional query language also opens up interesting possibilities for data exploration. Consider the formulation of a query to present a matrix of distances between CA carbons on columns and all other carbons on rows. Constructing this query in SQL would require use a complex PIVOT or JOIN. Unfortunately, the severe penalty for performing calculations or even just retrieving large data sets using MDX severely limits how it can be used and even if it will be a viable option for analysis of large data sets.

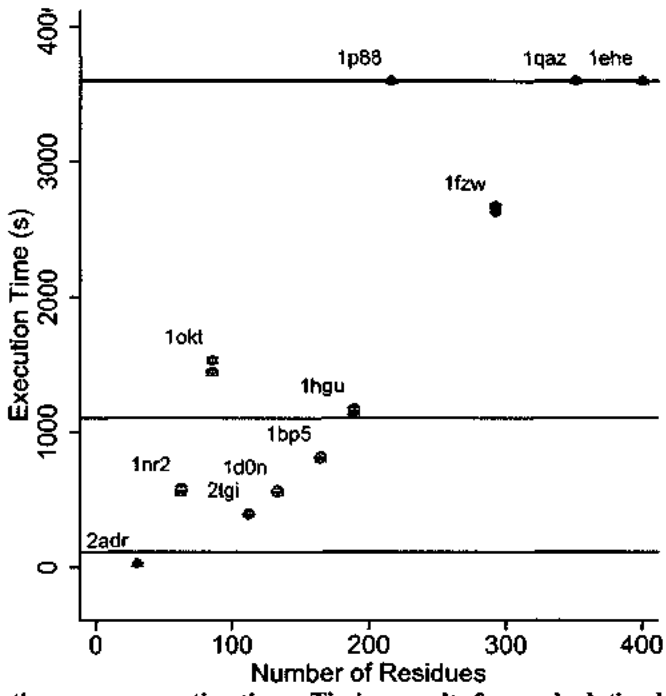


Figure 10. MDX calculation query execution times. Timing results for a calculating heavy atom contact distance in MDX, limited to 5 frames. Blue lines are as described in Figure 9. The query is extremely slow for all three cube variants tested, and the application of spatial indexing dimension has no effect on execution time.

There are many potential causes for the lack of performance in this model, some of which could be addressed through a different query design. For example, the spatial indexing implementation in SQL significantly reduces query time over that of exhaustive calculation. In this OLAP model, it appears to have no significant effect, indicating that the MDX query optimizer may incorrectly be calculating distances when it could utilize the SpatialIndex dimensions to avoid them. Another possibility is that cube model itself is less than optimal. It was designed to exactly mimic the rich set of data stored in the main data warehouse, exhaustively representing every dimensional property. Reducing the size and complexity of attributes within the existing dimensions could speed things up. However, the low performance could also be an issue with underlying XMLA protocol (Microsoft Corporation, 2002) used by SSAS to return result sets to clients. This protocol is XML based and contains a rich language for describing result values, dimensionality, and even display attributes such as text color to an application. Returning large numbers of values in a single result set may simply be impractical

with XMLA.

Conclusions

SSAS is a multidimensional database that includes a multidimensional query language called MDX. The fundamental multi-dimensional storage engine stores data very efficiently, achieving an 80% reduction in data size as compared to the source data stored in SQL Server. In addition, the MDX language offers great flexibility in assembling result sets, without out the fundamental column and row restrictions found in SQL. Unfortunately these valuable features are overshadowed by serious deficiencies in handling large result sets as well as poor query performance.

Chapter 4: Beyond the Relational Model: 3D Spatial Hashing

One of the most attractive features of molecular dynamics simulations of proteins is the ability to monitor the atomic interactions between pairs of atoms across the protein over time. Atomic contacts, when combined with experiment, provide insight into protein folding, dynamics, and function. The calculation of contacts is non-trivial in MD simulations, as the possible number of contacts increases exponentially with the number of amino acids. The resultant data of such a calculation is often larger in size than the original coordinate data from which it was derived. In this paper we describe the implementation of a spatial indexing algorithm, in our multi-terabyte MD simulation database (Dynameomics), to significantly speed up the discovery of atomic interactions in a simulation. Spatial indexing, also known as spatial hashing is a method that divides a finite 3 dimensional space into regular sized bins and applies an index to each bin and hence it can be used to decrease the time of calculating the distance of nearest neighbor objects in 3 dimensional space. Since, the calculation of contacts is an often used computationally demanding calculation in the simulation field; we also use this as the basis for testing compression of data tables. We investigate the compression of coordinate tables with different permutations of data and index compression within MS SQL SERVER 2008 R2. The effect of compression of tables on query times is also investigated.

Our implementation of spatial indexing speeds up the calculation of contacts over a 1ns window by between 14 and 90%. For a 'full' simulation trajectory (51 ns) spatial indexing has negative to no effect on the two smallest proteins, however the calculation speed up is between 31 and 81% for the remaining simulations. Testing all permutations of data and index compression revealed there was no significant difference in the total execution time for all the proteins in our test set. The greatest compression (~36%) was achieved using page compression on both the data and indexes.

We implement a spatial indexing scheme in our simulation database that significantly decreases the time taken to calculate atomic contacts opening the door for rapid cross simulation

analysis and on the fly calculation and visualization of contacts. Using page compression across the data and index for the atomic coordinate tables will save ~36% of space without any significant decrease in calculation time.

Introduction

Many laboratories use molecular dynamics (MD) simulations to study the dynamic and structural properties of proteins. MD simulations provide atomic level resolution of the protein and its surrounding solvent environment; there are currently no experimental techniques that can provide this level of detail. The key to a protein's dynamics across time is the multitude of atomic interactions that occur between bonded and nonbonded atoms. Fluctuations in these contacts in the protein dictate the conformations accessible to the protein and its overall behavior. The dynamics of a protein are key to understanding protein function (Karplus, 2005), protein folding and misfolding(Chiti, 2006; Fersht, 2002).

Our lab has recently undertaken and completed a large scale project, named Dynameomics, in which we have simulated the native states and unfolding pathways of representatives of essentially all autonomous protein fold families(van der Kamp, 2010). These fold families, or metafolds, were chosen based on a consensus between the SCOP, CATH and DALI domain dictionaries, which we call a consensus domain dictionary (CDD) ((Day, 2003; Schaeffer, 2011a). For our 2009 release set there are 807 metafolds, representing 95% of the known autonomous domains in the Protein Data Bank (PDB). The coordinates of the MD simulations and standard analyses are loaded into a relational database. Our Dynameomics database is implemented using Microsoft SQL server with the Windows Server operating system (see (Simms, 2008) for a more detailed description). For our Dynameomics simulations we run one native state simulation, and at least 5 thermal unfolding simulations. In order to explore the dynamics and folding in these simulations we often calculate the contacts between pairs of atoms. The calculation is non-trivial as all possible pairs of atoms in the system must be evaluated. The

average number of protein atoms in our Dynameomics set simulations is 2150 with the smallest system consisting of 494 protein atoms and the largest of 6584 protein atoms. This problem has been well studied and is also known as the nearest neighbor search problem (Clarkson, 2005). As the atoms in our system are in motion, all pairs of atoms need to be re-evaluated for each frame of the simulation, so in the case of a 51 ns native state simulation, we have 51,000 frames of pairs of contacts to evaluate. Whilst ad hoc one-off calculations of contacts are possible, calculating contacts for a large number of simulations, in a project like Dynameomics, without any acceleration method is simply not possible.

Spatial indexing overview

Spatial indexing is an often commonly used method by programmers of 3D video games, in which collision between particles/objects are detected (Lefebvre, 2006). In order to accelerate the detection of collisions the 3D space is split into many smaller 3D bins, which are often uniform in size. Each of the bins is then given an index and the particles/objects in the system are rapidly evaluated to determine which of the indexed bins it falls in. Collisions can then be detected by evaluating only those particles/objects in the same or immediately adjacent neighboring bins.

In our MD simulation engine (Beck, 2000-2011) we already implement a spatial indexing (hashing) algorithm for the calculation of nonbonded terms (Beck, 2004). Our MD simulations are carried out in a periodic box of water molecules where the protein is solvated in the center of the box; this is conceptually similar to an orthorhombic unit cell in crystallography. Since the dimensions of the periodic box are constant throughout the simulation, one can create a spatial hash that is consistent throughout the simulation. In practice we split our periodic box into smaller bins of at least 5.4 Å since this the maximum distance we consider a pair of atoms to be in contact (Beck, 2008) (Figure 11A). Each bin is then assigned an unique integer. We then only evaluate the pairs of atoms in the current bin and the immediately adjacent 26 bins

(Figure 11B), which are calculated using simple algebra.

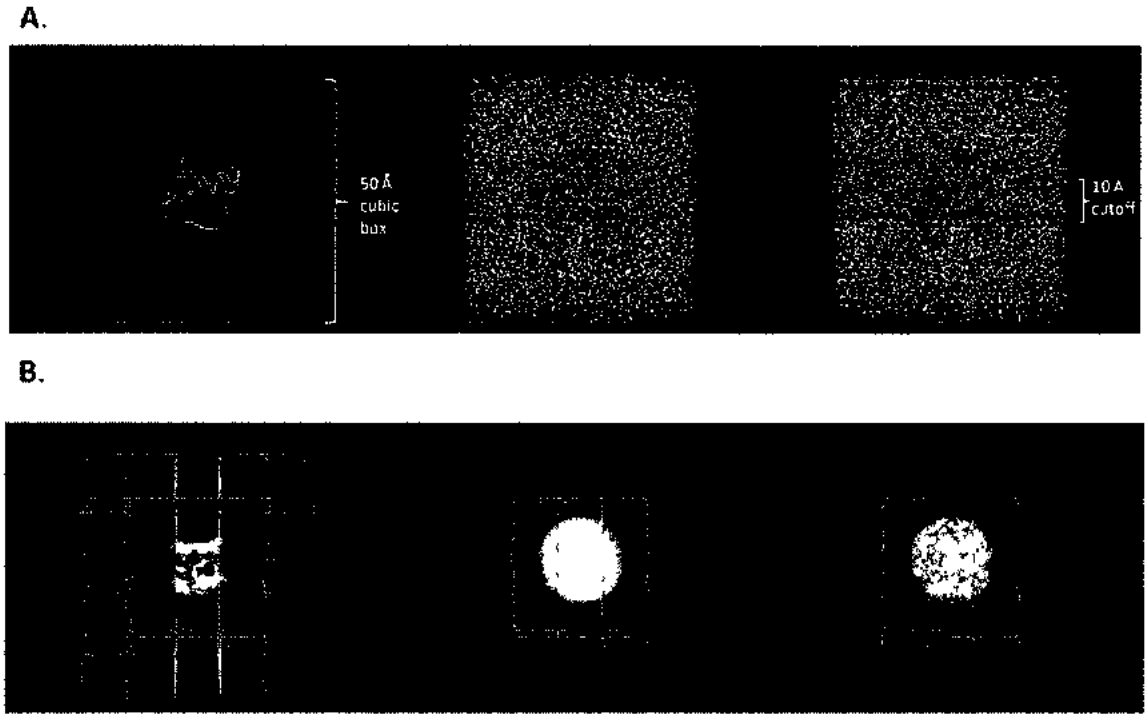


Figure 11. Illustration of spatial binning within a periodic box. The protein (1enh, the engrailed homeodomain) is simulated in a periodic box of water molecules with dimensions of 50 Å (A). The periodic box is split into smaller boxes of 10 Å, these are the 3 dimensional bins. Each bin is assigned an index and every atom at every time point will have associated X,Y, Z coordinates and a bin index. Illustration describing the evaluation of adjacent bins (B). Finding the distance to neighboring atoms within a prescribed cutoff is reduced to only evaluating euclidean distance between an atom and other atoms in the same bin or the 26 surrounding bins.

Results

We investigated the effect of using spatial indexing in the simulation coordinate table to accelerate the discovery of atomic contacts between pairs of atoms. We compared the execution times for the heavy atoms contact query for 1 ns (1000 frames) of each of our 11 representative metafolds (Figure 12, Table 15) and Figure 13 shows the average execution time for the 1 ns heavy atom contact queries with and without the use of spatial indexing. Table 16

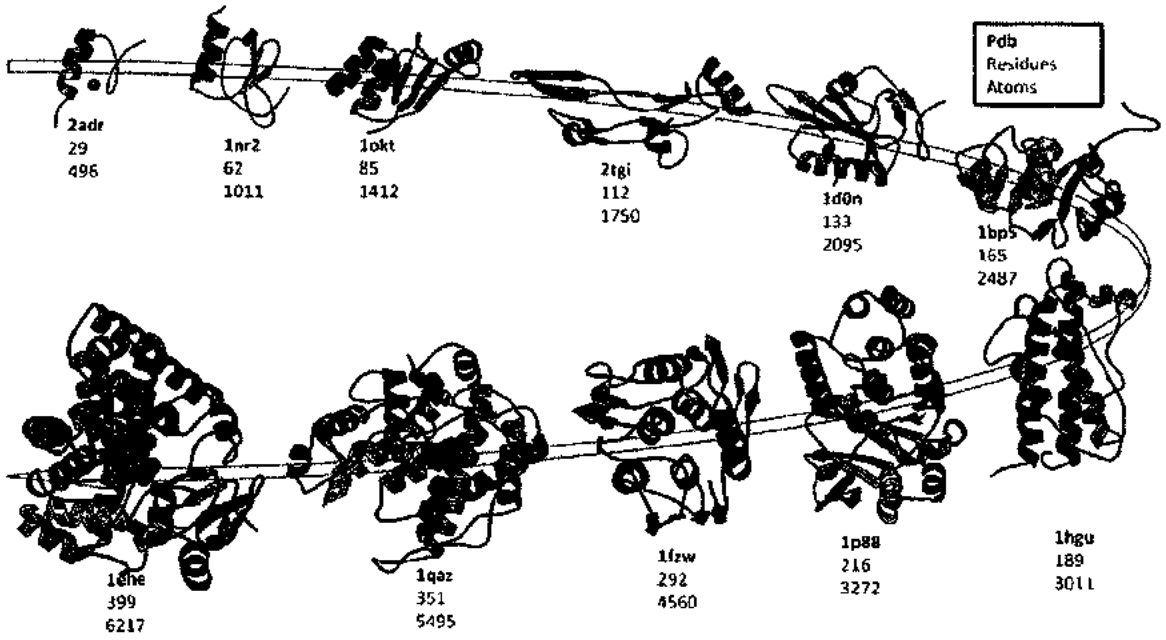


Figure 12. 11 metafolds representative of sequence length in Dynameomics. The proteins are ordered by the number of amino acid residues in each protein. See also, Table 1.

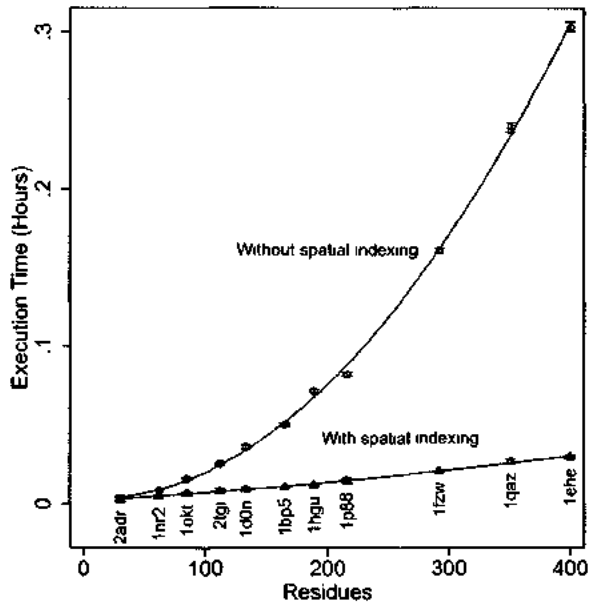


Figure 13. Contacts query execution times. Comparison of heavy atom contacts query with and without spatial indexing for 11 metafold representatives over 1ns. With no spatial indexing (circles) applied the calculation of heavy atom contacts over 1 ns (1000 frames) takes an average of ~20 minutes (n=6) for the largest protein 1ehe. For the smallest protein, 2adr, the average time taken is around 10 seconds. When spatial indexing is applied (triangles) the is a dramatic decrease in execution time for 1ehe from ~20 minutes to an average of 1 minute 46 seconds. There is almost no change in execution time for 2adr since it is an extremely small protein, spatial indexing has little effect.

Table 15. Test set definition. The test set consisting of 11 representative proteins taken from the Dynameomics project.

PDB4	Name	Residue Range	residues	protein atoms
2adr	Domain of Adr1 DBD from S. cerevisiae	102-130	29	496
1nr2	Thymus and activation-regulated chemokine	8-69	62	1011
1okt	Glutathione S-transferase	1-85	85	1412
2tgi	Domain of transforming growth factor-beta 2 (TGF-B2)	1-112	112	1750
1d0n	Horse plasma gelsolin	27-159	133	2095
1bp5	Domain of serum transferrin	82-246	165	2487
1hgu	Human growth hormone	2-190	189	3011
1p88	3-phosphoshikimate 1-carboxyvinyltransferase	25-240	216	3272
1fzw	Monomer of glucose-1-phosphate thymidylyltransferase	2-293	292	4560
1qaz	Alginate Lyase A1-III	4-354	351	5495
1ehe	Cytochrome P450nor	5-403	399	6217

shows in detail the comparison of the execution times with and without the spatial index. The results show that for 10 out of the 11 cases that we achieved a significant decrease in execution time when using spatial indexing. As expected, query times decreased as the number of distance calculations is significantly reduced ($p < 0.05$) for 11 metafolds. For one metafold (2adr) the

Table 16. Comparison of average execution times by protein. All observations indicated that the spatial index optimized query ran faster than its non-optimized counterpart and except in the case of 2adr, that observed speed improvement was statistically significant ($p < 0.0001$).

PDB	[1]Time (s)	[2]Time (s)	[3]Δ (s)	95% CI	p-value	Note
2adr	11.6	10	1.6	(-5.34, 8.55)	0.3092	No significant change
1nr2	29.3	16.7	12.6	(11.18, 14.10)	0	Significantly faster
1okt	56.1	23.7	32.5	(30.85, 34.09)	0	Significantly faster
2tgi	89.7	28.2	61.5	(56.53, 66.38)	0	Significantly faster
1d0n	127.9	32.6	95.3	(90.63, 99.99)	0	Significantly faster
1bp5	180.2	37.5	142.7	(138.40, 147.03)	0	Significantly faster
1hgu	256.4	42.4	214	(210.81, 217.25)	0	Significantly faster
1p88	294.4	51.7	242.7	(236.38, 248.92)	0	Significantly faster
1fzw	578.3	73.4	504.8	(498.92, 510.77)	0	Significantly faster
1qaz	860.4	95.1	765.3	(753.80, 776.85)	0	Significantly faster
1ehe	1091.8	105.6	986.2	(974.65, 997.79)	0	Significantly faster

[1]Average time not using spatial index, N=6. [2]Average time using spatial index, N=6. [3]mean difference.

heavy atom contact execution time did not significantly change ($P>0.05$) when using spatial indexing. Investigating further, we found that for 2adr that spatial indexing had little effect since 2adr is a very small protein. 2adr has a an average radius of gyration of 8.5 Å and since each spatial bin has the minimum dimensions of 5.4 Å by 5.4 Å by 5.4 Å , the entire protein is covered by only a small number of bins. In this instance there is no significant difference in the number of pairs of atoms considered when the query uses the spatial indexing. Also, since the execution time is so short, it is likely that the cost of selecting out only the immediately adjacent bins instead of joining all atomic coordinates is more apparent.

The significant decrease in execution time for the calculation of atomic contacts is important for 3 main reasons. First, the reduction in execution time enables us to calculate contacts in a tractable time frame for large proteins, considering the largest fold representative in our Dynameomics set, 1ehe which contains 399 residues, the average execution time reduced from around 18 minutes to just under 1 minute and 45 seconds. With such a tractable execution time, we can perform rapid ad hoc queries in our database, which is extremely useful in an exploratory sense and enables us to ask and quickly answer questions about the atomic interactions in a simulation. Second, the query execution time is quick enough to enable us to perform large-scale multi-simulation analysis. For example, if we wanted to find all the long-range contacts in the denatured state of the 807 metafold representatives and look for patterns, it would not be difficult to execute multiple contacts queries across multiple servers to return back that result rapidly. Third, since the calculation can be run in a short period of time, this analysis could be performed on the fly where the data would only need to be stored temporarily or regenerated rapidly when required. The size of the resultant contact data often exceeds the size of the original uncompressed tables they were derived from and hence we would need to more than double the size of our existing database configuration if we were to consider storing the result of contact queries for all simulations. The ability to run on fly analyses such as this in the database also

lends itself well to exploratory visualization tools which can connect directly to the database. For extremely large datasets like ours we have found that current commercial software is inadequate for our needs and our lab has developed a powerful data visualization engine dubbed DIVE (Data Intensive Visualization Engine) that can connect to our SQL database and rapidly visualize millions of data points in many dimensions (Bromley, 2010).

Since the heavy atom contact query is a computationally expensive calculation that queries the atomic coordinate tables, the largest tables in our database, we decided to use this query (utilizing spatial indexing) as the basis for testing permutations of data and index compression. The aim here was to find a data and index compression permutation on the coordinate tables that saved disk space but did not significantly affect query execution times. We looked at 9 permutations of data and index compression - we applied each of these to each of our 11 metafold representative coordinate tables. As an initial test we calculated the heavy atom contacts for the first nanosecond of each metafold with each permutation of compression for the coordinates table. Figure 14 shows the average execution times for calculating heavy atom contacts (with and without spatial indexing) for 1 ns for each metafold with every permutation of compression. Figure 15 shows a box plot which compares the average % compression for each permutation of data and index compression across all the 11 metafolds The average % compression ranges from 8 - 36%. Having no data compression and row index compression gives the smallest % compression and page data compression and page index compression giving the greatest % compression .

Comparing the total execution times (Figure 15) for the heavy atom contacts over 1 ns on the compressed tables versus the non compressed tables we observe that all permutations of compression fall within the standard deviation of the total execution times for the non-compressed tables. This result is important as it indicates that we can choose any permutation of compression and still retain the same execution time for the heavy atom contacts query. The tantalizing prospect of being able to compress our coordinate data by 36%, by using the page

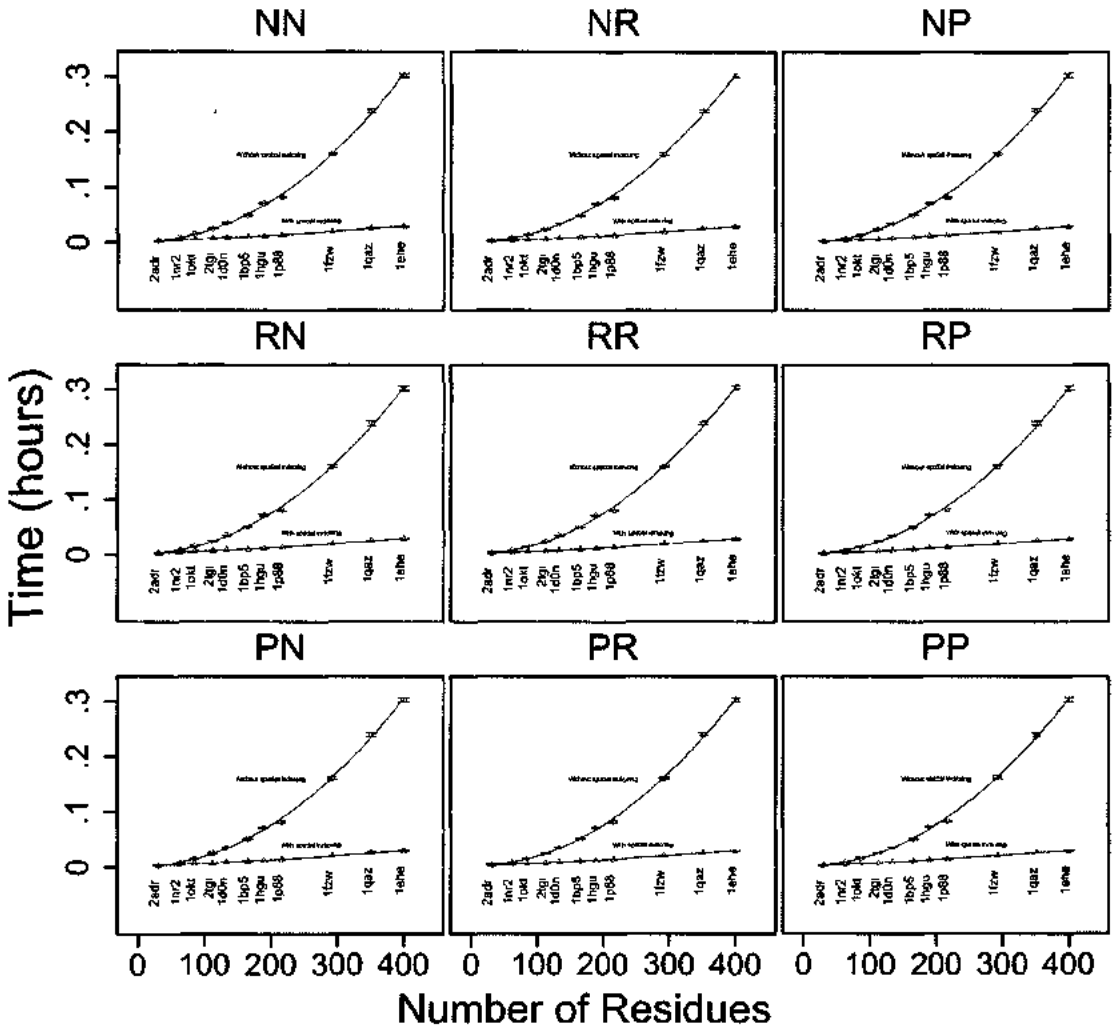


Figure 14. Compression and execution times. Comparison of 9 combinations of compression and their affect on query execution times with and without spatial indexing. P = page, R= row, N=none, e.g. PP represents Page compression on both the data and index where as NR represents no data compression but row compression on the index.

data and page index compression was investigated further by then calculating heavy atom contacts for the full 51 ns (51,000 frames) of each simulation. We compared the execution times (with and without spatial indexing) of the non-compressed coordinate tables to that of the 36% compressed data (Figure 16). We observe that that there is no significant difference (Table 17, b) in execution times for examining the full trajectories when calculating contacts from an uncompressed coordinate table and page/page compressed coordinate with and without spatial indexing, which confirms our earlier finding (Figure 15). The implication of this will be far

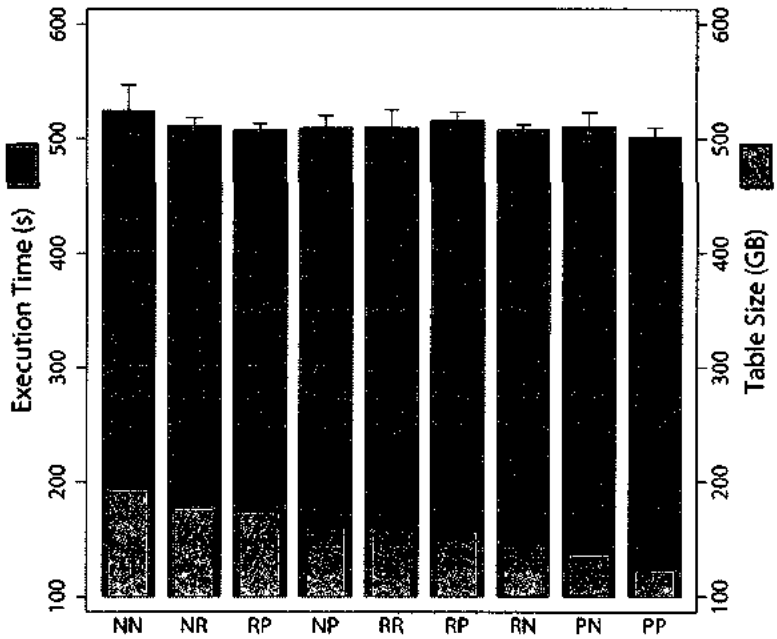


Figure 15. Comparison of total execution times and table sizes. Total table sizes for tables for all compression combinations and total execution times for the 1 ns contacts query. Total execution times are the sum of the individual representatives query times. Even with the largest compression using page compression on both the data and indexes, total execution times were comparable to the none compressed tables.

reaching in our lab since we can now proceed confidently in applying the page/page compression scheme across all our coordinate tables in our entire database. Since, 85% of our database is taken up with simulation atomic coordinate data a 36% space saving is extremely significant since our database comprised of 70 TB of uncompressed data across 6 servers. This compression scheme is oriented towards repeated values, such as those found in dimension keys. For those bioinformaticians with databases implemented in MS SQL Server 2008, compression permutations should be investigated with an appropriate representative data set.

Importantly, our implementation of a spatial indexing scheme in a SQL database to speed up the discovery of nearest neighbor atoms can be applied to other nearest neighbor problems, indeed the indexing is not bound to 3 dimensions. An indexing scheme, based on many dimensions is possible and thus adjacent bins in many dimensions could also be determined to speed up the detection of nearest neighbors in a many dimensional space -(need a good example here...hmmm)

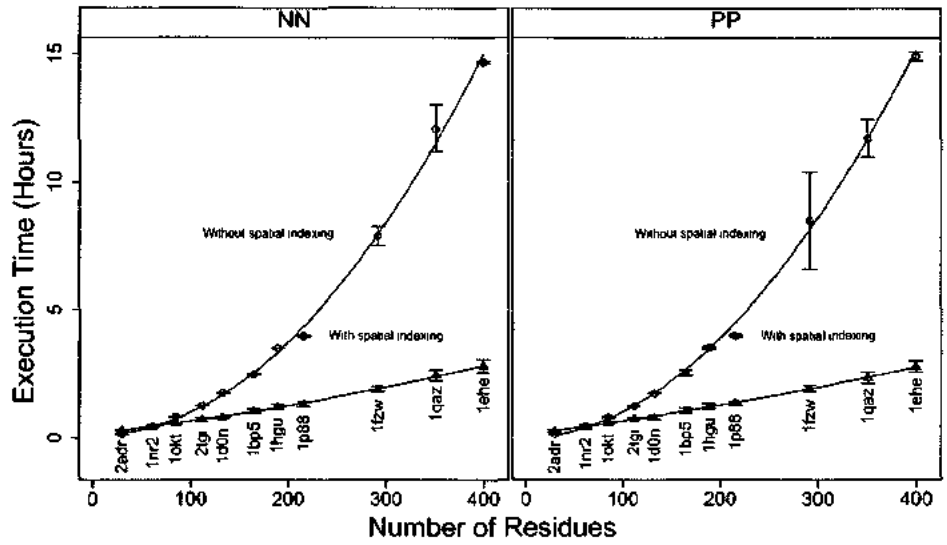


Figure 16. Comparison of compression. Execution time comparison for heavy atom contacts query, with and without spatial indexing, using uncompressed tables and page/page compression. NN denotes no data and no index compression while PP denotes page compression on the data and index. No significant difference is observed in execution times.

Conclusions

We investigated the use of spatial indexing to speed up the discovery of atomic contacts/interactions in our MD Dynameomics simulation database. We compared the discovery of contacts for 11 representative metafolds with and without spatial indexing and found that using spatial indexing decreases execution times by up to 90%. We note that for a small protein, 2adr, execution times using spatial indexing actually increased execution time. We also discovered that we could improve the execution time by providing the query optimizer a hint to use the correct index. Since the coordinate tables are the largest tables in our database we investigated permutations of page and row compression across the data and indexes. We determined that whichever permutation of compression we used the execution time for the heavy atom contact query was not significantly different. Further investigation into the compression permutation that gave us 36% savings (page/page compression) across the entire trajectories showed that this also applied to a large-scale query We can now proceed with applying the page/page compression across our entire database and make use of the space savings without losing query performance.

Table 17. Compression comparison. Comparison of non-compressed vs. page-page compressed tables both without and with spatial indexing. In no case was the observed difference in average execution time significant for non-compressed vs. page-page compress source tables for 51ns trajectories.

Spatial	PDB	[1]Time (s)	[2]Time (s)	[3]Δ	95% CI	p-value	Note
No	2adr	482.2	486.4	-4.2	(-26.19, 17.82)	0.6921	No significant difference
	1nr2	1592.9	1608.7	-15.8	(-59.20, 27.58)	0.8242	No significant difference
	1okt	2866.5	2918.4	-51.9	(-244.00, 140.28)	0.7758	No significant difference
	2tgi	4438.3	4464.8	-26.4	(-140.83, 87.96)	0.7233	No significant difference
	1d0n	6256.6	6252.9	3.7	(-149.68, 157.05)	0.4748	No significant difference
	1bp5	8922.4	9113.9	-191.5	(-1058.00, 674.98)	0.771	No significant difference
	1hgu	12578.6	12634	-55.4	(-472.06, 361.28)	0.677	No significant difference
	1p88	14310.3	14358.3	-48	(-326.01, 230.06)	0.6746	No significant difference
	1fzw	27756.2	30548.9	-2792.7	(-9959.08, 4373.64)	0.8184	No significant difference
	1qaz	43456.2	42074.9	1381.3	(-5140.15, 7902.75)	0.2873	No significant difference
	1ehe	52838.7	53669.9	-831.3	(-2309.40, 646.87)	0.9271	No significant difference
Yes	2adr	1064.1	963.7	100.5	(-61.07, 262.03)	0.0924	No significant difference
	1nr2	1632.1	1545.8	86.3	(-260.31, 433.00)	0.2841	No significant difference
	1okt	1977.1	2065.1	-88	(-294.22, 118.21)	0.8218	No significant difference
	2tgi	2508.5	2732.8	-224.3	(-480.51, 31.96)	0.9612	No significant difference
	1d0n	2854.3	2833.8	20.4	(-272.43, 313.32)	0.4373	No significant difference
	1bp5	3738	3790	-52	(-413.02, 309.00)	0.6231	No significant difference
	1hgu	4343.3	4395.7	-52.4	(-515.52, 410.65)	0.5971	No significant difference
	1p88	4688.2	4820.8	-132.6	(-502.85, 237.61)	0.7788	No significant difference
	1fzw	6821	6868.9	-48	(-575.63, 479.66)	0.5786	No significant difference
	1qaz	8751	8390.1	360.9	(-619.33, 1341.05)	0.2156	No significant difference
	1ehe	10049.6	9941.4	108.1	(-1004.09, 1220.36)	0.4163	No significant difference

[1]Average time for non-compressed tables. [2]Average time using page-page compressed tables. [3]mean difference.

Methods

MD Simulations

Details of how we selected the 807 metafolds for simulation in our Dynameomics project can be found elsewhere ((Schaeffer, 2011b; van der Kamp, 2010)). The MD simulations were performed using in lucem molecular mechanics (ilmm) (Beck, 2000-2011) following the Dynameomics protocol described by Beck et al. (Beck, 2008). Each of the metafolds was subjected to at least one native-state at 298 K simulation of at least 51 ns, and five to eight simulations at 498 K, with two of these simulations being at least 51 ns long. Structures were saved every 0.2 ps for the shorter runs and every 1 ps for the longer simulations. Coordinates and

analyses from the simulations were loaded into our Dynameomics database (Simms, 2008).

When a simulation is loaded into the database, it is assigned an integer identifier and a specific database. Three tables are created in the assigned database to hold the fact data for the simulation: a Coordinate table (abbreviated to "Coord"), a Box table, and Bins table. Each table is named by the simulation id, for example the tables for simulation id 37 would be "Coord_37," "Box_37," and "Bins_37." The Coord table contains columns for each of the three-dimensional coordinates, atom number, step, structure, and instance. The step and coordinate columns are considered fact data, the instance, structure and atom number are dimension data linking each fact back to a specific structure. The Box table has columns for the x, y, and z dimensions of the periodic box at each time point. The Bins table records the set of adjacent bins for each bin. All three tables have clustered primary keys and constraints; the Coord table also has a secondary covering index.

We selected 11 metafolds to represent the range in sequence size that our Dynameomics project covers from the smallest: ADR1 DNA-binding domain from Saccharomyces Cerevisiae (2adr, 30 residues,(Bowers, 1999)) to cytochrome P450 (1ehe, 400 residues,(Shimizu, 2000)). Figure 12, shows the metafolds selected. In the test conducted in this study we chose to look at the native (298 K) simulations for each of these proteins. Each 298 K simulation was 51 ns in length and coordinates were written out every 1ps.

Implementation of spatial indexing in the database

To calculate contacts in SQL, an expensive self-join of the coordinate table must be used in addition to joins with structural data tables. A version of this query is shown in Figure 17. Conditions in the JOIN clauses ensure that comparisons are made within the same frame (a.step = b.step) and with a granularity of 1ps (a.step % 500 = 0 and b.step % 500 = 0). Since distance is reflexive, we only calculate the distance from a heavy atom in "a" to another in "b" (a.atom_number < b.atom_number). We also exclude contacts in the same or adjacent residues

```
SELECT      j.sim_id
          , j.step
          , j.residue_id_x
          , j.atom_number_x
          , j.residue_id_y
          , j.atom_number_y
          , SQRT(j.distance) as distance
FROM (
   SELECT   a.sim_id
            , a.step
            , c.residue_id as residue_id_x
            , c.atom_number as atom_number_x
            , c.atom_type as atom_type_x
            , d.residue_id as residue_id_y
            , d.atom_number as atom_number_y
            , d.atom_type as atom_type_y
            , (b.x_coord - a.x_coord) * (b.x_coord - a.x_coord) +
              (b.y_coord - a.y_coord) * (b.y_coord - a.y_coord) +
              (b.z_coord - a.z_coord) * (b.z_coord - a.z_coord) as distance
   FROM (   dbo.Coord_2029 AS a
            INNER MERGE JOIN dbo.id as c
            ON (    c.heavy_atom = 1
                    AND a.struct_id   = c.struct_id
                    AND a.atom_number = c.atom_number
                    AND a.[step] % 500 = 0
                    AND a.[step] between 0 and 500000 ))
         JOIN dbo.Bins_2029 AS n
              ON ( n.hash3d_index = a.bin )
         INNER MERGE JOIN (  dbo.Coord_2029 AS b
               INNER MERGE JOIN dbo.id as d
                    ON (  d.heavy_atom = 1
                          AND b.struct_id   = d.struct_id
                          AND b.atom_number = d.atom_number
                          AND b.[step] % 500 = 0
                          AND b.[step] between 0 and 500000 ))
                    ON (    a.[step] = b.[step]
                            AND a.atom_number <b.atom_number
                            AND c.residue_id < d.residue_id-1
                            AND n.hash3d_index_neighbor = b.bin )) AS j
WHERE j.distance < (CASE WHEN j.atom_type_x = 'C'
                              AND j.atom_type_y = 'C' THEN 29.16
                         ELSE 21.16 END)
```

Figure 17. Heavy atom contacts query. The size of the coordinate table self-join is reduced by applying two right associative join clauses, shown in bold. Right associative joins are a mechanism to control the order of join evaluation. In this case we insure that only the rows meeting satisfying the given predicates participate in the final self-join (i.e. heavy atoms and only the first 1ns of simulation time). The spatial-index join in shown in bold-italics. This clause allows SQL to trim away most atoms outside the cutoff range without needing to perform the distance calculation, greatly reducing the number of operations as well as rows that would later be thrown away by the distance cutoff. Finally, MERGE joins are explicitly specified to avoid the optimizer choosing a HASH join for the coordinate table self-join.

(a.residue_id < b.residue_id - 1). Finally, the query only considers heavy atoms (c.heavy_atom=1 and d.heavy_atom=1).

There are three supported join types in SQL Server: Hash, Merge, and Loop. Normally queries are expressed using only the keyword JOIN, leaving the optimizer free to choose the join type when an execution plan for a query is prepared. Join types are described in detail elsewhere (Fritchey, 2009). The self-join of the coordinate table presents unique difficulty because of its size. We have observed that the optimizer will consistently choose a hash join, which will cause an expensive build of a temporary hash structure. In contrast, the merge join type does not require the temporary structure, and since the data are ordered based on the primary key, this approach is significantly faster.

We have optimized the structure of the query with the use of two right associative joins to cause early evaluation of the Coordinate and ID table joins. We have also pushed predicates directly into the join clauses. However, despite these optimizations a great deal of time is spent calculating distances for atoms that are far outside the 5.4 Å distance of interest. These additional calculations generate result rows that add a significant performance burden, making it impractical to run this query over more than a handful of trajectories.

We have implemented a spatial indexing algorithm in the database to accelerate the discovery of atomic contacts. In our implementation we subdivide the periodic box used for simulation and divide it into as many smaller cubes with sides of at least 5.4 Å. We then consistently number these cubes, creating a one-dimensional hash. For simulation data, the number of these smaller cubes in a simulation will never come close to 232-1 bins, so it is possible to represent in a SQL 32bit integer. We then iterate over all atoms in a simulation and map each atom's coordinates into a single bin using equations 1.1 and 1.2. This result is stored in the coordinate table. A second smaller table named bins_x (where x is the simulation id) is created for each simulation, which stores rows for each combination of a bin index and itself and the 26 possible adjacent bin indices. This table is populated using a C# user defined func-

tion at the time the simulation coordinate data are loaded.

With the Bins table in place, the contact query presented earlier can be modified slightly to filter coordinates considered using the bin column in the coordinate table. The modification is shown in bold below (Figure 18). This simple join allows the query optimizer to quickly remove distance calculations based on a comparison of integer columns instead of projecting and transforming x, y, z from each half of the join. The result is a spectacular increase in speed, as the Bins table acts as a highly optimized spatial index.

The spatial indexing optimization increases query performance significantly by reducing the number of pairs of atoms it has to evaluate. The first part of this paper looks at the performance gains when utilizing the spatial indexing by comparing the time taken to calculate contacts over a 1 ns window (1000 frames) of time over 11 representative proteins ranging from 30 to 400 amino acids (Figure 12, Table 15).

The second question we address is whether performance can be enhanced further by making I/O operations more efficient. SQL Server 2008 supports two types of compression, which can be applied separately to the data and indices associated with a table. Row compression is a more efficient representation of row data; the implementation involves storing fixed length columns in a manner similar to variable length columns. For coordinate columns, which are a set of 5 32 bit fixed length columns, the storage savings for row compression are small. Page compression, which is built on top of row compression, stores repeating values in a single structure and then references them. This can result in significant savings in a fact table since the table contains numerous constant dimension columns like sim_id. For the combination of data and index page compression, we observe a consistent 36% reduction in table storage space.

Although storage space reductions with data and index page compression are significant for coordinate data, a major concern was the potential for decompression to ruin the performance of analysis queries. To investigate this, we return to the contacts query introduced earlier in this section (since this is a commonly used and computationally expensive query in the lab)

and review performance data collected against all combinations of compression options across our sample set of 11 protein simulations. We also considered non-compressed and fully page compressed contact queries for the first 1 nanosecond that did not utilize the spatial indexing optimization.

Database and System setup

Two Dell R710 servers each equipped with dual hex-core processors were used to collect timing information. The base operating system is Windows Server 2008 Enterprise x64 R2 and the database engine used was SQL Server 2008 R2 Enterprise x64 R2. Detailed hardware and software configuration information is shown in Table 18.

One database called hash3d-700 was created on each testing server and populated with a set of coordinate trajectory tables and dimension tables from our primary data warehouse. The base coordinate tables were then copied to additional tables, adding an additional suffix to indicate data and index compression settings. After all coordinate tables were created and populated, identical primary keys, constraints and indexes were applied. Tables were then compressed using **ALTER TABLE** statements. A script was run on all the coordinate table compression combinations to create contact tables. The size of each hash3d-700 database size

Table 18. Test server hardware configuration, hardware and software.

Hardware	Description
Server	Dell R710
Processors	Dual Intel Xeon X5650s (x64 Hex Core)
Memory	48 GB
Storage	H700 Integrated RAID SAS Disk Controller
System Disks	136 GB on two 15K RPM 150GB SAS disks , RAID 1 (Mirrored)
Data Disks	7,450 GB on six 7200 RPM 2TB SAS disks, RAID 0 (Striped)

Software	Description
OS	Windows Server 2008 R2 Enterprise x64
Database	SQL Server 2008 R2 Enterprise x64
SQL	Enabled for all CPUs
SQL Memory	Limited to 40,960 MB (8GB for OS)
Anti-Virus	Sophos Endpoint Security and Control, version 9

```
CHECKPOINT;
DBCC FREESYSTEMCACHE (''ALL'') WITH MARK_IN_USE_FOR_REMOVAL;
DBCC DROPCLEANBUFFERS;
```

Figure 18. Cache clearing commands. These command are executed before each timing query to insure that SQL Server's cache is set to the same state as if the server were rebooted.

was then adjusted upwards to 1.2 TB and the SQL Server process shutdown. The **defrag.exe** utility was then run on the data and system partitions clean up file system fragmentation caused by auto-growth during loading.

In our primary data warehouse, a simulation's coordinate fact data are stored in three distinct tables: a coordinate table, a box table, and a bins table. Dimensional data describing simulation and structure parameters are stored in shared tables. For testing purposes, coordinate, box and bins tables were copied to each testing server and the set of dimensional meta-data for the simulations in our sample set were copied locally. This approach allows the fact and dimension data for these tests to be completely self-contained.

Queries were run in SQL Server Management studio running on a remote machine with a connection to the test database server. Queries were executed with **SET STATISTICS IO ON** and **SET STATISTICS TIME ON** to capture logical and physical read statistics. To control for performance gains caused by data and/or query plan caching; and background write operations from result tables, a series of three system statements were executed prior to running the test query (Figure 18). The **CHECKPOINT** statement insures that any dirty pages (such as those result rows written out by the previous query) are written to disk. The **FREESYSTEMCACHE** command eliminates any stored query or procedure plans. The **DROPCLEANBUFFERS** flushes out the current cache leaving it effectively cold, as though SQL Server had just started. During the collection of timing information, access to both servers was restricted and only the timing query was allowed to run.

Performance of heavy atom contacts query with and without spatial indexing

We calculated the pairs of heavy atom contacts for the 1st nanosecond of each simulation

and compared the execution times with and without spatial indexing. Queries were written in SQL and executed in MS SQL management studio as described in the above section. Heavy atom contacts were calculated 3 times for each simulation, ensuring the system cache was cleared between each run to obtain performance statistics. To utilize spatial indexing, a simple join to the 'Bin' table was employed, which ensured that only atoms within the current spatial bin and immediately adjacent bins were considered for evaluation. Statistics were calculated using a two sample two-sided t-test for unequal variances.

Comparison of page and row compression on data and indexes for coordinate tables

MS SQL Server 2008 supports two types of compression that can be applied to both data and indexes independently. We investigated 9 permutations of non-compressed, page compression and row compression on both data and indices for each coordinate table for each of the 11 simulation's coordinate table in our test set. We recorded the % compression of each compression permutation compared with the non-compressed coordinate tables. We then ran an initial test of performance by investigating the execution time and disk I/O operations of the heavy atom contacts query over the first nanosecond of the simulation. Performance of heavy atom contacts query Data (page) and Index (page) compression on data and indexes for coordinate tables. When compressing data and indices there is inherently a trade-off between the reduction in the size of the table and the time taken to decompress the table and access the data. Ideally, a data intensive query run on a compressed table would not take significantly longer to process than the same query on an uncompressed table. Based on the results obtained from analyzing the multiple compression permutations on the data and the related indexes we examined the execution time of the heavy atom contacts query over a full 51 ns (51,000 frames) trajectory for each of the proteins in our test set.

Chapter 5: Generation of a Consensus Domain Dictionary

The discovery of new protein folds is a relatively rare occurrence even as the rate of protein structure determination increases. This rarity reinforces the concept of folds as reusable units of structure and function shared by diverse proteins. If the folding mechanism of proteins is largely determined by their topology, then the folding pathways of members of existing folds could encompass the full set used by globular protein domains.

We have used recent versions of three common protein domain dictionaries (SCOP, CATH, and Dali) to generate a consensus domain dictionary (CDD). Surprisingly, 40% of the metafolds in the CDD are not composed of autonomous structural domains, i.e. they aren't plausible independent folding units. This finding has serious ramifications for informatics studies mining these domain dictionaries for globular protein properties. However, our main purpose in deriving this consensus domain dictionary was to generate an updated 2009 CDD to choose targets for MD simulation as part of our Dynameomics effort, which aims to simulate the native and unfolding pathways of representatives of all globular protein consensus folds (metafolds). Consequently, we also compiled a list of representative protein targets of each metafold in the CDD. This domain dictionary is available at www.dynameomics.org.

Introduction

Structurally similar proteins need not share significant sequence identity. The early observation of structurally and functionally similar proteins (such as hemoglobin and myoglobin) led to the partition of different sets of structurally similar proteins into folds (Kendrew, 1959; Perutz, 1960). However, as more structures were determined and more folds discovered, it became clear that not all members of a fold are necessarily linked by a common function (Nagano, 2002). Also, the determination of structures with conserved structural cores surrounded by variable regions complicated the classification of new structures into existing folds. What degree of structural variation is tolerable between a domain and a potential cousin before they no longer

can be considered to belong to the same fold?

The inconsistencies of analyzing and generating protein domain dictionaries are one component of the vigorous discussion surrounding the properties of protein 'fold space' (Csaba, 2009; Pascual-Garcia, 2009; Sam, 2006). Distinct folds can contain regions of shared structural similarity (Grishin, 2001). Folds are both populated to different degrees and structurally heterogeneous (Coulson, 2002; Majumdar, 2009; Wolf, 2000). This heterogeneity complicates estimates of the size and 'shape' of fold space, and is likely responsible for the wide range of the estimated number of protein folds. The presence of unclear domain boundaries in regions of fold space have led some to question the utility of a hierarchal definition (Kolodny, 2006). Furthermore, fold assignment is dependent on the prior problem of domain detection (Holland, 2006; Majumdar, 2009).

The gold standards among domain dictionaries, SCOP (Structural Classification of Proteins) (Murzin, 1995) and CATH (Class, Architecture, Topology, Homology) (Orengo, 1997), have been the subject of many detailed comparisons (Day, 2003; Hadley, 1999; Jefferson, 2008; Pascual-Garcia, 2009; Veretnik, 2004). In general, both dictionaries weigh potential functional and evolutionary relationships between fold members with different strengths at different levels of their hierarchies. The presence of shared fragments between differing folds and/or regions of "conserved" structure have been well documented and are one reason for the development of different empirical classification methodologies, as more knowledge of protein structural evolution emerges, hope remains that an evolutionary classification will be derived (Valas, 2009). In their early formulations, these domain dictionaries represented different design methodologies. Whereas SCOP was hand curated by experts, CATH was maintained by a combination of automated process and expert curation. However, SCOP has assumed more automated pre-classification of new structures in responses to the increasing rate of structure determination, diluting this methodological distinction (Andreeva, 2008).

Although individual domain dictionaries may contain their own biases, we can minimize

the effect of those differences by extracting a consensus from a group of such dictionaries. We previously demonstrated the application of this method to SCOP, CATH, and the Dali Domain Dictionary (Dietmann, 2001) to generate a consensus domain dictionary (CDD 2003 version, v2003) (Day, 2003). This domain dictionary was the basis of our initial high-throughput survey of native dynamics (Beck, 2008). Additionally, the concept of the metafold that we introduced in the v2003 CDD was further developed in a study of 'cradle-loop' structures (Alva, 2008). A subset of the representative domains from v2003 was used to conduct benchmark simulations of standard molecular dynamics (MD) force fields (Rueda, 2007).

Here we present an updated CDD (v2009) derived using recent versions of the input domain dictionaries, which incorporate many of the new structures determined since the v2003 CDD. The CDD is the backbone of our high-throughput molecular dynamics initiative, Dynameomics (Beck, 2008; van der Kamp, 2010). This project seeks to simulate the native and unfolding behavior of representatives of all protein folds. Consequently, we need an objective basis for selection of simulation targets. Therefore, it is important that the CDD be monitored so that we can identify novel topologies as they are classified and observe potential splits within, and mergers between, our metafolds as classifications shift. It is important that we identify domains that appear to be autonomous units, since we use the contents of the CDD as potential targets for simulation of folding/unfolding pathways. The selection process was complicated by the discovery that roughly a third of the consensus folds (metafolds) in the CDD are not autonomous structural units, but instead are dependent components of multi-domain or complex structures (or are small structural motifs).

We present our data model for representing domains and their metafolds over time in a relational database (Simms, 2008). We discuss the use of this data model to map domains and their annotations from older versions of our dictionary to the newer one (v2003 ® v2009). We present the full v2009 CDD consisting of 1695 metafolds. We then filter the set to remove metafolds that do not represent autonomous units or cannot be simulated for other reasons,

which yields 807 metafolds. In addition to being of use to our Dynameomics efforts, the filtered 807 target list is more appropriate for bioinformatics studies investigating globular protein properties than the full consensus domain dictionary or the three parent domain dictionaries by removing folds that do not represent autonomous folded structures.

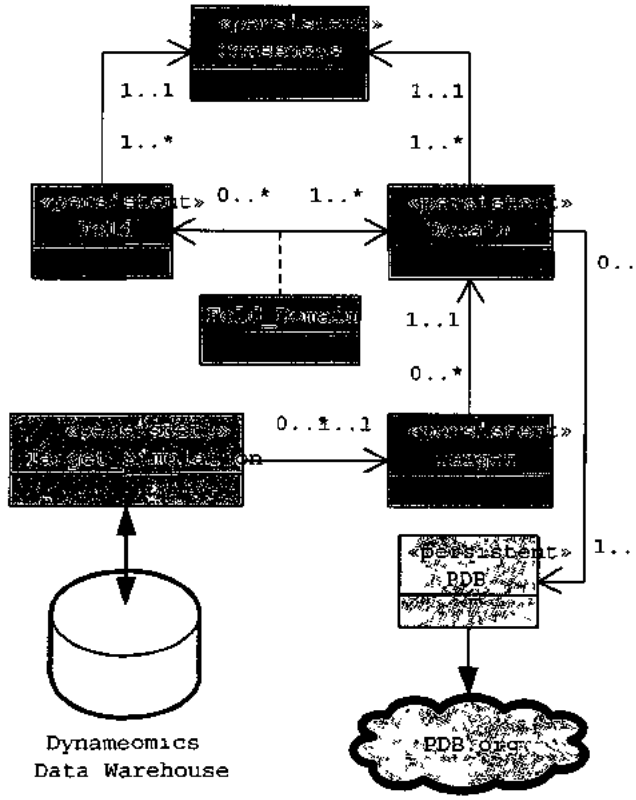
Methods

Relational model for consensus set data

The relational schema for the 'Target Selection and Preparation' (or 'Prep') database, which houses our CDD, is shown in Figure 19 in a unified modeling language (UML) representation (Simms *et al.*, 2008). Consensus domains are stored in the *Domain* table consisting of an identifier, PDB code, and fold identifiers from the SCOP, CATH, and Dali domain dictionaries. A domain must contain fold identifiers from at least two of the three input domain dictionaries. Metafold data are stored in the *Fold* table, which contains a metafold identifier, name, and the metafold's rank (based on domain population). Note that the *Fold* table is, in fact, a table of metafolds. There may be multiple versions of the same domain in the *Domain* table (due to multiple CDD versions), and these differing versions may link to multiple metafolds (also due to multiple CDD versions). The many-to-many relationship between *Fold* and *Domain* is implemented via the *Fold_Domain* table. Metafold representatives chosen for simulation are captured in the *Target* table.

As previously stated, domain classifications evolve over time, which can cause changes in the CDD. To capture these changes, the Fold, Fold_Domain, Domain, and Target tables include a consensus set identifier to allow multiple versions of metafold and domain definitions to be stored in the same primary tables. To facilitate cross-consensus set queries, fold identifiers

Figure 19. Target Selection and Preparation ('Prep') database schema. Prep schema as modified to account for multiple consensus sets and simulation associations. UML schema describes one-to-one (1..1) and one-to-many(1..*) relationships.



are maintained across consensus set releases where this is meaningful. It is possible for new identifiers to be introduced and existing identifiers to be removed in subsequent releases.

Domains and targets are both linked to external data sources. The Domain table contains a field for the PDB code, and we populate a local cache table (PDB) with specific information synthesized from a given structure's PDBml (Westbrook et al., 2005). Examples include a structure title, dates, methods, and source organism. These fields facilitate local searches and analysis. The Target_Simulation table links targets in the Prep database to simulations contained in the Dynameomics data warehouse (Simms, 2008).

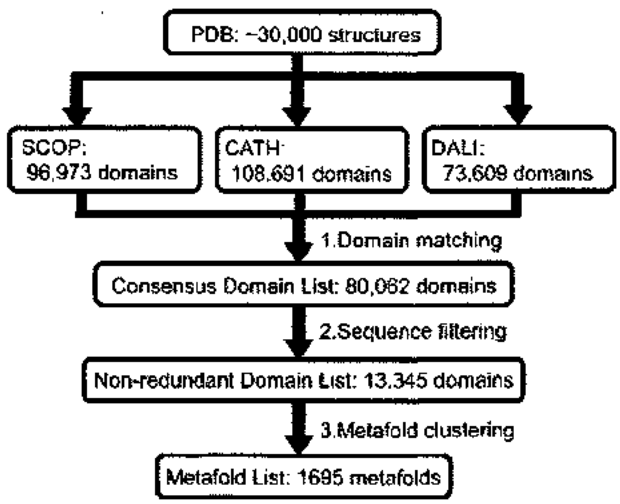
Generation of the v2009 CDD

The v2009 CDD was generated as described by Day et al. (Day, 2003). To generate the

CDD, we integrate recent versions of three major domain dictionaries: SCOP (Andreeva, 2008), CATH (Cuff, 2009), and Dali (Dietmann, 2001). SCOP v1.73, CATH v3.2, and a March 2005 download of the Dali Domain Dictionary were used as input for consensus generation. CDD generation is a two-step process: First, consensus domains are generated by pairwise comparison between domain dictionaries of residue ranges from the same chain. Where a significant overlap between input domains is detected, a consensus domain is assigned. Second; the set of consensus domains is filtered for sequence similarity and then clustered into a set of metafolds based on their composite fold identifiers. The set of consensus domains and metafolds comprise our CDD. The workflow of this process is outlined in Figure 20.

Our domain matching procedure follows the criteria specified by Dietmann and Holm (Dietmann, 2001). A given domain in one input dictionary is compared against analogous domains in the other domain dictionaries. Where the given domain and an analogous domain both overlap to a significant extent (80%) a consensus domain pair is assigned. If a given domain matches domains from both other input dictionaries, the three resulting domain pairs are col-

Figure 20. Overview of the consensus domain dictionary (CDD) generation process. Consensus domains are first found between pairs of input dictionaries. The resulting domain list is filtered for sequence identity. The resulting non-redundant domain list is clustered into a list of metafolds. The collected domain lists and metafold list are the contents of the CDD.



lapsed into a single consensus domain spanning analogous domains from all three input dictionaries. If a domain from any single domain dictionary has no consensus with any domain from either of the remaining domain dictionaries, it is discarded. Each consensus domain preserves the source data from its input dictionaries (PDB, chain, residue range, and fold identifier). This list is loaded into our database to assist with metafold representative selection and report generation. The schema is described in Figure 19.

The full domain list is filtered by sequence using the SCOP ASTRAL95 sequence-filtered domain list and the CATH 'SOLID' sequence identifiers (Chandonia, 2004; Greene, 2007). The non-redundant domain list produced by the sequence filter is used as the basis for generation of metafolds. Each domain contains a composite fold identifier derived from its input domain definitions. SCOP and CATH are hierarchal classifications, for SCOP we chose the 'Fold' level to cluster, for CATH we chose the 'Topology' level. Domains whose composite fold identifiers share two of three elements are clustered together into a metafold. Those metafolds are then sorted and ranked by their non-redundant population.

Mapping between CDD versions

The CDD is a product of clustering across input domain dictionaries. As these input dictionaries change with the release of new versions, so should the CDD. However, without a detailed description of the changes made, it can be difficult to assign equivalence between two domains from CDDs generated from different inputs. A mapping between the v2003 and v2009 CDD was generated based on domain identifier and fold identifier equivalence. Changes in fold representation in new versions of both CATH and Dali motivated the mapping criteria. Between the release of CATH v2.4 and v3.0, "working" CATH classes [6-9] were no longer included in production releases (Greene, 2007). Since the v2003 CDD included these classes,

criteria were chosen such that v2003 domains could be reassigned to regular (1-4) CATH classes. Since fold identifiers do not persist between v3.1b and the March 2005 version of Dali, identify between these versions could not be used as the basis for a mapping. In the period of time since we acquired this version of Dali, this domain dictionary has been discontinued (Holm, 2008).

Four mapping criteria were defined based on the mapping classification a domain possessed in the v2003 CDD. A v2003 domain possessing composite SCOP, CATH, and Dali fold identifiers is mapped to a v2009 domain if both the SCOP and CATH composite chain and domain (PDB6) and the v2009 Dali PDB6 is defined (though not necessarily equivalent). A v2003 domain possessing only SCOP and CATH fold identifiers is mapped to a v2009 domain if both the SCOP PDB6 identifier and CATH PDB6 identifier are equivalent. A v2003 domain possessing only CATH and Dali fold identifiers is mapped to a v2009 domain if the CATH PDB6 and fold identifiers are equivalent and the Dali fold identifier and PDB6 is defined. A v2003 domain possessing only SCOP and Dali identifiers is mapped to a v2009 domain if the SCOP PDB6 and fold identifiers are equivalent and the Dali fold identifier and PDB6 is defined.

Selection of domains as metafold representatives

We examined domains by manual inspection within each metafold to assess their suitability as a simulation target. We chose targets that were self-contained domains in a single protein chain that were less than 450 residues in length. Where the structure was determined by X-ray crystallography, we only chose crystal structures with resolutions higher than 3.0 Å. Domains with obligate cofactors (other than Zn2+, Ca2+, and heme) were rejected. Also, domains with multiple Zn2+, Ca2+, and heme sites were rejected, with a few exceptions (e.g. calbindin). Many of the domains rejected for this reason are chains where the cofactor is a major structural element. Domains with a single Zn2+, Ca2+, or heme were selected (i.e. myoglobin) regardless of whether folding information was available regarding the role of the cofactor. When multiple

domains within a metafold met our selection criteria, we preferred domains with biomedical relevance or with experimental folding studies available for comparison. The workflow for target selection where targets exist from a previous CDD is outlined in Figure 21.

The determination of whether a given domain was self-contained was primarily determined by manual inspection. Several factors could lead to the rejection of a domain as not self-contained; these factors could occur either in isolation or in concert with one another. Where a domain existed was deposited as a multi-domain structure, we used a simple "sheet of paper" criteria to examine the interface of the domain of interest with the rest of the protein. Where a domain could not be cleanly separated from the remainder of the protein, it was rejected for its convoluted interface. In addition, we examined the proposed biological unit from the deposited transform. Structures with extensive domain swapping or crystal contacts could be rejected even though appearing to be in isolation. Furthermore, structures that were 'irregular' (those that possessed little to no structure or hydrophobic core) could also be rejected for being not self-contained. This range of factors led to a broad spectrum of possible buried surface area in rejected metafolds (10% - 60%). Furthermore, where the domain boundary occurred in the middle of a significant secondary structure element (helix or beta sheet), this disruption could be

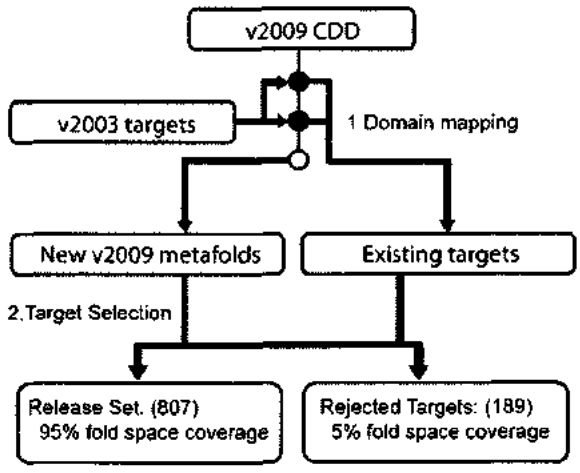


Figure 21. Overview of the mapping and target selection process. Existing v2003 targets are mapped to the v2009 CDD. (1)Where a mapped domain was selected or rejected in the v2003 CDD, this status is maintained in the v2009 CDD. (2)Where a new metafold is observed, targets are selected from available domains in that metafold.

used as a reason for rejection as not self-contained. This was not used as a basis for rejection where the secondary structure was a linking region and could be safely truncated to the previous loop region and where that truncation would not expose significant hydrophobic surface area.

Where a single suitable domain was selected as a target for simulation it was designated a representative for its metafold. If, after examining all domains within a metafold, a suitable domain could not be found, a domain was chosen as a fold representative and the reasons for its rejection were annotated. Once a domain was selected as a metafold representative, we chose a residue range to simulate that incorporated the input domain definitions such that we avoided disrupting secondary structure elements while removing long, unstructured tails (many of which are cloning artifacts).

Results

v2009 Consensus Domain Dictionary

The CDD consists of a set of consensus domains and a list of consensus fold identifiers binding these domains together into metafolds. The process of CDD generation is summarized in Figure 20. Consensus domains were identified between pairs of domain dictionaries (SCOP/CATH, SCOP/DALI, Dali/CATH). Summary statistics from each of the domain dictionaries are presented in Table 19. The agreement between domain dictionaries was measured as the fraction of shared consensus domains divided by the total number of domains originating from structures shared between the two dictionaries.

We reduced the effect of differing release dates (and thus different numbers of structures) by considering only shared structures. CATH and Dali have the highest agreement, with 96% of CATH domains and 90% of Dali domains included in the CATH/Dali consensus domain set. SCOP and CATH have the next highest agreement, with 79% of SCOP domains and 82% of CATH domains in the SCOP domains in the SCOP/CATH consensus domain set. Finally, SCOP

Table 19. SCOP, CATH, and Dali. Summary statistics of the SCOP, CATH, and Dali domain dictionaries used in the v2003 and v2009 CDD.

Version	Dictionary	Chains (C)	Domains (D)	Folds[1]	D/C[2]
v2003	SCOP	27,308	35,095	783	1.29
	CATH	25,622	36,480	1,453	1.42
	Dali	21,493	35,492	1,088	1.65
v2009	SCOP	74,608	96,973	1,280	1.29
	CATH	74,240	108,691	1,110	1.46
	Dali	52,740	73,609	2,783	1.39

[1]Number of unique folds at the chosen level within each domain dictionary
[2]Number of distinct domains (D) per distinct chain (C)

and Dali had the lowest agreement, with 65% of SCOP domains and 61% of Dali domains included in the SCOP/Dai consensus domain set. A consensus domain need not exist solely between a single pair of domain dictionaries. Where a consensus domain was determined by each of the three pairwise comparisons, it was collapsed into a single triple consensus domain in the CDD. Thus, four classes of consensus domains were created, SCOP/CATH, SCOP/Dali, Dali/CATH, and SCOP/CATH/Dali.

The v2009 CDD is composed of 80,062 domains, originating from 27,140 PDB structures. The total number of PDB structures considered is lower than the structures available due to the lag between PDB and domain dictionary releases. The domains of the CDD were distributed among the aforementioned classes as follows: 51% SCOP/CATH/Dali, 30% SCOP/CATH, 10% CATH/Dali, and 9% Dali/CATH. To generate the metafold list, the CDD first must be filtered by sequence identity. The nrCDD was composed of 13,345 domains. The domains in the CDD clustered into 1695 metafolds. On the whole, these metafolds incorporate 4217 unique consensus fold identifiers derived from 971 unique SCOP folds, 923 unique CATH topologies, and 2362 Dali folds. The distribution of domains per fold for the input domain dictionaries is shown in Figure 22.

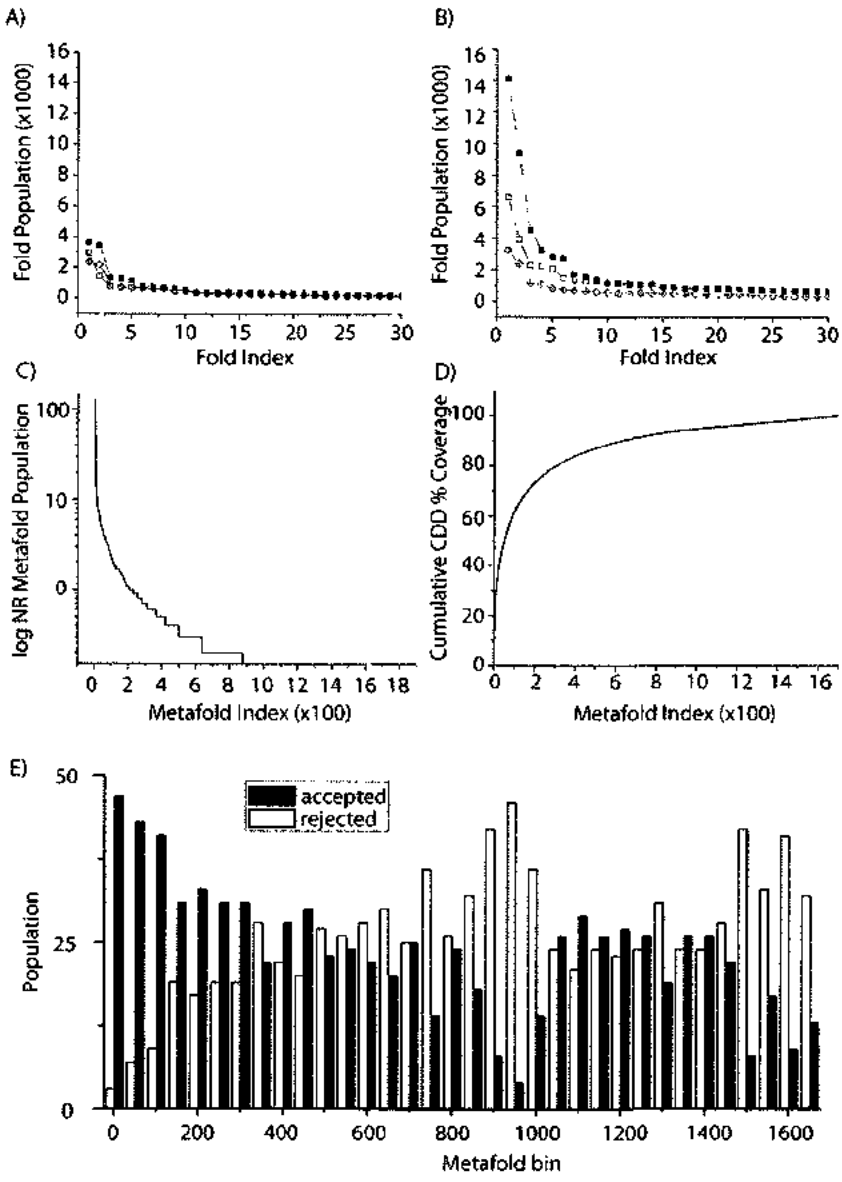


Figure 22. Distribution of domain populations between folds and metafolds. A) Population distribution of top 30 most populated folds in the SCOP (filled squares), CATH (open squares), and DALI (crossed diamond) dictionaries for the v2003 CDD. B) Population distribution of top 30 most populated folds in the SCOP, CATH, and DALI dictionaries for the v2009 CDD. C) Non-redundant population distribution of the top 100 most populated metafolds in the v2009 CDD. D) Cumulative percentage of domains represented by metafold rank. The most populated metafolds account for a large percentage of the domains in the CDD. E) Metafold distribution binned by 50-fold increments, sorted by rank into rejected and accepted populations

Comparison of v2009 to v2003

Both the residue range of a domain and its fold classification can change over time.

These changes affect the output of the metafold clustering and the domain contents of the CDD. Since our Dynameomics simulations are indexed against the CDD, it is necessary to track domains across multiple dictionary versions so that information about our simulated domains is current. Where possible, we generated a map between domains in our v2003 and v2009 CDD based on their fold identifiers. There were 31,141 domains in the v2003 CDD. From this dictionary, 4,693 domains could not be mapped forward from v2003 to v2009 and are considered obsolete (discussed below). 26,448 domains were mapped from v2003 to v2009. There are 53,614 domains in the v2009 dictionary that were not in the v2003 dictionary.

The domains that were not mapped from the v2003 CDD can be broadly partitioned into three categories: (1) domains from structures that were dropped from consideration in one of the input domain dictionaries, (2) domains whose boundaries changed significantly in one of the input domain dictionaries, and (3) domains that were split into multiple domains or merged into a single domain. From each of our input dictionaries used in v2003 CDD, 95% of the structures considered also had at least one domain in the input dictionaries used in the v2009 CDD. The ~5% of structures that were in the v2003 CDD but not in the v2009 CDD had the following properties: the structure was deemed obsolete by the PDB, the structure consisted primarily of nucleic acids, or the structure was a purely computational model. Of those chains that were removed from consideration that were not part of the aforementioned dropped structure set, the majority are rare cases arising from the presence of synthetic linkers and/or multi-chain domains arising from viral capsid structures. In some cases where neither the chain nor structure containing a domain were dropped, but it could still not be mapped, the domain boundaries in the structure were significantly altered. Alternatively a domain was split into multiple domains or merged with other domains. Although we can observe these transitions, we prefer to treat the resulting domain(s) as new. The 4,393 dropped domains from the v2003 ® v2009 CDD mapping originated from 2,198 PDB structures. 3,314 of those v2003 domains originate

from PDB structures that still contain domains in the v2009 CDD. There are 1,379 v2003 domains originating from 608 PDB structures not found in the v2009 CDD. 319 of these v2003 domains originate from structures that were superseded by newer structures in the PDB. The remaining 1,060 domains are dropped either because they were removed from one of the input dictionaries, or because the domain definition was changed in one or more of the input dictionaries, breaking the original v2003 consensus.

Domains that were mapped from v2003 to v2009 met specific criteria for their particular class (SCOP/CATH, SCOP/Dali, etc.). Of the 26,448 mapped domains, 15,735 were mapped using the SCOP/CATH/Dali class, 7,736 were mapped using the SCOP/CATH class, 1,734 were mapped using the SCOP/Dali class, and 995 were mapped using the CATH/Dali class. A majority of the domains in our CDD could be mapped based on their SCOP and CATH identifiers alone. The mapped domains originated from 11.896 PDB structures, leading to an average of 2.23 mapped domains per PDB structure. The mapped domains originate from 857 metafolds in the v2003 CDD and are mapped into 719 metafolds in the v2009 CDD, indicating that some v2003 metafolds and their domain contents were merged into larger v2009 metafolds. Any domains were also folded into larger metafolds as they gained a third input fold identifier. 6,613 mapped domains with defined SCOP, CATH, and Dali domain identifiers in the v2009 CDD contained only two fold identifiers in the v2003 CDD.

'New' domains are those that exist in the v2009 CDD and did not exist in the v2003 CDD. The 53,614 new domains originate from 17,949 PDB structures. These new domains fall into 1565 metafolds. There were 976 metafolds in the v2009 CDD that consisted entirely of new domains, 589 metafolds composed of a mix of mapped and new domains, and 130 metafolds that consist entirely of mapped domains. A majority of the new v2009 domains were placed into metafolds with other mapped domains. 8,401 v2009 domains fell into metafolds composed solely of new domains. The domain population was less than five for 633 of the new v2009 metafolds.

SCOP and CATH in the v2009 CDD

The consensus generation process can separate an input fold into multiple metafolds or merge multiple input folds into a single metafold. We examined the location of input folds from SCOP and CATH within the CDD closely because it indirectly addresses the continuity of fold space. This analysis also serves as an internal check of the consistency of our metafold clustering method. The domains of an input fold can be distributed into multiple metafolds and/or combined into a metafold with domains from other input folds. To quantify this effect, we analyzed the number of metafolds into which an input fold and its domains are distributed. An input fold can be distributed over many metafolds and yet the vast majority of that fold's domains can still be assigned to a single metafold. Thus, we are primarily interested in the fractional domain population of the metafold containing the majority of an input fold's domains, or the 'most populated metafold.' The net effect of this treatment is that outliers within a fold are partitioned into their own poorly populated or singleton metafolds (metafolds containing only a single domain).

Certain structurally variable topologies (such as the Rossmann folds) are split more evenly across a number of metafolds. The 860 input SCOP folds were spread over 815 CDD metafolds. 12 of these metafolds contained multiple SCOP input folds. The metafold containing the most SCOP input folds was metafold #2 (consisting of a number of Rossmann folds), followed by metafold #16 (consisting of parallel a-helical bundles), and metafold #1 (consisting of IgG-like b-sandwiches). These SCOP folds are bound together by highly populated CATH topologies. A full listing of merged SCOP folds is provided in Table S1. 815 metafolds contained only a single SCOP fold. Of these 815 metafolds, 290 also contained only a single non-redundant domain. We also examined those SCOP folds where the most populated metafold contained a diminished fraction of the total domains, indicating that the SCOP fold was distributed across multiple metafolds. 112 of the input SCOP folds had a fractional population within the most populated metafold of 80% or less. The significance of this fraction can vary, however, if the

input fold is poorly populated or if the input fold was not a child of one of the 4 main structural classes (all-a, all-b, a+b, or a/b).

The 892 input CATH folds were distributed over 862 metafolds. 26 metafolds contained domains from multiple CATH folds. The most populated metafold, consisting of IgG-like b-sandwiches, contained four CATH folds. Metafolds #16 and #46 contained three CATH folds. The remaining 23 metafolds each contained two CATH folds. The most populated metafold of the 30 most populated CATH folds is presented in Table S2. Of the 866 metafolds containing only a single CATH fold, 277 also contained only a single nonredundant domain, signifying singleton metafolds. The CATH Rossmann fold (3.40.50) was the most populated of the CATH folds that were significantly distributed over multiple metafolds. This fold was distributed over 42 metafolds, and the most populated metafold of these (#2) contained only 49% of the input fold.

The v2009 CDD has 881 unique SCOP folds from the 11 different SCOP classes (all-a, all-b, a+b, a/b, multidomain a and b, membrane and cell surface, small proteins, coiled coil, low resolution, peptides, and designed) There were 434 SCOP folds that only appeared in metafolds with a simulated metafold representative and 332 SCOP folds that were only found in rejected metafolds. The rejected SCOP folds represent about a third of the folds from each of the top four classes (all-a, all-b, a+b, a/b) found in our CDD, between 27 to 38% of each class. We rejected approximately 70% of each of the multidomain and membrane classes in our set. Similarly, there are 894 CATH topologies in our domain dictionary from the four CATH classes: mainly-a, mainly-b, mixed a-b, and irregular/few secondary structures. The majority (77%) of the irregular class CATH topologies are only found in rejected metafolds. The other three CATH classes all had between 36-47% of topologies found only in rejected metafolds. These classes had a similar number of topologies found only in selected metafolds (40-55%). This analysis of the SCOP and CATH folds reveals that we have not biased our set of selected metafolds towards any fold class or systematically rejected any class, except for unstructured

peptides and membrane proteins.

Selection of Metafold Representatives

The primary purpose of the CDD was to facilitate the simulation of both the native state dynamics and the unfolding behavior of at least one domain from each metafold. As such, we examined domains from each metafold to find a high quality structure suitable for simulation. Such domains were then selected as a 'metafold representative', or target, of that metafold and prepared for simulation. If no suitable domain could be found we chose one domain from the metafold to represent the reason that the metafold was rejected. The selected representatives for the top 30 most populated metafolds are presented in Figure 24, the full target set is provided in Table S3. Selected representatives could come from a variety of structural contexts; 387 representatives were the full contents of their PDB structure deposition, 165 representatives were a full chain from a multi-chain deposition, and 165 representatives were excised domains where a chain was chopped to select the domain.

We identified at least one domain suitable for simulation from 807 of 1695 metafolds in the v2009 CDD. Of the remaining 888 metafolds, 585 metafolds consisted of domains that were not self-contained and 87 metafolds consisted of domains that were irregular. Of these 672 metafolds, none were autonomous units (75% of the rejected metafolds or 40% of the total number of metafolds). A summary of the reasons a domain from a metafold was rejected is presented in Table 20. These rejected domains fell into three categories: domain-swapped dimers, domains with a large buried interface in the experimentally determined structure of a complex, and domains with secondary structure elements that continue into other domains of the protein (Figure 23). There was no significant bias in major fold class (all a, all b, mixed a/b) in the rejected metafolds. In 11 metafolds, no domains of less than 450 residues were present so the metafold was rejected for reasons of size. In 27 cases, the domains of the metafolds in question were contained a transmembrane region. There were 54 metafolds whose domains required an

Table 20. Justifications for rejection for 888 metafolds in the v2009 CDD.

Reject Reason	Definition	Metafolds
Not an autonomous domain	Poor interface, continuation of secondary structure into other domains, small with little secondary structure	672
Large gaps	Backbone gap of more than seven residues	85
Non-parameterized co-factors or structural ions	Structurally necessary non-protein molecules that have not been parameterized	57
Membrane	Domain penetrates membrane	27
Size	Larger than 450 residues	11
Resolution	Resolution lower than 3.0 Å	20
Rejected by simulation	Did not pass native (298 K) simulation quality control	14
Other	Structures in dispute	2

aStructure 1BEF was retracted from the PDB, causing rejection of domains 1BEFA01 and 1BEFA02. (Murthy, 2009)

obligate cofactor. There were 85 metafolds where each of the domains contained a large (greater than 7 residue) gap and were rejected. In 87 cases, the metafold consisted of domains that lacked regular secondary elements and/or were unstructured peptides. In 20 metafolds, all domains had a resolution lower than 3.0 Å. Finally there were two singleton metafolds that were rejected because their domains were of disputed structural validity at the time of writing (Murthy, 2009). In 14 cases, we selected a domain but the resulting native state simulation was not stable and the metafold was rejected. For these 'rejected by simulation' cases, no alternative replacement could be found from their respective metafolds (See (van der Kamp, 2010) for more details).

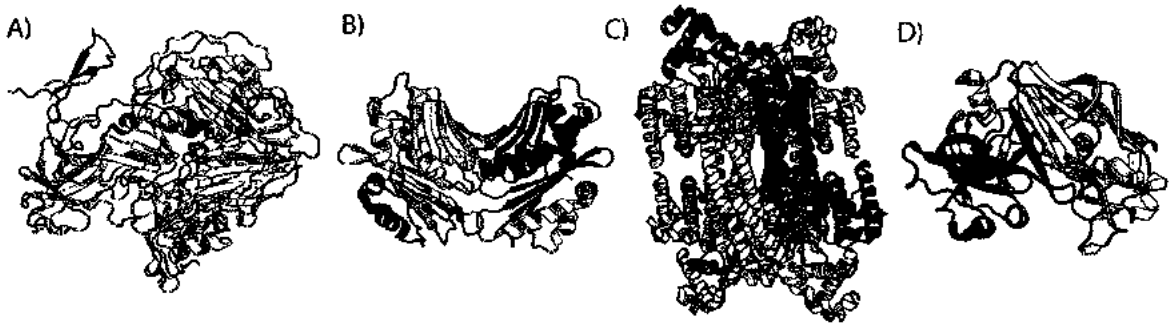


Figure 23. Example metafolds rejected for not being autonomous units. A) Metafold #232, chain 4 of P1/Mahoney poliovirus mutant (1AL2). B) Metafold #2232, Chain A of d-crystallin I (110A). C) Metafold #489, chain B of HSP33 (1HW7). D) Metafold #172, Chain C of cathepsin D (1LYA).

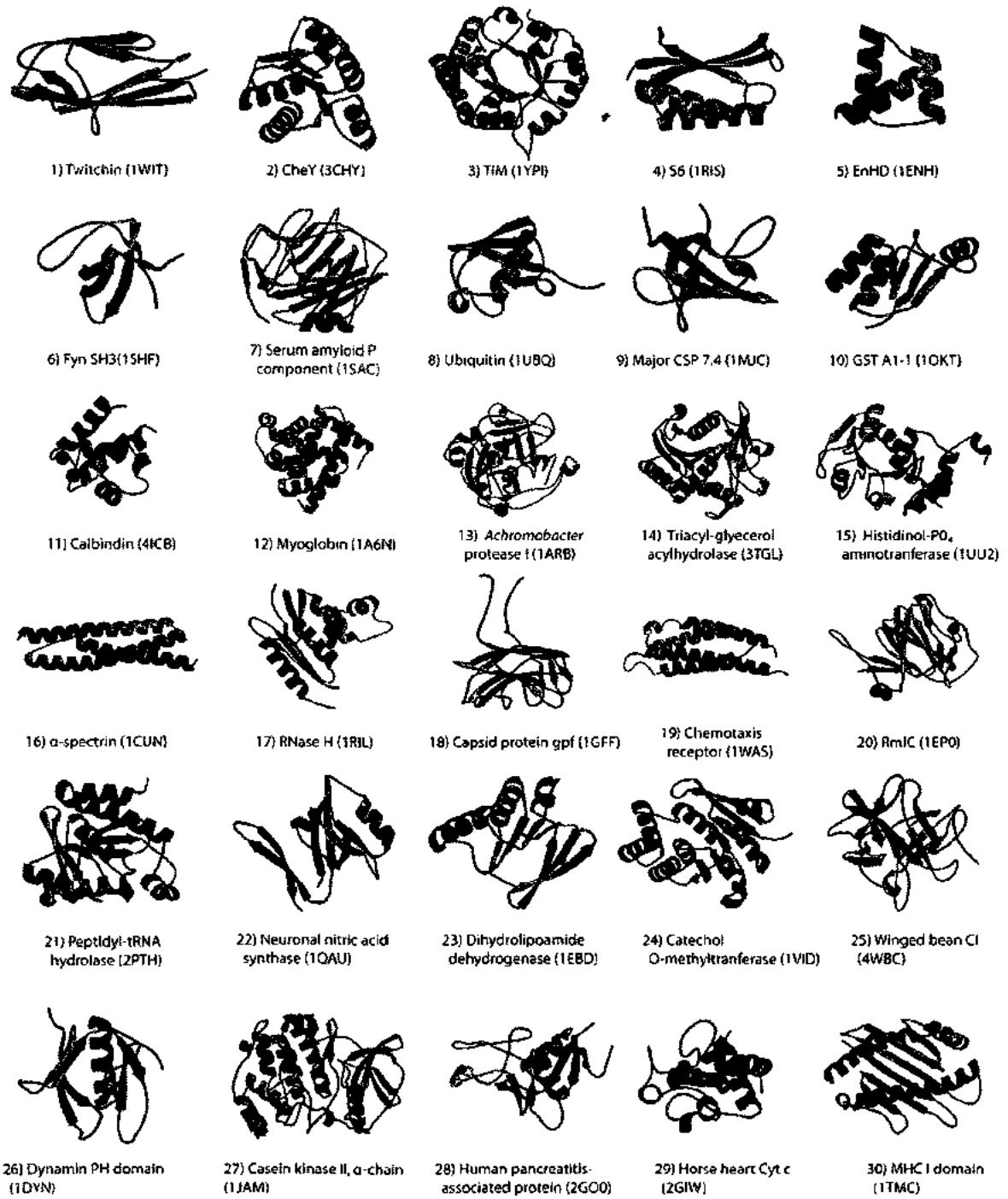


Figure 24. Structure representatives. Structures of the representative domains of the 30 most populated metafolds in the 2009 CDD. Domains are named based on their source structure, where a domain was an excised chain or domain, it is named according to the PDB-deposited name for its chain.

Discussion

The recognition of spatially distinct motifs and structural patterns is a long-standing component of structural protein studies (Phillips, 1967; Wetlaufer, 1973). The understanding of the term 'domain' to denote an autonomous, structurally cohesive unit is similarly well established (Levitt, 1976). However, the multiple extant definitions for 'domain' do not always converge (Majumdar, 2009; Sowdhamini, 1995). A spatially distinct region within a structure may not coincide with an autonomous, stable unit. Our interest in domain dictionaries is to establish a systematic, broad sampling of topologies that satisfy our autonomy criterion. The single most striking conclusion from this endeavor was that a significant fraction of metafolds generated by our consensus method contained no domain suitable for simulation. This occurred due to a variety of factors, but the single largest reason for rejection was that the domain was not self-contained. Identification of protein domains can be split into two problems: the partition of a chain into multiple domains, and the separation of domains into folds. The difficulty of partitioning a chain into domains has been well studied (Holland, 2006; Veretnik, 2004). The separation of domains into fold has been similarly examined. Both problems share similar elements. It may be that the smallest repeating structural element observed between two structures is not necessarily a shared domain. For example, if chain discontinuity is allowed within a domain to increase structural similarity of the domains in a fold, then the structural integrity of the excised region may be sacrificed. The problem becomes more complex when considering domains that are solely observed in the context of multimeric structures or in complexes. In our opinion, one must be very careful to consider the effect inadvertent inclusion of such domains may have on bioinformatics studies; they are not independent, globular structures. We note that the distribution of autonomous and non-autonomous domains is not necessarily related to the dependent or independent folding of these domains in nature. Indeed, discovering the folding behavior of the autonomous domains is one of the primary goals of the native and folding simulations we have performed of these domains.

We have generated a consensus domain dictionary (CDD) from three major domain dictionaries. This CDD contains 1695 metafolds. We have inspected each metafold and selected a single representative where suitable autonomous criteria were met. These representatives constitute our release set, which consists of 807 'simulatable' domains. This set of autonomous domains is the basis for our high-throughput MD simulation of representatives of all globular protein folds (Beck, 2008; van der Kamp, 2010). Also, to reduce artifacts, we would suggest that the reduced list of 807 metafolds be used for bioinformatics studies, not the full CDD, nor the domain dictionaries from which they were derived.

Chapter 6: The Molecular Mechanics Parameter Markup Language

Molecular dynamics is a method from theoretical physics used to study the motion of a system of particles and has been validated in many applications, including the study of protein motion in solvent. A significant issue in the application of molecular dynamics simulations to map macromolecular motion is the correct determination, representation, and management of force field parameters. Numerous incompatible formats exist, including everything from tables in the literature to proprietary file formats tied to specific software packages. Here we introduce the Molecular Mechanics Parameter Library (MMPL), an extensible public standard for developing and sharing force field parameters, as well as a collection of validated parameters for common chemical entities.

Introduction

Molecular dynamics (MD) is a simulation method from theoretical physics used to study the motion of a system of particles (Allen, 1987; Haile, 1992). The method has been applied and validated in many applications(Daggett, 2002; Giudice, 2002; Hansson, 2002; Jungwirth, 2002; Kremer, 2003; Norberg, 2002; Saiz, 2002; Wang, 2001; Warshel, 2002); here we focus on its practical application in the domain of protein dynamics, but the approach is general and the schema described here can accommodate any chemical moieties. At the core of MD are the classical equations of motion and an equation, known as a force field, which models the potential energy. The force field equation includes terms that capture the contribution of intra- and intermolecular interactions. These terms are parameterized using constants for the specific types of atoms involved as well as spatial configuration. A simulation engine is used to solve these equations numerically, yielding a set of atomic coordinates. The details of these calculations are covered elsewhere (Levitt, 1983; van Gunsteren, 1990).

Simulation engines read and write data in application specific file formats, including force field parameters. Parameter data formats are usually very compact, often only a thin syn-

tactical veneer over the FORTRAN, C or C++ data structures they will ultimately be loaded into. This approach minimizes the work to parse the data into memory structures by the simulation engine, but it does not facilitate data validation, annotation, or computability outside the simulation engine. Ultimately, these limitations make parameter data difficult to share, publish and maintain. We have addressed these limitations by designing the Molecular Mechanics Parameter markup Language (MMPL) and have implemented it in the *in lucem* Molecular Mechanics (*il*mm) simulation engine (Beck, 2000-2011).

Force Field Parameters

ilmm is a highly scalable MD simulation and analysis software package that is fully integrated with a data warehouse (Simms, 2008). It implements the Levitt et al. (Levitt, 1995) potential function shown in Eqs. (1-3), which is similar to the force fields implemented in AMBER (Pearlman, 1995), CHARMM (Brooks, 2009), and many other simulation engines, with the bonded Ub and nonbonded Unb terms separated.

This compact and elegant symbolic representation hides the complexity of all the bonded and nonbonded interactions, which must be mapped to the potentially thousands of specific atoms found in a simulation system. MMPL captures all the constants, bonds, and links to specific atoms, organizing them hierarchically as shown in Figure 25. This model, which is discussed in detail in the following sections, enables assembly of molecules from reusable

$$F = -\left(\frac{\partial U}{\partial x}\right) \quad (1)$$

$$U_b = \sum_{i}^{bonds} K_{b,i}\left(b_i - b_{0,i}\right)^2 + \sum_{i}^{bond\ angles} K_{\theta,i}\left(\theta_i - \theta_{0,i}\right)^2 + \sum_{i}^{torsion\ angles} K_{\varphi,i}\left\{1 - \cos\left[n_i\left(\varphi_i - \varphi_{0,i}\right)\right]\right\} \quad (2)$$

$$U_{nb} = \sum_{pairs\ i,j}\left[\varepsilon_{ij}\left(\frac{r_{0,ij}}{r_{ij}}\right)^{12} - 2\varepsilon_{ij}\left(\frac{r_{0,ij}}{r_{ij}}\right)^{6}\right] + 332\sum_{pairs\ i,j}\left(\frac{q_i q_j}{r_{ij}}\right) \quad (3)$$

components. Although designed and tested using ilmm, MMPL is general purpose and can be

easily adopted by other simulation packages. As an example, an MMPL file was created using parameter information from the published AMBER parameter file (Cornell, 1995; Wang, 2000) (parm99.dat) and is included in the supplementary data.

The MMPL Data Model

MMPL consists of a data model for describing chemical entities and parameters, a W3C XML Schema (W3C.org, 2004a; W3C.org, 2004b) for representing the data model as an XML document, and a sample set of parameters for chemical entities described by Levitt et al.(Levitt, 1995) and the F3C water model (Levitt, 1997). The data model defines a four-level hierarchy of structural entities and an ordered set of rules to map non-bonded interaction parameters to specific atoms. The highest level of structure within MMPL is a molecule; the lowest level is an atom. Molecules are composed of residues, which are in turn composed of groups, and groups are composed of individual atoms as illustrated in Figure 26. Nonbonded parameters are associated with individual atoms explicitly (van der Waals radius and energy), and through

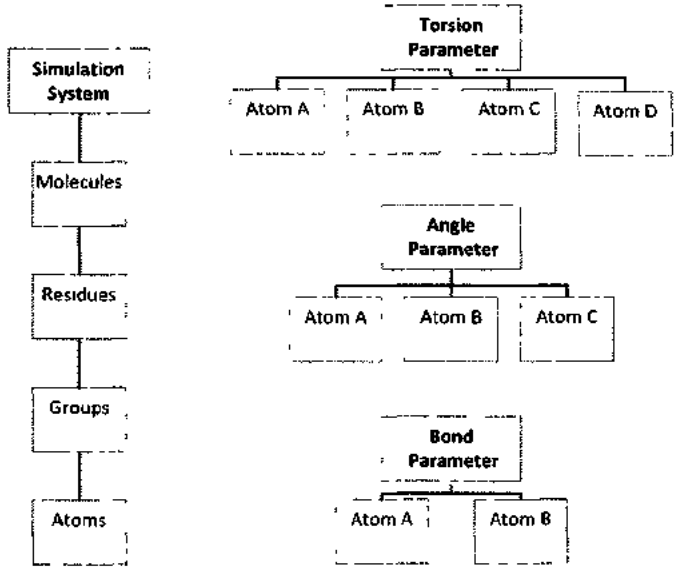


Figure 25. The MMPL as a multi-dimensional data model. MMPL has 4 hierarchies, the simulation system which consists of structural elements and three types of bond parameters. Non-bonded parameters for van der Waals radii and energy are stored at the atoms level; charge parameters are stored in atom at the groups level.

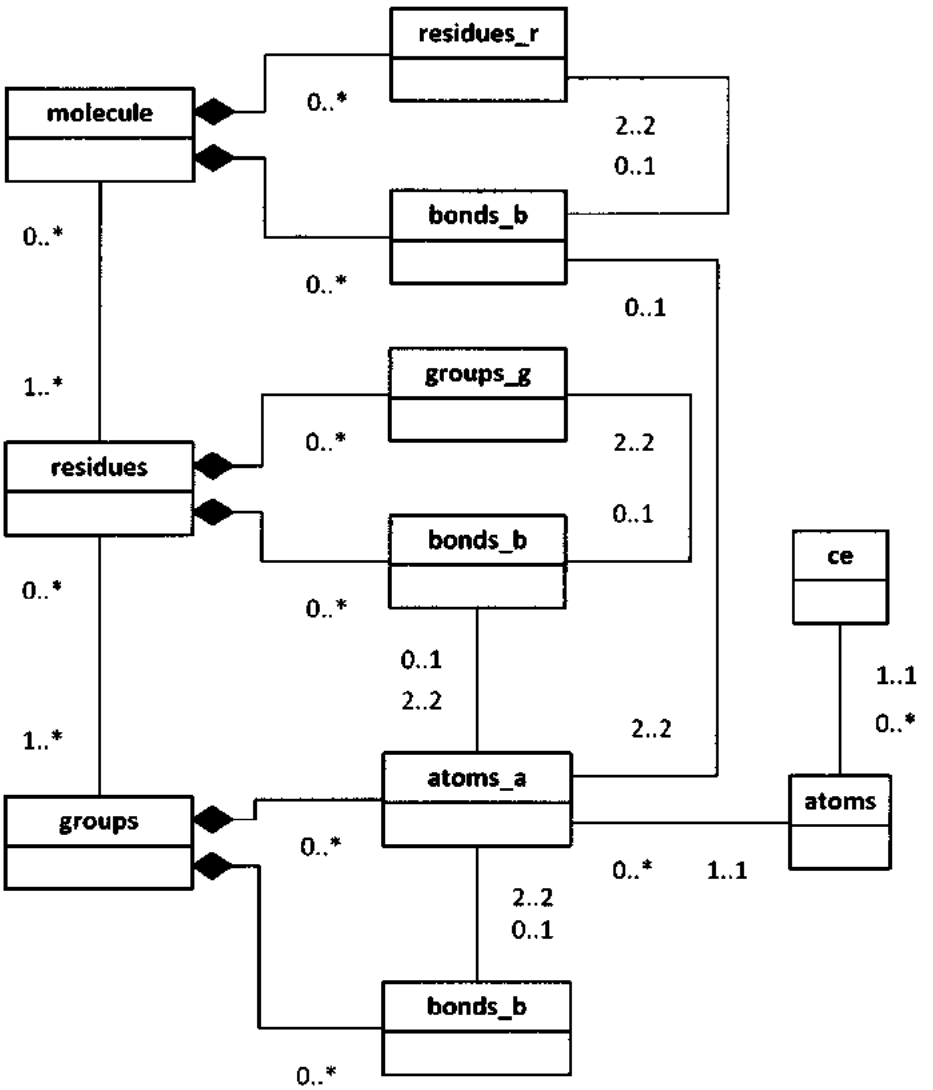


Figure 26. MMPL schema. A Unified Modeling Language (UML) representation of the MMPL structural hierarchy. Complex chemical structures are assembled from reusable components, atoms are assembled into groups, which are then composed into residues, and finally into molecules.

groups (charge). These parameters are base values, which may be overridden in a simulation. For example, a simulation may apply general a-scale or b-scale factors to interactions between non-bonded atoms (van der Waals attractive and repulsive portions of potential, respectively); may include and scale specific interactions between atoms (c-scale); or may apply a cutoff range to exclude interactions between non-bonded atoms. Bonded parameters are mapped using

a wildcard facility that matches on atom names. The set of structural entity and parameter matching rules are listed in Table 21 and described below. All primary elements support the concept of a comment and Digital Object Identifier field (DOI) for data provenance.

Table 21. MMPL Elements. This table lists the principle elements of MMPL.

Element	Type	Usage
mmpl	Root Element	Root of a valid MMPL document
atom	Structural	Defines an atom
group	Structural	Defines a collection of atom elements
residue	Structural	Defines a collection of group elements
molecule	Structural	Defines a collection of residue elements
ce	Annotation	Chemical element, used to classify atoms
bp	Parameter	Bond length parameter
ap	Parameter	Angle parameter
tp	Parameter	Torsion parameter

Atom elements

An **atom** element is used to define van der Waals radius, r, van der Waals energy, ε, and atomic mass, m, using the **r**, **epsilon**, and **mass** attributes, respectively. Atoms are referenced by name and can be assigned multiple names, so long as they are unique across all atoms. Since radius and energy parameters will vary depending on the surrounding structure, it is possible to define multiple atom elements for the same atom type. For example, a hydrogen atom may appear in a polar configuration, such as exists in a water molecule, and will have r = 0.91Å and ε = 0.01001 kcal/mol; a hydrogen atom in a nonpolar molecule will have r = 2.825Å and ε = 0.038 kcal/mol. Atoms can optionally be explicitly associated with a ce (chemical element definition) element.

Group elements

A **group** element is used to describe charges on individual atoms and to define bonds

between atoms in the group. Groups are keyed on a name attribute; atoms within a group linked to previously defined atoms by name. Each atom reference is assigned a locally unique integer index attribute, **idx**, and a charge, using the **q** attribute. If a group contains more than one atom, and these atoms participate in a chemical bond, **b** elements are used to describe the participating atoms by index. Figure 27 illustrates the relationship between the carbon atom in the group named "CH" and its definition (A), as well as assignment of a covalent bond to that same carbon and a hydrogen within the group (B).

Residue elements

A **residue** element corresponds to a reusable unit of structure such as an amino acid. A **residue** element contains a list of group elements; each referenced by the attribute name and assigned a locally unique integer index attribute idx. Similar to the group element, residue elements can also define bonds between atoms in different groups. Bonds at the residue level are bonds between groups. Thus, they reference the local group index group_idx_A and an atom index attribute atom_idx_A that identifies a specific atom within the referenced group. Figure 27 illustrates the relationship between a group named "CH" and its definition (C); bonds between groups, such as the covalent bond between carbon and nitrogen (D); and an indirect reference to the "C1" carbon via the attribute **atom_idx_B** (E).

Molecule elements

A **molecule** element represents the highest level of structure within MMPL. A molecule element contains a list of residues and two attributes: **name** and an optional **struct_id** (used to link to the Dynameomics data warehouse). Similar to elements described earlier, an r element is linked to a previously defined **residue** by the **name** attribute and is assigned a locally unique integer index attribute **idx**. Unlike other elements, the **r** also supports the **pdb_num** and **pdb_icode** attribute which allows the element to be linked to a Protein Databank (PDB) residue number and insertion code, respectively. Similar to the residue element, the molecule element

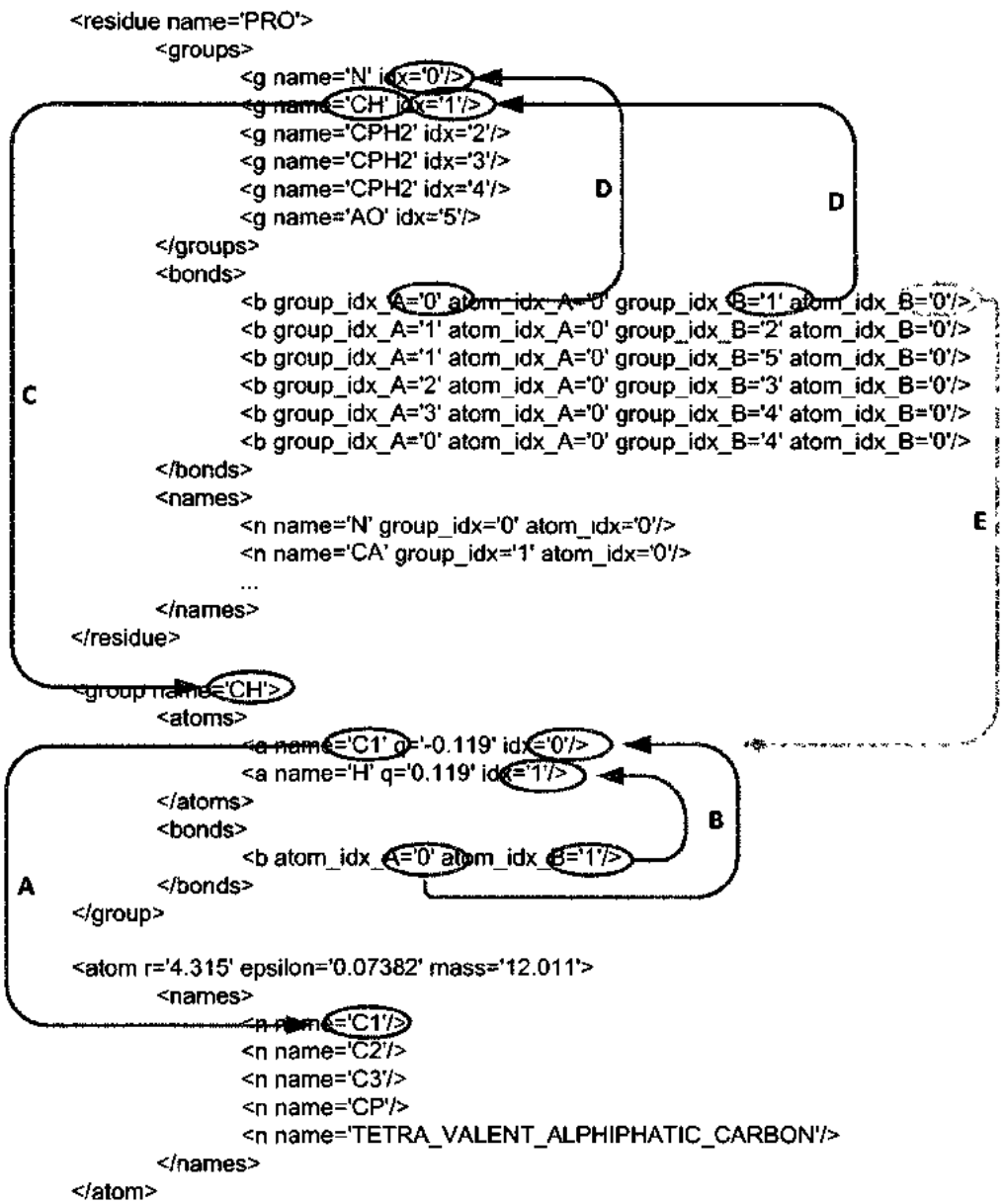


Figure 27. Illustration of relationships between structural elements. A carbon-hydrogen group named "CH" references a carbon atom named "C1" (A), and defines a bond between the carbon and hydrogen atoms (B). This group is part of a proline amino acid residue, and is linked via atom_idx_B attribute (C). The group is bonded to a nitrogen group (D) specifically to atoms specified by the atom_idx_A and atom_idx_B attributes. The atoms referenced are indexed at the group level; atom_idx_B is shown (E).

supports defining bonds between atoms in different residues.

Bonded parameter elements bp, ap, and tp

Bonded parameter elements **bp**, **ap**, and **tp** use a simple pattern match facility to associate parameters to specific configurations of atoms. These correspond to the bond, bond angle and torsion angle terms, respectively, of Eq. (2). This facility allows bonded parameters to be expressed concisely without having to exhaustively enter elements for each atom combination. Ambiguity is resolved by the order that rules appear—the last rule to match takes precedence. Match attributes are always two characters. The first character may consist of an upper-case letter **A-Z** or '**?**'. The second character can consist of a letter **A-Z**, a number **0-9**, a dash '**-**', a period '**.**', apostrophe '**'**', or double-quote '**"**'. Apostrophes and double quotes correspond to notational naming conventions of "prime" and "double-prime", respectively. These values are always URL encoded ("**'**" or "**"**") to avoid conflicts with XML text delimiters.

The **bp** element consists of two atom name reference pattern attributes (**A**, **B**) and two parameters: ideal bond length attribute **l** (b_0) and energy attribute **k** (K_b). The atoms referenced are separated by one covalent bond. The ap element consists of three atom name reference pattern attributes (**A**, **B**, **C**) and two parameters, ideal angle attribute theta (θ0) and energy attribute k (Kθ). The first and last atoms referenced are separated by two covalent bonds. The tp element consists of four atom name references (**A**, **B**, **C**, **D**), and four parameters: a torsion angle **type** attribute (0 = normal and 1 = out-of-plane), ideal angle attribute **phi** (φ_0), energy attribute **k** ($K\varphi$) periodicity attribute **n**. Here the first and last atoms are separated by three covalent bonds. All three types of bonded parameters are illustrated graphically in Figure 28.

The match algorithm for bonded parameters is illustrated in Figure 29. Bond length, angle, and torsion parameters are matched against atoms in the appropriate bond configuration. Because more than one rule can match a given set of atoms, the algorithm will keep reading rules and storing any match until there are no more rules.

Chemical Element (ce) Definitions

The final type of top level element is the chemical element definition **ce**. These elements form a simple list containing a symbol and name for the set of chemical elements defined in the periodic table (CRC Press, 2010). Linking **atom** elements to **ce** elements enables a variety of secondary analyses and reporting.

Validation of elements and relationships

Although MMPL top-level elements can occur in any order, extensive explicit constraints are used to ensure that parameter data stored in MMPL are valid. First, domain level constraints are used wherever possible to limit attributes to a specific data type and a range of correct values. For example, we use the standard XML Schema type "nonNegativeInteger" for

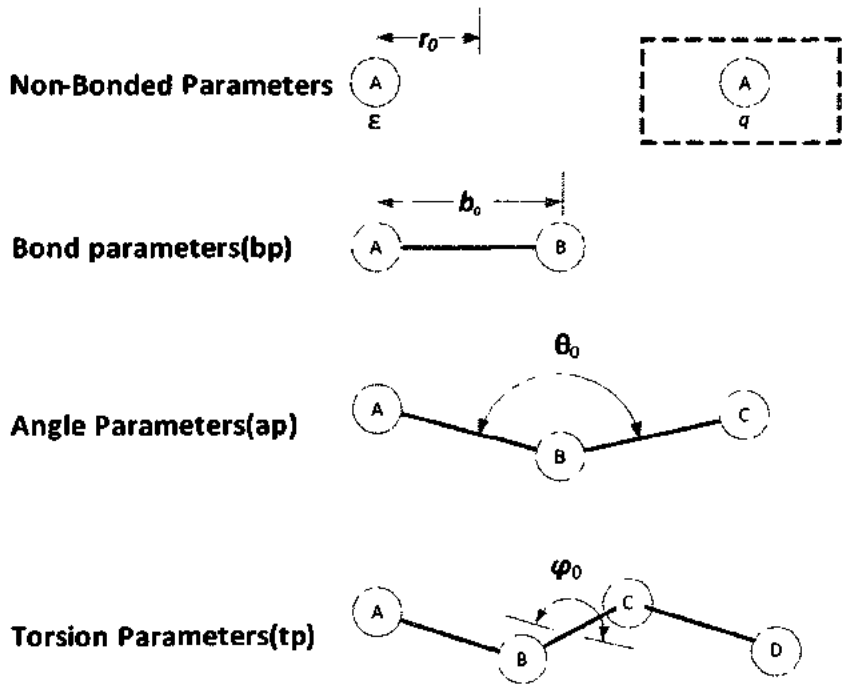


Figure 28: Parameter types. The non-bonded parameters van der Waals radius and van der Waals energy are encoded directly in the atom element using the r and epsilon attributes, respectively. The non-bonded charge parameter is encoded at the group level (hashed box) in a group/atoms/a element, allowing different charges to be assigned to the same atom based on local structure. In contrast, bonded parameters are matched to specific sets of atoms using patterns, allowing rules to be reused.

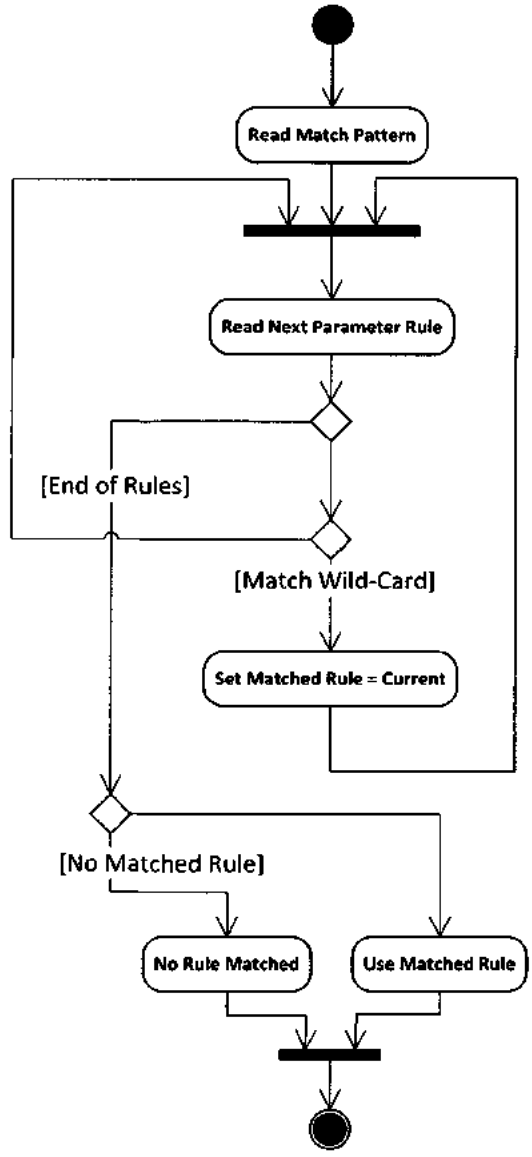


Figure 29. Parameter mask matching algorithm.

idx attributes and "double" for floating point values such as distances and charges. This allows downstream consumers of MMPL data to map these values to an appropriately typed data structure in their programming language of choice.

Structured cardinality constraints are used to ensure required elements are present and to explicitly define optional attributes and elements. For example, it is possible to define a valid parameter library that does not include torsion parameters, thus **tp** top level elements are

optional. However, if a torsion parameter is later added, additional constraints ensure that the **tp** element will contain all required fields.

Finally, relational constraints are enforced using XML Schema *key* and *keyref* concepts. For example, this allows the **ce** attribute of an atom element to be assigned only to one predefined chemical element. Similar restrictions allow groups to be composed from previously defined atoms, residues from groups, and so on. Currently, XML schema can only be used to define direct foreign key relationships. Indirect relationships, such as the atom index attribute in a residue pointing to a group atom index, cannot be expressed as an explicit key/keyref constraint (See Figure 27, relationship **E**). Encoding all of these constraints in the schema relieves MMPL client code from having to implement these error checks. It should be emphasized that although these constraints help reduce or eliminate certain types of errors, there are still many other errors that cannot be detected by these mechanisms.

MMPL Components and Extending the Parameter Library

MMPL is distributed as a set of files as outlined in Table 22 and accessible via http://www.dynameomics.org/mmpl/v2009/sample_parmlib.xml. The files listed in Table 22 of type "XML Fragment" contain the parameters defined in Levitt et al. (Levitt, 1995) encoded as MMPL elements. The schema defines a single XML namespace: http://www.dynameomics.org/schemas#mmpl. The mmpl document element is defined as a complex type containing an unbounded choice group, meaning the allowed element types can occur in any order. This element type enables the use of the Document Type Definition (DTD) system facility for including base definitions and then adding additional definitions and molecules. A sample empty parameter library, containing only base definitions is shown in Figure 30. A typical usage pattern is to store reusable components (such as new atom types) in a fragment file accessible via http, and then assemble molecules of interest in a local file. This local file should be edited using an XML aware editor, maintaining the assigned schema definition

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mmpl [
        <!ENTITY celements SYSTEM "http://mmpl/v2009/elements.xml">
        <!ENTITY atoms SYSTEM "http://mmpl/v2009/atoms.xml">
        <!ENTITY groups SYSTEM "http://mmpl/v2009/groups.xml">
        <!ENTITY residues SYSTEM "http://mmpl/v2009/residues.xml">
        <!ENTITY molecules SYSTEM "http://mmpl/v2009/molecules.xml">
        <!ENTITY bonds SYSTEM "http://mmpl/v2009/bonds.xml">
        <!ENTITY angles SYSTEM "http://mmpl/v2009/angles.xml">
        <!ENTITY torsions SYSTEM "http://mmpl/v2009/torsions.xml">
]>
<mmpl       name="empty"
        version="2009.03.13"
        xmlns="http://www.dynameomics.org/schemas#mmpl"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.dynameomics.org/schemas#mmpl
                http://www.dynameomics.org/mmpl/mmpl.xsd">
        &celements;
        &atoms;
        &groups;
        &residues;
        &molecules;
        &bonds;
        &angles;
        &torsions;
</mmpl>
```

Figure 30. A minimal parameter library. This valid XML document imports components from all the base fragment files using the Document Type Definition (DTD) SYSTEM import facility.

similar to the example. The resulting document can then be easily confirmed as being properly formed XML and explicitly validated against the schema. All validation issues should be resolved prior to using any parameter library—simulating a system based on an invalid MMPL document will yield unpredictable results.

The parameters included with MMPL are sufficient for many types of simulations, but they do not include many other parameters of interest such as metals. These can be added by creating additional XML fragment files with the atoms, groups, or residues needed. New parameters can take advantage of a key feature of the MMPL XML Schema—support for **com-**

Table 22. MMPL File Manifest.

File	Type	Description
angles.xml	XML Fragment	Standard library bond angle parameters
atoms.xml	XML Fragment	Standard library atoms
bonds.xml	XML Fragment	Standard library bond length parameters
elements.xml	XML Fragment	Definition of chemical elements and symbols
groups.xml	XML Fragment	Standard library groups
mmpl.xsd	W3C XML Schema	MMPL Annotated Schema
residues.xml	XML Fragment	Standard library amino acids
sample_parmlib.xml	XML	MMPL Standard Library example
torsions.xml	XML Fragment	Standard library torsion parameters

ment and **DOI** elements. The former allows descriptive text to be stored directly in the XML with the parameters, and the latter provides a standard mechanism to directly link a parameter to a literature citation. Use of both of these fields is optional but strongly encouraged. Once validated, new parameter files can be shared and easily incorporated into new simulation systems.

Conclusions

MMPL is a comprehensive data standard for representing molecular mechanics/dynamics force field parameters. It includes a rigorous and fully annotated XML Schema as well as an extensive library of previously published and validated parameters based on Levitt et al. (Levitt, 1995) and the F3C water model (Levitt, 1997). Key features include a component oriented design that allows atoms, groups, and residues to be assembled into complex chemical structures; comprehensive explicit constraints to prevent data errors; and data provenance through comments and DOI references. The data model and schema facilitate the independent development, sharing, and publication of new parameters for force fields similar to Eqs. (1-3)

and can easily be extended to accommodate parameters unique to other force fields, as has already been done for AMBER.

Chapter 7: Conclusions and Future Directions

The availability of multi-terabyte disk drives and processors with multiple 64 bit cores has made it both possible and less expensive than ever to build large, fast servers. Commercial and open source software can take advantage of these large, cheap servers; allowing very large data repositories to be created. However, software and hardware alone do not magically organize data into a model that can be easily mined and maintained. Instead, building large systems requires extensive planning and design to arrive at a flexible data model that can scale to the size of the data and to the processing capacity required to mine the data. It is inevitable that the initial design costs coupled with ongoing operational costs will be similar to or even exceed the cost of the sensor network or computing network that creates the data being stored. Despite this reality, data repository design, ongoing operation, and mining of data are considered to be somehow tangential to the "science" of a grant proposal. This perception must change if data intensive science is to succeed.

Paying for Storage Infrastructure

The previous chapters have described the methods and the design behind the Dynameomics data warehouse and laid out a path to building large scale repositories. However, these methods and design do not grapple with the fundamental economics of scientific research funding. With the notable exception of calculation resources (super computers, compute-clusters); storage infrastructure is not generally covered. The reasons for this are understandable. Building out and maintaining a large storage server facility, with power, cooling, and network connectivity for an undetermined period of time and with an unknown number of potential users cannot be easily justified or budgeted. At the same time, if a large data resource were made generally available, it could open doors to many areas of discovery not imagined by the original creators.

The fundamental problem is one of appropriate monetization of data infrastructure.

Labs tend to be largely capable of serving their own data needs, especially when rigorous data design methods and technologies are used. However, this model breaks when they attempt to share those data with other researchers. The two options have been either scaling up the primary data repository to support multiple labs, or replicating the data to multiple labs. In the scale up scenario, the original lab builds up their own infrastructure to support sharing partners, purchasing and deploying sufficient machines and network bandwidth to support their own needs and the needs of their collaborators. In the replication scenario, each sharing lab builds up their own infrastructure and effectively copies the original data for their own use. Both of these solutions are wrought with problems.

In the scaling scenario, beyond some number of collaborations (usually 0) the ability of the primary lab to simply absorb the cost of supporting partner labs will cease. This means that alternative arrangements must be made to share costs. These could include charging fees, jointly purchasing hardware, or sharing costs of power and cooling. These arrangements are complicated for a single partner, and grow exponentially with multiple partners. In the replication scenario, each lab must duplicate some portion of the original infrastructure and take on the role of building and maintaining it. In addition, as research continues, the data set at each site has the potential to diverge. This creates problems of ongoing synchronization and/or creating multiple independent versions of the data.

Cloud Computing

Cloud computing at first glance offers a third potential solution, placing data at third party that completely manages server rooms, machines, and network connectivity. As an example, Amazon's Elastic Compute Cloud (EC2) and Simple Storage Service (S3) (Amazon Web Services) are part of a larger "computing as a service" product line that allows anyone to rent whatever number of processors and storage they need and for whatever period of time is

required. Microsoft's Windows Azure (Microsoft Corporation, 2011) is another alterative, also supplying an effectively unlimited number of processors and storage on an as needed rental basis. These solutions effectively address the dynamic upsizing and downsizing of resources required by research; you simply rent exactly what you need for only as long as you need it. Unfortunately, the pricing models for these services are based on processor time, network bandwidth, and storage capacity over time. In a sense these services provide a better implementation of scaling and replication, but they do not help distribute the costs. However, a newly available cloud service can provide the means to solve this problem.

Amazon's S3 Infrastructure supports sharing of data restricted to requests marked with a "user pays" header. Although the owner is responsible for paying the monthly cost of maintaining S3 storage units (called buckets), it is possible for transfers to and from buckets to be billed to the user of the buckets, as opposed to the owner. The concept of user pays has been generalized and extended in Microsoft's Windows Azure Marketplace DataMarket (Microsoft Corporation), which supports the creation of data products available through free or paid subscriptions. In a sense the DataMarket takes on all the responsibility of managing access to the data—managing accounts, regulating access, providing a generalized data interface, and even collecting subscription revenue.

Moving to the Cloud

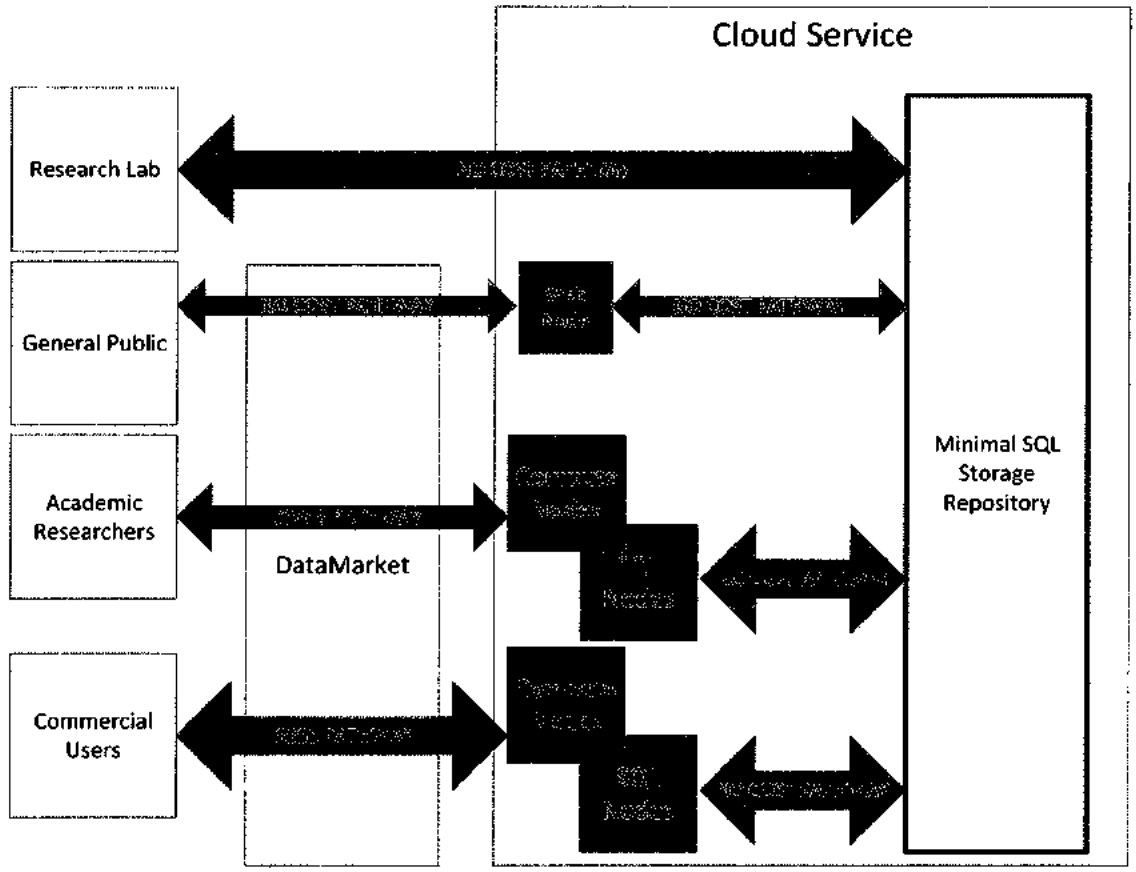
The Dynameomics Data Warehouse is currently hosted on 6 primary database servers ranging in size from 12TB to 25TB. These 6 servers contain 74 databases, over 90 thousand tables, and 725 billion rows of information. In addition, a small subset of simulations is hosted on two small externally facing machines (one web server, one database server). This is sufficient for the small number of external users who hit www.dynameomics.org, but these machines are limited to a few terabytes of storage and can only serve simple queries. In order to make all of

the data available to external users, a different approach is required.

Figure 31 describes a high level architecture that is based on several discussions held with Microsoft SQL Azure and DataMarket teams. The general idea is that minimal SQL storage would be provided to house the data in the cloud. This storage would be implemented using storage nodes only powerful enough to receive data from the lab, and to serve data to standard data nodes, which would be "rented" on-demand by consumers of the data. The DataMarket would provide user license validation (e.g. commercial or academic) and authentication, and optionally collect revenue from use of the data.

For the Dynameomics project, this model offers several benefits. First, the lab would gain an active offsite repository hosted at a professionally maintained, high availability data

Figure 31. Cloud services and repositories. An architecture to support large scale sharing of scientific data through cloud services.



center. Although it is anticipated that most analysis would continue to take place on local servers, this opens the door to utilizing compute nodes in the cloud. Second, lab could eliminate ongoing support of its externally facing web server and database server, replacing them with a small web server node that could brows the entire repository. Management of licenses and accounts would absorbed by the DataMarket. The general public gains a more reliable web server, and are no longer limited to just looking at a limited number of trajectories. For academic and commercial researchers, they gain access the entire data set. This allows them to select portions of interest and replicate it SQL Azure nodes they control and pay for. They can add compute nodes and do analysis in the cloud, or transfer the data to their own machines for analysis. Finally, the Microsoft cloud teams (DataMarket and Azure) gain customers through the rental of computing and SQL resources on their cloud services.

Conclusions

Dimensional modeling has shown great flexibility organizing protein simulation data. The implementation of the dimensional model in a relational database has successfully organized over 100TB of simulation data and continues to scale as more data are generated. The SQL Server Analysis Services (SSAS) implementation of Online Analysis Processing (OLAP) showed some potential as a tool for specific analyses, but appears to lack the scalability of SQL Server. Spatial indexing was shown to be highly effective optimization for developing analysis directly in the database. The Consensus Domain Dictionary demonstrated how domain knowledge can be linked across repositories. MMPL described an XML schema for organizing force field parameters. Finally, although sharing large data remains unsolved and a significant challenge; the potential of cloud services to change the economics of data intensive computing holds great promise.

BIBLIOGRAPHY

Adler J. 2010. R in a Nutshell. O'Reilly Media. Sebastopol, Calif.

Allen M. P. & D. J. Tildesley. 1989. Computer Simulation of Liquids. Oxford University Press. New York.

Alva V., K. K. Koretke, M. Coles et al. 2008. Cradle-loop barrels and the concept of metafolds in protein classification by natural descent. Curr. Opin. Struct. Biol. 18: 358-365.

Amazon Web Services. Amazon Web Services. 2011.

Amazon Web Services. 2010. AWS SDK for .NET API Reference. 12/28.

Amazon Web Services. 2010. AWS SDK for .NET Getting Started Guide.

Andreeva A., D. Howorth, J. M. Chandonia et al. 2008. Data growth and its impact on the SCOP database: new developments. Nucleic Acids Res. 36: D419-25.

Beck D. A. C., D. O. V. Alonso & V. Daggett. 2000-2011. in lucem Molecular Mechanics (ilmm).

Beck D. A. C. & V. Daggett. 2004. Methods for molecular dynamics simulations of protein folding/unfolding in solution. Methods. 34: 112-120.

Beck D. A. C., A. L. Jonsson, R. D. Schaeffer et al. 2008. Dynameomics: mass annotation of protein dynamics and unfolding in water by high-throughput atomistic molecular dynamics simulations. Protein Eng Des Sel. 21: 353-68.

Beck D. A. C. & V. Daggett. 2004. Methods for molecular dynamics simulations of protein folding/unfolding in solution. Methods in Enzymology. 34: 112-120.

Berman H., K. Henrick & H. Nakamura. 2003. Announcing the worldwide Protein Data Bank. Nat. Struct. Biol. 10: 980.

Berman H. M., J. Westbrook, Z. Feng et al. 2000. The Protein Data Bank. Nucleic Acids Res. 28: 235-242.

Bowers P. M., L. E. Schaufler & R. E. Klevit. 1999. A folding transition and novel zinc finger accessory domain in the transcription factor ADR1. Nat Struct Biol. 6: 478-85.

Branden C. & J. Tooze. 1999. Introduction to Protein Structure. Garland Publishing, Inc. New York, NY.

Brin S. & L. Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine.

Bromley D., S. Rysavy, D. A. Beck et al. 2010. DIVE: A Data Intensive Visualization Engine. In Microsoft Research eScience Workshop.

Brooks B. R., C. L. Brooks III, A. D. Mackerell et al. 2009. CHARMM: The Biomolecular Simulation Program. Journal of Computational Chemistry. 30: 1545-1614.

Celko J. 2005. Joe Celko's SQL for Smarties : Advanced SQL Programming. Morgan Kaufmann. Amsterdam ; Boston.

Celko J. 2008. Joe Celko's Thinking in Sets. Elsevier / Morgan Kaufmann. Amsterdam ; Boston.

Chandonia J. M., G. Hon, N. S. Walker et al. 2004. The ASTRAL Compendium in 2004. Nucleic Acids Res. 32: D189-92.

Chiti F. & C. M. Dobson. 2006. Protein misfolding, functional amyloid, and human disease. Annu Rev Biochem. 75: 333-66.

Clarkson K. 2005. Nearest-neighbor searching and metric space dimensions. In Nearest-Neighbor Methods for Learning and Visions: Theory and Practice. Anonymous MIT press. Cambridge, MA.

Codd E. F. 1970. A relational model of data for large shared data banks. Commun ACM. 13: 377-387.

Codd E. F., S. B. Codd & C. T. Salley. 1993. Providing OLAP to User-Analysts: An IT Mandate.

Cornell W. D., P. Cieplak, C. I. Bayly et al. 1995. A Second Generation Force Field for the Simulation of Proteins, Nucleic Acids, and Organic Molecules. J. Am. Chem. Soc. 117: 5179-5197.

Coulson A. F. & J. Moult. 2002. A unifold, mesofold, and superfold model of protein fold use. Proteins. 46: 61-71.

CRC Press. 2010. CRC Handbook of Chemistry and Physics. CRC Press.

Csaba G., F. Birzele & R. Zimmer. 2009. Systematic comparison of SCOP and CATH: a new gold standard for protein structure analysis. BMC Struct. Biol. 9: 23.

Cuff A., O. C. Redfern, L. Greene et al. 2009. The CATH hierarchy revisited-structural divergence in domain superfamilies and the continuity of fold space. Structure. 17: 1051-1062.

Daggett V. 2002. Molecular dynamics simulations of the protein unfolding/folding reaction. Acc. Chem. Res. 35: 422-429.

David M. 1999. Advanced ANSI SQL Data Modeling and Structure Processing. Artech House. Boston.

Day R., D. A. C. Beck, R. S. Armen et al. 2003. A consensus view of fold space: combining SCOP, CATH, and the Dali Domain Dictionary. Protein Sci. 12: 2150-2160.

Dietmann S., J. Park, C. Notredame et al. 2001. A fully automatic evolutionary classification of protein folds: Dali Domain Dictionary version 3. Nucleic Acids Res. 29: 55-57.

Dietrich S. W. & S. D. Urban. 2005. An Advanced Course in Database Systems Beyond Relational Databases. Pearson. Upper Saddle River, NJ.

Dudley J. T. & A. J. Butte. 2009. A quick guide for developing effective bioinformatics programming skills. PLoS computational biology. 5: e1000589.

Elmasri R. & S. B. Navathe. 2005. Fundamentals of Database Systems. Addison Wesley.

Fersht A. R. 1999. Structure and Mechanism in Protein Science. W. H. Freeman and Company. New York, NY.

Fersht A. R. & V. Daggett. 2002. Protein folding and unfolding at atomic resolution. Cell. 108: 573-82.

Fletcher P. T., C. Lu, S. M. Pizer et al. 2004. Principal geodesic analysis for the study of nonlinear statistics of shape. IEEE Trans. Med. Imaging. 23: 995-1005.

Frenkel M., R. D. Chirico, V. V. Diky et al. 2003. ThermoMLAn XML-Based Approach for Storage and Exchange of Experimental and Critically Evaluated Thermophysical and Thermochemical Property Data. 1. Experimental Data. Journal of Chemical & Engineering Data. 48: 2-13.

Fritchey G. & S. Dam. 2009. SQL Server 2008 Query Performance Tuning Distilled. Apress. New York.

Garey M. R. & D. S. Johnson. 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman.

Giudice E. & R. Lavery. 2002. Simulations of nucleic acids and their complexes. Acc. Chem. Res. 35: 350-357.

Gorbach I., A. Berger & E. Melomed. 2009. Microsoft SQL Server 2008 Analysis Services Unleashed.

Greene L. H., T. E. Lewis, S. Addou et al. 2007. The CATH domain structure database: new protocols and classification levels give a more comprehensive resource for exploring evolution. Nucleic Acids Res. 35: D291-7.

Grishin N. V. 2001. Fold change in evolution of protein structures. J. Struct. Biol. 134: 167-185.

Hadley C. & D. T. Jones. 1999. A systematic comparison of protein structure classifications: SCOP, CATH and FSSP. Structure. 7: 1099-1112.

Haile J. M. 1992. Molecular Dynamics Simulation: Elementary Methods. Wiley. New York.

Hansson T., C. Oostenbrink & W. van Gunsteren. 2002. Molecular dynamics simulations. Curr. Opin. Struct. Biol. 12: 190-196.

Harinath S., M. Carroll, S. Meenakshisundaram et al. 2009. Professional Micrsoft SQL Server Analaysis Services 2008 with MDX. Wiley Publishing, Inc. Indianapolis, IN.

Harold E. R. 2004. Effective XML. Addison-Wesley. Boston, MA.

Henrick K., Z. Feng, W. F. Bluhm et al. 2008. Remediation of the protein data bank archive. Nucleic Acids Res. 36: D426-33.

Henry E. R., M. Levitt & W. a. Eaton. 1985. Molecular dynamics simulation of photodissociation of carbon monoxide from hemoglobin. Proc. Natl. Acad. Sci. U. S. A. 82: 2034-2038.

Holand S. M. 2008. Principal Components Analysis (PCA). 2011: 8.

Holland T. A., S. Veretnik, I. N. Shindyalov et al. 2006. Partitioning protein structures into domains: why is it so difficult? J. Mol. Biol. 361: 562-590.

Holm L., S. Kaariainen, P. Rosenstrom et al. 2008. Searching protein structure databases with DaliLite v.3. Bioinformatics. 24: 2780-2781.

Hubbard T. J., B. Ailey, S. E. Brenner et al. 1999. SCOP: a Structural Classification of Proteins database. Nucleic Acids Res. 27: 254-256.

Hucka M., A. Finney, H. M. Sauro et al. 2003. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics. 19: 524-531.

IEEE Computer Society Standards Committee, IEEE Standards Board & American National Standards Institute. 1985. IEEE standard for binary floating-point arithmetic.

Intel Corporation. 2007. First the Tick, Now the Tock: Intel® Microarchitecture (Nehalem).

Intel Corporation. 2009. An Introduction to the Intel® QuickPath Interconnect. : 1-22.

International Organization for Standardization & International Electrotechnical Commission. 2001. Information Technology: Database Languages: SQL. Part 1, Framework (SQL/framework). ISO/IEC. Geneva.

International Standards Organization. 199. SQL99. In Anonymous .

Janert P. K. 2010. Data Analysis with Open Source Tools. O'Reilly. Farnham.

Jefferson E. R., T. P. Walsh & G. J. Barton. 2008. A comparison of SCOP and CATH with respect to domain-domain interactions. Proteins. 70: 54-62.

Jones N. C. & P. A. Pevzner. 2004. An Introduction to Bioinformatics Algorithms. MIT Press. Boston, MA.

Jungwirth P. & D. J. Tobias. 2002. Ions at the air/water interface. J Phys Chem B. 106: 6361-6373.

Karplus M. 2003. Molecular dynamics of biological macromolecules: a brief history and perspective. Biopolymers. 68: 350-358.

Karplus M. & J. A. McCammon. 2002. Molecular dynamics simulations of biomolecules. Nat. Struct. Biol. 9: 646-652.

Karplus M. & J. Kuriyan. 2005. Molecular dynamics and protein function. Proc Natl Acad Sci U S A. 102: 6679-85.

Kazmirski S., A. Li & V. Daggett. 1999. Analysis Methods for Comparison of Molecular Dynamics Trajectories: Applications to Protein Unfolding Pathways and Denatured Ensembles. Journal of Molecular Biology. 290: 283-283-304.

Kazmirski S. L. & V. Daggett. 1998. Non-native interactions in protein folding intermediates: molecular dynamics simulations of hen lysozyme. J. Mol. Biol. 284: 793-806.

Kazmirski S. L., A. Li & V. Daggett. 1999. Analysis methods for comparison of multiple molecular dynamics trajectories: applications to protein unfolding pathways and denatured ensembles. J. Mol. Biol. 290: 283-304.

Kehl C., A. M. Simms, R. D. Toofanny et al. 2008. Dynameomics: a multi-dimensional analysis-optimized database for dynamic protein data. Protein Eng Des Sel. 21: 379-86.

Kendrew J. C. 1959. Structure and function in myoglobin and other proteins. Fed. Proc. 18: 740-751.

Kimball R., L. Reeves, M. Ross et al. 1998. The Data Warehouse Lifecycle Toolkit. Wiley Pub.

Kimball R. & M. Ross. 2002. The Data Warehouse Toolkit : The Complete Guide to Dimensional Modeling. Wiley. New York.

Kolodny R., D. Petrey & B. Honig. 2006. Protein structure comparison: implications for the nature of 'fold space', and structure and function prediction. Curr. Opin. Struct. Biol. 16: 393-398.

Kremer K. 2003. Computer Simulations for Macromolecular Science. Macromolecular Chemistry and Physics. 204: 257-264.

Langtangen H. P. 2009. A Primer on Scientific Programming with Python. Springer Berlin Heidelberg. Berlin, Heidelberg.

Lefebvre S. & H. Hoppe. 2006. Perfect Spatial Hashing. ACM Transactions on Graphics. 25: 579-588.

Levitt M. 1983. Molecular dynamics of native protein. I. Computer simulation of trajectories. J. Mol. Biol. 168: 595-617.

Levitt M. 1983. Molecular dynamics of native protein. II. Analysis and nature of motion. J. Mol. Biol. 168: 621-657.

Levitt M. & C. Chothia. 1976. Structural patterns in globular proteins. Nature. 261: 552-558.

Levitt M., M. Hirshberg, R. Sharon et al. 1995. Potential energy function and parameters for simulations of the molecular dynamics of proteins and nucleic acids in solution. Comput. Phys. Commun. 91: 215-231.

Levitt M., M. Hirshberg, R. Sharon 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: 5051-5061.

Levitt M. & H. Meirovitch. 1983. Integrating the equations of motion. J. Mol. Biol. : 617-620.

Lifson S. & A. Warshel. 1968. Consistent Force Field for Calculations of Conformations , Vibrational Spectra , and Enthalpies of Cycloalkane and n-Alkane Molecules. Chem. Phys. 49: 5116.

Lin F. & R. Wang. 2010. Systematic Derivation of AMBER Force Field Parameters Applicable to Zinc-Containing Systems. Journal of Chemical Theory and Computation. 6: 1852-1870.

Lutz M. 2009. Learning Python. O'Reilly Media. Sebastopol, Calif.

Mahé P. & J. Vert. 2009. Virtual Screening with Support Vector Machines and Structure Kernels. Comb. Chem. High Throughput Screen. 12: 409-423.

Majumdar I., L. N. Kinch & N. V. Grishin. 2009. A database of domain definitions for proteins with complex interdomain geometry. PLoS One. 4: e5084.

McCammon J. A., B. R. Gelin & M. Karplus. 1977. Dynamics of folded proteins. Nature. 167: 585-590.

Menasce D. & V. Almeida. 2000. Scaling for E-Business, Technologies, Models, Perforamnce, and Capacity Planning. Prentice Hall, PTR. Upper Saddle River, NJ.

Microsoft Corporation. Windows Azure Marketplace DataMarket. 2011.

Microsoft Corporation. 1997. OLEDB for OLAP.

Microsoft Corporation. 2007. SQL Server 2008. 2008 Enterprise Edition R2 x64.

Microsoft Corporation. 2007. SQL Server 2008 Analysis Services. 2008 Enterprise Edition R2 x64.

Microsoft Corporation. 2007. Windows. 2008 Server R2 Enterprise Edition x64.

Microsoft Corporation. 2010. SQL Server Books Online.

Microsoft Corporation. 2011. Windows Azure. 2011.

Microsoft Corporation & Hyperion Corporation. 2002. XML for Analysis Specification. : 1.

Muenchen R. A. & J. Hilbe. 2010. R for Stata Users. Springer. New York.

Murray-Rust P. & H. S. Rzepa. 1999. Chemical Markup, XML, and the Worldwide Web. 1. Basic Principles. J. Chem. Inf. Comput. Sci. 39: 928-942.

Murthy H. M., S. Clum & R. Padmanabhan. 2009. Dengue virus NS3 serine protease. Crystal structure and insights into interaction of the active site with substrates by molecular modeling and structural analysis of mutational effects. J. Biol. Chem. 284: 34468.

Murzin A. G., S. E. Brenner, T. Hubbard et al. 1995. SCOP: a structural classification of proteins database for the investigation of sequences and structures. J. Mol. Biol. 247: 536-540.

Nagano N., C. A. Orengo & J. M. Thornton. 2002. One fold with many functions: the evolutionary relationships between TIM barrel families based on their sequences, structures and functions. J. Mol. Biol. 321: 741-765.

Norberg J. & L. Nilsson. 2002. Molecular dynamics applied to nucleic acids. Acc. Chem. Res. 35: 465-472.

Orengo C. A., A. D. Michie, S. Jones et al. 1997. CATH—a hierarchic classification of protein domain structures. Structure. 5: 1093-1108.

Pascual-Garcia A., D. Abia, A. R. Ortiz et al. 2009. Cross-over between discrete and continuous protein structure space: insights into automatic classification and networks of protein structures. PLoS Comput. Biol. 5: e1000331.

Patel S., A. D. Mackerell & C. L. Brooks. 2004. CHARMM fluctuating charge force field for proteins: II protein/solvent properties from molecular dynamics simulations using a nonadditive electrostatic model. Journal of computational chemistry. 25: 1504-1514.

Pearlman D., D. Case, J. Caldwell et al. 1995. AMBER, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules. Comput. Phys. Commun. 91: 1-41.

Pearson W. 2008. Dimensional Model Components: Dimensions Part I. Database Journal.

Pearson W. 2009. Introduction to Attribute Discretization. Database Journal.

Perutz M. F., M. G. ROSSMANN, A. F. CULLIS et al. 1960. Structure of haemoglobin: a three-dimensional Fourier synthesis at 5.5-A. resolution, obtained by X-ray analysis. Nature. 185: 416-422.

Phillips D. C. 1967. The hen egg-white lysozyme molecule. Proc Natl Acad Sci USA. 57: 483-483-495.

Phillips J. C., R. Braun, W. Wang et al. 2005. Scalable molecular dynamics with NAMD. Journal of computational chemistry. 26: 1781-1802.

Rueda M., C. Ferrer-Costa, T. Meyer et al. 2007. A consensus view of protein dynamics. Proc. Natl. Acad. Sci. U. S. A. 104: 796-801.

Rutherford K. & V. Daggett. 2009. A hotspot of inactivation: The A22S and V108M polymorphisms individually destabilize the active site structure of catechol O-methyltransferase. Biochemistry. 48: 6450-6460.

Saiz L., S. Bandyopadhyay & M. L. Klein. 2002. Towards an understanding of complex biological membranes from atomistic molecular dynamics simulations. Biosci. Rep. 22: 151-173.

Sam V., C. H. Tai, J. Garnier et al. 2006. ROC and confusion analysis of structure comparison methods identify the main causes of divergence from manual protein classification. BMC Bioinformatics. 7: 206.

Schaeffer R. D., A. L. Jonsson, A. M. Simms et al. 2011. Generation of a Consensus Protein Domain Dictionary. Bioinformatics. 27: 46-54.

Schaid D. J. 2010. Genomic Similarity and Kernel Methods I: Advancements by Building on Mathematical and Statistical Foundations. Hum. Hered. 70: 109-131.

Shimizu H., S. Park, D. Lee et al. 2000. Crystal structures of cytochrome P450nor and its mutants (Ser286—>Val, Thr) in the ferric resting state at cryogenic temperature: a comparative analysis with monooxygenase cytochrome P450s. J. Inorg. Biochem. 81: 191-205.

Simms A. M., D. A. C. Beck, A. L. Jonsson et al. 2011. The Molecular Mechanics Parameter Markup Language (in preparation).

Simms A. M. & V. Daggett. 2011. Protein Simulation Data in the Relational Model (in preparation).

Simms A. M., R. D. Toofanny, C. Kehl et al. 2008. Dynameomics: design of a computational lab workflow and scientific data repository for protein simulations. Protein engineering, design & selection. 21: 369-377.

Smith B. C., R. C. Clay & Hitachi Consulting. 2009. SQL Server 2008 MDX Step by Step. Microsoft Press.

Sowdhamini R. & T. L. Blundell. 1995. An automatic method involving cluster analysis of secondary structures for the identification of domains in proteins. Protein Sci. 4: 506-520.

Stephenson F. H. 2003. Calculations for Molecular Biology and Biotechnology A Guide to Mathematics in the Laboratory. Academic Press. San Diego, CA.

Stonebraker M., L. A. Rowe, B. G. Lindsay et al. 1990. Third-generation database system manifesto. SIGMOD Rec. 19: 31-44.

Tennick A. 2010. Practical MDX Queries for Microsoft SQL Server Analysis Services. McGraw-Hill.

Toofanny R. D., A. M. Simms, D. A. C. Beck et al. 2011. Implementation of 3D spatial indexing and compression in a large-scale molecular dynamics simulation database for rapid atomic contact detection (in preparation).

UCLA: Academic Technology Services, Statistical Consulting Group. Regression with Stata. 2011.

Valas R. E., S. Yang & P. E. Bourne. 2009. Nothing about protein structure classification makes sense except in the light of evolution. Curr. Opin. Struct. Biol. 19: 329-334.

van der Kamp M. W., R. D. Schaeffer, A. L. Jonsson et al. 2010. Dynameomics: a comprehensive database of protein dynamics. Structure. 18: 423-35.

van Gunsteren W. F. & H. J. C. Berendsen. 1990. Computer Simulation of Molecular Dynamics: Methodology, Applications, and Perspectives in Chemistry. Angewandte Chemie International Edition in English. 29: 992-1023.

Vapnik V. N. 1999. An overview of statistical learning theory. IEEE Trans. Neural Netw. 10: 988-999.

Veretnik S., P. E. Bourne, N. N. Alexandrov et al. 2004. Toward consistent assignment of structural domains in proteins. J. Mol. Biol. 339: 647-678.

W3C.org. 2004. XML Schema Part 1: Structures Second Edition.

W3C.org. 2004. XML Schema Part 2: Datatypes Second Edition.

Wagner B. 2010. Effective C#: 50 Specific Ways to Improve Your C#, 2nd Ed. Pearson Education. Boston, MA.

Wang J., P. Cieplak & P. A. Kollman. 2000. How well does a restrained electrostatic potential (RESP) model perform in calculating conformational energies of organic and biological molecules? Journal of Computational Chemistry. 21: 1049-1074.

Wang W., O. Donini, C. M. Reyes et al. 2001. Biomolecular simulations: recent developments in force fields, simulations of enzyme catalysis, protein-ligand, protein-protein, and protein-nucleic acid noncovalent interactions. Annu. Rev. Biophys. Biomol. Struct. 30: 211-243.

Warshel A. 2002. Molecular dynamics simulations of biological reactions. Acc. Chem. Res. 35: 385-395.

Webb C., A. Ferrari & M. Russo. 2009. Expert Cube Development with Microsoft SQL Server 2008 Analysis Services.

Westbrook J., N. Ito, H. Nakamura et al. 2005. PDBML: the representation of archival macromolecular structure data in XML. Bioinformatics. 21: 988-992.

Westreich D., J. Lessler & M. J. Funk. 2010. Propensity score estimation: neural networks, support vector machines, decision trees (CART), and meta-classifiers as alternatives to logistic regression. J. Clin. Epidemiol. 63: 826-833.

Wetlaufer D. B. 1973. Nucleation, rapid folding, and globular intrachain regions in proteins. Proc. Natl. Acad. Sci. U. S. A. 70: 697-701.

White T. 2009. Hadoop. O'Reilly Media, Inc. Sebastopol, Calif.

Whitehorn M., R. Zare & M. Pasumansky. 2006. Fast Track to MDX. Springer. New York ; London.

Wilf H. S. Algorithms and Complexity. Prentice Hall. Englewood Cliffs, NJ.

Wolf Y. I., N. V. Grishin & E. V. Koonin. 2000. Estimating the number of protein folds and families from complete genome data. J. Mol. Biol. 299: 897-905.

Wyke R. A. & A. Watt. 2002. XML Schema Essentials. John Wiley & Sons, Inc.

VITA

Andrew M. Simms was born in Ann Arbor, Michigan. He has lived in several states, including New Jersey, California, Massachusetts, Arizona, and Washington. At the University of Michigan, he earned a Bachelor of Science in Computer Science, and a Master of Science in Biomedical Health Informatics at the University of Washington, School of Medicine. In 2011, he earned a Doctor of Philosophy at the University of Washington School of Medicine in Biomedical Health Informatics.