Modeling ToolKit++ (MTK++)
v0.2.0

April 2010
Last revised, March 2011

List of Tables

List of Figures

Authors

Martin B. Peters (martin.b.peters@gmail.com)
Kenneth Ayers
Andrew Wollacott
Duane E. Williams
Benjamin P. Roberts
Kenneth M. Merz Jr. (merz@qtp.ufl.edu)

Contributors

Yue Yang
Daniel Sindhikara
Mark Benson
Roger Martin
Jianzhong Liu
Xiaohua Zhang
Lance Westerhoff

Acknowledgments

The authors would like to thank the NIH and NSF for funding this software development.

1 Introduction

Here we outline the design and development of a C++ package called Modeling ToolKit++ (MTK++). MTK++ was designed from the ground up to be used in areas of metalloprotein modelling and *in silico* Structure Based Drug Design (SBDD).

This package contains functions to handle molecular structures ranging from proteins to small molecules, that may be utilized to calculate molecular mechanics energies and gradients, to perceive atom hybridizations, and evaluate bond orders, formal charges, rings and functional groups. Utilities to add Hydrogen atoms to structures have been developed; this code was created to deal with metalloprotein systems where no other software could satisfactorily do so.

MTK++ also has the capability to perform conformational searching of drug molecules using a systematic approach where the molecular mechanics code was a prerequisite. Also an algorithm to perform clique detection of molecular features was implemented to superimpose two molecular species on to one another for use in ligand and receptor based drug design. Additionally, the MTK++ package contains other general purpose libraries for parameter optimization, graph utilities, and statistical methods.

2 Design

MTK++ is an object oriented C++ package of molecular modeling libraries including Molecular Mechanics (MM), Genetic Algorithm (GA), file processing and conversion (Parsers), statistical and molecular tools to be used in SBDD and other computational chemistry fields. The Basic Linear Algebra Subprograms (BLAS), Linear Algebra PACKage (LAPACK), Boost, and eigen3 were used in the development. XML parsing libraries tinyxml, xerces-c [1], and Qt can be used.

2.1 Library Hierarchy

Figure 1 shows the hierarchy of the MTK++ package. At the center of the package of libraries is a group of utility routines which are used in all other packages. These include constants definition, diagonalization functions, an indexing class for easy sorting of objects, and a class called vector3d for atomic coordinate storage and transformation. The Parsers library takes care of reading and writing of files and it requires the Molecule and GA Libraries. Also the Molecule library uses the Graph library for ring perception and other recursive functions. The design of the individual libraries is discussed further in the sections below.

2.2 Molecule Library

The core of the MTK++ package is the Molecule library and its most important classes are shown in Fig. 2. This library can handle multiple molecules at a time and these are stored in the collection class. The collection class also takes care of all the elements (this information only needs to be stored once, not for every molecule), and parameter and fragment information for MM calculations. The molecules themselves are of type “molecule” and this class stores submolecule or residue information. This division is analogous to that of amino acids in protein or nucleotides in

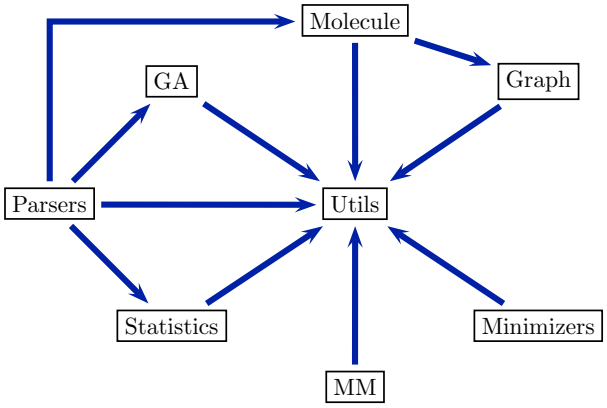


Figure 1: Library Hierarchy as Implemented in MTK++. The Library where the tail of the arrow starts uses the library where the head of the arrow ends, e.g. The Parsers library uses the Molecule, Utils, Statistics, and GA Libraries.

DNA or in fact fragments in small organic molecule. The submolecule class stores a list of atoms and the atom class stores pointers to objects such as its element and coordinates which are a vector of three double precision numbers (vector3d).

The parameters class stores information for MM calculations as structures, such as atom types, bond, angle, torsion, improper (force constants, equilibrium values) and non-bonded (charges, Lennard-Jones values) parameters as shown in Fig. 3. The stdLibrary class is the main object which deals with the storage and function of a molecular fragment library as shown in Fig. 4. stdLibrary stores a list of stdGroups and a stdGroup is a storage container for stdFrag's. For example a stdGroup could store the 20 amino acids, each a stdFrag, of proteins or a list of functional groups in drug design. The stdFrag class contains information about it atoms, stdAtom, bonds, stdBonds, features, stdFeature, etc.

The functionality available to molecules such as proteins, DNA, and small organics originate from the molecule class in the Molecule library as shown in Fig. 7. molecule stores a lists of bonds, (a vector of Bond objects in C++), angles, torsions, and impropers. The connectivity information is determined in the connections class. This class can perceive bonds using distance and other geometric information, and also determine bonds through user defined databases of molecular structures. For example the connectivity of an alanine residue in a protein doesn't need to be perceived since it is known *a priori* if the names of the atoms are known. Disulfide bonds between Cysteine residues, as shown in Fig. 5, of proteins are automatically perceived using

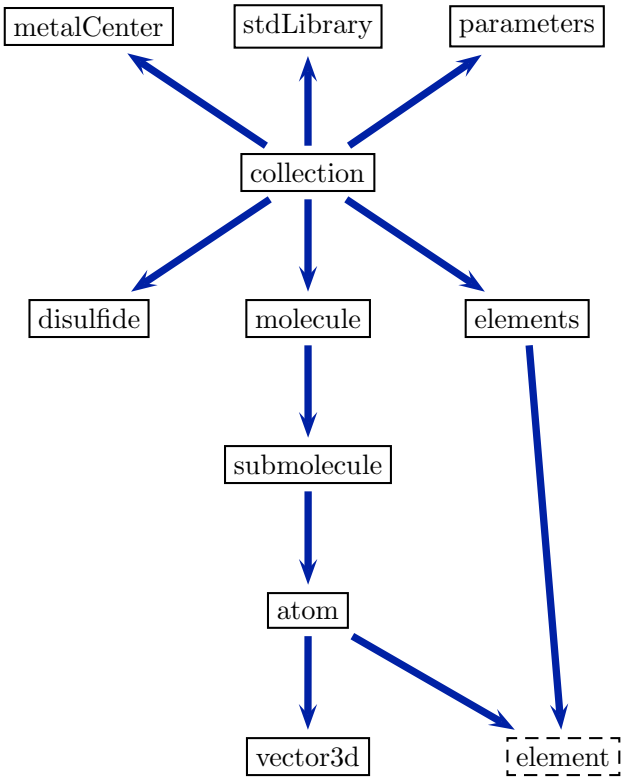


Figure 2: Core Class Hierarchy of the Molecule Library as Implemented in MTK++. Solid line boxes denotes a class, while a dashed box signifies a structure. A class where the tail of the arrow starts uses or contains a class or structure where the head of the arrow ends. e.g. The elements class contains or uses the element structure.

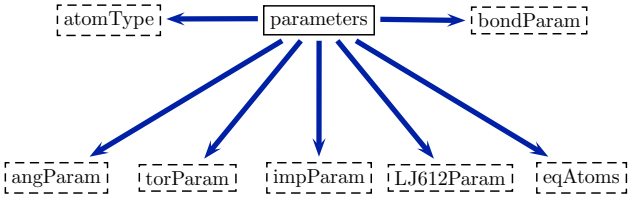


Figure 3: Class Hierarchy of the Parameters Component of the Molecule Class as Implemented in MTK++. Solid line boxes denotes a class, while a dashed box signifies a structure. A class where the tail of the arrow starts uses or contains a class or structure where the head of the arrow ends. e.g. The parameters class contains or uses the atomType structure.

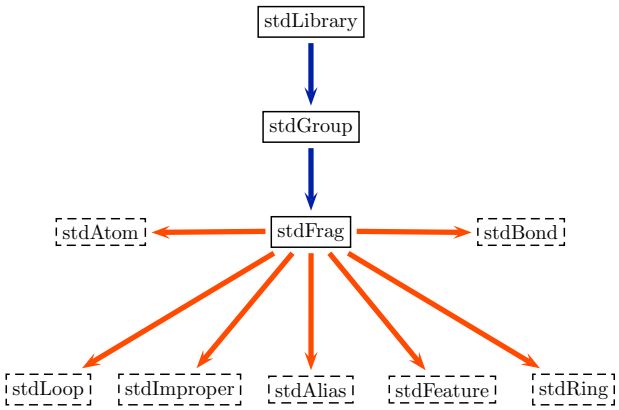


Figure 4: Class Hierarchy of the Standard Library Component of the Molecule Class as Implemented in MTK++. Solid line boxes denotes a class, while a dashed box signifies a structure. A class where the tail of the arrow starts uses or contains a class or structure where the head of the arrow ends. e.g. The stdFrag class contains or uses the stdAtom structure.

the parameters in table 1 [2]. If the Cysteine's SG atoms are within *dCutoff* of each other and $S - S_{Energy}$ from Eq. 1 is less than *eCutoff* they are considered bonded. The protonation states of Histidine residues bound to a metal ion are also perceived using a bond distance cutoff of 2.3 Ångström. If the HIS@NE2 (epsilon nitrogen of Histidine) atom is within this cutoff the residue is set to HID type. If the HIS@ND1 is within this cutoff the residue is set to HIE type. If both HIS@NE2 and HIS@ND1 are bonded to a metal atom within 2.3 Å then the residue is set to HIN type such as the bridging histidine residue in Copper-Zinc Superoxide Dismutase [3]. Bond order, hybridization and formal charge of atoms for small molecule are determined in the hybridize class which is discussed in more detail below in section 3.

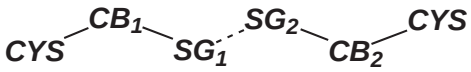


Figure 5: Disulfide Bond in Proteins.

$$S - S_{Energy} = E^{Bond}_{SG1-SG2} + E^{Angle}_{CB1-SG1-SG2} + E^{Angle}_{CB2-SG2-SG1} \quad (1)$$

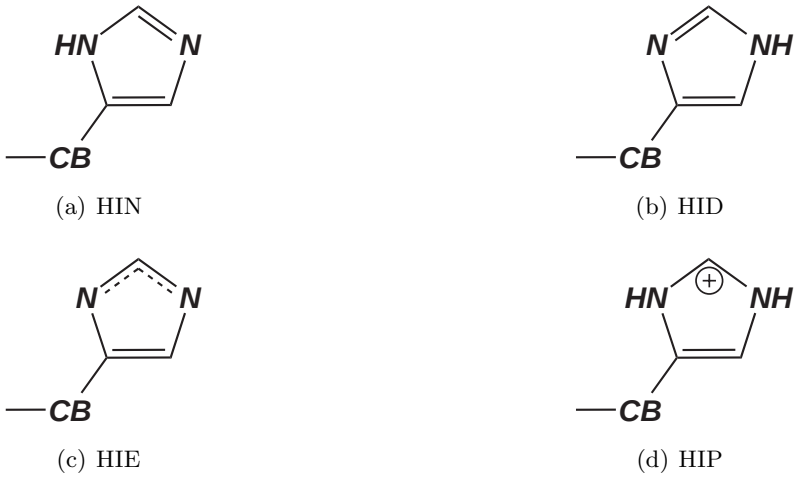
$$where:$$

$$E^{Bond}_{SG-SG} = ssBondKeq * (distance_{SG-SG} - ssBondReq)^2 \quad (2)$$

$$E^{Angle}_{CB-SG-SG} = CBSGSGKeq * (angle_{CB-SG-SG} - CBSGSGReq)^2 \quad (3)$$

Table 1: Disulfide Bond Prediction Parameters.

Parameter	Value
$dCutoff$	2.5
$ssBondReq$	2.038
$ssBondKeq$	166.0
$CBSGSGReq$	103.7
$CBSGSGKeq$	68.0
$eCutoff$	30.0



(a) HIN (b) HID

(c) HIE (d) HIP

Figure 6: The Structural Types of the Histidine Residue.

Ring moieties are perceived within the rings class and each ring found is stored in a ring structure. The perception of rings is discussed further in section 4. The functionalize class determines which functional groups are present in a molecule using a predefined database of fragments. The implementation details of the functionalize class is outlined in section 7.

The fingerprint class contains rudimentary functionality for molecular fingerprinting. A fingerprint is defined as information that describes a molecule in 1-D. The fingerprint in MTK++ is represented as a vector of integers with the following form: "atom info, bond type, # of rings ring info". The number of atoms from Hydrogen through Iodine are stored in the first 52 positions, another 52 positions store the number of each of the following bond types B-H, C-H, N-H, O-H, S-H, B-C, B=C, B-O, B-N, B-O, B-F, B-S, B-Cl, B-Br, B-I, C-C, C=C, C%C, N-N, N=N, C-N, C=N, C%N, N-O, N=0, N-P, N-Se, N=Se, O-O, C-O, C=O, O-Si, O-S, O=S, O-Se, O=Se, C-F, S-S, C-S, C=S, S-N, C-Cl, P-P, P-C, P-O, P=O, P-S, P-Se, Se-Se, C-Se, C=Se, N-Se, where "-", "=", "%" denote a single, double and triple bond respectively. Finally the 105^{th} position stores the number of rings in the molecule or fragment. The size, planarity, aromaticity, heterocyclic boolean,

and the number of nitrogens, oxygens and sulfur atoms of each ring is also stored after the 105^{th} position. Thus the length of the vector depends on the number of rings present in the molecule or fragment. Fingerprinting in MTK++ is primarily used in conjunction with the functionalize class. Fingerprints are used to screen out fragments that could not be apart of a molecule based on elements, bond types, and rings present, thus speeding up the functionalization of molecules.

A pharmacophore is commonly defined as the three dimensional geometric arrangement of molecular features that are necessary for biological activity. Pharmacophores between two molecules are detected using a feature (H-bond Donor/Acceptor, Pi Center, Positive/Negative Center, Hydrophobicity) matching algorithm in the pharmacophore class. The features common to both molecules are stored in a clique structure. A full description of the algorithm implemented in MTK++ is outlined in section 8.

The protonate class carries out the addition of Hydrogen atoms to macromolecules (proProtonate), ligands (ligProtonate), and water (watProtonate) molecules. proProtonate uses information from user defined libraries to add Hydrogens while ligProtonate is used when no such library is available. Water molecules often surround structures derived from X-ray crystallographic data but no Hydrogen atom positions are provided. Hydrogens are added separately to water molecules after they are added to macromolecules and ligands. The algorithmic details of the three protonate classes are described in section 5.

Conformational searching of drug-like molecules is carried out in the conformer class using graph theory methods. Each conformer of a molecule is stored in a conformer structure. The internal workings of this class are described in section 6. A integral part of conformational searching is determine the amount of conformational space sampled. To determine this requires being able to superimpose a conformer onto some reference structure and calculate the root-mean-squared deviation. The superimposition of two molecules is carried out in the superimpose class and is discussed below in section 9.

The selection class is used to parse strings that represent subsets of molecular data and is essential in providing an API for users of MTK++. The data structure in the Molecule library has a hierarchical definition. Atom information is stored in submolecule; bonds, angles, torsions, impropers, and submolecules are stored in molecule and finally all molecules are stored in collection. The atom class is at the bottom of the hierarchy, while collection is at the top. Thus to retrieve for example all atoms which a specific name in all molecules of the collection would require a certain syntax. The syntax used in the selection class resembles that of a UNIX operating system such as "/collection/molecule/submolecule/atom" For example, providing the following string: "/COL/MOL/ALA-10/.CA." would select the atom ".CA.", alpha carbon, in alanine with residue number 10 (ALA-10) and that's part of the molecule named MOL in the COL collection. The "/" on the left hand side of the string assumes that the selection is starting from the top of the structural hierarchy. The following selection string does not begin with a slash: "ALA-10/.CA." and represents parsing the hierarchy from the bottom up; all alpha carbons of molecules in the collection with alanine at position 10 will be selected. This syntax can handle molecule/ submolecule/ atom names, numbers, or a combination (name-number), such as ALA-10.

The atomTyper class assigns molecular mechanics atom types to the atoms of a molecule using user defined fragment libraries. The hydrophobic regions of molecules is determined using an atom

additive approach as outlined by Wang and Zhou in 1998 [4].

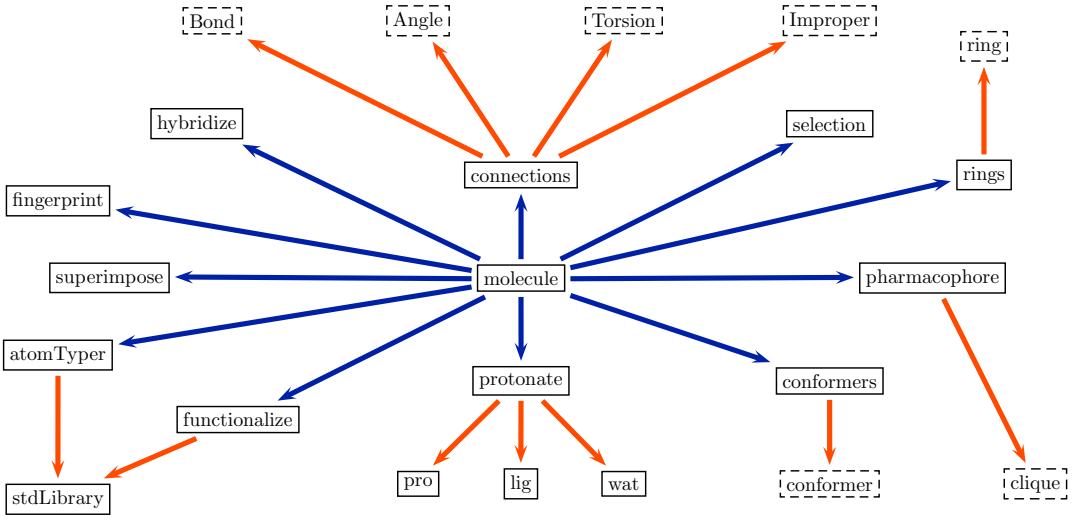


Figure 7: Class Hierarchy of the Molecule Component of the Molecule Class as Implemented in MTK++. Solid line boxes denotes a class, while a dashed box signifies a
structure. A class where the tail of the arrow starts uses or contains a class or structure where the head of the arrow ends. e.g. The molecule class contains or uses the hybridize class.

2.3 Graph Library

2.3.1 Graph Theory

Many problems in cheminformatics such as finding the shortest path from one atom to another, ring and substructure searching are solved using graph theory and recursive algorithms [5]. Graph theory is a well establish area and is commonly used in computer networking [6]. A graph G consists of a set of n vertices, V, and a set of m edges, E, where an edge is an unordered pair of vertices. $V = \{v_1, v_2, v_3, v_4, v_5, ..., v_n\}$, $E = \{e_{12}, e_{23}, e_{34}, e_{37}, ..., e_m\}$, and $G = \{V, E\}$. The order and size of a graph is the number of vertices, n, and the number of edges, m, respectively. The degree of a vertex v of G is the number of edges incident upon v. Connected graphs contain a route from every vertex to every other. Multigraphs (multiple bond containing molecules) are graphs which contain repeated edges between vertices while a simple graph does not contain any. A directed graph, or digraph, is a graph with directions assigned to each edge. Complete graphs are denoted by K_n and are graphs where an edge connects every pair of vertices. A labeled graph is one where the vertices

(a) Sample Graph (b) Molecular Graph

Figure 8: Graphs to Link the Terminology used in Graph Theory and Chemistry.

and/ or edges are given labels. A weighted graph is a type of labeled graph where the labels are real numbers.

A walk in G is a sequence of vertices $w = [v_1, v_2, ...v_k], k \geq 1$, such that $[v_j, v_{j+1}] \in E$ for $j = 1, ..., k-1$. The walk is closed if $k > 1$ and $v_k = v_1$, and open if they are different. A walk is called a path if there are no repeated vertices. A closed walk with no repeated vertices other than its first and last one is called a cycle. The length l of a walk is the number of edges it contains (open walks: $l = s - 1$, closed walks: $l = s$, where s is the number of vertices visited). The terms path and chain describe an open walk and a walk in which all vertices (and edges) are distinct, respectively. Cycles and paths of size n are denoted by C_n and P_n, respectively. A block is a group of vertices such that all edges between them are involved in one or more cycles. An open acyclic vertex is a vertex that is not located between two blocks while a closed acyclic vertex is located between two blocks.

The graph in Fig. 8(a) contains a cycle, $R = \{R_V, R_E\}$ where $R_V = \{v_7, v_8, v_9\}$ and $R_E = \{e_{78}, e_{89}, e_{97}\}$, of type C_3. R is a subgraph of G where the vertices and edges of R are subsets of G or in other words R is isomorphic to a subgraph of G. Reversely, G is a supergraph of R. The determination if the graph G_1 is isomorphic to a subgraph of G_2 is a known as the subgraph isomorphism problem which is NP-complete (Non-deterministic Polynomial time). The term clique is used for a set of vertices where an edge exists between each pair. A clique is a subgraph of G and itself is a complete graph. A k-clique is a clique of order k. Clique detection or maximum common subgraph isomorphism is a method to find the largest subgraph of G_1 isomorphic to a subgraph of G_2. The subgraph isomorphism and maximum subgraph isomorphism problems are known in cheminformatics as substructure searching and pharmacophore mapping. A molecule can be represented as a graph where the atoms are the vertices and bonds are the edges as in Fig. 8(b). This is a labeled or colored graph, in other words each vertex is labeled with the element type and each edge is colored with the bond order. The similarity between the two structures (graph and molecule) can be seen in Figures 8(a) and 8(b). A dictionary of terms is compiled in Table 2.

A tree, $T = (T_V, T_E)$, is a connected acyclic graph. Trees contain leaves which are vertices of degree 1 and non-leaf vertices. A root is a vertex where all edges point away from it. A forest is a set of disjoint trees while a "k-ary tree is a rooted tree in which every vertex has k children". Trees are often used in conformational searching and other combinatorial problems.

A matching or edge independent set, M, of G is a subset of the edges, such that no two edges

Table 2: Correspondence between Graph Theory and Chemical Terminology.

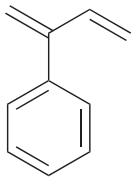
Graph Theory	Chemistry
Connected Graph	Molecule
Graph Order	Number of Atoms
Graph size	Number of bonds
Vertex Degree	Number of bonded atoms
Leaf Vertex	Terminal atom
Closed path/ Cycle	Ring
Cycle Type	Ring Size
Chain	Chain
Block	Cluster of Rings
Subgraph Isomorphism Problem	Substructure Searching
Maximum Common Subgraph Isomorphism	pharmacophore mapping

in E shares a vertex. There are three types of matching called maximum, maximal, and perfect. A maximum matching is a matching of highest cardinality. Maximal matching is a matching where no other edges can be added, while a perfect matching contains all vertices of the graph. A matching is maximum if and only if it has no augmenting path. An augmenting path is an alternating path which starts and ends with free or unmatched vertices. An alternating path describes a matching where the edges are alternately in M and not in M. For molecular graphs the maximum weighted matching algorithm is a technique of assigning double and triple bonds and corresponds to maximizing the number of double bonds in a pi system [7].

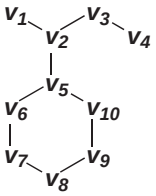
There are various ways of traversing or searching a graph. One such technique is the depth-first search. This is implemented as a recursive routine and tracks which vertex and edge are encountered thus only visiting each once.

2.3.2 Library Design

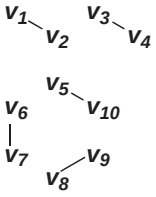
The graph library contains classes as shown in Fig. 10 to handle molecular graphs. This library is used to find rings and to determine whether graphs are isomorphic. Also it is used to traverse the torsional tree for systematic conformational searching. Tree and graph traversal is carried out using the depth-first search algorithm. The graph class stores both vertices and edges with the edge struct storing pointers to two vertex objects. Both vertices and edges store a boolean to describe whether each has been visited during a traversal and a numerical variable to describe its color or label. The vertex class also stores a list of its neighbors and which layer it is placed on.



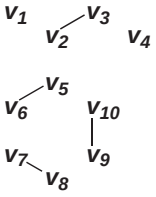
(a) Molecular Graph



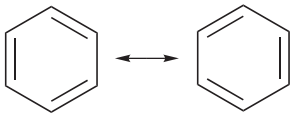
(b) Graph



(c) Maximum Matching



(d) Maximal Matching



(e) Kekule Structures of Benzene

Figure 9: Graphs to Link the Terminology used in Graph Theory and Chemistry.

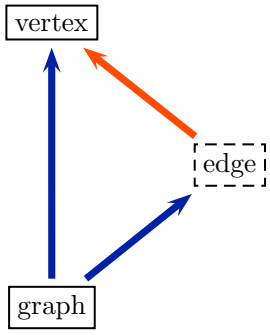


Figure 10: Class Hierarchy of the Graph Library as Implemented in MTK++. Solid line boxes denotes a class, while a dashed box signifies a structure. A class where the tail of the arrow starts uses or contains a class or structure where the head of the arrow ends. e.g. The graph class contains or uses the edge structure.

2.4 MM Library

2.4.1 Background

Molecular Mechanics (MM) force fields such as AMBER [8, 9, 10, 11], CHARMM [12], MMFF [13, 14, 15, 16, 17, 18, 19, 20], OPLS [21], and MM3 [22] can be used to calculate the enthalpic component of the binding free energy between the receptor and ligand.

The AMBER energy function, Eq. 4, contains bond, angle, dihedral, and non-bonded terms. The bond and angle terms are represented by harmonic expressions. The van der Waals term is a 6-12 potential, and the electrostatic is expressed as a Coulombic interaction with atom centered point charges.

$$E_{\text{total}} = \sum_{\text{bonds}} K_r(r - r_{eq})^2 + \sum_{\text{angles}} K_\theta(\theta - \theta_{eq})^2 + \sum_{\text{dihedrals}} \frac{V_n}{2}[1 + \cos(n\phi - \gamma)] + \sum_{i<j} \left[\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} + \frac{q_i q_j}{\varepsilon r_{ij}} \right] \quad (4)$$

where $K_r(kcal/mol\text{Å}^2)$, and $K_\theta(kcal/(mol\ Radian^2))$ are the force constants for bond length and angle, respectively, while r_{eq} and θ_{eq} are the equilibrium bond distances and angles.

A truncated Fourier series represents the dihedral term, where V_n is the barrier height, n is the periodicity, ϕ is the calculated dihedral angle and γ is the phase difference.

The fourth term describes the steric interaction as a Lennard-Jones potential, where r_{ij} is the distance between atoms i and j. $A_{ij} = \varepsilon_{ij} r_{ij}^{*12}$ and $B_{ij} = 2\varepsilon_{ij} r_{ij}^{*6}$ are parameters that define the shape of the potential where $r_{ij}^* = r_i^* + r_j^*$ in Å, r_i^* is the van der Waals radius for atom i, and $\varepsilon_{ij} = \sqrt{\varepsilon_i * \varepsilon_j}$, ε_i is the van der Waals well depth in kcal/mol and q are the atom-centered point charges.

2.4.2 Library Design

The MM library contains classes and functions to carry out Molecular Mechanics minimizations as shown in Fig. 11. Currently, the AMBER function is used. The amber class contains the driver functions for the lower level classes ambBond, ambAngle, ambTorsion, and ambNonBonded that contain the AMBER energy/gradient functions. The mmPotential class is the controller for all MM functions which could be developed. It performs all the memory allocation/deallocation. The MTK++ was designed as to easily allow the extension of its features. For example, cross terms such as bond-stretch and non-harmonic terms such as the Morse potential for bonded atoms are now possible within the MM library.

2.5 Parsers Library

The Parsers library contains classes to read and write molecular file types. XYZ, MOL, MOL2, PDB, SD, and ZMAT file formats are supported. All classes inherit baseParser as shown in Fig.

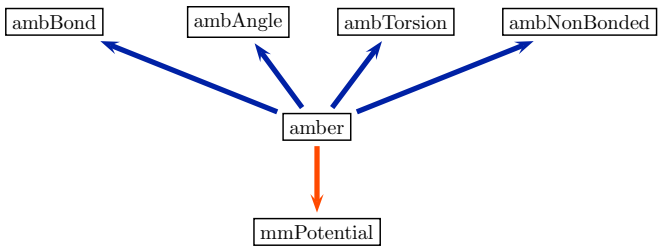


Figure 11: Class Hierarchy of the MM library as Implemented in MTK++. Solid line boxes denotes a class and a class where the tail of the arrow starts uses or contains a class where the head of the arrow ends. e.g. The amber class contains or uses the ambBond class. The orange arrow is used to represent a public inheritance relationship between classes, i.e. amber is of type mmPotential.

12. baseParser controls the error handling of all classes in a uniform way. The xml file parsers also inherit xmlConvertors and domErrorHandler from the xerces-c library which deal with errors in the xml files. Input and output files of programs such as DivCon and Gaussian (both cartesian and internal coordinates) are handled. The element parser reads the elements.xml file stored in the MTK++ distribution and populates the elements object which the collection class stores. For each atom in the periodic table the following information is stored: atomic number, mass, group, period, valence, full shell size, covalent radius, van der Waals radius, Pauling's electronegativity value, and which semiempirical Hamiltonians are available to a given atom. The stdLib parser handles the library xml files described in the previous section and populates the stdLibrary and stdGroup classes. The param parser handles the parameter files associated with the fragment library such as parm94 and GAFF from AMBER. The GA parser handles the files associated with the GA library of MTK++. The amber parser can export and import AMBER topology and coordinate files.

3 Atom Type and Bond Perception

It is often the case in SBDD that the design process starts with an x-ray crystal structure of a target molecule in complex with some bound substrate. This poses the challenge of determining atomic hybridizations, formal charges and bond orders of the small molecule due to the fact that there are no Hydrogen atoms present. Numerous algorithms have been published [23, 24, 25, 26, 27] but the algorithm by Labute in 2005 [7] to perceive atom hybridizations, bond orders and formal charges of drug-like molecules was implemented in MTK++ as it was shown to be superior to the others. Other methods to perceive atom types and bond information include antechamber by Wang *et al.* [28]. The Labute algorithm takes ten steps to determine the atom hybridizations, formal charges, and bond orders. A ligand that binds to PPARγ (PDB: 1FM9) as shown in Fig. 13(a) is used to illustrate the algorithm where $x_1, ..., x_n$ denote the 3D coordinates of n atoms with atomic number $Z_1, ..., Z_n$, and the number of bonded atoms for each atom is Q_i and $r_{ij} = |x_i - x_j|$ is the distance

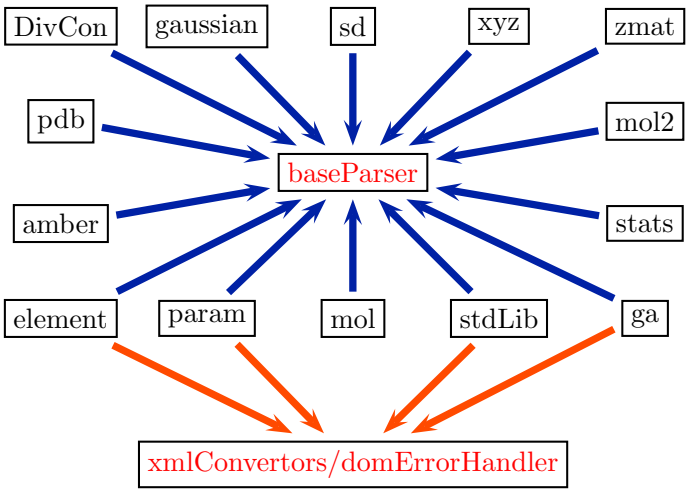


Figure 12: Class Hierarchy of the Parsers Library as Implemented in MTK++. Solid line boxes denotes a class and a class where the tail of the arrow starts uses or contains a class where the head of the arrow ends. e.g. The element class contains or uses the xmlConvertors class. The blue arrow is used to represent a public
inheritance relationship between classes, i.e. pdb is of type baseParser.

between two atoms. Bonds are perceived by first producing a candidate list and then refining it using geometry. Covalent radii, R_i, from Meng [23] shown in table 3 are used in Eq. 5 to determine the candidate bond list. Then for each atom, i, a "dimension", d_i, is assigned based on a principal component analysis of the Gram Matrix, D, defined in Eq. 6 where i is the current atom index, k is the number of bonded atoms and $\bar{q}$ is the geometric center as shown in Eq. 7.

$$0.1 < r_{ij} < R_i + R_j + 0.4 \tag{5}$$

$$D = \sum_{i=0}^{k} (q_i - \bar{q})(q_i - \bar{q})^T \tag{6}$$

$$\bar{q} = \frac{1}{k} \sum_{i=0}^{k} q_i \tag{7}$$

d_i is set to k if $k < 2$ otherwise, d_i is the number of positive eigenvalues of D with square root greater than 0.2. d_i will be 0 for isolated atoms, 1 for terminal and linear atoms with at least 2 bonds, 2 for planar atoms (e.g., sp^2 or square planar), and 3 otherwise (e.g. tetrahedral or sp^3d). The d_i numerical values for 1FM9 are shown in Fig. 13(b). Following the assignment of d_i an upper bound, B_i, for the number of bonds allowed by an atom is determined using d_i and Z_i as shown in

Table 3: Meng Atomic Covalent Radii.

Atom	Radii	Atom	Radii	Atom	Radii	Atom	Radii
H	0.23	P	1.05	Ni	1.5	Nb	1.48
He	1.5	S	1.02	Cu	1.52	Mo	1.47
Li	0.68	Cl	0.99	Zn	1.45	Tc	1.35
Be	0.35	Ar	1.51	Ga	1.22	Ru	1.4
B	0.83	K	1.33	Ge	1.17	Rh	1.45
C	0.68	Ca	0.99	As	1.21	Pd	1.5
N	0.68	Sc	1.44	Se	1.22	Ag	1.59
O	0.68	Ti	1.47	Br	1.21	Cd	1.69
F	0.64	V	1.33	Kr	1.5	In	1.63
Ne	1.5	Cr	1.35	Rb	1.47	Sn	1.46
Na	0.97	Mn	1.35	Sr	1.12	Sb	1.46
Mg	1.1	Fe	1.34	Y	1.78	Te	1.47
Al	1.35	Co	1.33	Zr	1.56	I	1.4
Si	1.2						

Table 4. Only the shortest B_i are retained. At this point all atom hybridizations and bond orders are set to zero or undefined. The next phase assigns obvious hybridizations based on d, Z, and

Table 4: Labute Algorithm Upper Bound Bond Conditions.

B_i	Condition
0	$d_i = 0$
1	$Z_i < 3 (H, He)$
2	$d_i = 1, Z_i > 2$ (sp hybridized and linear)
3	$d_i = 2, Z_i < 11$ (sp^2 hybridized for 2^{nd} row elements)
4	$d_i = 2, Z_i > 10$ or $d_i = 3, Z_i < 11$ (square planar or sp^3 hybridized)
7	otherwise

Q. Each row of table 5 is carried out one at a time with each row only being applied to atoms with unassigned hybridization resulting in Fig. 13(c). Only atoms with unassigned hybridizations have $d < 3$, $Z =$ (C,N,O,Si,P,S,Se), $Q < 4$ and at least one bonded neighbor with an unassigned hybridization. At this stage all bond orders, b_{ij} in which atom i or j has non-zero hybridization are set to 1.

A dihedral test is used to identify bonds of order 1. The smallest out-of-plane dihedral is computed using: $\min_{j,k} |P_{ijkl}|, |\pi - P_{ijkl}|, | - \pi - P_{ijkl}|$. If this dihedral is greater than 15° then b_{ij} is set to 1. Results of this step are shown in Fig. 13(d).

The following table 6 of lower bound single bond lengths and $|x_i - x_j| > L_{ij} - 0.05$, where L_{ij} is the reference bond length, are used to identify single bonds. The bonds identified using this step

Table 5: Labute Algorithm Atom Hybridization Assignment.

hybridization	Condition
sp^3	$Z_i = 1, 2$
sp^3d	$Q_i > 4, Z_i =$ (Group 5) and $Q_i = 5, Z_i =$ Group 4,5,6,7,8
sp^3d^2	$Q_i > 4, Z_i =$ (Group 6) and $Q_i = 6, Z_i =$ Group 4,5,6,7,8
sp^3d^3	$Q_i > 4, Z_i =$ (Group 7) and $Q_i = 7, Z_i =$ Group 4,5,6,7,8
sp^3d^2	$Q_i = 4, Z_i > 10, d_i = 2$
sp^3d^2	$Z_i =$ (Transition Metal)
sp^3d^2	$Q_i > 4, Z_i > 10$ and not Si, P, S, Se
sp^3	$Q_i > 4, Z_i > 10$ and Si, P, S, Se
sp^3	$(Q_i = 4)$ and $(Q_i = 3,\ d_i = 3)$
sp^3	$Q_i > 2, Z_i =$ Group 6,7,8
sp^3	Z_i not (C,N,O,Si,P,S,Se)
sp^3	All atoms where none of its bonded atoms have zero hybridization

are shown in Fig. 13(e)

Table 6: Labute Algorithm Lower Bound Single Bond Lengths.

bond	dist	bond	dist
C-C	1.54	C-N	1.47
C-O	1.43	C-Si	1.86
C-P	1.85	C-S	1.75
C-Se	1.97	N-N	1.45
N-O	1.43	N-Si	1.75
N-P	1.68	N-S	1.76
N-Se	1.85	O-O	1.47
O-Si	1.63	O-P	1.57
O-S	1.57	O-Se	1.97
Si-Si	2.36	Si-P	2.26
Si-S	2.15	Si-Se	2.42
P-P	2.26	P-S	2.07
P-Se	2.27	S-S	2.05
S-Se	2.19	Se-Se	2.34

After steps 5 and 6 the hybridizations of all uncharacterized atoms not involved in a bond of unknown order are set to sp^3 as shown in Fig. 13(f).

A molecular graph is formed including only atoms (vertices) that have undefined hybridization and bonds (edges) that have unknown order. This graph is then divided into components or subgraphs. Each subgraph is analyzed independently and bond orders are assigned as shown in

Fig. 13(g). Edge weights are assigned with the following equation $w_{ij} = u_i + u_j + 2\delta(r_{ij} < L_{ij} - 0.11) + \delta(r_{ij} < L_{ij} - 0.25)$ using the atom parameters, u, defined in Table 7 (3^{rd} and 4^{th} row elements are mapped to the corresponding 2^{nd} row with 0.1 been subtracted, -20.0 for all other atoms). Results are shown in Fig. 13(h).

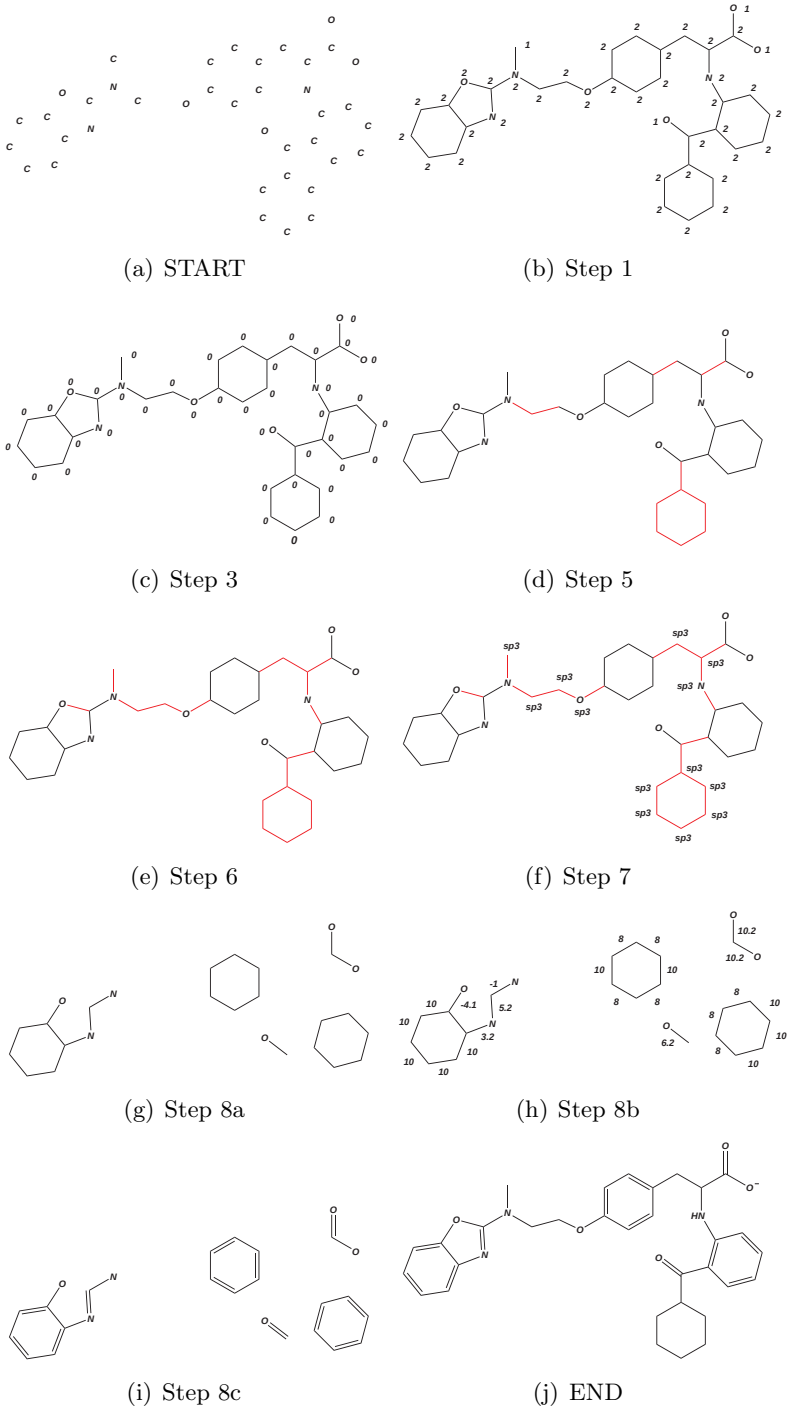
Table 7: Labute Algorithm Bond Weights.

atom	Q=1	Q=2	Q=3
C-O	1.3	4.0	4.0
C-N	-6.9	4.0	4.0
C	0.0	4.0	4.0
N-C-O	-2.4	-0.8	-7.0
N-C-N	-1.4	1.3	-3.0
N	1.2	1.2	0.0
O-C-O	4.2	-8.1	-20.0
O-C-O	4.2	-8.1	-20.0
O	0.2	-6.5	-20.0

A Maximum Weighted Matching Algorithm is employed to find the best arrangement of double/triple bonds in each subgraph resulting in the pattern shown in Fig. 13(i). Ionization states and formal charges are perceived from the connectivity and bond order. The formal charge of atom i, f_i, is calculated as follows: $f_i = c_i - o_i + b_i$, where: c_i is the atom group in the periodic table, o_i is the nominal octet (2 for Hydrogen, 6 for Boron, 8 for Carbon and all other sp^3 atoms in groups 5,6,7,8) and b_i is the sum of the atom bond orders. The final stage of the algorithm determines the correct bonding and charge state for the following functional groups: nitro, alphiatic amines, carboxylic acids, sulfonic acids, phosphonic acids, amidines, guanidines, and sulfonamides.

4 Ring Perception

The algorithm used is in close agreement with that published by Fan, Panaye, Doucet, and Barbu in 1993 [29]. The functions contained in rings determines the smallest set of smallest rings (SSSR) from a molecule graph. The SSSR of a molecule is represented as $S(m_1, m_2, ...)$ where m_i are the ring sizes. Take for example the following molecule shown in Fig. 14(a) with all open acyclic nodes highlighted in Fig. 14(b) are removed resulting in the structure shown in Fig. 14(c). Then all closed acyclic nodes are removed as highlighted in figures 14(d) and 14(e). The structure is then separated into blocks as shown in Fig. 14(f). The question then arises how many rings are there in the current block as shown in Fig. 14(g)? Allowing the first node to be the root node, numerous ring systems can be found including $R_V^1 = \{v_1, v_2, v_3, v_{15}, v_{13}, v_{14}\}$, $R_V^2 = \{v_1, v_2, v_3, v_{15}, v_{16}, v_{10}, v_{11}, v_{12}, v_{13}, c_{14}\}$, ..., R_V^n. The closed path found containing the root node is recursively searched until it can not be reduced further, in other words R_V^1 is found as shown in 15. Once an irreducible closed path is found all nodes with two links are removed. Nodes 2, 1,



(a) START (b) Step 1

(c) Step 3 (d) Step 5

(e) Step 6 (f) Step 7

(g) Step 8a (h) Step 8b

(i) Step 8c (j) END

Figure 13: Hybridization, Bond Order, and Formal Charge Perception Using the Labute Algorithm.

and 14 are then removed. The algorithm then picks another root node and the next ring is found until all rings are found. Once all rings are found in a molecule, an aromaticity test is applied. The algorithm used is in close agreement with that published by Roos-Kozel, and Jorgensen in 1981 [30]. Rings are classified as aromatic (AR), antiaromatic (AA), or nonaromatic (NA). A ring is assigned to be nonaromatic if it contains no intra-ring double bonds, contains a quaternary atom, contains more than one saturated carbon, contains a monoradical, or contains a sulfoxide or sulfone. A ring system is aromatic if and only if it contains 4n+2 (n=0,1,2,3,4,...) pi electrons (Hückel rule) and is planar (10 ° tolerance). The number of pi electrons of a ring is determine using the following rules: cationic carbon and boron contribute 0, saturated heteroatoms give 2, an anionic carbon has 2, and atoms on intra-ring pi bond contribute 1. If a rings contains exocyclic pi bond(s) (Carbon double bonded to a heteroatom), then 1 pi electron is removed. Some rings correctly perceived by this algorithm are shown in Fig. 16. All are perceived to be aromatric except for cyclooctatetraene (COT). COT contains alternating single and double bonds but it is non-planar and is correctly assigned to be antiaromatric.

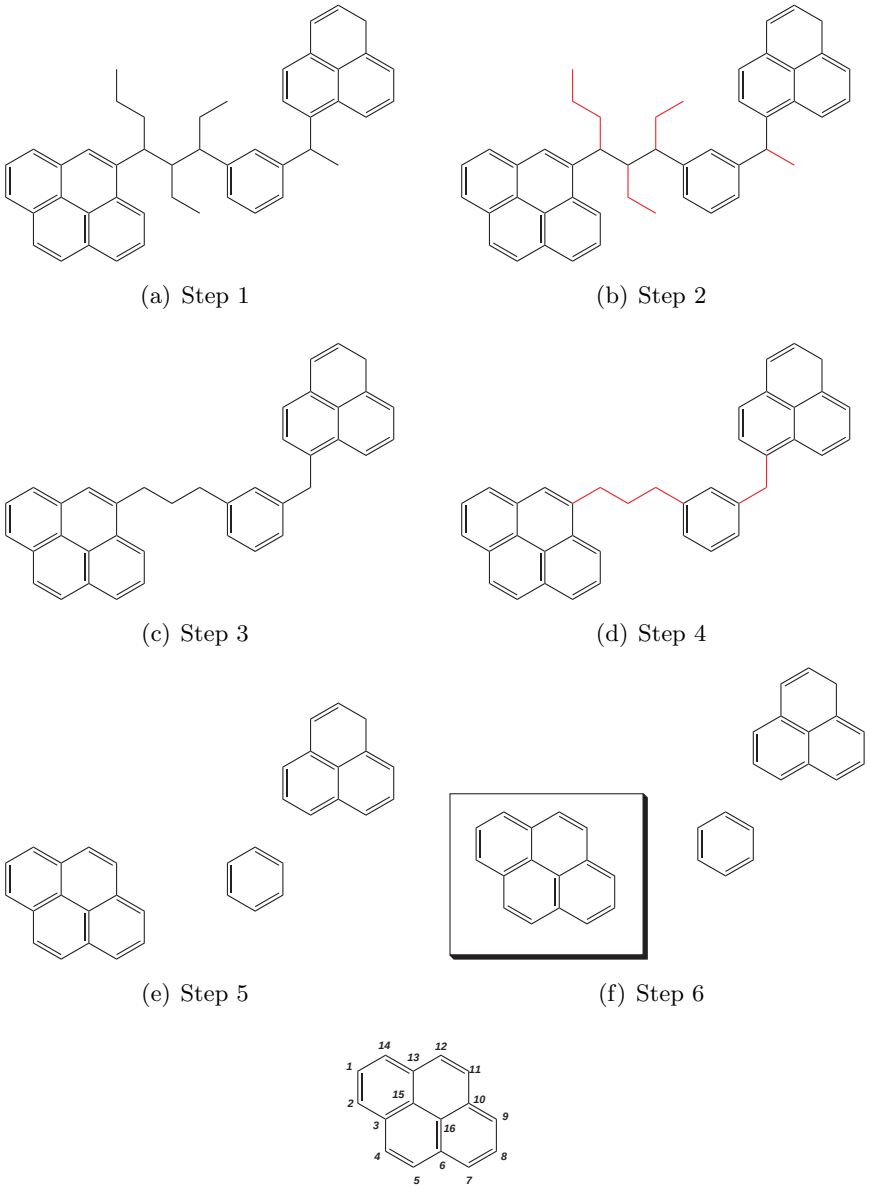
The ring centroid, plane and normal are also calculated for uses in pharmacophore matching and molecular alignment which will be discussed later in this chapter. The centroid is calculated using Eq. 8 where k is the size of the ring and q_i are the atomic coordinates. The ring plane and normal are computed by carrying out the principal component analysis of the Gram matrix as previously described in section 3. Matrix D is evaluated, Eq. 9, and then diagonalized with the first two eigenvectors defining the ring plane and the third being the ring normal.

$$c = \frac{1}{k}\sum_{i=0}^{k} q_i \tag{8}$$

$$D = \sum_{i=0}^{k}(q_i - c)(q_i - c)^T \tag{9}$$

5 Addition of Hydrogen Atoms to Molecules

The addition of Hydrogen atoms to proteins, DNA or water molecules is carried out using a predefined library. Small molecules with known atom hybridization, ring information and bond orders are dealt with using the following algorithm. First, Hydrogen atoms are added to polar (N, O, and S) atoms, followed by ring systems, then terminal atoms. All other unprotonated atoms are dealt with at the end. The number of Hydrogen atoms to add is defined by the current valence and the ideal full shell value of the atom to which the Hydrogen will be added. The bond lengths used are defined in Table 8. The only distinction of type of atom to which a Hydrogen is added is the element type, in other words the bond distance for a Carbon sp^3 or sp^2 to Hydrogen is the same. The angle to which a Hydrogen atom is added is defined either by the hybridization or type of the connecting atom or the type of bond between the connecting atom and 1-3 atom as shown in Table 9. The dihedral angle to add a Hydrogen atom is the most complex component of this algorithm.



(a) Step 1

(b) Step 2

(c) Step 3

(d) Step 4

(e) Step 5

(f) Step 6

(g) Step 7

Figure 14: Ring Perception.

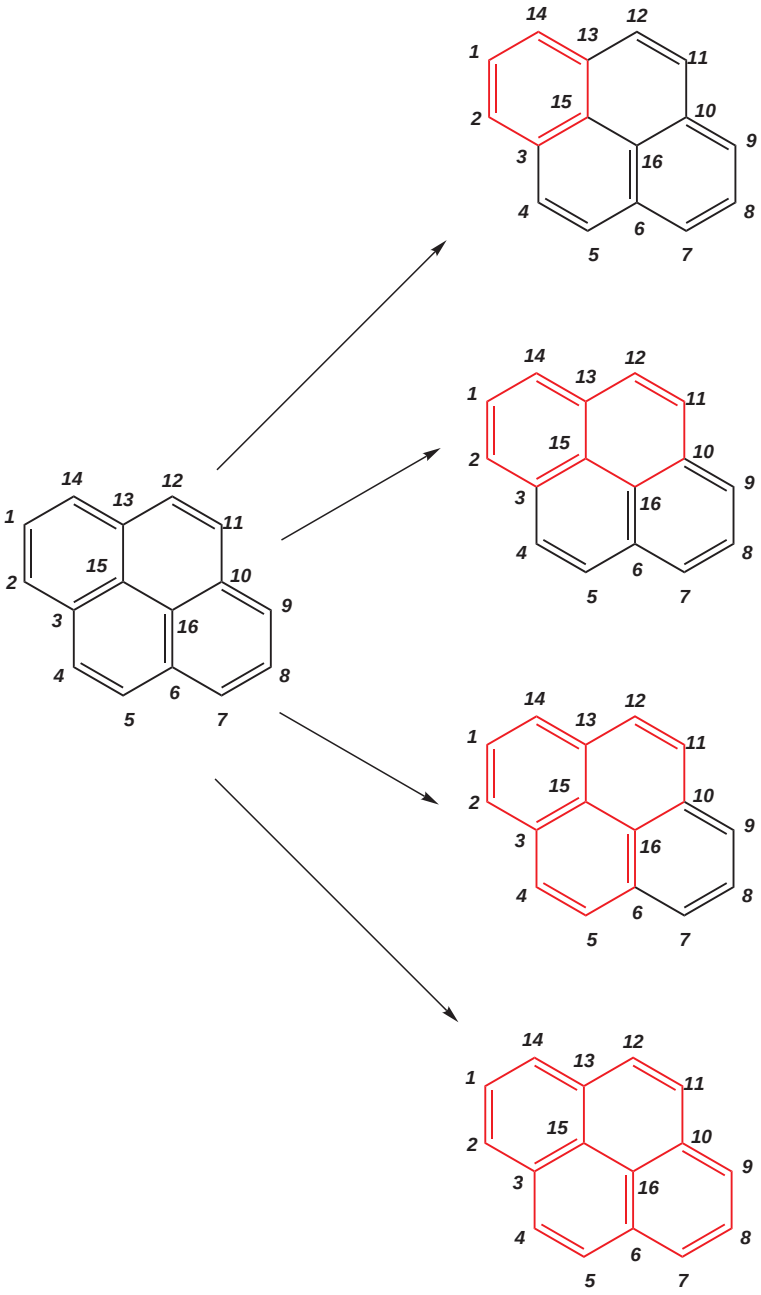
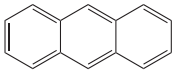
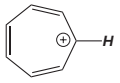


Figure 15: Ring Perception Step 8.

(a) Benzene (AR)



(b) Anthracene (AR)



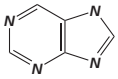
(c) Cycloheptatriene (AR)

(d) Cyclopentadienyl Anion (AR)

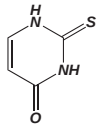
(e) Pyridine (AR)

(f) Thiophene (AR)

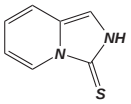
(g) Pyrimidine (AR)



(h) Purine (AR)



(i) 2-thioxo-2,3-dihydropyrimidin-4-one (AR)



(j) imidazo-pyridine-3-thione (AR)

(k) Cyclooctatetraene (AA)

Figure 16: Ring structure which are correctly assigned aromatic (AR), non-aromatic (NA) and anti-aromatric (AA).

Suppose you want to add a proton, A, on to atom B which is bonded to atom C and is $1-3$ bonded to atom D. First a list is compiled of all torsional angles $XBCD$ already occupied, where X is any heavy atom bonded to atom A. The dihedral then used is defined by the hybridization of atom B and built using atoms BCD. If B is sp^3 hybridized then a Hydrogen atom is placed 120° from other bonded atoms. Dihedral angles of 0° and 180° are used when B is sp^2 and 180° for sp hybridized. Aromatic rings are a special case of sp^2 hybridized atoms where only a torsion of 180° is allowed. The dihedral values of polar Hydrogens are optimized to maximize intra-molecular Hydrogen bonding using Eq. 10 where θ_{D-H-A} is the angle between the donor, Hydrogen and acceptor atoms and r_{H-A} is the Hydrogen-acceptor distance. If θ_{D-A-AA} or θ_{D-H-A} is greater

than 90°, or r_{D-A} is less than 3.5Å then no Hydrogen bond is considered.

$$E_{HB} = cos^2\left(\theta_{D-H-A}\right) * e^{\left(-(r_{H-A}-2.0)^2\right)} \tag{10}$$

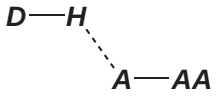


Figure 17: Hydrogen Bond.

Table 8: Hydrogen Bond Lengths.

atom	Bond Length (Å)
C	1.09
N	1.008
O	0.95
S	1.008
Se	1.10
Default	1.05

Table 9: Hydrogen Bond Angles.

atom / Bond	Angle (Degrees)
sp / triple	180.0
sp^2 / double	120.0
sp^3 / single	109.47
Aromatic Ring	$((360 - ((ringSize - 2) * 180)/ringSize)/2)$
Default	109.47

6 Conformational Sampling

Conformational searching of drug-like molecules in MTK++ is carried out using a systematic approach. GAFF [31] atom types are assigned to the atoms in a particular molecule using ANTECHAMBER and CM2 charges are generated using the DivCon program. The atomic hybridizations and bond orders defined in the hybridize class are used to mark which single or double bonds are rotatable. If either of the atoms in a bond are described as terminal then the bond is removed

Table 10: Hydrogen Bond Dihedrals.

atom	Dihedral (Degrees)
sp	180.0
sp^2	0.0, 180.0
sp^3	120.0
Aromatic Ring	180.0

from the list of rotatable bonds. If both of the atoms are members of a ring then the bond is also removed, thus removing ring flexibility. The incorporation of ring flexibility is planned in later releases of the MTK++ code. Then for each rotatable bond that remains a torsion is sought after. The total number of molecular conformations, $N_{conformers}$, is then defined by Eq. 11, where i is a rotatable bond index, R is the range of the associated torsion $(0 - 360^\circ)$, and δ is the rotation increment (120° for $sp^2 - sp^3$). The increment currently used are tabulated in Table 11. Once the number and location, as shown in Fig. 18, of each rotatable bond is determined a graph is formed as described in Fig. 19 [32]. Each rotatable bond is defined as a layer with each unique torsional value information contained in a vertex upon this layer. Graph edges are then defined between each vertex of one layer to every vertex one layer below. Once formed the graph is traversed and the AMBER energy, E_{MM}, is calculated for each conformer. The lowest energy conformers are stored, based on some user provided criteria, for later use.

$$N_{conformers} = \prod_{i=1}^{n} \frac{R_i}{\delta_i} \tag{11}$$

Table 11: Dihedral Angles Available based on Bond Type.

Bond Type	Angles
$sp^3 - sp^3$	60, 180, 300
$sp^2 - sp^2$	0, 30, 150, 180, 210, 330
$sp^2 - sp^3$	0, 30, 60, 90, 120, 150, 180, 210, 240, 270, 300, 330

In Fig. 20(a) we have an organic molecule which binds to the Peroxisome Proliferator-Activated Receptor γ. This is a functional group rich molecule containing phenyl rings, a carboxylate, a heterocycle (2,4,5-oxazole), a ketone, an amine, and an ether moiety. This structure has 12 rotatable bonds as shown in Fig. 20(b). Using the torsion resolution definitions in table 11 would lead to 4,353,564,672 conformers! Even on modern computer hardware this number is too large. Taking a closer look at this structure, Fig. 20(c), the symmetry of the functional groups becomes apparent. For example, the carboxylate group, shown in green, is C_2 symmetric since the negative charge is

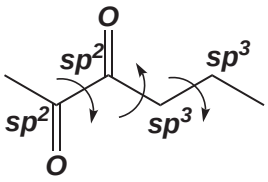


Figure 18: Rotatable Bond Types.

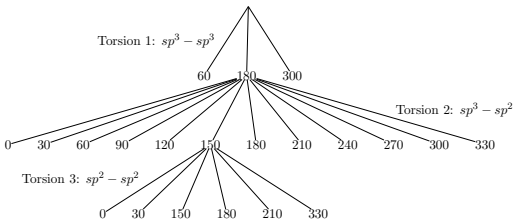
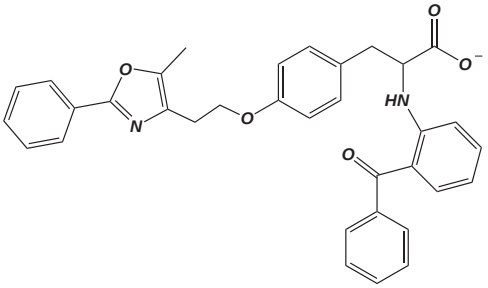
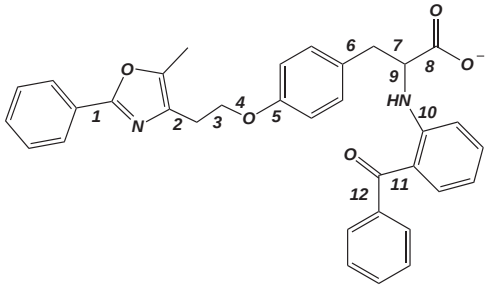


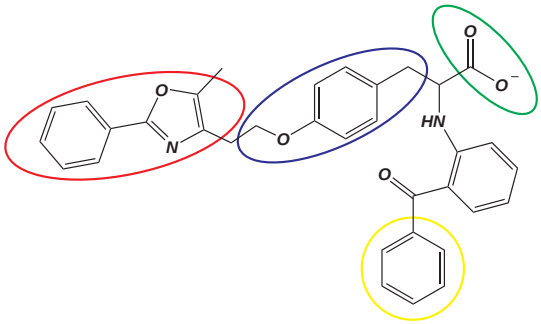
Figure 19: Systematic Conformational Searching. Torsion 1 forms the first layer containing three values or vertices. Followed by layer containing 12 vertices and finally the third layer with six vertices. This graph would result in 216 conformers been formed.



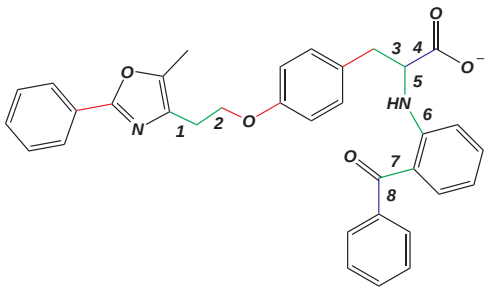
(a) PPARγ Agonist



(b) Number of Rotatable Bonds



(c) Symmetric Regions



(d) Reduced number of Rotatable Bonds

Figure 20: Conformer Generation.

not solely placed on one of the oxygen atoms and the phenyl group shown in yellow is also symmetric. Removing these symmetric torsions from the total number results in 544,195,584 conformers. Chemical knowledge of the torsional profile between the phenyl and oxazole groups enclosed in the red oval suggests even fewer available torsions due to conjugation. Thus reducing the total number of discrete conformers to 181,398,528. MTK++ attempts to reduce this number even further by recognizing "privileged fragments" with known torsion profiles. The tyrosine-like group enclosed in the blue oval is one such fragment and is stored in the "cores" library of the package. Removing these rotatable bonds results in 419,904 conformers. In Fig. 20(d) highlights the rotatable bonds: green represents unrestricted, blue are restricted by symmetry, while red bonds are frozen.

The systematic approach works extremely well for molecules with approx. 12 rotatable bond or less. When the number of potential conformers exceeds two million the searching algorithm reverts to using the GA library of MTK++. During a GA search of conformational space the user is required to provide the maximum number of MM calculations which are allowed. Other searching tools such as MD and MD-LES are recommended for large peptidic molecules that bind certain proteins such HIV Protease and Endothiapepsin.

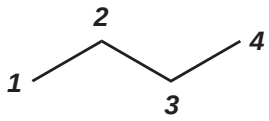
7 Substructure Searching/ Functionalize

To functionalize a molecule involves searching it for chemical substructures. Substructures searching is known as the subgraph isomorphism problem of graph theory and belongs to the class of NP-complete computational problems. Due to the NP-complete nature of substructure searching usually a screen is carried out to eliminate subgraphs that cannot be contained in the molecule. The fingerprint class in the molecule library carries out this screening process between fragments and a molecule.

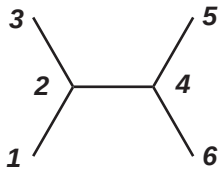
The brute-force algorithm for subgraph isomorphism begins by generating the adjacency matrices A and B for the fragment and the molecule containing P_A and P_B atoms respectively. Then an exhaustive search involves generating $P_B!/[P_A!(P_B - P_A)!]$ combinations of P_A and determining whether any combinations are matches to a portion of the molecule. The algorithm used is in close agreement with that published by Ullmann [33], and Willett, Wilson, and Reddaway [34]. Ullmann first noticed that using a depth-first backtracking search dramatically increases efficiency, while Willett used a labeled graph and a non-binary connection table to increase algorithm speed.

The functionality of finding substructures in molecules was developed to carry out functional group alignment of drug-like molecules and to optimize fragment positions in drug-protein complexes during the lead optimization stage of the drug design process.

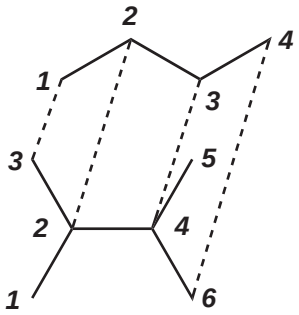
The algorithmic details of the functionalize code are as follows (this example was adapted from Molecular Modelling, Principles and Applications 2nd Edition by Andrew R. Leach [35]). Take for example a fragment and molecule shown in Fig. 21(a) and 21(b). The corresponding adjacency matrices are shown in Eq. 12 and 13. The Ullman algorithm tries to find the match between the fragment and the molecules (Fig. 21(c)). Mathematically this is represented as the matrix A, Eq. 14, which satisfies $A(AM)^T$ as shown in Eq. 15.



(a) Fragment Structure



(b) Molecule Structure



(c) Alignment

Figure 21: Ullman Subgraph Isomorphism Illustration.

$$F = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{12}$$

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \tag{13}$$

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{14}$$

$$A(AM)^T = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = F \tag{15}$$

This depth-first backtracking algorithm uses a General match matrix, M that contains all the possible equivalences between atoms from A and B. The elements of this matrix, $m_{ij}(1 \leq i \leq P_a; 1 \leq j \leq P_b)$ are such that:

$$m_{ij} = \begin{cases} 1 \text{ if the } i^{th} \text{ atom of } A \text{ can be mapped to the } j^{th} \text{ atom of } B, \\ 0 \text{ otherwise.} \end{cases} \tag{16}$$

The Ullmann heuristic states that "if a fragment atom a_i has a neighbor x, and a molecule atom b_j can be mapped to a_i , then there must exist a neighbor of b_j, y, that can be mapped to x" and is mathematically written in Eq. 17.

$$m_{ij} = \forall \quad x(1...P_A)[(a_{ix} = 1) \quad \Rightarrow \quad \exists \quad y(1...P_B)(m_{xy}b_{jy} = 1)] \tag{17}$$

If at any state during the search an atom i in A such that $m_{ij} = 0$ for all atoms in B then a mismatch is identified as defined in Eq. 18 and the match is discarded.

$$mismatch = \exists \quad i(1...P_A)[(m_{ij} = 0 \quad \forall \quad j(1...P_B)] \tag{18}$$

The complete algorithm to perceive the functional groups in a molecule in described using pseudo code in Algorithm 1. The algorithm begins by reading user created fragment libraries and molecules which are to be studied. Fingerprints of each fragment and molecule are then created. For each molecule under consideration rings, atom hybridizations, bond orders and formal charges are assigned using the algorithm previously described. If Hydrogens are not present on the molecule then they are added using an algorithm described later in this chapter.

Then for each fragment store in memory its fingerprint is compared to that of the molecule. If there is a fingerprint match then the Ullmann/Willett algorithm is invoked. If the subgraph isomorphism algorithm results in a match then the fragment code is assigned to the molecule.

8 Clique Detection/ Maximum Common Pharmacophore

As outlined in Section 2.3.1 a 3D molecular clique is defined as a group of pharmacophore points and the geometric distances between all points in that group. Fig. 22 illustrates the clique detection algorithm implemented in MTK++ [36]. Take for example two estrogen receptor ligands (PDBID: 1ERR and 3ERT) and finding the pharmacophore points such as Hydrogen bond acceptor/donor, positive/negative charge centers, hydrophobes, rings, and ring Normals as shown in Fig. 22(b) certain molecular features can be mapped to one another, as shown in Fig. 22(c). The mapping,

Algorithm 1: Functionalize Algorithm.

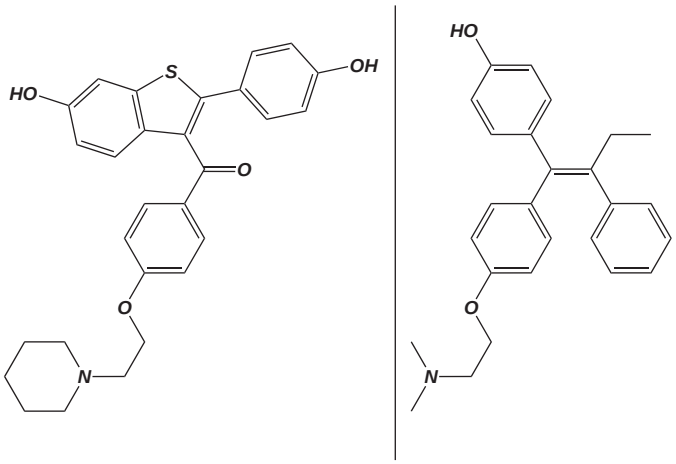
```
Data: Fragment Libraries and MDL Files
Result: Functional group assignment
begin
    Read Fragment Libraries;
    Read in molecules to functionalize;
    Generate fingerprints for all fragments;
    for i → nMolecules do
        Determine Rings;
        Perceive Hybridizations, Bond Order, and Formal Charge;
        Add Hydrogens;
        Generate fingerprint;
        for j → nFragments do
            bool bMatch1 = Compare molecule to simple fingerprint (Screening);
            if bMatch1 then
                bool bMatch2 = Match fragment to molecule using the Ullmann and Willett
                algorithm of subgraph isomorphism;
                if bMatch2 then
                    assign fragment code to molecule;
                end
            end
        end
    end
end
```

Eq. 19, results in a valid clique because the inter point distances, Eq. 20, are compatible within some tolerance. However, adding the mapping $M = [F_1 \leftrightarrow F_2]$, does not result in a valid clique as $d^1_{CF} \not\approx d^2_{CF}$. The clique detection algorithm thus requires a method of pruning a potentially large set of mapping which is carried out by allowing each to be a seed and growing cliques using heuristic criteria. Obviously certain seed mappings will lead to equivalent cliques however a diverse set are often found. Cliques are then scored or ranked to determine the best overall matching using Eq. 21 where D are the inter-point distances and d^1_i is a distance between two features of molecule 1 and d^2_i is the distance between equivalent features in molecule 2. The function for the two mapping $A_1 \leftrightarrow A_2$ and $B_1 \leftrightarrow B_2$ reaches a value of 1.0 when $d^1_{AB} = d^2_{AB}$. The parameter δd_{max} controls how rapidly the match score drops off as the distances becomes less compatible.

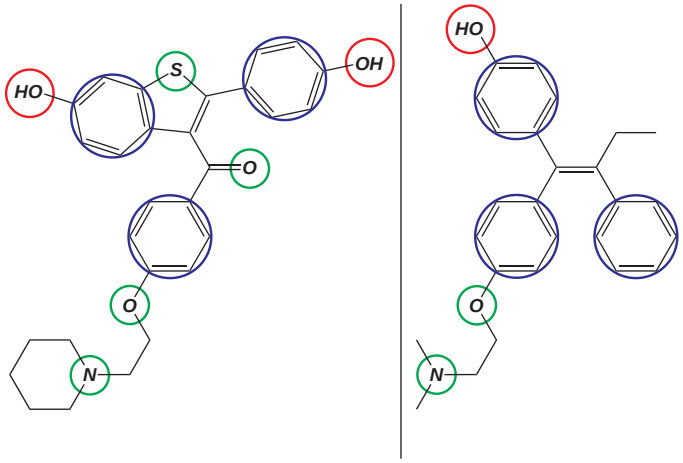
$$P = [A_1 \leftrightarrow A_2, B_1 \leftrightarrow B_2, C_1 \leftrightarrow C_2, D_1 \leftrightarrow D_2, E_1 \leftrightarrow E_2] \tag{19}$$

$$D = \left[d^1_{AB} \approx d^2_{AB}, d^1_{BC} \approx d^2_{BC}, d^1_{CD} \approx d^2_{CD}, d^1_{DE} \approx d^2_{DE}, \ldots\right] \tag{20}$$

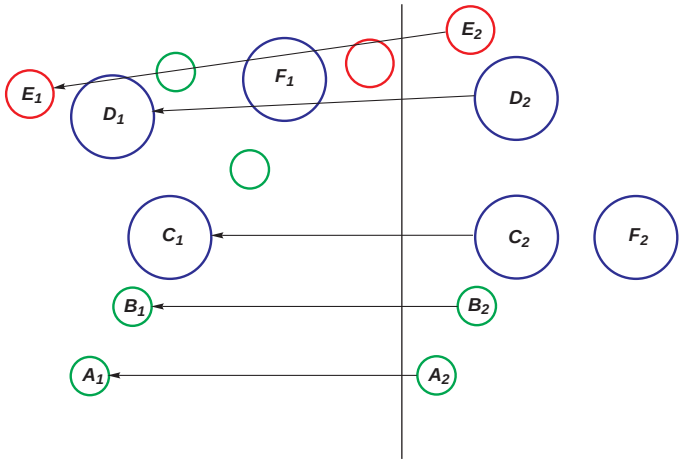
$$Score = \sum_i^D exp\left[-\left(\frac{d^1_i - d^2_i}{\Delta d_{max}}\right)^2\right] \tag{21}$$



(a) Estrogen Ligands (PDB: 1ERR and 3ERT.)



(b) Chemical Features Highlighted



(c) Chemical Feature Mappings

Algorithm 2: Find Maximum Common Pharmacophore (MCP)

```
Data: Two Molecules
Result: Maximum Common Pharmacophore Between the Two Molecules
begin
    Generate Feature Correspondence Matrix between the two molecules;
    Get Threshold Feature Score, TFS;
    bestClqScore = 0; bestClqSize = 0;
    for i → CorrespondenceMatrix do
        getPair ⟵ 1; curClq ⟵ i; curClqScore ⟵ 0;
        while getPair do
            getPair ⟵ 0; maxScore ⟵ 0; pair ⟵ 0;
            for j → CorrespondenceMatrix do
                testScore ⟵ 0;
                for k → curClq do
                    jkScore = exp(−((d_j − d_k)/d_m)^2);
                    if jkScore > TFS then
                        testScore+ = jkScore
                    else
                        break;
                    end
                end
                if testScore > maxScore then
                    Pair ⟵ j;
                    maxScore ⟵ testScore;
                end
            end
            if Pair then
                Add Pair to curClq;
                curClqScore+= maxScore;
                getPair ⟵ 1;
            end
        end
        store ⟵ 0;
        if curClqScore > bestClqScore + 0.1 then
            store = 1;
        end
        if curClqScore > bestClqScore − 0.1 then
            if curClqSize > bestSize then
                store = 1;
            end
            if curClqSize == bestSize then
                if curClqDist > bestDist + 0.1 then
                    store = 1;
                end
            end
        end
        if store then
            bestClqScore = curClqScore; bestClqSize = curClqSize;
            bestClqDist = curClqDist; bestClq ⟵ curClq;
        end
    end
    return bestClq
end
```

9 Superimposition

Molecular superimposition is carried out using a rigid body least squares procedure from Kearsley [37] and Kabsch [38, 39]. The rotation matrix to minimize the sum of the squared distances between atoms of two molecules, Eq. 22 is solved using quaternions and eigen methods as described by Kearsley in 1989.

$$F = \sum_i |x_i - x_i'|^2 \tag{22}$$

A requirement of this procedure is that atom i in molecule A corresponds to atom i in molecule B. For example if you wanted to measure the rmsd between two benzoic acid conformers as shown in figures 23(a) and 23(b) would require a certain correspondence to remove artificial differences attributed to automorphisms or self-symmetry. This is carried out by generating all matchings of non-Hydrogen atoms by type or element kind and assigning the lowest rmsd as the true value [40].

(a) Conformer 1 (b) Conformer 2

Figure 23: Illustration of the requirement of atom correspondence for molecularsuperposition.

10 Metalloproteins

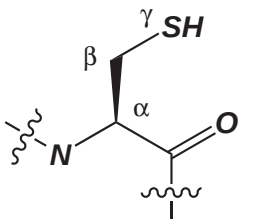
Metalloproteins are a key subset of proteins in the body which bind a transition metal. The metal ion acts as a Lewis acid (electron pair acceptor) towards amino acids or other molecules which are Lewis bases (possess one or more lone pairs) and are called ligands. The bonding of ligands to a metal ion is described as dative when the ligand donates one or more lone pairs to the metal. Metals can be coordinated to any number of ligands with four, five and six being the most common in biosystems. Thus the most likely geometries include tetrahedral, square planar, trigonal pyramidal, and octahedral. The amino acids that most commonly bind to a metal ion in metalloproteins are shown in Fig. 24. The side chain of each amino acid is labeled with greek letters and these are used when referring to which atom of an amino acid bonds to the metal, e.g. Zn-CYS@SG would translate that the gamma Sulfur of a cysteine residue is bound to a Zinc ion.

Iron, Copper, and Zinc are the most abundant transition metals in the human body. Metal ions in biological systems have both structural and functional roles. They are termed structural when no chemical reaction takes place at the metal site but aide in the stabilization of the protein structure whereas functional metalloproteins carry out chemical reactions.

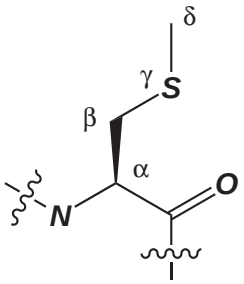
Zinc proteins are both structural and catalytic. Zinc acts as a superacid and promotes the hydrolysis or cleavage of chemical bonds. For example, Human Carbonic Anhydrase II (HCA II), catalyses the conversion of CO_2 into bicarbonate or vice versa. HCA II contains a tetrahedral zinc at its active site as shown in Fig. 25(a). The Zinc atom is bound to three histidine residues and a water molecule (pH$<$ 7) or a hydroxyl ion. Farnesyl Transferase (FTase) is a zinc metalloenzyme that removes the diphosphate group from the farnesyl diphosphate substrate and connects the resulting farnesyl moiety to the cysteine. The full active site of 1QBQ is shown in Fig. 25(b). Other Zn metalloproteins include carboxypeptidase which cleaves the terminal carboxy group from peptides and alcohol dehydrogenase which converts alcohol to acetaldehyde.

Metalloproteins that contain Copper are also both structural and functional. Copper can change oxidation state and is often involved in electron-transfer reactions. Human Antioxidant protein (HAH1) contains a tetrahedral Cu(I) bound by four cysteine residues as shown in Fig. 26(a). HAH1 is involved in the transporting of Copper in the body and is labeled a chaperone. Amicyanin is a tetrahedral Cu(II) containing protein which binds two histidines, a methionine, and a cysteine residue as shown in Fig. 26(b). This protein is called a blue copper protein due to its spectroscopic properties arising from cysteine to Cu(II) charge-transfer [41, 42, 43, 44, 45, 46].

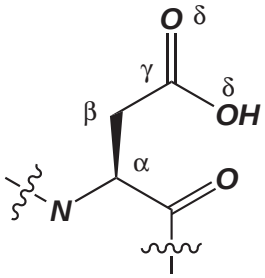
Metalloproteins can also contain multiple metals in close proximity. For example, Aminopeptidase is a di-zinc protein from *Aeromonas. proteolytica* (AAP) which catalytically cleaves the N-terminus of polypeptides. The active site of Aminopeptidase is shown in Fig. 27(a) where the zinc ions are bound to histidine, aspartic acids and are bridged with a water molecule. Urease from *Bacillus pasteurii*, is a di-nickel enzyme that catalyzes the hydrolysis of urea to ammonia and carbon dioxide. Its active site is shown in Fig. 27(b), where two nonequivalent Ni(II) atoms (3.5Å separation) are bound to two histidines each and a bridging carbamylated lysine. An aspartate residue, two waters and a bridging water/hydroxyl ion complete the coordination sphere. The geometry of both Ni centers can be described as square pyramidal and octahedral [47, 48, 49, 50]. Both Aminopeptidase and Urease are homo-nuclear proteins but hetero-nuclear metalloproteins



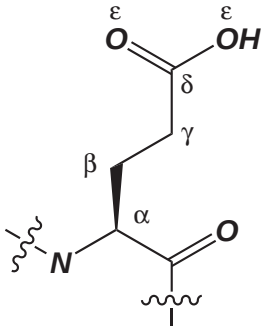
(a) Cysteine (CYS)



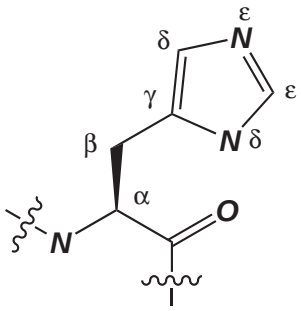
(b) Methionine (MET)



(c) Aspartic Acid (ASP)

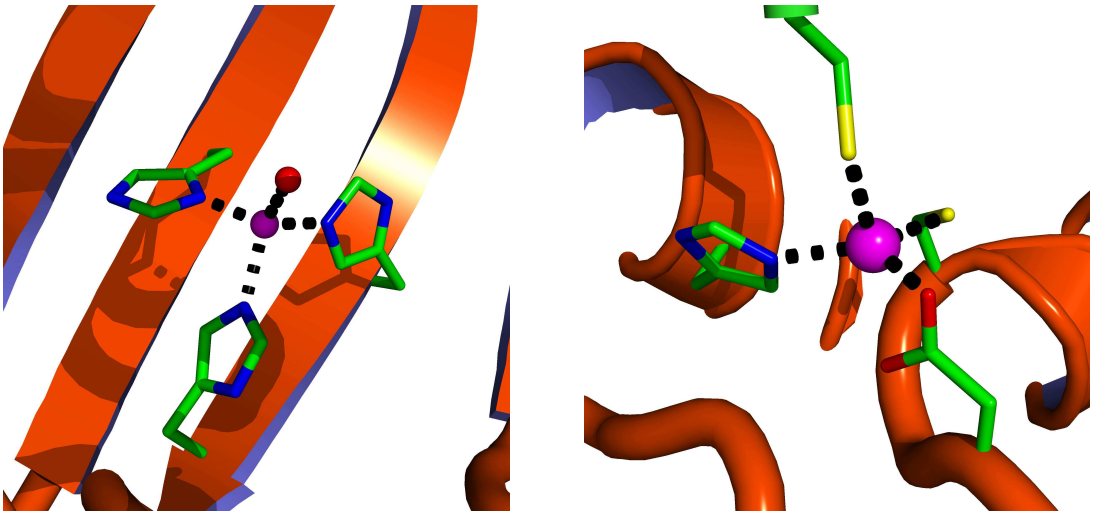


(d) Glutamic Acid (GLU)



(e) Histidine (HIS)

Figure 24: Most Common Amino Acid Residues which Bond to Metal Ions.



(a) 1CA2 Active Site (b) 1QBQ Active Site

Figure 25: Zinc Metalloproteins. The Zinc ion is shown in purple while the Oxygen atom of the water molecule in 1CA2 bound to Zinc is shown in red.

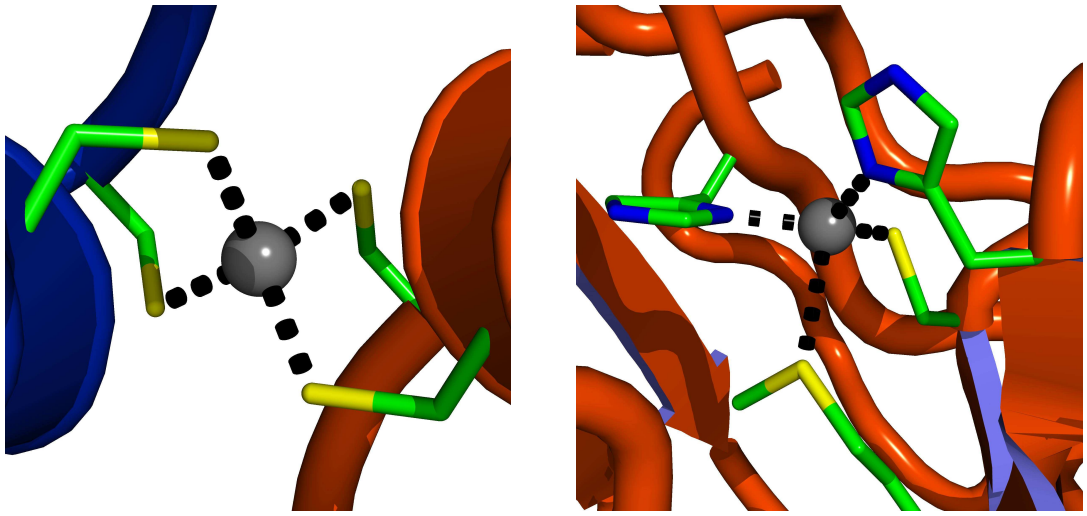
also exist. Copper-Zinc Superoxide Dismutase, Cu,Zn-SOD, is one such protein as shown in Fig. 28.

There are currently 52550 structures in the Protein Data Bank (PDB) [51] and searching for "metal" results in over 18,000 hits with the break down shown in table 12. Metal ions play a vital role in protein function, structure, and stability, with zinc, copper, and iron playing the biggest role as described in Section 10.

Table 12: Metal Ions in the Protein Data Bank (Accessed on April 5^{th} 2007).

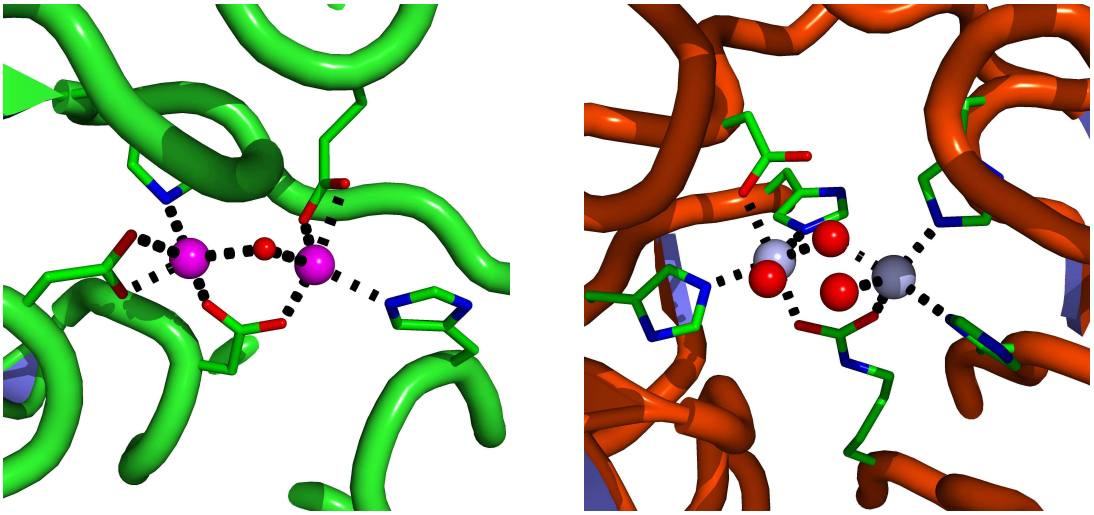
Metal	Hits	Metal	Hits	Metal	Hits
Na	2149	V	12	Pd	1
Mg	3467	Cr	7	Ag	9
K	632	Mn	984	Cd	361
Ca	3601	Fe	2022	Ir	6
		Co	340	Pt	62
		Ni	310	Au	28
		Cu	589	Hg	323
		Zn	3427		
Total = 18330					

It is desirable to model metalloprotein systems using MM models because one can carry out



(a) 1FEE Active Site (b) 1AAC Active Site

Figure 26: Copper Metalloproteins. The Copper ion is shown in grey.



(a) 1AMP Active Site (b) 2UBP Active Site

Figure 27: Homo-Nuclear Metalloproteins. The Zinc and Nickel ions are shown in purple and grey respectively, while Oxygen atoms of water molecules are shown in red.

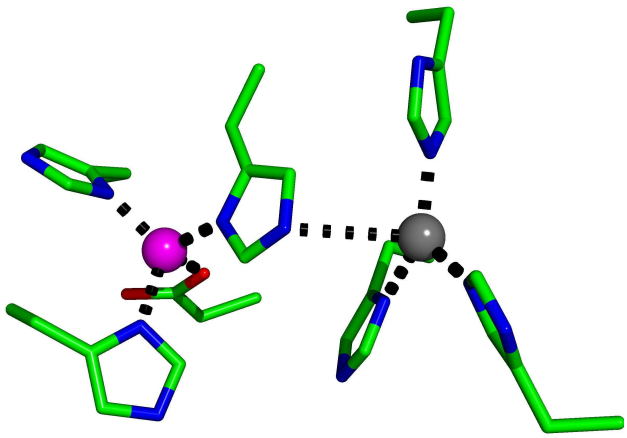
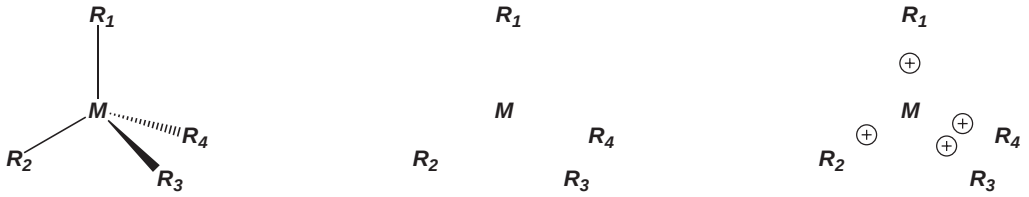


Figure 28: Hetero-Nuclear Metalloproteins. The Active Site of Copper-Zinc Superoxide Dismutase, Cu,Zn-SOD, (PDB ID: 1CBJ) is shown. The Zinc and Copper ions are shown in purple and grey respectively.

simulations to address important structure/function and dynamics questions that are not currently attainable using QM and QM/MM based methods due to unavailability of parameters or system size.

There are a number of approaches to incorporating metal ions into FFs. The **Bonded Model** defines bonds, angles, and torsion's between the metal ion and its ligand which are added to the FF plus the van der Waals component of the non-bonded function. Hancock [52] used this approach to study systems including Copper and Nickel. The **Bonded plus electrostatics Model** defines bonds and angles between the metal ion and its ligand as well as electrostatic potential (ESP) charges (Fig. 29(a)) [53]. This method attempts to define the correct electrostatic representation of the metal active site as assigning a plus two charge to a divalent metal ion would not describe reality though formally correct. The partial atomic charges can be calculated using the RESP approach [54] or the CMX models of Truhlar and Cramer [55]. The bond and angle force constants are derived from experiment or calculated using *ab initio* or DFT methods while the torsion term has so far been neglected. The **Non Bonded Model** does not define any extra bonds and places integer charge on the metal ion [56]. Electrostatic and Lennard-Jones terms describe the interactions. Modifications to this model to include polarization and charge transfer effects have been developed (Fig. 29(b)) [57]. The **Cationic Dummy Atom Model** is related to the non bonded method where it places dummy atoms (cations) to mimic valence electrons around the metal ion [58].

Electrostatic and Lennard-Jones terms between the dummy atoms and ligating residues describe the metal-ion interactions (Fig. 29(c)) [59, 60]. Other methods include those of Vedani *et al.* which



(a) Bonded Model (b) Non-Bonded Model (c) Cationic Dummy Atom Model

Figure 29: Three Approaches to Incorporate Metal Atoms into Molecular Mechanics Force Fields. The bonded model defines bonds, angles, and dihedrals between the metal and ligands, while the non-bonded model does not and uses electrostatics and van der Waals to model the interactions. The cationic dummy atom model is a derivative of the non-bonded model where cations are placed near the metal center to mimic valence electrons around the metal.

is a compromise between the bonded and non-bonded methods and is implemented in the YETI program [61], the SIBFA of Gresh and co-workers [62, 63] and the Universal Force Field (UFF) of Goddard and Rappe and co-workers [64, 65, 66]. These methods do not use a pairwise additive potential or are not readily available in typical biomolecular modeling packages.

Carrying out MM modeling or MD simulations of metal containing proteins is a complicated procedure using the bonded plus electrostatics model. Incorporating metals into protein force fields is a convoluted process due to the plethora of QM Hamiltonians, basis sets and charge models to choose from. Also it has generally been carried out by hand without extensive validation for specific metalloproteins. Some of the published force fields for Zinc, Copper, Nickel, Iron, and Platinum containing systems using the bonded plus electrostatics model are listed in Table 13. There have been numerous other FFs containing various metals published including ruthenium(II)-polypyridyl [67], cobalt corrinoids [68, 69, 70, 71], Staphylococcal Nuclease [72], alcohol dehydrogenase [73, 74, 75], and metalloporphyrins [76, 77, 78, 79, 80].

Automated procedures for the parameterization of MM functions for inorganic coordination chemistry have been developed over the last number of years by Norrby and co-workers [81, 82]. Their attempts have focused on generating parameters using experimental, structural data from databases such as the Cambridge Crystallographic Structural Database (CCSD) and quantum mechanical reference data using a version of the MM3 force field [22].

Table 13: Published Metalloprotein Force Fields Using the Bonded Plus Electrostatics Model.

Metal	Protein	References
Zinc	Human Carbonic Anhydrase II	[53, 83, 84]
	Beta-lactamase	[85, 86, 87, 88, 89, 90, 91, 92]
	ZAFF	[93]
	Dinuclear Beta-lactamase	[94, 95]
	Farnesyl Transferase	[96]
Copper	Blue Copper Proteins	[41, 42, 43, 44, 45, 46]
Nickel	Urea Amidohydrolase	[47, 48, 49, 50]
	NikR	[97]
Iron	Cytochrome P450	[98, 99]
Platinum	DNA/Cisplatin	[100]
Copper, Zinc	Superoxide Dismutase	[3]

11 Metal Center Perception

Now with the ability to build and validate metal FFs established the task of generating a generalized FF was initiated. Zinc was chosen as a considerable number of proteins contain that metal as highlighted in Table 12, while also being computationally well behaved. Metalloproteins containing zinc are both structural and functional proteins as described in section 10 and in general Zn is four coordinate, sometimes five or six coordinate when multiple ASP/GLU residues or water molecules bind. It was then necessary to determine all Zn environments which exist in proteins. This was carried out using a program called pdbSearcher to analyze all structures currently in the PDB. pdbSearcher was developed using the API provided by MTK++. All X-ray crystal structures with a resolution below 3.0 Å were extracted from a local mirror of the PDB for further analysis. For each metal site the primary and secondary shell ligands were determined using Harding's bond cut-off values as shown in Table 14 [101, 102, 103, 104, 105, 106]. These values were determined from a series of papers describing metal coordination in the CCSD. A donor atom is considered primary coordinated to a metal if it is within the target distance as shown in Table 14 plus some tolerance (0.5 Å was used). Metal-donor distances lying between the target distance plus the tolerance and the target distance plus a second tolerance (1.0 Å was used) were defined secondary ligands. For example, if a Zn atom is less than 2.53 Å from a Histidine ND1 or NE2 atom then it is considered a primary ligand. If it was less than 3.03 Å away then that ligand is labeled as secondary, otherwise it is unbound. Once the number of primary and secondary shell ligands were determined, the geometry of the metal centers were evaluated. The coordination states allowed include octahedral, Fig. 30(a), Trigonal Bipyramid, Fig. 30(b), Square Pyramid, Fig. 30(c), tetrahedral, Fig. 30(d), square planar, Fig. 30(e), and tetrahedral plus a non-bonded contact, Fig. 30(f). From Fig. 30 we can see that the coordination number alone is not enough to assign a metal geometry. Thus the root mean square deviation (RMSD) of the geometry angles from those in a regular polyhedron were calculated. Equation 23 was used to distinguish between square planar and tetrahedral geometries

with the ideal angles used in Table 15. Likewise, equations 24 and 25 were used for five and six coordinate metals respectively. The atom indices in Table 15 correspond to those atoms in Fig. 30. This indexing is useful to differentiate between axial/equatorial and cis/trans ligands. The coordination state with the lowest rms was assigned to the metal and its ligands.

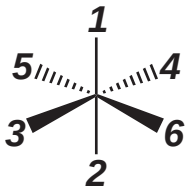
Table 14: Metal-Donor Bond Target Lengths in Å. The following donor atoms of residues are implied: HOH@O, ASP@OD1/OD2, GLU@OE1/OE2, HIS@ND1/NE2, CYS@SG, MET@SG, SER@O, THR@O, TYR@O and the amino acid backbone carbonyl oxygen atom CRL. If a metal-donor distance is within these target distances plus some tolerance (0.5Å) it is considered a primary interaction.

Metal	HOH	ASP/GLU	HIS	CYS/MET	SER/THR	TYR	CRL
Na	2.41	2.41					2.38
Mg	2.07	2.07			2.10	1.87	2.26
K	2.81	2.82					2.74
Ca	2.39	2.36			2.43	2.20	2.36
Mn	2.19	2.15	2.21	2.35	2.25	1.88	2.19
Fe	2.09	2.04	2.16	2.30	2.13	1.93	2.04
Co	2.09	2.05	2.14	2.25	2.10	1.90	2.08
Ni	2.09	2.05	2.14	2.25	2.10	1.90	2.08
Cu	2.13	1.99	2.02	2.15	2.00	1.90	2.04
Zn	2.09	1.99	2.03	2.31	2.14	1.95	2.07

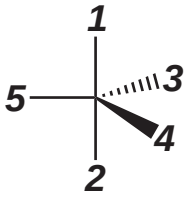
$$\delta_{tet/sqp} = \left[\frac{1}{6}\sum_{i=1}^{6}(a_i - a_{ideal})^2\right]^{1/2} \tag{23}$$

$$\delta_{tbp/ttp} = \left[\frac{1}{10}\sum_{i=1}^{10}(a_i - a_{ideal})^2\right]^{1/2} \tag{24}$$

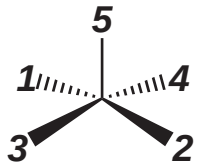
$$\delta_{oct} = \left[\frac{1}{15}\sum_{i=1}^{15}(a_i - a_{ideal})^2\right]^{1/2} \tag{25}$$



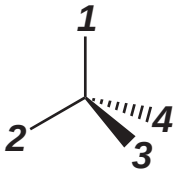
(a) Octahedral



(b) Trigonal Bipyramid

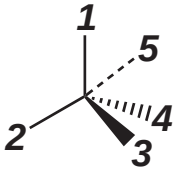


(c) Square Pyramid



(d) Tetrahedral

(e) Square Planar



(f) Tetrahedral plus Non-Bonded Contact

Figure 30: Metal Ligand Geometries Perceived Using Harding's Rules.

Table 15: Ideal Angles Used to Calculate Root Mean Square Deviations for Tetrahedral, Square Planar, Trigonal Bipyramidal, Square Pyramid and Octahedral Geometries. The notation a_{12} describes the angle between atom 1, the metal and atom 2. The atom indices correspond to Fig. 30. b_m is the mean of the four angles between the apical bond and the basal bonds in square pyramid geometries.

Type	Coordination	Angle (Deg)	Atoms
ML_4	Tetrahedral	109.5	
	Square Planar	180.0	a_{12}, a_{34}
		90.0	All others
ML_5	Trigonal Bipyramidal	180.0	a_{12}
		120.0	a_{34}, a_{45}, a_{35}
		90.0	All others
	Square Pyramid	$b_m = \frac{1}{4}\sum_{i=1}^{4} a_{i5}$	$a_{15}, a_{25}, a_{35}, a_{45}$
		$(360.0 - 2b_m)$	a_{12}, a_{34}
		$2\sin^{-1}\left(2^{-1/2}\left[\sin(180.0 - b_m)\right]\right)$	$a_{13}, a_{23}, a_{14}, a_{24}$
ML_6	Octahedral	180.0	a_{12}, a_{34}, a_{56}
		90.0	All others

12 Metal Center Parameter Builder

The goal of this research was to provide a platform to rapidly build, prototype, and validate MM models of metalloproteins using the bonded plus electrostatics model for the AMBER suite of programs [11]. The bonded plus electrostatics model was chosen over the other approaches as the resulting parameters lend themselves to be readily added to FFs such as those in AMBER [8] and CHARMM [12]. Also the functions used in these programs are pairwise additive meaning there are no cross-terms and are thus easier to parameterize and less computationally expensive. The latter is a key point when considering fully solvated metalloproteins in MD simulations can have many hundreds of thousands of atoms. A computer program, MCPB (Metal Center Parameter Builder), to generate FF parameters for metalloproteins was developed to this end. MCPB was not build to supersede the approaches developed by Norrby described above but instead to incorporate a realistic bonded and electrostatic model of the metallocenter into the AMBER FF. The nature of these parameters was investigated in a systematic manner with the objective of creating a generalized metal FF within the bonded plus electrostatics framework. The MCPB program was built using the MTK++ Application Program Interface (API). A complete work flow of MCPB can be seen in Fig. 31. The MCPB program carries out the following steps after a structure is downloaded from the PDB. First the program checks whether the structure contains a transition metal. If the structure does not contain a metal then the program terminates. Otherwise MCPB attempts to determine the primary and secondary ligands of the metal using rules described by Harding [101, 102, 103, 104, 105, 106] which will be described in more detail later in this chapter. Once a

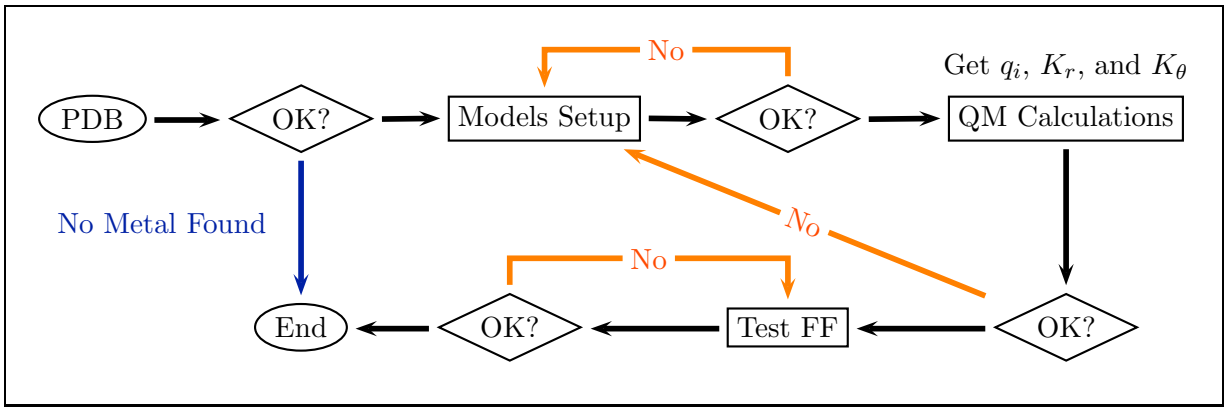


Figure 31: MCPB Flow Diagram where a biomolecular structure is downloaded from the PDB and tested whether it contains a transition metal. If the structure contains a metal ion the MCPB program is used to build and test molecular mechanics force field parameters.

metal site is found, MCPB creates model structures of the metal's first coordination sphere with which *ab initio* calculations can be performed on to generate AMBER-like FF parameters. These models include one to generate charges, q_i, and another to determine bond, K_r, and angle, K_θ, force constants. The AMBER function includes bond, angle, torsion, improper, van der Waals and electrostatic terms; however, only bond, angle and electrostatic terms are parameterized under the assumption made by Loops *et al.* that dihedral terms can be ignored. Lennard-Jones parameters are also not parameterized here due to the fact that most metals are buried and that van der Waals interactions are not as important as the electrostatics [82]. Lennard-Jones parameters for the most common metal ion in biology were taken from the literature [107, 108, 109, 110, 111, 112, 113, 114]. The methods of incorporating the bond, angle and charge parameters are outlined below. Once a FF is produced it is tested using minimization techniques to observe its stability. Further tools such as comparing the frequencies from both *ab initio* and the resulting FF could also be used [115].

12.1 Equilibrium Bond Lengths and Angles

Equilibrium values for bond, r_{eq}, angles, θ_{eq}, can be determined through *ab initio* calculations or taken directly from the crystal structure in the PDB. There are pros and cons for using values from both methods. *Ab initio* calculations are generally carried out in the gas phase but solvent effects can be incorporated with PCM but with an added cost. Crystal structures may contain spurious values and may not be representative of all structures with this bond type. Therefore the values from *ab initio* calculations were used here. Alternatively, data from the CSD could be utilized as well, but as with the approaches described above this method has pros and cons as well. Given our focus on metalloproteins and not solid state metal clusters we have not compared with or used CSD derived information.

12.2 Force Constants

Force constants, K_r and K_θ, are calculated by first creating a model (model 1) of the metal site, adding Hydrogen atoms using the methods described in Section 5 and then optimizing it in the gas phase. The residues bound to the metal are approximated, for example, cysteine by a thiolate or histidine by a methyl-imidazole, to reduce the computational cost of the minimization. However, all bonds and angles missing from the FF were accounted for. Once a minimum is found the second derivatives are determined.

The Cartesian Hessian matrix is shown in Eq. 26, which is the second derivative of energy with respect to coordinates. The eigen-analysis of k provides the force constants, λ_i and the normal modes, $\hat{\nu}_i$ as shown in Eq. 27. The interatomic force constant, K_{AB}, between atoms A and B is required to determine the force on atom A by displacing atom B as shown in Eq. 28 which is required for a MM function.

$$[k] = k_{ij} = \frac{\partial^2 E}{\partial x_i \partial x_j} \tag{26}$$

$$F_i = -[k]\hat{\nu}_i \delta r = -\lambda_i \hat{\nu}_i \delta r \tag{27}$$

$$\delta F_A = [k_{AB}]\delta r_B \tag{28}$$

From the minimized structure of model 1 the metal-Ligand bond and angle force constants are evaluated. The force constants are converted from Cartesian into internal coordinates using the Gaussian program [116] providing the following keyword (iop(7/33=1)). Force constants were also determined using a method described by Seminario [117]. Here the force constants are calculated from sub matrices of the Cartesian Hessian matrix. The bond force constant is calculated using Eq. 29 where λ_i^{AB} are the eigenvalues of the AB Hessian sub matrix, $\hat{\nu}_i^{AB}$ are the corresponding eigenvectors and $\hat{u}^{AB}$ is the unit vector. Similarly, the angle force constants were calculated using the Eq. 30. The Seminario method has the advantage over the "Traditional" method of determining force constants as it avoids defining internal coordinates. The MCPB program then reads either the internal force constant matrix or the Seminario derived parameters and assigns the values to the appropriate bonds and angles.

The MCPB program uses conversion factors of 627.5095 and 2240.87 for $Hartree$ to $kcal/mol$ and $Hartree/Bohr^2$ to $kcal/mol$Å^2 respectively.

$$k_{AB} = \sum_{i=1}^{3} \lambda_i^{AB} |\hat{u}^{AB} \cdot \hat{\nu}_i^{AB}| \tag{29}$$

$$\frac{1}{k_{ABC}} = \frac{1}{R_{AB}^2 \sum_{i=1}^{3} \lambda_i^{AB} |\hat{u}^{PA} \cdot \hat{\nu}_i^{AB}|} + \frac{1}{R_{CB}^2 \sum_{i=1}^{3} \lambda_i^{CB} |\hat{u}^{PC} \cdot \hat{\nu}_i^{CB}|} \tag{30}$$

where:

$$\hat{u}^{ABC} = \frac{\hat{r}^{CB} \times \hat{r}^{AB}}{|\hat{r}^{CB} \times \hat{r}^{AB}|} \quad (31)$$

$$\hat{u}^{PA} = \hat{u}^{ABC} \times \hat{r}^{AB} \quad (32)$$

$$\hat{u}^{PC} = \hat{r}^{CB} \times \hat{u}^{ABC} \quad (33)$$

12.3 Point Charges

The atom centered partial charges were derived using the Merz-Singh-Kollman (MK) [118] and the Restrained ElectroStatic Potential (RESP) [119, 120, 121] schemes using a second model (model 2) of the metal center. This model included all atoms of a bound residue which were capped with acetyl (ACE) and N-methylamine (NME) residues. If two ligating residues were less than five residues apart then they were tethered with glycine residues and the chain capped with ACE and NME. Hydrogen atoms were added using the methods described in section 5. This model was not allowed to relax to save computational expense and to keep the crystallographic geometry. The van der Waals radii for the metals used in the MK scheme were taken from the literature. The MK/RESP scheme was favored over other charge model schemes because its ability to adjust the charge of the capped or linking residues to an integer value, thus allowing the formal charge of the cluster to disperse over the metal and the bound ligands.

Four different methods to develop charges are implemented within MCPB. The first method allows all atoms of the bound residue to change (ChgModA), the second technique restrains the backbone heavy atoms (CA, N, C, O) to those values found in the AMBER parm94 force field (ChgModB), the third one restrains all the backbone atoms (CA, H, HA, N, HN, C, O) to the AMBER parm94 force field values (ChgModC) while the fourth one (ChgModD) also adds carbon beta (CB) into the restraint list.

13 Development History

- **2011-1-15 Martin Peters**
  - Added Eigen3 library
- **2010-8-25** Martin Peters
  - AmberTools Transfer
- **2010-8-19** Martin Peters
  - Added latex documentation
- **2010-3-18** Martin Peters
  - Added tools: capActiveSite, frcmod2xml, func, hybrid, mmE, prep2xml, protonator, sequenceAligner, stats, superimposer, MCPB
  - Updated tests: fileFormatsTest, hybridizeTest, mmTest, ringTest
  - Added test: linearAlgebra
  - Added support for apache-log4cxx-0.10.0
  - Added Diagnostics library (contributed by QuantumBio Inc.)
  - Added reduced boost library v1.38.0
  - Updated version number to 0.2.0
- **2010-3-9** Martin Peters
  - Added reduced boost library v1.33.0
  - Updated version number to 0.1.9
- **2010-2-1** Martin Peters
  - Added tinyxml library for xml parsing
  - Updated version number to 0.1.8
- **2009-4-2** Martin Peters
  - Major overall to mtkppParser
  - Much better support of multi metal containing proteins
  - Updated version number to 0.1.7
- **2008-2-26** Martin Peters
  - Added Trolltech's Qt library for xml parsing

  - Updated version number to 0.1.6

- **2008-1-2** Martin Peters
  - Added seqAlign to molecule for protein alignment
  - Updated version number to 0.1.5
- **2007-11-2** Martin Peters
  - Update Contributed by Lance Westerhoff (QuantumBio)
  - Added support in all Makefiles to configure/build library in a directory other then the main source directory e.g.:

    ```
    cd MTKpp/.. ; mkdir build ; cd build ; ../MTKpp/configure ...
    ```
- **2007-9-28** Martin Peters
  - Added error handling library called Log
  - Removed most programs from tools directory and created a new repository called MTKpp-tools
  - Updated version number to 0.1.4
- **2007-6-2** Martin Peters
  - Added metalCenter to molecule for metalloprotein perception
  - Added amberParser to Parsers for writing prmtop and inpcrd files to interface with AMBER
  - Added more fragments to fragLib
  - Added program for SE-COMBINE
  - Updated version number to 0.1.3
- **2007-3-16** Martin Peters
  - Upgraded superimpose to carry out smart superimposition and rmsd
  - Now using unsigned long long on 64-bit machines for torsion indexing
  - Upgraded watProtonate
  - Added frcmodParser
  - Added copyright to Utils
  - Added pdbSearcher, prep2xml, frcmod2xml, confAnalysis, MTKppConstants and printHeader to tools directory
  - Updated version number to 0.1.2

- **2007-2-7** Martin Peters
  - Upgraded the Statistics Library to use Boost
  - Added dMParser to populate the Sheet and Table classes
  - Added acParser and prepParser
  - Added FragGen, func, aSiteFeatures, capActiveSite, CheckPDB, DataBaser, nmrSimilarity and stats to tools directory
  - Added the hydrophobize class
  - Updated version number to 0.1.1
- **2006-12-9** Martin Peters
  - Added hybridize, bond order and formal charge perception capability
  - Updated version number to 0.1.0
- **2006-11-20** Martin Peters
  - Added Graph class
  - Added MKL support
  - Added mmE and nmrParms to tools dir
  - Updated version number to 0.0.9
- **2006-10-6** Martin Peters
  - Added selection ability
  - Now using the namespace MTKpp
  - Added first derivatives to MM code
  - Updated version number to 0.0.8
- **2006-8-16** Martin Peters
  - Added Maximum Common Pharmacophore detection
  - Added XML based fragment library
  - Added protonator, torProfiler, superimpose, and stdLib2Sdf to tools dir
  - Updated version number to 0.0.7
- **2006-5-23** Martin Peters
  - Added functional group recognition
  - Added basic molecular fingerprint generation
  - Major update to MM code

  - Updated version number to 0.0.6

- **2006-4-19** Martin Peters
  - Added files to the tests directory
  - fileFormats, mm, and ring code is tested
- **2006-3-16** Martin Peters
  - Added tests and tools directories
- **2006-3-15** Martin Peters
  - Added molecule conformer capability - rotatable bond based conformer class
  - Updated version number to 0.0.5
- **2006-3-15** Martin Peters
  - Added molecule superimpose capability
  - Updated version number to 0.0.4
- **2006-2-9** Duane Williams
  - Added pdbParser Write function
- **2006-2-9** Martin Peters
  - Added dcParser Write function
- **2006-1-28** Martin Peters
  - Added hydrogen addition capability - Relies on all the molecular information to be present
  - Updated version number to 0.0.3
- **2006-1-27** Martin Peters
  - Updated sdfParser and molParser Write functions
- **2006-1-19** Martin Peters
  - Added ring perception capability
  - Updated version number to 0.0.2
- **2006-1-15** Martin Peters
  - Updated AUTHORS

- **2006-1-14** Martin Peters
  - Added doxygen documenting ability
- **2006-1-01** Martin Peters
  - Project Started

14 Tests

After MTK++ has been configured and compiled the tests should be run to determine if the package is working correctly using the following commands:

```
cd builddir
make check
```

After the tests have run the following output should be generated:

```
Making check in tests
make  fileFormatsTest linearAlgebraTest ringTest mmTest hybridizeTest
make[2]: 'fileFormatsTest' is up to date.
make[2]: 'linearAlgebraTest' is up to date.
make[2]: 'ringTest' is up to date.
make[2]: 'mmTest' is up to date.
make[2]: 'hybridizeTest' is up to date.
make  check-local
-------------------------------------------------------
Test 'fileFormatsTest' PASSED.
-------------------------------------------------------
Test 'linearAlgebraTest' PASSED.
-------------------------------------------------------
Test 'ringTest' PASSED.
-------------------------------------------------------
Test 'mmTest' PASSED.
-------------------------------------------------------
Test 'hybridizeTest' PASSED.
-------------------------------------------------------
```

Three files are generated for each test: a log, out, and diff files. The log file contains information dumped by the code during execution. The out file contains results of the calculation while the diff file contains the difference between the out file and the expected results.

14.1 File Formats

The file formats test reads and writes of the following file types:

1. elements xml file (Read only)
2. MM parameter xml files (Read/Write)
3. MM standard residue xml files (Read/Write)
4. table xml files (Read/Write)

5. pdb files (Read/Write)
6. state xml files (Write)
7. MDL mol files (Read/Write)
8. sd files (Read/Write)
9. mol2 files (Read/Write)
10. xyz files (Read/Write)
11. pam matrix files (Read)

14.2 Hybridize

The hybridize test assesses MTK++ ability to predict atom hybridizations, bond orders, and formal charges of 261 ligands obtained from the PDB and the literature list in Table 16. See section 3 for more details of how the algorithm works.

14.3 Linear Algebra

This is a very simple test of the connection between MTK++ and boost. Matrix diagonalization and Singular Value Decomposition (SVD) methods are tested.

14.4 Molecular Mechanics

The MM tests evaluates the energy function within MTK++ for 20 pentapeptides contains all amino acids listed below:

```
GGAGG, GGCGG, GGDGG, GGEGG, GGFGG, GGGGG, GGHGG, GGIGG, GGKGG, GGLGG
GGMGG, GGNGG, GGPGG, GGQGG, GGRGG, GGSGG, GGTGG, GGVGG, GGWGG, GGYGG
```

This test outputs the bond, angle, torsion, improper, van der Waals, electrostatic, 1-4 van der Waals and 1-4 electrostatic components of the MM energy function:

```
### GGAGG Energies  9.75857 24.571 16.9042 -4.62884 -361.285 10.4182 313.053 8.79176
```

14.5 Ring

This test also reads the 261 ligands from Table 16 and outputs the number of rings, the SSSR, and whether the rings are planar and/or aromatic:

```
1A42_lig
nRings(2)
SSSR(5,6)
 ring [1] : 2042 2053 2054 2055 2056 planarity = 1 aromaticity = 1
 ring [2] : 2044 2054 2055 2057 2060 2043 planarity = 0 aromaticity = 0
```

See section 4 for more details of the algorithm.

Table 16: Hybridize Ligand Data Set.

Receptor	PDB-ID
MMP-1	**966C**, 1CGL, 1HFC, 2TCL
MMP-3	**1HY7**, 1B3D, 1BQO, 1CIZ, 1D8F, 1G49, 1SLN, 2USN, 1B8Y, 1C3I, 1D5J, 1D8M, 1G4K, 1USN, 1BIW, 1CAQ, 1D7X, 1G05, 1HFS, 2D1O, 1HY7
MMP-7	**1MMQ**, 1MMP, 1MMR
MMP-8	**1I76**, 1A85, 1I73, 1JAP, 1JJ9, 1MNC, 1ZVX, 1A86, 1JAN, 1JAQ, 1KBC, 1ZP5, 2OY2, 1BZS, 1JAO, 1JH1, 1MMB, 1ZS0, 1I76
MMP-12	**1Y93**, 1JIZ, 1JK3, 1RMZ, 1ROS, 2OXW
MMP-13	**830C**, 1CXV, 1XUD, 1YOU, 2D1N, 456C, 1XUC, 1XUR, 1ZTQ, 2OW9
TNF-α converting enzyme	**2DDF**, 1BKC, 1ZXC, 2A8H, 2FV5, 2FV9
Adamalysin II	**4AIG**, 2AIG, 3AIG
Trypsin	1PPH, 1TNH, 1TNI, 1TNJ, 1TNK, 1TNL, 3PTB P0XX (96 Other ligand from literature
Carboxypeptidase A	1CBX, 2CTC, 3CPA
Glycogen Phosphorylase	1GPY, 3GPB, 4GPB, 5GPB
Immunoglobin	1DBB, 1DBJ, 1DBK, 1DBM, 2DBL
Streptavidin	1SRE, 1SRF, 1SRG, 1SRH, 1SRI, 1SRJ
Dihydrofolate Reductase	1DHF, 4DFR
Estrogen Receptor	1ERR, 3ERT
Peroxisome Proliferator-Activated Receptor γ	1FM6, 1FM9
Human Carbonic Anhydrase II	1A42, 1BN1, 1BN3, 1BN4, 1BNM, 1BNN, 1BNQ, 1BNT, 1BNU, 1BNV, 1BNW, 1CIL, 1CIM, 1CIN, 1CNX, 1EOU, 1G1D, 1G52, 1G53, 1G54, 1I8Z, 1I90, 1I91, 1IF4, 1IF5, 1IF6, 1IF7, 1IF8, 1IF9, 1KWQ, 1KWR, 1OKL, 1OKN, 1OQ5, 1TTM, 1XPZ, 1XQ0, 1YDA, 1ZE8
Thrombin	1DWC, 1DWD
Elastase	1ELA, 1ELB, 1ELC, 1ELD, 1ELE
Thermolysin	1TLP, 1TMN, 2TMN, 3TMN, 4TLN, 4TMN, 5TMN
HIV-Protease	1HIV
Endothiapepsin	2ER7, 4ER1, 4ER2, 5ER1, 5ER2
Human Rhinovirus	2R04, 2R06, 2RR1, 2RS3

15 Examples

Most of the examples below use the human Carbonic Anhydrase II co-crystallized structure PDBID: 1A42. All examples can be found in the 'examples' directory that was distributed with the MTK++ code.

15.1 Active Site Capping (capActiveSite)

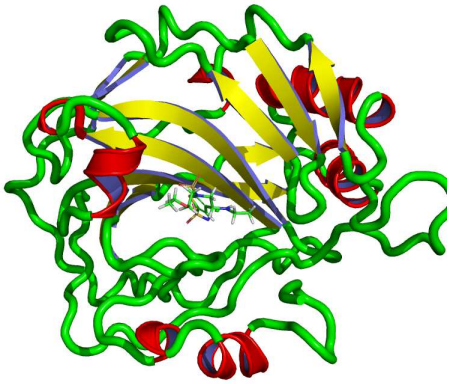
Given a co-crystallized PDB file this utility can cut out the active site and cap the resulting amino acid strands.

The program has the following options:

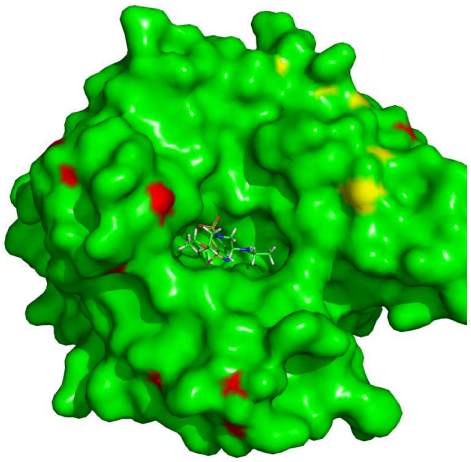
```
capActiveSite: Caps active site using a cutoff

  usage: capActiveSite [flags] [options]

options:
        -r receptor pdb file
        -l ligand mol/pdb file
        -c distance cutoff [10.0]
        -o output pdb file
        -a log file

  flags:
        -h help
```

The capActiveSite example cuts the active site from 1A42 with a distance cut off of 5.0 Ångström. The active site is saved to a pdb file and the resulting model can be visualized in Fig. 32.

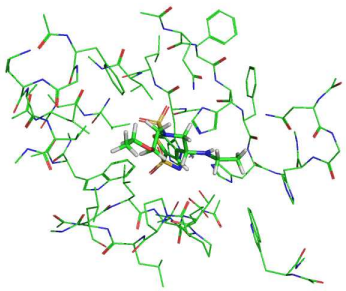
```
#!/bin/csh -f
echo " "
echo "  MTK++ capActiveSite example"
echo " "

../../tools/capActiveSite -r 1A42_rec.pdb \
                          -l 1A42_lig_w.mol \
                          -c 5.0 \
                          -o 1A42_aSite.pdb \
                          -a 1A42_aSite.log  || goto error

echo No errors detected
exit(0)
```



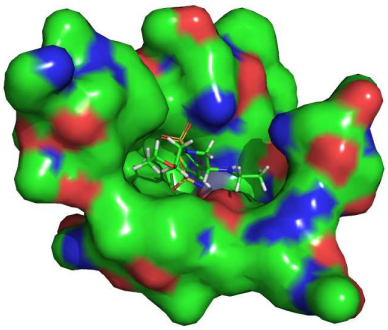
(a) 1A42 Complex - Cartoon/Sticks



(b) 1A42 Complex - Surface



(c) 1A42 Active Site - Sticks



(d) 1CTT Active Site - Surface

Figure 32: Active Site Capping.

```
error:
echo Problem: check .out and try again
exit(1)
```

15.2 File Conversion (frcmod2xml, prep2xml)

The frcmod2xml and prep2xml utility program convert AMBER formatted parameter files into XML files which are used by MTK++. frcmod files are necessary to supplement the core GAFF forcefield with new atom type, bond, angles etc. While the prep file contains the atom names and type, connectivity and partial charges of the newly parameterized compound.

The frcmod2xml program has the following options:

```
frcmod2xml: Converts frcmod file to MTK++ xml file

  usage:  frcmod2xml [flags] [options]

options:
        -i frcmod file
        -o parameter xml file
        -n parameters name
        -a log file

  flags:
        -h help
```

The frcmod2xml example converts the frcmod file created by ANTECHAMBER of the brinzolamide ligand bound to hCAII in the PDB file 1A42. The A42.xml can then be read by other MTK++ tools.

```
#!/bin/csh -f
echo " "
echo "  MTK++ frcmod2xml example"
echo " "

../../tools/frcmod2xml -i A42.frcmod \
                       -o A42.xml \
                       -n A42 \
                       -a A42.log || goto error

echo No errors detected
exit(0)

error:
echo Problem: check .out and try again
exit(1)
```

The prep2xml program has the following options:

```
  prep2xml: Converts prepin file to MTK++ xml file

    usage:  prep2xml [flags] [options]

  options:
          -i prep file
          -o lib xml file
          -g group name
          -f frag name
          -n mol name
          -l hybridize name
          -a log file

    flags:
          -h help
```

The prep2xml example similarly converts the 1A42 ligands prep file in a XML file. Here the user is required to specify the group, fragment and molecule name for the compound.

```
#!/bin/csh -f
echo " "
echo "  MTK++ prep2xml example"
echo " "

../../tools/prep2xml -i A42.prep \
                     -o A42.xml \
                     -g HCAII \
                     -f A42 \
                     -n A42 \
                     -a A42.log || goto error

echo No errors detected
exit(0)

error:
echo Problem: check .out and try again
exit(1)
```

15.3 Hybridize

The hybrid program determine the atom hybridizations and bond orders of molecules in pdb format using the Labute algorithm [7].

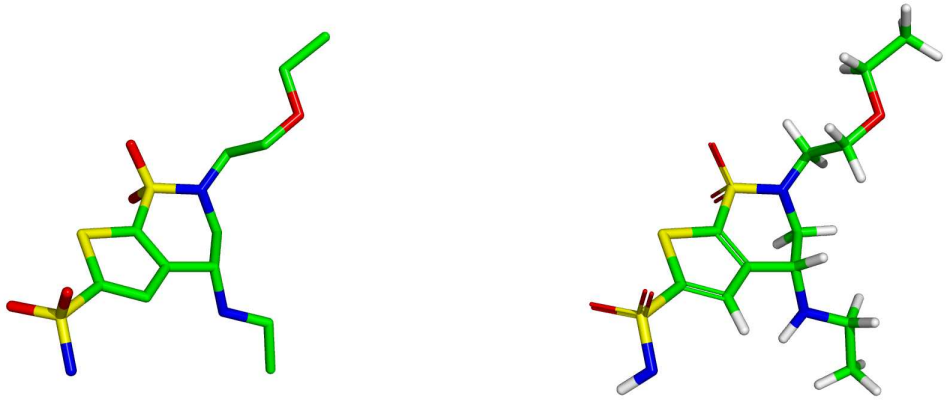
The program takes a pdb file as input and output either a pdb or mol file. A Labute parameter file can be provided but is not required in most cases.

```
  hybrid: Determines bond orders of ligands

    usage:  hybrid [flags] [options]

  options:
          -i input pdb file
          -p parameters file
          -o output mol/pdb file
          -l log file

    flags:
          -h help
```

The hybrid example is run using the following script:

```
#!/bin/csh -f
echo " "
echo "  MTK++ hybrid example"
echo " "

../../tools/hybrid -i 1A42_lig.pdb \
                   -o 1A42_hybrid.mol \
                   -l 1A42_hybrid.log  || goto error

echo No errors detected
exit(0)

error:
echo Problem: check .log and try again
exit(1)
```

The pre and post hybrid structure of the ligand (BZO) in 1A42 are shown in Fig. 33(a) and 33(b) respectively.



(a) 1A42 PDB Ligand (b) 1A42 Post hybrid

Figure 33: Ligand Hybridization.

15.4 Functionalize (func)

MTK++ has a build-in fragment library of over 460 fragments and cores. Using this library the func program reads a MDL mol file and outputs a sd file where properties fields are populated with information regarding which fragments were found.

The func program has the following options:

```
func: Determines the functional groups in ligands

  usage: func [flags] [options]

options:
        -i input MDL mol file
        -o output file
        -a log file

  flags:
        -h help
```

The func example reads the MDL mol file of brinzolamide outputted from the hybrid program and outputs an sd with the fragments listed. The core fragment 'F01' is found to be contained in this ligand. The fragment is a member of the HCAII group and is listed in the sdf file as 'FuncGroup:HCAIIF01'.

```
#!/bin/csh -f
```

```
echo " "
echo "  MTK++ func example"
echo " "

../../tools/func -i 1A42_lig_w.mol \
                 -o 1A42_func.sdf \
                 -a 1A42_func.log  || goto error

echo No errors detected
exit(0)

error:
echo Problem: check .out and try again
exit(1)
```

15.5 MM Energy

The mm program calculates the AMBER gas phase Energy of provided pdb file.
The options of the program are show below:

```
mmE: Calculates the AMBER Energy/Gradients

usage: mmE [options] pdbFile

options:
        -s Standard Library XML File
        -f Frcmod XML File
        -c Non Bonded Cutoff [100.0]
        -b Calculate Bond Energy [1]
        -a Calculate Angle Energy [1]
        -t Calculate Torsion Energy [1]
        -i Calculate Improper Energy  [1]
        -n Calculate Non Bonded Energy [1]
        -w Calculate H Bond Energy [0]

        -m Minimize [0]
          - 0 None
          - 1 Hydrogens Only
          - 2 All Atoms
        -k Minimize Method [2]
          - 0 Steepest Descents
          - 1 Conjugate Gradient (non implemented)
          - 2 lBFGS
        -q Minimize steps [100]
        -p Write Output Every N Steps [1]
        -o Output file
        -z Log file

   flags:
        -v Verbose [1]
        -h Help
```

The mm example calculates the MM energy of the pentapeptide, GGMGG, show in Fig. 34.

```
#!/bin/csh -f
echo " "
echo "  MTK++ mmE example"
echo " "
```

```
../../tools/mmE -z mmE.log -o mmE.out GGMGG.pdb || goto error

echo No errors detected
exit(0)

error:
echo Problem: check .out and try again
exit(1)
```

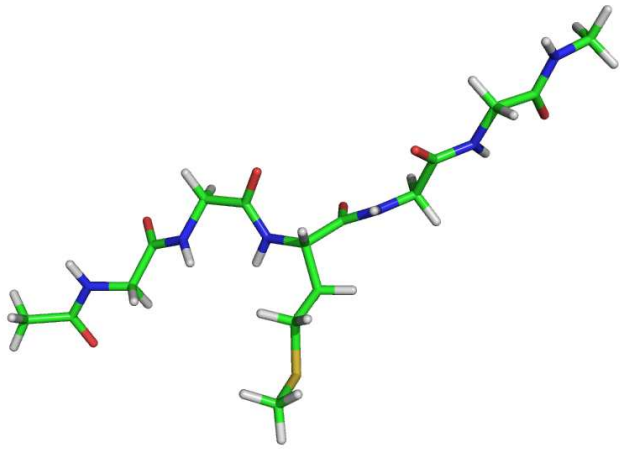


Figure 34: GGMGG Pentapeptide.

The output file contains the following:

```
PDB File                    : GGMGG.pdb
Library File                :
Frcmod File                 :
Non Bonded Cut Off          : 100
Calculate Bond Energy       : 1
Calculate Angle Energy      : 1
Calculate Torsion Energy    : 1
Calculate Improper Energy   : 1
```

```
Calculate NonBonded Energy : 1
Calculate H-Bond Energy    : 0
Minimize                   : 0
Number of Atoms     = 57
Number of Bonds     = 56
Number of Angles    = 97
Number of Torsions  = 139
Number of Impropers = 12
MM Setup took 0 seconds.

 (1)  BOND ENERGY               = 6.63744
 (2)  ANGLE ENERGY              = 23.9241
 (3)  TORSION ENERGY            = 16.7491
 (4)  IMPROPER ENERGY           = 2.20171
 (5)  DIHEDRAL ENERGY (3+4)     = 18.9508
 (6)  1-4 VDW ENERGY            = 11.9489
 (7)  VDW ENERGY                = -4.87678
 (8)  TOTAL VDW ENERGY (6+7)    = 7.0721
 (9)  ELE ENERGY                = -357.212
(10)  1-4 ELE ENERGY            = 309.548
(11)  TOTAL ELE ENERGY (9+10)   = -47.6634
(12)  NON-BONDED ENERGY (8+11)  = -40.5913
(13)  H-BOND ENERGY             = 0
-------------------------------------------------
      TOTAL ENERGY (1+2+5+12)   = 8.92099
      R^6 LENNARD JONES ENERGY  = -29.0228
      R^12 LENNARD JONES ENERGY = 36.0949
```

15.6 Protonate

The protonate programs adds Hydrogen atoms to proteins, ligands and water molecules. It also adds missing atoms from standard residues. The position of polar Hydrogens are also optimized to improve H-Bonding contacts.

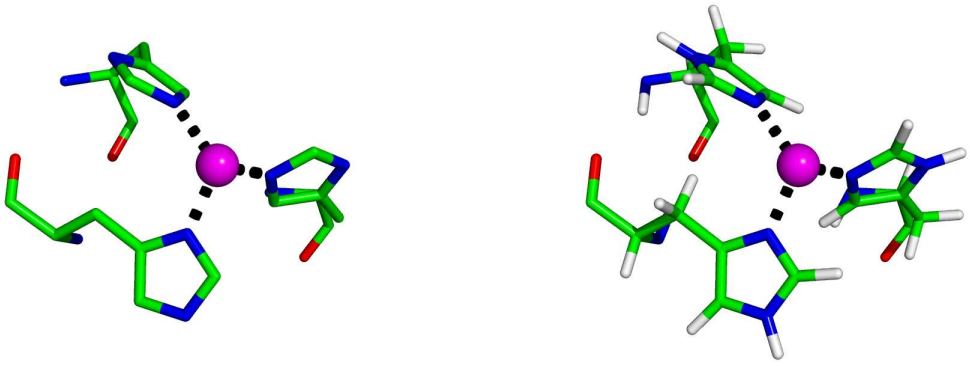
The program takes a pdb file as input and returns a pdb file.

```
  protonator: Adds Hs to Pro/Lig/Wat Molecules

    usage:  protonator [flags] [options]

  options:
          -i input pdb file
          -o output pdb file
          -l log file

    flags:
          -h help
```

The example is executed using the shell script:

```
#!/bin/csh -f
echo " "
echo "  MTK++ protonator example"
echo " "

../../tools/protonator -i 1A42_rec.pdb \
                       -o 1A42_protonator.pdb \
                       -l 1A42_protonator.log  || goto error

echo No errors detected
exit(0)

error:
echo Problem: check .out and try again
exit(1)
```

The addition of Hydrogen to 1A42 is show in Fig. 35(a) and 35(b) to illustrate its ability to handle metalloproteins.



(a) 1A42 Pre protonator (b) 1A42 Protonated

Figure 35: Hydrogen Atom Addition.

15.7 Sequence Alignment

The sequenceAligner programs gives the user the ability to superimpose two structures when the residue and/or the atom numbers are out of sync. This is common when dealing with a family of protein-ligand complexes and researcher would like to see the ligands superimposed using the backbones of the proteins.

The program has two input arguments. The first is a control file and the second is a log file.

```
sequenceAligner: Sequence Alignment and Structural
                 Superimposition

  usage:  sequenceAligner [flags] [options]

options:
        -i input file
        -o log file

  flags:
        -h help
```

An example of where two HCAII complexes (1A42 and 1XQ0) are superimposed is show below.

```
#!/bin/csh -f
echo " "
echo "  MTK++ sequenceAligner example"
```

```
echo " "

../../tools/sequenceAligner -i 1A42_1XQ0.in \
                            -o 1A42_1XQ0_sequenceAligner.log || goto error

echo No errors detected
exit(0)

error:
echo Problem: check .out and try again
exit(1)
```

The input file starts by sourcing a setting file which provides the location of libraries required by MTK++ and can be reused for many sequencedAligner runs. Then the two complexes are read followed by the assignment of disulfide bond, atom types and finally all the atom connections are made.

Then it is necessary to define the template and query doing the alignment. Here 1XQ0 will be superimposed onto 1A42. The runAlign command is used to carry out the sequence alignment. It takes four parameters: algorithm type, gap penalty type, gap open, gap extend. The Needleman-Wunsch (1) and Smith-Waterman (2) algorithms are available. There are three gap penalty types including the constant gap penalty (1), then linear gap penalty (2), and the affine gap penalty (3). For further details of these and the gap open and extend parameters please read elsewhere. The result of the sequence alignment is used to guide the structural superimposition. The superimpose takes one parameter from alphaCarbons, bb, or bbb. alphaCarbons implies that only the coordinates of the 'CA' carbons are used in the superimposition. The bb parameter is used if the user wants to align using the 'CA, N, C, O' atoms while the bbb parameter supplements this with 'CB'. The transformation matrix of this superimposition is saved internally for later use. The tansform command uses this transformation matrix from the previous step to adjust the coordinates of the query structure. The writePdb command saves the alignment structure to a file.

```
source 1A42_1XQ0_settings.txt
#
readPdb 1A42_recLig.pdb
readPdb 1XQ0_recLig.pdb
removeHs
#
assignDisulfideBonds
atomType
assignConnectivity
#
setTemplate 1@start
setQuery 2@start
runAlign 1 3 2.0 0.5
```

```
superimpose alphaCarbons
transform 2@start 2@end alphaCarbons
#
writePdb 2@start 2@end 1A42_1XQ0_sequenceAligner.pdb
quit
```

The settings file contains the following:

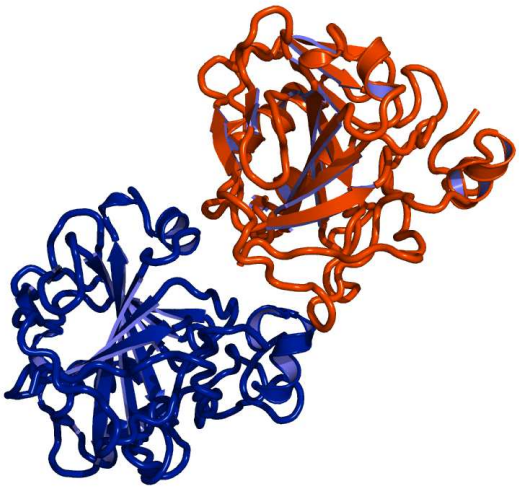
```
# USER INPUT
set MTKppData ../../../../dat/mtkpp
setLoggingLevel 4
#
# READ ELEMENTS FILE
loadElements MTKppData/elements.xml
#
# READ PARAMETER FILES
loadParam MTKppData/parm94.xml
loadParam MTKppData/parm_gaff.xml
#
# READ LIBRARY FILES
loadLib MTKppData/aminont94.xml
loadLib MTKppData/aminoct94.xml
loadLib MTKppData/amino94.xml
#
# READ PAM FILES
loadPam MTKppData/PAM/PAM250
```

The results are placed in the log file. The details of the calculation are summarized including the alignment algorithm used and the resulting RMSD is printed:

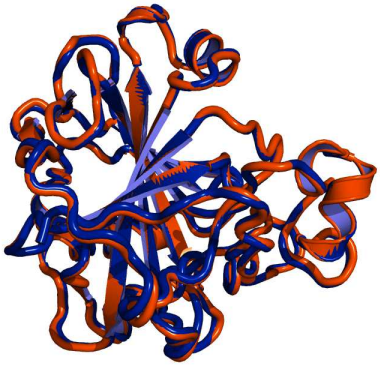
```
### Message:  Results
 Options used:
  Algorithm Type = 1
  Gap Penalty Type = 3
  Gap Open Value = 2
  Gap Extend Value = 0.5
  Maximum Gap Value = 10
  PAM File = AMBERHOME/dat/mtkpp/PAM/PAM250

KEY:
  | == similar
  : == similar
  # == dissimilar
```

```
  ^ == gap

ALIGNMENT
  HWGYGKHNGPEHWHKDFPIAKGERQSPVDIDTHTAKYDPSLKPLSVSYDQATSLRILNNGHAFNV
  ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
  HWGYGKHNGPEHWHKDFPIAKGERQSPVDIDTHTAKYDPSLKPLSVSYDQATSLRILNNGHAFNV

  EFDDSQDKAVLKGGPLDGTYRLIQFHFHWGSLDGQGSEHTVDKKKYAAELHLVHWNTKYGDFGKA
  ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
  EFDDSQDKAVLKGGPLDGTYRLIQFHFHWGSLDGQGSEHTVDKKKYAAELHLVHWNTKYGDFGKA

  VQQPDGLAVLGIFLKVGSAKPGLQKVVDVLDSIKTKGKSADFTNFDPRGLLPESLDYWTYPGSLT
  ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
  VQQPDGLAVLGIFLKVGSAKPGLQKVVDVLDSIKTKGKSADFTNFDPRGLLPESLDYWTYPGSLT

  TPPLLECVTWIVLKEPISVSSEQVLKFRKLNFNGEGEPEELMVDNWRPAQPLKNRQIKASF
  ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
  TPPLLECVTWIVLKEPISVSSEQVLKFRKLNFNGEGEPEELMVDNWRPAQPLKNRQIKASF

STATS
Identicals = 256/256 (100)
Similars = 256/256 (100)
Dissimilars = 0/256 (0)
Gaps = 0/256 (0)

RMSD OF ALPHA-CARBONS = 0.371362
```

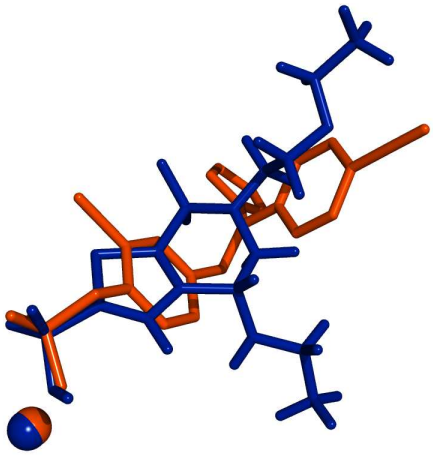
The structures of the pre-aligned pdb files is shown in Fig. 36(a). The aligned proteins and ligands are shown in Fig. 36(b) and 36(c) respectively.



(a) 1A42-1XQ0 Pre Alignment



(b) 1A42-1XQ0 Protein Aligned



(c) 1A42-1XQ0 Ligand Alignment

Figure 36: Structural Alignment and Superimposition.

15.8 Superimposer

The superimposer program carries out structural superimposition of molecules such as proteins or ligands.

```
superimposer: Structural Superimposition

  usage:  superimposer [flags] [options]

options:
        -t template file
        -l template lib xml file
        -f flexible file
        -e flexible lib xml file
        -p aligned file
        -k calculation kind
          :- 1 coordinate RMSD [no fitting]
          :- 2 atom type based RMSD [no fitting] default
          :- 3 RMSD
          :- 4 atom type based RMSD
          :- 5 RMSD & Molecules Superimposed
        -v verbose
          :- 0 rmsd is printed [default]
          :- 1 all output
        -a log file
        -o output file

  flags:
        -h help
```

The superimposer example carries out the superimposition of a conformer of BZO onto the original structure found in the pdb. The novelty lies in the fact that the atom order is not the same in the two files and so the program uses atom type matching functionality to find the optimal alignment. To confirm this the first five atoms from the original file are as follows:

```
ATOM      1  C7  A42     1      -0.878  10.903  15.178 -0.095810
ATOM      2  H5  A42     1       0.092  10.547  14.830  0.053860
ATOM      3  H6  A42     1      -1.194  11.750  14.569  0.034250
ATOM      4  H7  A42     1      -0.798  11.215  16.220  0.036590
ATOM      5  C6  A42     1      -1.915   9.769  15.059  0.126490
```

While the first five atoms from the conformer file are:

```
ATOM      1  C1  A42     1      -5.511   3.062  16.093 -0.202100
```

```
ATOM      2  N1  A42     1      -4.717   1.279  17.877 -1.001370
ATOM      3  O1  A42     1      -5.486   3.439  18.546 -0.682280
ATOM      4  S1  A42     1      -5.693   2.367  17.638  1.271670
ATOM      5  C2  A42     1      -4.138   5.122  11.957  0.131970
```

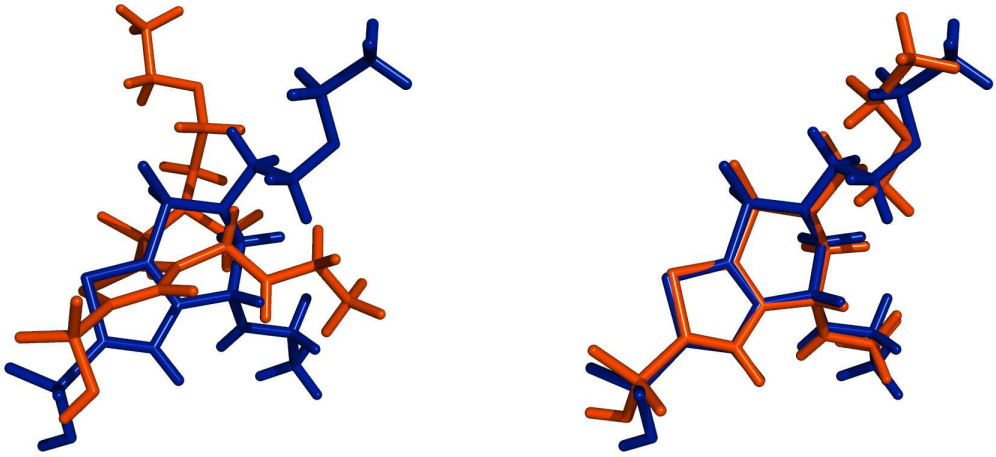
The superimposer example looks like the following:

```
#!/bin/csh -f
echo " "
echo "  MTK++ superimposer example"
echo " "

../../tools/superimposer -t 1A42_lig_H.pdb \
                         -l A42.xml \
                         -f 1A42_conformer.pdb \
                         -k 4 \
                         -v 0 \
                         -p 1A42_superimposer.pdb \
                         -a rmsd.log \
                         -o rmsd.out || goto error

echo No errors detected
exit(0)

error:
echo Problem: check .out and try again
exit(1)
```

The results of the calculation are placed in the out file:

```
RMSD = 0.514339
TYPE = |Atom Type Based RMSD|
```

The pre and post superimposer structures of BZO are shown in Fig. 37(a) and 37(b) respectively.



(a) 1A42-Conformer Pre Alignment (b) 1A42-Conformer Aligned

Figure 37: Ligand Atom Type Based Superimposition.

15.9 pdbSearcher

The pdbSearcher programs gives the user the ability to search a local copy of the Protein Data Bank for metal containing proteins. The program returns hits and geometric information regarding the metal center.

The program has two input arguments. The first is a control file and the second is a log file.

```
  pdbSearcher: Searches a local copy of the
                  Protein Data Bank

    usage:  pdbSearcher [flags] [options]

  options:
          -i input file
          -l log file

    flags:
          -h help
```

```
#!/bin/csh -f
echo " "
echo "  MTK++ pdbSearcher example"
echo " "
```

```
../../tools/pdbSearcher -i list.in \
                        -l list.log || goto error

echo No errors detected
exit(0)

error:
echo Problem: check .out and try again
exit(1)
```

The input file contains a series of commands which pdbSearcher executes.

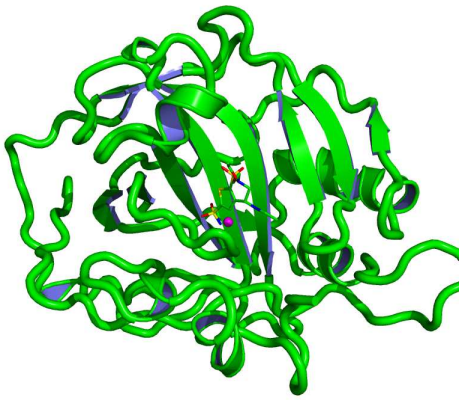
```
# Set up parameters
writeMetalEnvironment 1
expTechniques X-RAY NMR UNKNOWN
resolution 3.0

# Read input list
#  Contains two file names: 1CTT.pdb.Z and 1A42.pdb.Z.
readPDBList pdbList list.txt

# Determine which files have zn
hasMetal pdbList znList zn

# output the geometry of the metal atoms
getEnvironment znList znEnvironments zn

# Clean up
quit
```

Information such as resolution, experiement technique, primary/secondary shell residues is outputted by pdbSearcher:

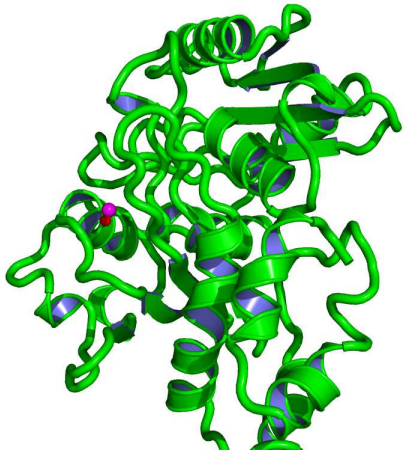
```
FILE_NAME,RESOLUTION,EXP_TECH,TOTAL_NUM_ATOMS,NUM_METAL_ATOMS,RES_NAME,RES_ID,AT_NAME,
     AT_ID,B_FACTOR,PRIM_SHELL,SEC_SHELL,GEOM,GEOM_RMS,ERROR
1CTT.pdb.Z,2.2,UNKNOWN,2286,1, ZN,296,ZN  ,2238,22.4,HCCO,,tet,7.73763,0
1A42.pdb.Z,2.25,X-RAY,2118,1, ZN,262,ZN  ,2041,6.76,HHHX,,tet,9.47689,0
```

More verbose data is also outputted by the program at the atom level:

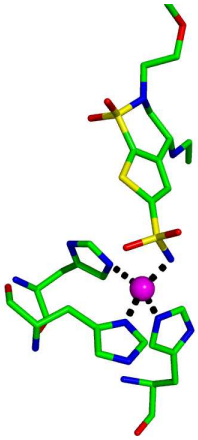
```
FILE_NAME,METAL_RES_NAME,METAL_RES_ID,METAL_NAME,METAL_ID,RES_NAME,RES_ID,AT_NAME,
    AT_ID,DISTANCE,TYPE,B_FACTOR,PRIM_SHELL,SEC_SHELL,GEOM1,GEOM1_RMS,GEOM2,GEOM2_RMS
1CTT, ZN,296,ZN  ,2238,HIS,102, ND1,761,2.0181,p,26.25,CCHO,,tet,7.73763,sqp,41.1116
1CTT, ZN,296,ZN  ,2238,CYS,129, SG ,962,2.4226,p,23.56,CCHO,,tet,7.73763,sqp,41.1116
1CTT, ZN,296,ZN  ,2238,CYS,132, SG ,982,2.11325,p,19.31,CCHO,,tet,7.73763,sqp,41.1116
```



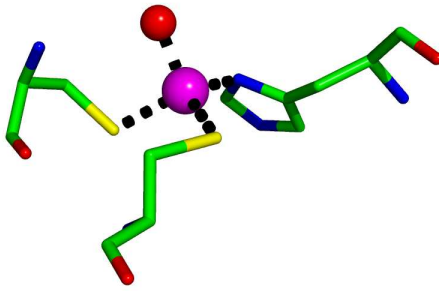
(a) 1A42



(b) 1CTT



(c) 1A42 Metal Center



(d) 1CTT Metal Center

Figure 38: Metal Environment Perception.

```
1CTT, ZN,296,ZN  ,2238,HOH,700, O  ,2287,1.84415,p,13.66,CCHO,,tet,7.73763,sqp,41.1116
1A42, ZN,262,ZN  ,2041,HIS,94, NE2,732,2.11157,p,5.75,HHHX,,tet,9.47689,sqp,39.157
1A42, ZN,262,ZN  ,2041,HIS,96, NE2,753,1.99282,p,2,HHHX,,tet,9.47689,sqp,39.157
1A42, ZN,262,ZN  ,2041,HIS,119, ND1,930,1.97059,p,2.88,HHHX,,tet,9.47689,sqp,39.157
1A42, ZN,262,ZN  ,2041,BZO,555, N21,2051,1.97794,p,2.77,HHHX,,tet,9.47689,sqp,39.157
```

15.10 MCPB

The MCPB programs gives the user the ability to create MM parameters for metal containing proteins using the bonded plus electrostatics model. MCPB uses MTK++'s selection syntax to select subsets of molecular data to operate on, please consult section 2.2 of this manual for more details.

The program has two input arguments. The first is a control file and the second is a log file. A full listing of all the commands used by MCPB can be found by using the -f flag.

```
MCPB: Semi-automated tool for metalloprotein
        parameterization

  usage:  MCPB [flags] [options]

options:
        -i script file
        -l log file
  flags:
        -h help
        -f function list
```

The MCPB example carries out the active site parameterization of a di-zinc system (PDB ID: 1AMP) shown above in Fig. 27(a). The parameterization is broken down into stages since MCPB relies on external packages for results including Gaussian and RESP. Most of the steps are carried out using MCPB but some require user input and instruction. The full example can be run using the run.MCBP.csh script supplied. The Gaussian steps do not need to be carried out as the output necessary can be found in the data folder.

15.10.1 Schematic Generation

The first step in generating parameters for a metal active site is to prepare a schematic diagram similar to the one shown in Fig. 39 using a tool such as ChemDraw.

15.10.2 Source PDB File

The second step is to source a PDB file for the protein of interest. Here the 1AMP file was downloaded from the PDB site and placed in the $AMBERHOME/examples/mtkpp/MCPB/data folder. The 1AMP.pdb file was manually edited where the header and the connect information was removed. Also residues 256 and 935 were renamed HID and MOH respectively. The latter was carried out so that the hydroxyl state was parameterized. This file was saved as called 1AMP_OH_fixed.pdb in the data folder.

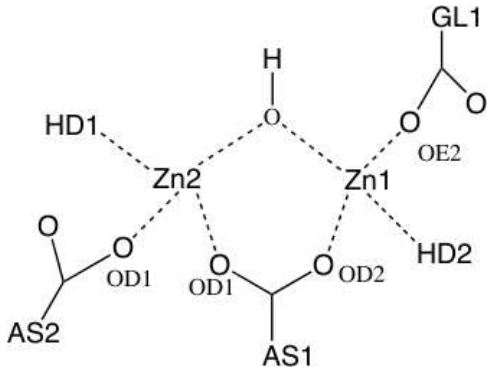


Figure 39: 1AMP schematic.

15.10.3 Generate the MCPB scripts

The script found in genMetalFF.sh is used to generate various files that are used during the creation of FF parameters. The name of the protein is passed to the shell script and it is this name that is used to distinguish parameter sets.

```
sh genMetalFF.sh -n 1AMP_OH
```

15.10.4 Settings file

One of the file generated in the previous step is the settings file 1AMP_OH_settings.bcl. This file is not run directly but is instead "sourced" in most of the other bcl scripts. It contains a number of commands, including setting the variable NAME and loading the desired parameter databases and fragment libraries. An example of a settings file looks like the following where AMBERHOME is replaced by the users environment variable.

```
set NAME 1AMP_OH

# Load AMBER Parameters
loadParam AMBERHOME/data/parm94.xml
loadParam AMBERHOME/data/parm_gaff.xml
loadParam AMBERHOME/data/metals/metalParm.xml

# Load AMBER libraries
loadLib AMBERHOME/data/amino94.xml
loadLib AMBERHOME/data/aminont94.xml
loadLib AMBERHOME/data/aminoct94.xml
loadLib AMBERHOME/data/fragLib/terminal.xml
loadLib AMBERHOME/data/metals/metals.xml
```

The settings file can be alter to use other libraries and parameters. For example, the ff99SB (param99.xml) forcefield can be used instead of the default FF94 (param94.xml) or add non standard aminos such lysine NZ-carboxylic acid or carboxymethylated cysteine (AMBERHOME/-dat/mtkpp/nonStandardAAs/nonstandardAAs.xml).

15.10.5 Structural Preparation

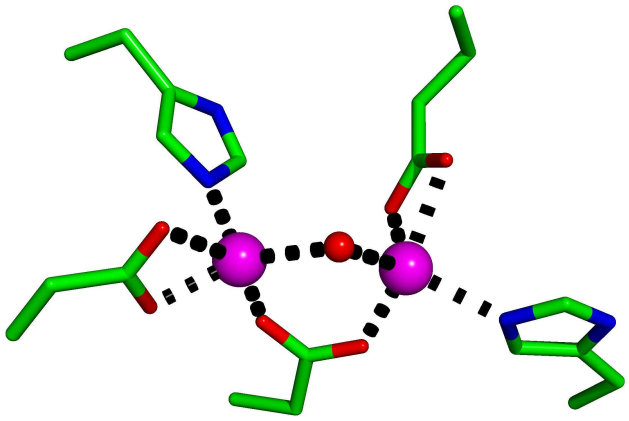
To add Hydrogens, define disulfide bonds, and set Histidine residue names to the fixed pdb file the following command is used:

```
MCPB -i 1AMP_OH_addHs.bcl \
     -l 1AMP_OH_addHs.bcl.log
```

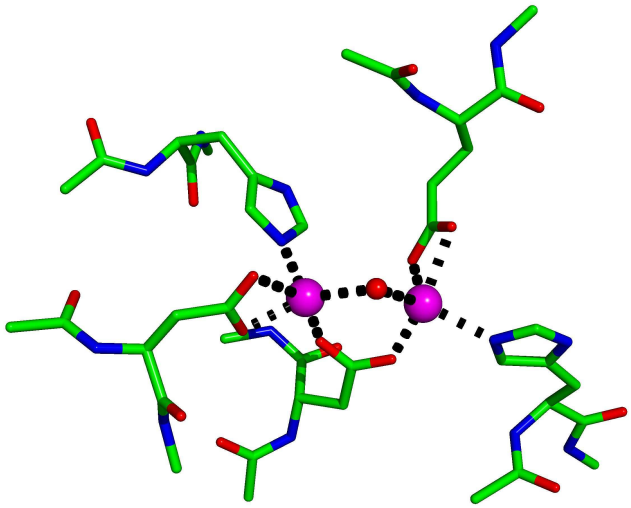
15.10.6 Side Chain Model

The next step and probably the most time consuming is the preparation of the MCPB script files to create the first of the two models shown in Fig. 40. The first model (Model 1 (Fig. 40(a))) is required to determine the new bond and angle force constants while the second (Model 2, Fig. 40(b)) is used to generate partial charges. Model 1 is created using the 1AMP_OH_sidechain.bcl file stored in the data folder. This file is not created by the genMetallFF script and must be created by hand. Most of the file should be fairly self-explanatory once the comments are considered.

```
#
# Load settings
#
source 1AMP_OH_settings.bcl

#
# Create non-standard residue library
#
createStdGroup NAME
copyStdResidue aminoAcids94/ASP NAME/AS1
copyStdResidue aminoAcids94/HID NAME/HD1
copyStdResidue aminoAcids94/GLU NAME/GL1
copyStdResidue aminoAcids94/HID NAME/HD2
copyStdResidue aminoAcids94/ASP NAME/AS2
copyStdResidue aminoAcids94/MOH NAME/OH1
copyStdResidue metals/.ZN NAME/ZN1
copyStdResidue metals/.ZN NAME/ZN2

#
# Create new atom types
#
copyAtomType parm94/O2 NAME/OA
copyAtomType parm94/O2 NAME/OB
copyAtomType parm94/O2 NAME/OC
```



(a) 1AMP Model 1



(b) 1AMP Model 2

Figure 40: 1AMP MCPB Models

```
# periods are replaced by spaces internally
setAtomType NAME/AS1/.OD1 NAME/OA
setAtomType NAME/AS1/.OD2 NAME/OA
setAtomType NAME/GL1/.OE1 NAME/OB
setAtomType NAME/GL1/.OE2 NAME/OB
setAtomType NAME/AS2/.OD1 NAME/OC
setAtomType NAME/AS2/.OD2 NAME/OC

#
# write standard library
#
writeLib NAME NAME.xml

#
# Open PDB file
#
readPdb NAME NAME_fixed.pdb

# Assign S-S bonds
assignDisulfideBonds

# Atom type
atomType

# Assign bonds, angles, and torsions
assignConnectivity

# Add hydrogens
addHs /NAME/

#
# Build Cluster
#

# Create molecule named CLR
createMolecule CLR

# Create HD1-1
createResidue HD1 in CLR
addToResidue /NAME/CLR/HD1 /NAME/1/HID-97 not bb

# Create AS2-2 from ASP-179
createResidue AS2 in CLR
addToResidue /NAME/CLR/AS2 /NAME/1/ASP-179 not bb

# Create AS1-3 from ASP-117
createResidue AS1 in CLR
```

```
addToResidue /NAME/CLR/AS1 /NAME/1/ASP-117 not bb

# Create GL1-4 from GLU-152
createResidue GL1 in CLR
addToResidue /NAME/CLR/GL1 /NAME/1/GLU-152 not bb

# Create HD2-5 from HID-256
createResidue HD2 in CLR
addToResidue /NAME/CLR/HD2 /NAME/1/HID-256 not bb

# Create ZN1-6 from ZN-501
createResidue ZN1 in CLR
addToResidue /NAME/CLR/ZN1 /NAME/2/.ZN-501/ZN..

# Create ZN2-7 from ZN-502
createResidue ZN2 in CLR
addToResidue /NAME/CLR/ZN2 /NAME/3/.ZN-502/ZN..

# Create OH1-8 from MOH-935
createResidue OH1 in CLR
addToResidue /NAME/CLR/OH1 /NAME//MOH-935

# Methyl terminating groups (Add these after all other residue have been added)
addFragment terminal/CH3 bd /NAME/CLR/HD1-1/.CB. ag /NAME/CLR/HD1-1/.CG. tr /NAME/CLR/HD1-1/.ND1 87.0
addFragment terminal/CH3 bd /NAME/CLR/AS2-2/.CB. ag /NAME/CLR/AS2-2/.CG. tr /NAME/CLR/AS2-2/.OD2 170.0
addFragment terminal/CH3 bd /NAME/CLR/AS1-3/.CB. ag /NAME/CLR/AS1-3/.CG. tr /NAME/CLR/AS1-3/.OD2 200.0
addFragment terminal/CH3 bd /NAME/CLR/GL1-4/.CB. ag /NAME/CLR/GL1-4/.CG. tr /NAME/CLR/GL1-4/.CD. 288.0
addFragment terminal/CH3 bd /NAME/CLR/HD2-5/.CB. ag /NAME/CLR/HD2-5/.CG. tr /NAME/CLR/HD2-5/.ND1 185.0

# Create bonds with zinc1
createBond /NAME/CLR/ZN1/ZN.. /NAME/CLR/HD2/.NE2
createBond /NAME/CLR/ZN1/ZN.. /NAME/CLR/AS1/.OD2
createBond /NAME/CLR/ZN1/ZN.. /NAME/CLR/GL1/.OE2
createBond /NAME/CLR/ZN1/ZN.. /NAME/CLR/OH1/.O..

# Create bonds with zinc2
createBond /NAME/CLR/ZN2/ZN.. /NAME/CLR/HD1/.NE2
createBond /NAME/CLR/ZN2/ZN.. /NAME/CLR/AS1/.OD1
createBond /NAME/CLR/ZN2/ZN.. /NAME/CLR/AS2/.OD1
createBond /NAME/CLR/ZN2/ZN.. /NAME/CLR/OH1/.O..

# Atomtype
atomType /NAME/CLR

# Assign bonds and angles
assignParameters /NAME/CLR
```

```
# Add Bond and Angle Parameters
addBondAndAngleParameters /NAME/CLR NAME

# Write parameters file
writeParams NAME NAME_params.xml

# Write pdb/mol files (Good idea to view one of these files
#                      before running the Gaussian calc.)
writePdb /NAME/CLR NAME_sidechain.pdb
writeMol /NAME/CLR NAME_sidechain.mol
writeSdf /NAME/CLR NAME_sidechain.sdf

# Gaussian options
levelOfTheory B3LYP
basisSet 6-31G*
clusterCharge CLR 0
clusterSpin 1
gaussianNProc 2
gaussianMem 3000MB

# Set Gaussian input name --> Optimize and get force constants
gaussianOptAndFC /NAME/CLR NAME_sidechain.com

# Exit MCPB
quit
```

The small model is generated using the following command:

```
MCPB -i 1AMP_OH_sidechain.bcl \
     -l 1AMP_OH_sidechain.bcl.log
```

15.10.7 Standard Molecule

This step adds information to the final xml library file which can be used to determine if another pdb file contains a particular active site. The file 1AMP_OH_addStdMol.bcl is also built by hand but it is very similar to the sidechain bcl file.

15.10.8 Side Chain Model Optimization/Frequency Calculation

It is assumed that the users of MCPB can use Gaussian to carry out optimization and frequency calculations. The files 1AMP_OH_sidechain_opt.com and 1AMP_OH_sidechain_fc.com generated by genMetallFF are used.

15.10.9 Large Model

This step builds a large metal cluster containing all parts of all residues forming the metal cluster. Residues are capped with acetyl and N-methylamino groups. Where two residues participating in

the cluster have exactly one residue between them, it is easier to include the extra residue, instead of replacing it with ACE and NME. An example bcl script is shown below:

```
# Load settings
source 1AMP_OH_settings.bcl

loadParam NAME_params.xml
loadLib NAME.xml

# Open PDB file
readPdb NAME NAME_fixed.pdb

# Reset name of HIS-256 to HID
# setResidueName /NAME/1/HIS-256 to HID

# setMoleculeName /NAME//HOH-935 to MOH
# setResidueName /NAME//HOH-935 to MOH

# Assign S-S bonds
assignDisulfideBonds

# Atom type
atomType

# Assign bonds, angles, and torsions
assignConnectivity

# Add hydrogens
addHs /NAME

#
# Cluster
#

# Create molecule named CLR
createMolecule CLR
setMaxFileID /NAME/CLR /NAME/1

# Create ACE-1 from GLY-96
createResidue ACE in CLR
addToResidue /NAME/CLR/ACE-1 /NAME/1/GLY-96/.CA.
setAtomName /NAME/CLR/ACE-1/.CA. to .CH3
```

```
addToResidue /NAME/CLR/ACE-1 /NAME/1/GLY-96/.C..
addToResidue /NAME/CLR/ACE-1 /NAME/1/GLY-96/.O..

# Create HD1-2
createResidue HD1 in CLR
addToResidue /NAME/CLR/HD1 /NAME/1/HID-97

# Create NME-3 from LEU-98
createResidue NME in CLR
addToResidue /NAME/CLR/NME-3 /NAME/1/LEU-98/.N..
addToResidue /NAME/CLR/NME-3 /NAME/1/LEU-98/.CA.
setAtomName /NAME/CLR/NME-3/.CA. to .CH3

# Create ACE-4 from LEU-178
createResidue ACE in CLR
addToResidue /NAME/CLR/ACE-4 /NAME/1/LEU-178/.CA.
setAtomName /NAME/CLR/ACE-4/.CA. to .CH3
addToResidue /NAME/CLR/ACE-4 /NAME/1/LEU-178/.C..
addToResidue /NAME/CLR/ACE-4 /NAME/1/LEU-178/.O..

# Create AS2-5 from ASP-179
createResidue AS2 in CLR
addToResidue /NAME/CLR/AS2 /NAME/1/ASP-179

# Create NME-6 from VAL-180
createResidue NME in CLR
addToResidue /NAME/CLR/NME-6 /NAME/1/MET-180/.N..
addToResidue /NAME/CLR/NME-6 /NAME/1/MET-180/.CA.
setAtomName /NAME/CLR/NME-6/.CA. to .CH3

# Create ACE-7 from ASP-116
createResidue ACE in CLR
addToResidue /NAME/CLR/ACE-7 /NAME/1/ASP-116/.CA.
setAtomName /NAME/CLR/ACE-7/.CA. to .CH3
addToResidue /NAME/CLR/ACE-7 /NAME/1/ASP-116/.C..
addToResidue /NAME/CLR/ACE-7 /NAME/1/ASP-116/.O..

# Create AS1-8 from ASP-117
createResidue AS1 in CLR
addToResidue /NAME/CLR/AS1 /NAME/1/ASP-117

# Create NME-9 from ASP-118
```


```
createResidue NME in CLR
addToResidue /NAME/CLR/NME-9 /NAME/1/ASP-118/.N..
addToResidue /NAME/CLR/NME-9 /NAME/1/ASP-118/.CA.
setAtomName /NAME/CLR/NME-9/.CA. to .CH3

# Create ACE-10 from GLU-151
createResidue ACE in CLR
addToResidue /NAME/CLR/ACE-10 /NAME/1/GLU-151/.CA.
setAtomName /NAME/CLR/ACE-10/.CA. to .CH3
addToResidue /NAME/CLR/ACE-10 /NAME/1/GLU-151/.C..
addToResidue /NAME/CLR/ACE-10 /NAME/1/GLU-151/.O..

# Create GL1-11 from GLU-152
createResidue GL1 in CLR
addToResidue /NAME/CLR/GL1 /NAME/1/GLU-152

# Create NME-12 from VAL-153
createResidue NME in CLR
addToResidue /NAME/CLR/NME-12 /NAME/1/VAL-153/.N..
addToResidue /NAME/CLR/NME-12 /NAME/1/VAL-153/.CA.
setAtomName /NAME/CLR/NME-12/.CA. to .CH3

# Create ACE-13 from ILE-255
createResidue ACE in CLR
addToResidue /NAME/CLR/ACE-13 /NAME/1/ILE-255/.CA.
setAtomName /NAME/CLR/ACE-13/.CA. to .CH3
addToResidue /NAME/CLR/ACE-13 /NAME/1/ILE-255/.C..
addToResidue /NAME/CLR/ACE-13 /NAME/1/ILE-255/.O..

# Create HD2-14 from HID-256
createResidue HD2 in CLR
addToResidue /NAME/CLR/HD2 /NAME/1/HID-256

# Create NME-15 from THR-257
createResidue NME in CLR
addToResidue /NAME/CLR/NME-15 /NAME/1/THR-257/.N..
addToResidue /NAME/CLR/NME-15 /NAME/1/THR-257/.CA.
setAtomName /NAME/CLR/NME-15/.CA. to .CH3

# Create ZN1-16 from ZN-501
createResidue ZN1 in CLR
addToResidue /NAME/CLR/ZN1 /NAME/2/.ZN-501/ZN..
```

```
# Create ZN2-17 from ZN-502
createResidue ZN2 in CLR
addToResidue /NAME/CLR/ZN2 /NAME/3/.ZN-502/ZN..

# Create OH1-18 from HOH-935
createResidue OH1 in CLR
addToResidue /NAME/CLR/OH1 /NAME//MOH-935

# Create bonds with zinc
createBond /NAME/CLR/ZN1/ZN.. /NAME/CLR/HD2/.NE2
createBond /NAME/CLR/ZN1/ZN.. /NAME/CLR/AS1/.OD2
createBond /NAME/CLR/ZN1/ZN.. /NAME/CLR/GL1/.OE2
createBond /NAME/CLR/ZN1/ZN.. /NAME/CLR/OH1/.O..

createBond /NAME/CLR/ZN2/ZN.. /NAME/CLR/HD1/.NE2
createBond /NAME/CLR/ZN2/ZN.. /NAME/CLR/AS1/.OD1
createBond /NAME/CLR/ZN2/ZN.. /NAME/CLR/AS2/.OD1
createBond /NAME/CLR/ZN2/ZN.. /NAME/CLR/OH1/.O..

# Atomtype
atomType /NAME/CLR

# Add Hs
addHs /NAME/CLR

# Write pdb/mol files
writePdb /NAME/CLR NAME_large.pdb
writeMol /NAME/CLR NAME_large.mol
writeSdf /NAME/CLR NAME_large.sdf

# Gaussian options
levelOfTheory B3LYP
basisSet 6-31G*
clusterCharge CLR 0
clusterSpin 1
gaussianNProc 2
gaussianMem 3000MB
setMKRadii zn 1.1

# Set Gaussian input name
gaussianCharges /NAME/CLR NAME_large.com
```

```
# Exit MCPB
quit
```

15.10.10 Large Model Charge Calculation

Once the large model is created the partial charges can be calculated using Gaussian and the 1AMP_OH_large_mk.com file.

15.10.11 RESP

Once the large model Gaussian calculation is complete the RESP calculations can be carried out. The genMetalFF script produces four files relating to the different charge models. Each of these supplies different constraints to RESP.

```
MCPB -i 1AMP_OH_large_mk1.bcl \
     -l 1AMP_OH_large_mk1.bcl.log
mv 1AMP_OH_large_respAdds 1AMP_OH_large_respAdds1
sh ./getCharges.sh 1
```

15.10.12 Create XML Libraries

Once RESP is finished then the final xml files (ChgMod A-D) can be generated:

```
MCPB -i 1AMP_OH_large_chg1.bcl \
     -l 1AMP_OH_large_chg1.bcl.log
```

15.10.13 Create FF Modification Files

This step requires the output from the frequency calculation of the sidechain model. You will need to convert the chk file into a formatted checkpoint file. The following command will produce an MTK++ formatted xml file containing the Seminario derived bond and angle parameters.

```
MCPB -i 1AMP_OH_sidechain_fc_sem.bcl \
     -l 1AMP_OH_sidechain_fc_sem.bcl.log
```

15.10.14 Create AMBER prep and frcmod Files

To use the MCPB generated parameters and charges the MTK++ file must be converted to something that AMBER utilities can use. The following command creates an frcmod file and a prep file to be used with leap.

```
MCPB -i 1AMP_OH_toAmberFormats_sem.bcl \
     -l 1AMP_OH_toAmberFormats_sem.bcl.log
```

15.10.15 Control Script Syntax

- **addBondAndAngleParameters** <selection> <group name>
  Add bond and angle parameters if missing
- **addFragment** <fragment> bd <atom1> ag <atom2> tr <atom3>
  Add Fragment to an atom.
  syntax: addFragment 6MemRings/6CH bd /col/Mol//34 ag /col/Mol//27 tr /col/Mol//9 180.0
- **addHs** <selection>
  Add Hydrogens to the selection
- **addStdMol** <molecule selection> <group>
  Add standard molecule to lib file
- **addToResidue** <residue selection 1> <residue selection 2> and/not <bb>
  Adds atom from one residue into another. bb values:
  - bb_heavy == backbone [ca, n, c, o]
  - bb == backbone [ca, h, ha, n, nh, c, o]
  - bbb == backbone [ca, h, ha, n, nh, c, o, cb]

  syntax: addToResidue /1FWJ/CLR/H11-5 /1FWJ/7/HIS-104 not bb
- **appendResidue** <residue selection> <atom selection>
  Adds selected atom into selection residue.
  syntax: appendResidue /1FEE/cuCYM4/CY1 /1FEE/1/CYS-12/.CB.
- **assignConnectivity**
  Assigns all bonds, angles, torsions and impropers in collection.
- **assignDisulfideBonds**
  Assigns all disulfide bonds (Needs to be carried out before atom typing).
- **assignParameters** <selection>
  Assigns bond/angle/torsion/improper parameters.
- **assignStdFeatures** <selection>
  Assigns standard features to molecule.
- **atomType**
  Assigns atom types in the collection.

- **basisSet** <options>
  Set Gaussian Basis Set. options:
  - Basis set Name. e.g. 6-31G*
  - Gen Basis with a supplied basis set file. e.g. GEN.6D.7F bs.txt.
- **capResidue** <atom selection> <ACE or NME>
  Cap residue residues with NME or ACE.
  syntax: capResidue /1FEE/cuCYM4/CY1/.N.. ACE
- **clusterCharge** <cluster> <charge>
  Set cluster charge for Gaussian calculation.
- **clusterSpin** <cluster> <spin>
  Set cluster spin state for Gaussian calculation.
- **copyAtomType** <existing atom type> <new atom type>
  Copy atom type details into a new atom type.
  syntax: copyAtomType parm94/NB 1CA2/NX
- **copyStdResidue** <existing residue> <new residue>
  Copy residue to create a new residue.
  syntax: copyStdResidue aminoAcids94/HIS myLib/HS1
- **createBond** <atom selection 1> <atom selection 2>
  Create bond.
- **createMolecule** <molecule name>
  Create molecule.
- **createResidue** <residue name> in <molecule selection>
  Create residue within a molecule.
- **createStdGroup** <name>
  Create standard group.
- **findMetalCenters**
  Find all metal centers in the collection.
- **g03Charges** <molecule selection> <filename>
  Generate a Gaussian input file for partial-charge computation.

- **g03Mem** <memory requirement>

  Set amount of memory requested for g03.

- **g03MoldenFormat**

  Request Gaussian log files formatted for viewing in Molden (adds the GFINPUT and IOP(6/7=3) keywords).

- **g03nProc** <number>

  Set number of processors requested for g03.

- **g03OptAndFC** <molecule selection> <filename>

  Generate Gaussian input files for optimisation and force constants.

- **g03Verbosity** <level>

  Set the verbosity of Gaussian output ([T]erse, [N]ormal, [P]rolix).

- **levelOfTheory** <theory>

  Set Gaussian Theory Level.

- **listFragments** <library name>

  List available fragments in a particular library.

- **loadLib** <library file>

  Loads AMBER library files into MCPB.

- **loadParam** <parameter file>

  Loads AMBER Parameters into MCPB.

- **modRedundant** <filename>

  Add modReduntant definitions to gaussian input file.

- **nmodeMatch** <filename>

  Compare nMode and Gaussian Normal Modes.

- **optimizePolarHs**

  Optimize polar hydrogens in a collection.

- **print** <selection>

  Print to screen details of structure.

- **printAtomTypes**

  Print Available atom types.

- **pseudoPotentials** <filename>
  Add pseudo potential definitions to gaussian input file.
- **quit**
  Exit MCPB.
- **readFormattedChkPtFile** <fchk file>
  Read Formatted Checkpoint file.
- **readG03Output** <gaussian output file>
  Read G03 Output.
- **readMolZmatMapping** <mapping file>
  Read Molecule to Z-Matrix mapping file
- **readNMode** <molecule selection> <nmode output file>
  Read frequencies from nmode.
- **readNModeVectors** <molecule selection> <nmode vecs file> <molden filename>
  Read eigenvalues and eigenvectors from nmode vecs file.
- **readPdb** <object name> <pdb file>
  Reads a PDB file.
- **readRespCharges** <molecule selection> <resp chg file>
  Read RESP charges into molecule.
- **readSdf** <object name> <filename>
  Read sd file.
- **renumber**
  Renumbers atoms and residues in the collection.
- **respgenAdditions** <groupname> <filename> <bb>
  Add info to respgen files. bb Definitions:
  - 0 No restraints
  - 1 Heavy Atoms in Backbone (bb_heavy, [ca, n, c, o]) set to parm94 values
  - 2 Atoms in Backbone (bb, [ca, h, ha, n, nh, c, o]) set to parm94 values
  - 3 Atoms in Backbone plus CB (bbb, [ca, h, ha, n, nh, c, o, cb]) set to parm94 values

- **set** <variable name> <value>

  Set variable.

- **setAtomName** <atom selection> to <atom name>

  Set atom name.

  syntax: setAtomName /1CA2/znCLR/ACE-1/.CA. to .CH3

- **setAtomType** <atom selection> <atom type>

  Set atom type.

  syntax: setAtomType myLib/HS1/.NE2 1CA2/NX

- **setFormalCharge** <atom selection> <value>

  Set Formal Charge on atom.

- **setLoggingLevel** <value>

  Set the verbosity of error/warning/info messages. Values:

  - 1 - Error
  - 2 - Warning
  - 3 - Debug
  - 4 - Info

- **setMaxFileID** <molecule selection 1> <molecule selection 2>

  Set the id of a new atom in the collection.

- **setMKRadii** <element> <value>

  Set Merz-Kollman radii for element.

- **setResidueName** <residue selection> to <3 Letter code>

  Set residue name.

  syntax: setResidueName /1CA2/1/HIS-119 to HIE

- **source** <file name>

  Sources a global file.

- **updateForceConstants** <molecule selection> <group> <X> <Y>

  Determine force constants and add them to parm file. X Values:

  - 0 Do not update bonds and angle equilibrium values
  - 1 Do update bonds and angle equilibrium values (req)

Y Values:

- 0 Seminario Method
- 1 Z-matrix Method

- **updateFrequencies** <molecule selection> <value>
  Scale frequencies.
- **updateRespCharges** <molecule selection> <group>
  Add RESP charge to lib file.
- **writeData** <filename>
  Write contents.
- **writeFrcmodFile** <frcmod file> <object>
  Writes AMBER Parameters.
- **writeLeap** <name> <pdb file>
  Write the metal center bonding info for leap.
- **writeLib** <group name> <file name>
  Write standard library.
- **writeMol** <molecule selection> <file name>
  Write mol file.
- **writeParams** <group name> <filename>
  Write all new parameters.
- **writePdb** <selection> <pdb file>
  Write pdb file.
- **writePrepFile** <prep file> <object>
  Writes AMBER prep file.
- **writePrmtop** <coord file> <prmtop file>
  Write prmtop and coordinate files.
- **writeSdf** <selection> <file name>
  Write sd file.
- **writeState** <file name>
  Write state xml file.

References

[1] The Apache Project. *Xerces-C++ Parser*. http://xml.apache.org/xerces-c/ (accessed Oct 1, 2005).

[2] A. M. Wollacott. *Computational studies of the applicability of semiempirical quantum mechanical methods to study protein structure*. PhD thesis, The Pennsylvania State University, 2005.

[3] R. J. F. Branco, P. A. Fernandes, and M. J. Ramos. Molecular dynamics simulations of the enzyme cu, zn superoxide dismutase. *J. Phys. Chem. B*, 110(33):16754–16762, 2006.

[4] T. Wang and J. J. Zhou. 3DFS: A new 3D flexible searching system for use in drug design. *J. Chem. Inf. Comput. Sci.*, 38(1):71–77, 1998.

[5] P. Willett. Searching techniques for databases of two- and three-dimensional chemical structures. *J. Med. Chem.*, 48(13):4183–4199, 2005.

[6] R. Diestel. *Graph theory*. Springer, Berlin, 2005.

[7] P. Labute. On the perception of molecules from 3D atomic coordinates. *J. Chem. Inf. Model.*, 45(2):215–221, 2005.

[8] W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz Jr., D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell, and P. A. Kollman. A 2nd Generation Force-Field for the Simulation of Proteins, Nucleic-Acids, and Organic-Molecules. *J. Am. Chem. Soc.*, 117(19):5179–5197, 1995.

[9] W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz Jr., D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell, and P. A. Kollman. A second generation force field for the simulation of proteins, nucleic acids, and organic molecules. *J. Am. Chem. Soc.*, 118(9):2309–2309, 1996.

[10] D. A. Case, T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz Jr., A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods. The AMBER biomolecular simulation programs. *J. Comput. Chem.*, 26(16):1668–1688, 2005.

[11] D. A. Case, T. A. Darden, T. E. Cheatham, III, C. L. Simmerling, J. Wang, R. E. Duke, R. Luo, K. M. Merz Jr., D. A. Pearlman, M. M. Crowley, R. C. R.C. Walker, W. W. Zhang, B. Wang, S. Hayik, A. Roitberg, G. Seabra, K. F. Wong, F. Paesani, X. Wu, S. Brozell, V. Tsui, H. Gohlke, L. Yang, C. Tan, J. Mongan, V. Hornak, G. Cui, P. Beroza, D. H. Mathews, C. Schafmeister, W. S. Ross, and P. A. Kollman. AMBER 9, 2006.

[12] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. CHARMM - a Program for Macromolecular Energy, Minimization, and Dynamics Calculations. *J. Comput. Chem.*, 4(2):187–217, 1983.

[13] T. A. Halgren. Merck molecular force field.5. Extension of MMFF94 using experimental data, additional computational data, and empirical rules. *J. Comput. Chem.*, 17(5-6):616–641, 1996.

[14] T. A. Halgren. Merck molecular force field.3. Molecular geometries and vibrational frequencies for MMFF94. *J. Comput. Chem.*, 17(5-6):553–586, 1996.

[15] T. A. Halgren. Merck molecular force field.2. MMFF94 van der waals and electrostatic parameters for intermolecular interactions. *J. Comput. Chem.*, 17(5-6):520–552, 1996.

[16] T. A. Halgren. Merck molecular force field.1. Basis, form, scope, parameterization, and performance of MMFF94. *J. Comput. Chem.*, 17(5-6):490–519, 1996.

[17] T. A. Halgren. Representation of van der Waals (vdW) Interactions in Molecular Mechanics Force-Fields - Potential Form, Combination Rules, and vdW parameters. *J. Am. Chem. Soc.*, 114(20):7827–7843, 1992.

[18] T. A. Halgren and R. B. Nachbar. Merck molecular force field.4. Conformational energies and geometries for MMFF94. *J. Comput. Chem.*, 17(5-6):587–615, 1996.

[19] T. A. Halgren. MMFF VII. Characterization of MMFF94, MMFF94s, and other widely available force fields for conformational energies and for intermolecular-interaction energies and geometries. *J. Comput. Chem.*, 20(7):730–748, 1999.

[20] T. A. Halgren. MMFF VI. MMFF94S Option for Energy Minimization Studies. *J. Comput. Chem.*, 20(7):720–729, 1999.

[21] W. L. Jorgensen and J. Tiradorives. The OPLS Potential Functions for Proteins - Energy Minimizations for Crystals of Cyclic-Peptides and Crambin. *J. Am. Chem. Soc.*, 110(6):1657–1666, 1988.

[22] N. L. Allinger, Y. H. Yuh, and J. H. Lii. Molecular Mechanics - the MM3 force-field for Hydrocarbons.1. *J. Am. Chem. Soc.*, 111(23):8551–8566, 1989.

[23] E. C. Meng and R. A. Lewis. Determination of Molecular Topology and Atomic Hybridization States from Heavy-Atom Coordinates. *J. Comput. Chem.*, 12(7):891–898, 1991.

[24] F. H. Allen, O. Kennard, D. G. Watson, L. Brammer, A. G. Orpen, and R. Taylor. Tables of Bond Lengths Determined by X-Ray and Neutron-Diffraction.1. Bond Lengths in Organic-Compounds. *J. Chem. Soc., Perkin Trans. 2*, (12):S1–S19, 1987.

[25] J. C. Baber and E. E. Hodgkin. Automatic Assignment of Chemical Connectivity to Organic-Molecules in the Cambridge Structural Database. *J. Chem. Inf. Comput. Sci.*, 32(5):401–406, 1992.

[26] A. G. Orpen, L. Brammer, F. H. Allen, O. Kennard, D. G. Watson, and R. Taylor. Tables of Bond Lengths Determined by X-Ray and Neutron-Diffraction.2. Organometallic Compounds and Co-Ordination Complexes of the D-Block and F-Block Metals. *J. Chem. Soc., Dalton Trans.*, (12):S1–S83, 1989.

[27] M. Hendlich, F. Rippmann, and G. Barnickel. BALI: Automatic assignment of bond and atom types for protein ligands in the Brookhaven Protein Databank. *J. Chem. Inf. Comput. Sci.*, 37(4):774–778, 1997.

[28] J. M. Wang, W. Wang, P. A. Kollman, and D. A. Case. Automatic atom type and bond type perception in molecular mechanical calculations. *J. Mol. Graphics Modell.*, 25(2):247–260, 2006.

[29] B. T. Fan, A. Panaye, J. P. Doucet, and A. Barbu. Ring Perception - A New Algorithm for Directly Finding the Smallest Set of Smallest Rings from a Connection Table. *J. Chem. Inf. Comput. Sci.*, 33(5):657–662, 1993.

[30] B. L. Roos-kozel and W. L. Jorgensen. Computer-Assisted Mechanistic Evaluation of Organic-Reactions.2. Perception of Rings, Aromaticity, and Tautomers. *J. Chem. Inf. Comput. Sci.*, 21(2):101–111, 1981.

[31] J. M. Wang, R. M. Wolf, J. W. Caldwell, P. A. Kollman, and D. A. Case. Development and testing of a general amber force field. *J. Comput. Chem.*, 25(9):1157–1174, 2004.

[32] M. Lipton and W. C. Still. The multiple minimum problem in molecular modeling - tree searching internal coordinate conformational space. *J. Comput. Chem.*, 9(4):343–355, 1988.

[33] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.

[34] Wilson T. Willett, P. and S. F. Reddaway. Atom-by-atom searching using massive parallelism. implementation of the ullmann subgraph isomorphism algorithm on the distributed array processor. *J. Chem. Inf. Model.*, 31(2):225–233, 1991.

[35] A. R. Leach. *Molecular modelling: principles and applications.* Prentice Hall, Harlow, England; New York, 2nd edition, 2001.

[36] E. J. Barker, D. Buttar, D. A. Cosgrove, E. J. Gardiner, P. Kitts, P. Willett, and V. J. Gillet. Scaffold hopping using clique detection applied to reduced graphs. *J. Chem. Inf. Model.*, 46(2):503–511, 2006.

[37] S. K. Kearsley. On the Orthogonal Transformation Used for Structural Comparisons. *Acta Crystallogr., Sect. A: Found. Crystallogr.*, 45:208–210, 1989.

[38] W. Kabsch. Solution for Best Rotation to Relate 2 Sets of Vectors. *Acta Crystallogr., Sect. A: Found. Crystallogr.*, 32:922–923, 1976.

[39] W. Kabsch. Discussion of Solution for Best Rotation to Relate 2 Sets of Vectors. *Acta Crystallogr., Sect. A: Found. Crystallogr.*, 34:827–828, 1978.

[40] G. Carta, V. Onnis, A. J. S. Knox, D. Fayne, and D. G. Lloyd. Permuting input for more effective sampling of 3D conformer space. *J. Comput.-Aided Mol. Des.*, 20(3):179–190, 2006.

[41] G. M. Ullmann, E. W. Knapp, and N. M. Kostic. Computational simulation and analysis of dynamic association between plastocyanin and cytochrome f. consequences for the electron-transfer reaction. *J. Am. Chem. Soc.*, 119(1):42–52, 1997.

[42] J. O. A. De Kerpel and U. Ryde. Protein strain in blue copper proteins studied by free energy perturbations. *Proteins: Struct. Funct. Genet.*, 36(2):157–174, 1999.

[43] M. H. M. Olsson and U. Ryde. The influence of axial ligands on the reduction potential of blue copper proteins. *J. Biol. Inorg. Chem.*, 4(5):654–663, 1999.

[44] R. Remenyi and P. Comba. A new general molecular mechanics force field for the oxidized form fo blue coppper proteins. *J. Inorg. Biochem.*, 86(1):397–397, 2001.

[45] P. Comba, A. Lledos, F. Maseras, and R. Remenyi. Hybrid quantum mechanics/molecular mechanics studies of the active site of the blue copper proteins amicyanin and rusticyanin. *Inorg. Chim. Acta*, 324(1-2):21–26, 2001.

[46] P. Comba and R. Remenyi. A new molecular mechanics force field for the oxidized form of blue copper proteins. *J. Comput. Chem.*, 23(7):697–705, 2002.

[47] D. Suarez, N. Diaz, and K. M. Merz Jr. Ureases: Quantum chemical calculations on cluster models. *J. Am. Chem. Soc.*, 125(50):15324–15337, 2003.

[48] G. Estiu and K. M. Merz Jr. Enzymatic catalysis of urea decomposition: Elimination or hydrolysis? *J. Am. Chem. Soc.*, 126(38):11832–11842, 2004.

[49] G. Estiu and K. M. Merz Jr. Catalyzed decomposition of urea. Molecular dynamics simulations of the binding of urea to urease. *Biochemistry*, 45(14):4429–4443, 2006.

[50] G. Estiu, D. Suarez, and K. M. Merz Jr. Quantum mechanical and molecular dynamics simulations of ureases and Zn beta-lactamases. *J. Comput. Chem.*, 27(12):1240–1262, 2006.

[51] F. C. Bernstein, T. F. Koetzle, G. J. B. Williams, E. F. Meyer, M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasumi. Protein Data Bank - Computer-Based Archival File for Macromolecular Structures. *J. Mol. Biol.*, 112(3):535–542, 1977.

[52] R. D. Hancock. Molecular Mechanics Calculations as a Tool in Coordination Chemistry. *Prog. Inorg. Chem.*, 37:187–291, 1989.

[53] S. C. Hoops, K. W. Anderson, and K. M. Merz Jr. Force-Field Design for Metalloproteins. *J. Am. Chem. Soc.*, 113(22):8262–8270, 1991.

[54] Cieplak P. Cornell W. Bayly, C. I. and P. A. Kollman. A well-behaved electrostatic potential based method using charge restraints for deriving atomic charges: the resp model. *J. Phys. Chem.*, 97(40):10269–10280, 1993.

[55] J. B. Li, T. H. Zhu, C. J. Cramer, and D. G. Truhlar. New class IV charge model for extracting accurate partial charges from wave functions. *J. Phys. Chem. A*, 102(10):1820–1831, 1998.

[56] R. H. Stote and M. Karplus. Zinc binding in proteins and solution: a simple but accurate nonbonded representation. *Proteins*, 23(1):12–31, 1995.

[57] D. V. Sakharov and C. Lim. Zn protein simulations including charge transfer and local polarization effects. *J. Am. Chem. Soc.*, 127(13):4921–4929, 2005.

[58] J. Aqvist and A. Warshel. Computer simulation of the initial proton transfer step in human carbonic anhydrase i. *J. Mol. Biol.*, 224(1):7–14, 1992.

[59] Y. P. Pang, K. Xu, J. E. Yazal, and F. G. Prendergas. Successful molecular dynamics simulation of the zinc-bound farnesyltransferase using the cationic dummy atom approach. *Protein Sci.*, 9(10):1857–65, 2000.

[60] Y. P. Pang. Successful molecular dynamics simulation of two zinc complexes bridged by a hydroxide in phosphotriesterase using the cationic dummy atom method. *Proteins*, 45(3):183–9, 2001.

[61] A. Vedani and D. W. Huhta. A New Force-Field for Modeling Metalloproteins. *J. Am. Chem. Soc.*, 112(12):4759–4767, 1990.

[62] N. Gresh, J. P. Piquemal, and M. Krauss. Representation of Zn(II) complexes in polarizable molecular mechanics. Further refinements of the electrostatic and short-range contributions. Comparisons with parallel ab initio computations. *J. Comput. Chem.*, 26(11):1113–30, 2005.

[63] N. Gresh. Development, validation, and applications of anisotropic polarizable molecular mechanics to study ligand and drug-receptor interactions. *Curr. Pharm. Des.*, 12(17):2121–58, 2006.

[64] A. K. Rappe, C. J. Casewit, K. S. Colwell, W. A. Goddard, and W. M. Skiff. UFF, a Full Periodic-Table Force-Field for Molecular Mechanics and Molecular-Dynamics Simulations. *J. Am. Chem. Soc.*, 114(25):10024–10035, 1992.

[65] A. K. Rappe, K. S. Colwell, and C. J. Casewit. Application of a Universal Force-Field to Metal-Complexes. *Inorg. Chem.*, 32(16):3438–3450, 1993.

[66] J. M. Sirovatka, A. K. Rappe, and R. G. Finke. Molecular mechanics studies of coenzyme B-12 complexes with constrained Co-N(axial-base) bond lengths: introduction of the universal force field (UFF) to coenzyme B-12 chemistry and its use to probe the plausibility of an axial-base-induced, ground-state corrin butterfly conformational steric effect. *Inorg. Chim. Acta*, 300:545–555, 2000.

[67] P. Brandt, T. Norrby, E. Akermark, and P. O. Norrby. Molecular mechanics (MM3*) parameters for ruthenium(ii)-polypyridyl complexes. *Inorg. Chem.*, 37(16):4120–4127, 1998.

[68] H. M. Marques and K. L. Brown. A Molecular Mechanics Force-Field for the Cobalt Corrinoids. *J. Mol. Struct. (Theochem)*, 340:97–124, 1995.

[69] K. L. Brown, X. Zou, and H. M. Marques. NMR-restrained molecular modeling of cobalt corrinoids: cyanocobalamin (vitamin B-12) and methylcobalt corrinoids. *J. Mol. Struct. (Theochem)*, 453:209–224, 1998.

[70] H. M. Marques and K. L. Brown. The structure of cobalt corrinoids based on molecular mechanics and NOE-restrained molecular mechanics and dynamics simulations. *Coord. Chem. Rev.*, 192:127–153, 1999.

[71] H. M. Marques, B. Ngoma, T. J. Egan, and K. L. Brown. Parameters for the AMBER force field for the molecular mechanics modeling of the cobalt corrinoids. *J. Mol. Struct.*, 561(1-3):71–91, 2001.

[72] J. Aqvist and A. Warshel. Free-Energy Relationships in Metalloenzyme-Catalyzed Reactions - Calculations of the Effects of Metal-Ion Substitutions in Staphylococcal Nuclease. *J. Am. Chem. Soc.*, 112(8):2860–2868, 1990.

[73] U. Ryde. Molecular-Dynamics Simulations of Alcohol-Dehydrogenase with a 4-Coordinate or 5-Coordinate Catalytic Zinc Ion. *Proteins: Struct. Funct. Genet.*, 21(1):40–56, 1995.

[74] U. Ryde. On the Role of Glu-68 in Alcohol-Dehydrogenase. *Protein Sci.*, 4(6):1124–1132, 1995.

[75] U. Ryde. Carboxylate binding modes in zinc proteins: A theoretical study. *Biophys. J.*, 77(5):2777–2787, 1999.

[76] R. D. Hancock, J. S. Weaving, and H. M. Marques. A Molecular Mechanics Model of the Metalloporphyrins - the Role of Steric Hindrance in Discrimination in Favor of Dioxygen Relative to Carbon-Monoxide in Some Heme Models. *J. Chem. Soc., Chem. Commun.*, (16):1176–1178, 1989.

[77] H. M. Marques and I. Cukrowski. Molecular mechanics modelling of porphyrins. using artificial neural networks to develop metal parameters for four-coordinate metalloporphyrins. *Phys. Chem. Chem. Phys.*, 4(23):5878–5887, 2002.

[78] H. M. Marques and K. L. Brown. Molecular mechanics and molecular dynamics simulations of porphyrins, metalloporphyrins, heme proteins and cobalt corrinoids. *Coord. Chem. Rev.*, 225(1-2):123–158, 2002.

[79] C. E. Skopec, J. M. Robinson, I. Cukrowski, and H. M. Marques. Using artificial neural networks to develop molecular mechanics parameters for the modelling of metalloporphyrins.

III. five coordinate Zn(II) porphyrins and the metalloprophyrins of the early 3d metals. *J. Mol. Struct.*, 738(1-3):67–78, 2005.

[80] C. E. Skopec, I. Cukrowski, and H. M. Marques. Using artificial neural networks to develop molecular mechanics parameters for the modelling of metalloporphyrins: Part IV. Five-, six-coordinate metalloporphyrins of Mn, Co, Ni and Cu. *J. Mol. Struct.*, 783(1-3):21–33, 2006.

[81] P. O. Norrby and T. Liljefors. Automated molecular mechanics parameterization with simultaneous utilization of experimental and quantum mechanical data. *J. Comput. Chem.*, 19(10):1146–1166, 1998.

[82] P. O. Norrby and P. Brandt. Deriving force field parameters for coordination complexes. *Coord. Chem. Rev.*, 212:79–109, 2001.

[83] K. M. Merz Jr. CO2 Binding to Human Carbonic Anhydrase-II. *J. Am. Chem. Soc.*, 113(2):406–411, 1991.

[84] K. M. Merz Jr., M. A. Murcko, and P. A. Kollman. Inhibition of Carbonic-Anhydrase. *J. Am. Chem. Soc.*, 113(12):4484–4490, 1991.

[85] N. Diaz, D. Suarez, and K. M. Merz Jr. Hydration of zinc ions: theoretical study of [Zn(H2O)(4)](H2O)(8)(2+) and [Zn(H2O)(6)](H2O)(6)(2+). *Chem. Phys. Lett.*, 326(3-4):288–292, 2000.

[86] N. Diaz, D. Suarez, and K. M. M. Merz Jr. Zinc metallo-beta-lactamase from Bacteroides fragilis: A quantum chemical study on model systems of the active site. *J. Am. Chem. Soc.*, 122(17):4197–4208, 2000.

[87] N. Diaz, D. Suarez, and K. M. Merz Jr. Molecular dynamics simulations of the mononuclear zinc-beta-lactamase from bacillus cereus complexed with benzylpenicillin and a quantum chemical study of the reaction mechanism. *J. Am. Chem. Soc.*, 123(40):9867–9879, 2001.

[88] N. Diaz, D. Suarez, T. L. Sordo, and K. M. Merz Jr. A theoretical study of the aminolysis reaction of lysine 199 of human serum albumin with benzylpenicillin: Consequences for immunochemistry of penicillins. *J. Am. Chem. Soc.*, 123(31):7574–7583, 2001.

[89] N. Diaz, D. Suarez, T. L. Sordo, and K. M. Merz Jr. Acylation of class a beta-lactamases by penicillins: A theoretical examination of the role of serine 130 and the beta-lactam carboxylate group. *J. Phys. Chem. B*, 105(45):11302–11313, 2001.

[90] D. Suarez and K. M. Merz Jr. Molecular dynamics simulations of the mononuclear zinc-beta-lactamase from Bacillus cereus. *J. Am. Chem. Soc.*, 123(16):3759–3770, 2001.

[91] N. Diaz, T. L. Sordo, K. M. Merz Jr., and D. Suarez. Insights into the acylation mechanism of class A beta-lactamases from molecular dynamics simulations of the TEM-1 enzyme complexed with benzylpenicillin. *J. Am. Chem. Soc.*, 125(3):672–684, 2003.

[92] N. Diaz, D. Suarez, K. M. Merz Jr., and T. L. Sordo. Molecular dynamics simulations of the TEM-1,beta-lactamase complexed with cephalothin. *J. Med. Chem.*, 48(3):780–791, 2005.

[93] Martin B. Peters, Yue Yang, Bing Wang, László Füsti-Molnár, Michael N. Weaver, and Kenneth M. Merz Jr. Structural Survey of Zinc Containing Proteins and the Development of the Zinc AMBER Force Field (ZAFF). *J Chem Theory Comput*, 6(9):2935–2947, Sep 2010.

[94] D. Suarez, E. N. Brothers, and K. M. Merz Jr. Insights into the structure and dynamics of the dinuclear zinc beta-lactamase site from Bacteroides fragilis. *Biochemistry*, 41(21):6615–6630, 2002.

[95] D. Suarez, N. Diaz, and K. M. Merz Jr. Molecular dynamics simulations of the dinuclear zinc-beta-lactamase from bacteroides fragilis complexed with imipenem. *J. Comput. Chem.*, 23(16):1587–1600, 2002.

[96] G. Cui, B. Wang, and K. M. Merz Jr. Computational studies of the farnesyltransferase ternary complex - Part I: Substrate binding. *Biochemistry*, 44(50):16513–16523, 2005.

[97] Daniel J Sindhikara, Adrian E Roitberg, and Kenneth M Merz. Apo and nickel-bound forms of the pyrococcus horikoshii species of the metalloregulatory protein: Nikr characterized by molecular dynamics simulations. *Biochemistry*, 48(50):12024–33, Dec 2009.

[98] J. R. Collins, D. L. Camper, and G. H. Loew. Valproic Acid Metabolism by Cytochrome-P450 - a Theoretical-Study of Stereoelectronic Modulators of Product Distribution. *J. Am. Chem. Soc.*, 113(7):2736–2743, 1991.

[99] J. R. Collins, P. Du, and G. H. Loew. Molecular-Dynamics Simulations of the Resting and Hydrogen Peroxide-Bound States of Cytochrome-C Peroxidase. *Biochemistry*, 31(45):11166–11174, 1992.

[100] S. J. Yao, J. P. Plastaras, and L. G. Marzilli. A Molecular Mechanics Amber-Type Force-Field for Modeling Platinum Complexes of Guanine Derivatives. *Inorg. Chem.*, 33(26):6061–6077, 1994.

[101] M. M. Harding. The geometry of metal-ligand interactions relevant to proteins. *Acta Crystallogr., Sect. D: Biol. Crystallogr.*, 55:1432–43, 1999.

[102] M. M. Harding. The geometry of metal-ligand interactions relevant to proteins. II. angles at the metal atom, additional weak metal-donor interactions. *Acta Crystallogr., Sect. D: Biol. Crystallogr.*, 56:857–67, 2000.

[103] M. M. Harding. Geometry of metal-ligand interactions in proteins. *Acta Crystallogr., Sect. D: Biol. Crystallogr.*, 57:401–11, 2001.

[104] M. M. Harding. Metal-ligand geometry relevant to proteins and in proteins: sodium and potassium. *Acta Crystallogr., Sect. D: Biol. Crystallogr.*, 58:872–4, 2002.

[105] M. M. Harding. The architecture of metal coordination groups in proteins. *Acta Crystallogr., Sect. D: Biol. Crystallogr.*, 60:849–59, 2004.

[106] M. M. Harding. Small revisions to predicted distances around metal sites in proteins. *Acta Crystallogr., Sect. D: Biol. Crystallogr.*, 62:678–82, 2006.

[107] J. Aqvist. Ion Water Interaction Potentials Derived from Free-Energy Perturbation Simulations. *J. Phys. Chem.*, 94(21):8021–8024, 1990.

[108] A. Bondi. van Der Waals Volumes + Radii. *J. Phys. Chem.*, 68(3):441–451, 1964.

[109] S. S. Batsanov. van der Waals radii of elements. *Inorg. Mater.*, 37(9):871–885, 2001.

[110] S. S. Batsanov. The determination of van der Waals radii from the structural characteristics of metals. *Russ. J. Phys. Chem.*, 74(7):1144–1147, 2000.

[111] D. Asthagiri, L. R. Pratt, M. E. Paulaitis, and S. B. Rempe. Hydration structure and free energy of biomolecularly specific aqueous dications, including Zn2+ and first transition row metals. *J. Am. Chem. Soc.*, 126(4):1285–1289, 2004.

[112] C. S. Babu and C. Lim. Empirical force fields for biologically active divalent metal cations in water. *J. Phys. Chem. A*, 110(2):691–699, 2006.

[113] C. S. Babu and C. Lim. A new interpretation of the effective born radius from simulation and experiment. *Chem. Phys. Lett.*, 310(1-2):225–228, 1999.

[114] C. S. Babu and C. Lim. Theory of ionic hydration: Insights from molecular dynamics simulations and experiment. *J. Phys. Chem. B*, 103(37):7958–7968, 1999.

[115] A. C. Vaiana, A. Schulz, J. Wolfrum, M. Sauer, and J. C. Smith. Molecular mechanics force field parameterization of the fluorescent probe rhodamine 6G using automated frequency matching. *J. Comput. Chem.*, 24(5):632–639, 2003.

[116] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, Jr J. A. Montgomery, T. Vreven, K. N. Kudin, J. C. Burant, J. M. Millam, S. S. Iyengar, J. Tomasi, V. Barone, B. Mennucci, M. Cossi, G. Scalmani, N. Rega, G. A. Petersson, H. Nakatsuji, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, M. Klene, X. Li, J. E. Knox, H. P. Hratchian, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, P. Y. Ayala, K. Morokuma, G. A. Voth, P. Salvador, J. J. Dannenberg, V. G. Zakrzewski, S. Dapprich, A. D. Daniels, M. C. Strain, O. Farkas, D. K. Malick, A. D. Rabuck, K. Raghavachari, J. B. Foresman, J. V. Ortiz, Q. Cui, A. G. Baboul, S. Clifford, J. Cioslowski, B. B. Stefanov, G. Liu, A. Liashenko, P. Piskorz, I. Komaromi, R. L. Martin, D. J. Fox, T. Keith, M. A. Al Laham, C. Y. Peng, A. Nanayakkara, M. Challacombe, P. M. W. Gill, B. Johnson, W. Chen, M. W. Wong, C. Gonzalez, and J. A. Pople. Gaussian 03, revision c.02. Gaussian, Inc., Wallingford, CT, 2004.

[117] J. M. Seminario. Calculation of intramolecular force fields from second-derivative tensors. *Int. J. Quantum Chem*, 60(7):59–65, 1996.

[118] U. C. Singh and P. A. Kollman. An approach to computing electrostatic charges for molecules. *J. Comput. Chem.*, 5(2):129–145, 1984.

[119] B. H. Besler, K. M. Merz Jr., and P. A. Kollman. Atomic Charges Derived from Semiempirical Methods. *J. Comput. Chem.*, 11(4):431–439, 1990.

[120] C. I. Bayly, P. Cieplak, W. D. Cornell, and P. A. Kollman. A Well-Behaved Electrostatic Potential Based Method Using Charge Restraints for Deriving Atomic Charges - the RESP Model. *J. Phys. Chem.*, 97(40):10269–10280, 1993.

[121] P. Cieplak, W. D. Cornell, C. Bayly, and P. A. Kollman. Application of the Multimolecule and Multiconformational RESP Methodology to Biopolymers - Charge Derivation for DNA, RNA, and Proteins. *J. Comput. Chem.*, 16(11):1357–1377, 1995.