Pacific Symposium on Biocomputing 5:251-262 (2000)

THE ENSEMBLE/LEGACY CHIMERA EXTENSION: STANDARDIZED USER AND PROGRAMMER INTERFACE TO MOLECULAR ENSEMBLE DATA AND LEGACY MODELING PROGRAMS

DAVID E. KONERDING
Graduate Group in Biophysics, Box 0446 UCSF
San Francisco, CA 94143-0446

Ensemble/Legacy is a toolkit extension of the Object Technology Framework (OTF) [1] that exposes an object oriented interface for accessing and manipulating *ensembles* (collections of molecular conformations that share a common chemical topology) and driving Legacy programs (such as MSMS[2], AMBER[3], X-PLOR[4], CORMA/MARDIGRAS[5], Dials and Windows[6], and CURVES[7]). Ensemble/Legacy provides a natural programming interface for running Legacy programs on ensembles of molecules and accessing the resulting data. Using the OTF reduces the time cost of developing a new library to store and manipulate molecular data and also allows Ensemble/Legacy to integrate into the Chimera[8] visualization program. The extension to Chimera exposes the Legacy functionality using a graphical user interface that greatly simplifies the process of modeling and analyzing conformational ensembles. Furthermore, all the C++ functionality of the Ensemble/Legacy toolkit is "wrapped" for use in the Python[9] programming language. More detailed documentation on using Ensemble/Legacy is available online (http://picasso.nmr.ucsf.edu/~dek/ensemble.html).

1 Introduction

1.1 Flexibility and Dynamics of Biomolecules

Biophysical analysis of molecules in solution reveals that proteins and nucleic acids are flexible and dynamic. This flexibility plays a critical role in many biological situations, such as cellular regulation, genomic replication, and environmental interaction. In some cases, molecules are rigid and the only flexibility is small positional fluctuation around average atomic positions, while in other cases there may be large conformational rearrangements from one conformational subfamily to another. Being able to extract and visualize relevant dynamic information from experimental data and theoretical predictions is a major challenge. Management and reduction of data to produce useful information is often hampered by the sheer volume of the data to be analyzed, and by the difficulty of converting the information into a form required by analysis programs.

1.2 The Ensemble/Legacy Toolkit

To address the issue of modeling and visualizing molecular ensemble data we have developed the Ensemble/Legacy toolkit. Ensemble/Legacy provides the biomolecular scientist with both a GUI (graphical user interface) and a programming library to analyze and visualize ensembles of structures. The Ensemble/Legacy library is built on top of Chimera and the OTF (see figure 1), eliminating the need to develop a new molecular visualization tool designed specifically for ensembles. This design decision allows programmers using the Ensemble/Legacy extension to focus on implementing code that performs the required functionality (such as an interface to a pre-existing Legacy program) and allows users to learn only one molecular visualization interface.

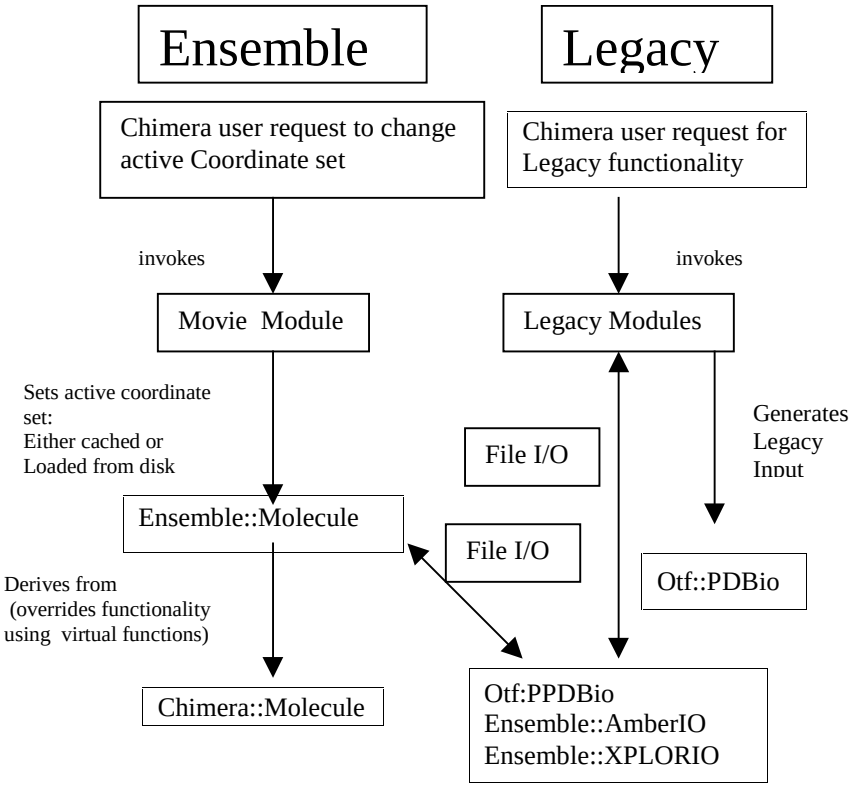

Figure 1. Structure of the Ensemble/Legacy toolkit and its interaction with Chimera. The Ensemble interface replaces Chimera's normal Molecule class with a derived class "Ensemble::Molecule".

1.3 OTF: a Molecular Applications Framework

Because the Ensemble/Legacy Toolkit depends heavily on the OTF, we will describe the design philosophy and interface structure of the OTF first. The Object Technology Framework is freely available software (http://www.cgl.ucsf.edu/otf) being developed by the Computer Graphics Lab at UCSF to automate the process of generating C++ classes to facilitate rapid biomolecular application development. The OTF stores molecular data using C++ classes known as otf::Molecule, otf::Residue, and otf::Atom. The otf::Molecule, otf::Residue and otf::Atom classes represent a molecule by storing its atom type information (carbon, oxygen, nitrogen, etc), topological structure (bonds), and atomic coordinates. Multiple structures are accommodated by storing multiple coordinate sets for a given molecule. Logically, an instance of otf::Molecule contains otf::Residue instances (one for each residue in the molecule), each of which contains otf::Atom instances (one for each atom in the residue). This demonstrates the straightfoward link between reality, where "molecules are composed of residues which are composed of atoms", and the class structure of the OTF.

2 Design Structure of the Ensemble/Legacy Toolkit

2.1 Ensemble::Molecule class

The Ensemble::Molecule C++ class is the primary data object used by Ensemble/Legacy to store molecular ensembles. This class inherits from the otf::Molecule class, and mostly duplicates its behavior. The primary difference in the Ensemble::Molecule class is that it overrides the normal otf::Molecule behavior when coordinate data is read. Otf::Molecule reads all the coordinate sets from a molecular structure at once; however, Ensemble::Molecule reads one frame of coordinates in at a time (on demand) and caches the coordinate sets for later use. This design decision was made to reduce the start-up time cost and memory footprint when reading large ensembles of data but still allow high performance for both random and sequential access of ensemble structures. Typical users of the otf::Molecule class will be loading a structure with a single coordinate set; consuming a modest amount of memory (10000 atoms * 3 coordinates/atom * 8 bytes/float <= 1MB) while users of the Ensemble::Molecule class may be reading up to 1000 or more coordinates sets (230MB). Since a typical workstation has only 128MB of RAM, it is clear that there is a significant advantage to loading only frames that are going to be used.

2.2 Ensemble::Legacy class

Ensemble/Legacy also has an Ensemble::Legacy class that forms an abstract representation of running a legacy program as a subprocess. Legacy programs are well-established as useful tools, but are not easilly automated or have other deficiencies which limit their usability in an automated modelling process. Legacy programs often require complicated input files which are challenging to build correctly by hand, and they frequently require customized versions of PDB[10] files (or other molecular data formats) which do not conform to the file format standard. The Ensemble/Legacy toolkit knows what each different legacy program expects in terms of input files, and automatically generates the appropriately formatted input files. Support for individual Legacy programs such as MSMS (a program to calculate molecular surfaces), AMBER (a suite of molecular modelling tools), X-PLOR (a package for crystallographic and NMR structure refinement), CORMA (a program to compute the R factor of an NMR structure), Dials and Windows and CURVES (programs which determine the helical structure of a DNA or RNA duplex) exists in subclasses of the Ensemble::Legacy class which provide customized code specific to the Legacy application.

2.3 The Chimera Movie GUI

The Movie GUI is a plug-in tool for the Chimera visualization program which allows the user to visualize an ensemble as a movie. Each frame of the movie maps one-to-one with a coordinate set in the ensemble. Frames can be single-stepped in forward or reverse direction or run as a movie where each frame is displayed rapidly in sequence. The movie feature is useful for qualitatively assessing a molecular dynamics or Monte Carlo simulation, while the single-step feature allows detailed scrutiny of individual structures in an ensemble. Furthermore, the Movie GUI supports all of the legacy interfaces, exposing their features through an intuitive graphical interface. At every step of the movie, the user can interactively run a legacy program on the current frame and immediately see the results.

The Movie GUI has evolved through several incarnations. The original Movie GUI was written as a delegate for the Midas display system[11]. The delegate mechanism allowed Midas to be extended with new features not anticipated by the original authors without the need for modification of the Midas source code. However, the delegate system was awkward, and allowed for only limited extension of Midas. The second Movie GUI was written as a standalone program with its own molecular visualization program. While this allowed for detailed control by the Movie GUI, developing a molecular visualization program is a major undertaking in itself; this version of Movie was primarily used as a feature testbed. For the third incarnation of Movie, we have chosen to use Chimera. Chimera is a fully extensible molecular visualization program and exposes much of its functionality to C++ and

Python programs. This allows the Movie GUI extension to have detailed control over Chimera without the need to modify the Chimera source code, and allows efficient communication of molecular structure data between the Movie GUI and Chimera. This efficiency and control are absolutely necessary for Movie to be a useful and effective program.

3 The Ensemble/Legacy Toolkit Python/C++ Interface

3.1 "Wrapping" C++ classes for Python

As mentioned earlier, the Ensemble/Legacy toolkit is written primarily in C++. However, all of the functionality is "wrapped" to be available to the Python programming language as modules. Python is a high-level object oriented interpreted programming language that is easy to learn and use. In fact, Python code is so readable it is often called "executable pseudocode". Python supports heterogeneous lists, hash arrays (called "dictionaries"), and other high-level data structures which are absent from the core C and C++ languages. Python provides a number of advantages over other high-level object-oriented interpreted programming languages. First and foremost, Python is is easy to learn and use, unlike C, or C++. Second, it is easy to understand Python programs written by others, unlike perl, because the Python syntax enforces readability. By using a language which is easy to learn and use, we make it more likely that Ensemble/Legacy will be adopted by users in the scientific community. Also, Python is open source, cross-platform (Unix, Windows, and Macintosh), and is being adopted to solve problems in many different application spaces. There is at least one other[12] published example using Python in molecular modelling and visualization.

One of the most powerful features of the OTF (and the Ensemble/Legacy library as a result) is the ability to automatically "wrap" C++ code as a Python extension. This allows programs written in Python to access OTF data and code directly from Python. The extension method of Python works as follows: a programmer writes code in C++ which declares the structure of the C++ data and code in a manner than Python can understand. This C++ code is compiled to a shared object module (also known as a "dynamic link library" in Windows terminology). A running Python script can "import" a C++ shared object module in the same way it can "import" a regular Python module. Methods within the C++ module which are called by the Python module execute as natively compiled code. When the native code finishes, control returns to the Python intepreter at the point immediately following the method invocation. From the perspective of the Python program, the method call

was simply a call to another Python module. For this to work correctly, the C++ module code must be carefully written to use Python data structures to communicate with the calling Python code.

Although the extension mechanism is powerful, it requires a fair amount of work to "wrap" C++ library code which has already been written. For each public method and data item in the C++ code, a "wrapper" function which handles the Python to C++ (and back) translation must be written. If the C++ code is undergoing development, changes to the C++ interface must be reflected in the extension code. To simplify the process of "wrapping", the OTF provides a powerful tool called "wrappy". Wrappy takes as input a C++ header file which defines the interface to a C++ class and outputs source code for the Python extension module. The extension has all the necessary support for accessing public methods and data of the C++ class. This automation greatly reduces the time necessary to build the Python/C++ interface and allows the programmer to focus on developing robust code. Ensemble/Legacy uses wrappy to wrap the AMBER molecular structure I/O code, which is written in C++ for maximum speed.

3.2 Example: the AMBER Legacy Interface

The most useful and powerful legacy interface is to the AMBER suite of programs. The AMBER legacy interface can convert a protein or nucleic acid stored in the Ensemble::Molecule format so that it can be used by AMBER for molecular mechanics/dynamics, or free energy perturbation simulation. Every AMBER option is exposed as a parameter in the AMBER legacy interface, thus allowing direct control over the course of the simulation. The Ensemble/Legacy toolkit legacy interfaces can automate an entire simulation methodology, from initial model construction to the final analysis stages. Furthermore, since the simulation methodology is stored as a Python program, it is very easy to change simulation parameters and analyze how the changes affect the simulation. The following example demonstrates how a programmer can use the AMBER legacy interface to run minimization and molecular dynamics on a structure from a previous trajectory.

```
##Construct an "Ensemble.Molecule" object from the
##our starting structure
m = Ensemble.Molecule(startingStructure)
##Create an "AMBER Legacy Object"from the Molecule
##object
a = Ensemble.Legacy.AMBER(m)
## Now minimize the molecule for 500 steps
## using the AMBER suite
minimizedStructure = a(type=MINIMIZE, steps=500)
```

```
## Now run molecular dynamics on the minimized structure
## for 1000 steps
m = Ensemble.Molecule(minimizedStructure)
a = Ensemble.Legacy.AMBER(m)
dynamicsStructures = a(type=DYNAMICS, steps=1000)
```

Note that in the case of a minimization, only a single structure is returned (the minimized structure) while in the case of dynamics, an ensemble of structures is returned (each structure is a step in the dynamics simulation). Since these structures are stored in the Ensemble::Molecule format, they can be submitted to Dials and Windows or CORMA for analysis as shown in the examples below.

3.3 Example: the Dials and Windows Legacy interface.

The "Dials and Windows" program requires a collection of single-structure PDB files representing the ensemble of molecular structures for which the helical parameters are to be computed. The Ensemble::Legacy::Dials class has functionality for writing ensembles (using any supported file format) to disk as PDB files using the otf::PDBio class. It is necessary for the Ensemble::Legacy::Dials class to ensure that all the residue names in the molecules written to the PDB file format conform to the file format expected by Dials. Dials requires that the nucleic acid residue names follow a particular convention (ADE, GUA, THY, CYT, URA). This can conflict with AMBER, which uses more descriptive nucleic acid residue names such as DG5 to represent a 5' terminal deoxyguanosine. Ensemble::Legacy::Dials then generates a Dials input file and runs Dials as a child process. Dials computes the helical parameters for the ensemble of structures. When control returns to the Ensemble::Legacy::Dials module, it determines whether Dials executed successfully, and if so, parses the output file to read in the helical parameters. These data are stored in a collection of hash arrays, indexed by the type of helical parameter.

While the graphical user interface has been written to support common requests (such as running Dials & Windows on an ensemble of nucleic acid structures then plotting the results) it is intended that advanced users will use the programming interface to perform tasks which are not specifically supported by the GUI. To demonstrate the straightforward mapping between the GUI and the progrmaming interface, we will demonstrate using an example which runs Dials & Windows on an ensemble and then plots the results. Each program statement corresponds to a GUI window in the figures.

```
d = Ensemble.Legacy.Dials(dynamicStructures)
##Run Dials and Windows on the ensemble
```

```
data = d()
## See figure 2; only plot backbone data
paramType = BACKBONE
## See figure 3; Only plot the pucker and amplitude
parameters = (PUCKER,AMPLITUDE)
## See figure 4; only plot bases 1 and 2
bases = (1,2)
## See figure 5; now plot the data
Ensemble.Analysis.Dials.PlotData(data, bases,
parameters)
```

Figure 2. The Dials and Windows Parameter Type Selection Dialog allows the user to select which type of Dials and Windows parameter to plot. Dials and Windows computes three types of parameters: “base data”, “axis data”, and “backbone data”. In this figure, the user has selected “backbone data”, which presents further choices.

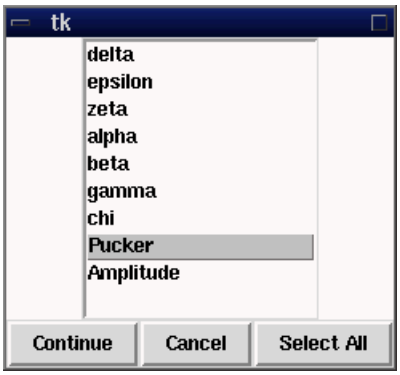

Figure 3. The Dials and Windows Backbone Parameter Selection Dialog allows the user to select which type of backbone data to plot. In this figure, the user has selected “Pucker” to plot the time course of the Pucker variable.

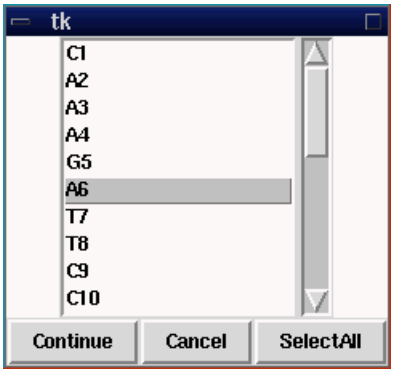

Figure 4. The Dials and Windows Base Selection Dialog allows the user to select which bases in the molecule to plot. This list is built at run-time depending on the constituents of the molecule. In this figure the user has selected base "A6" to plot.

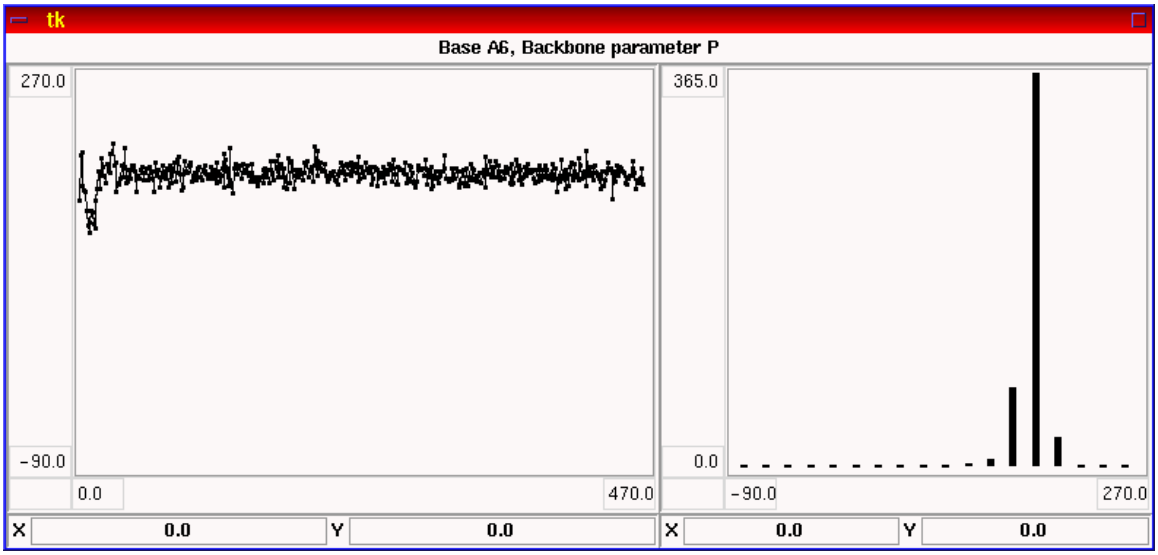

Figure 5. The Dials and Windows Data Plot Window plots the data requested by the user. Both the time course and a histogram are plotted. If the Movie GUI is running the user may "jump" to a particular frame by clicking on the corresponding data point in the time course plot.

3.4 Example: the CORMA Legacy Interface

The CORMA program computes the R factor of a model structure given experimental NMR data. CORMA requires PDB files representing the molecular

structure for which the NMR R factor is to be computed. CORMA requires the PDB files to contain experimental correlation times stored in the "temperature factor" field of the PDB file. The Ensemble::Legacy::Corma class has functionality for taking structures stored in the Ensemble::Molecule class and writing them to disk as PDB files using the otf::PDBio class with user-supplied correlation times. Since correlation times can either be constant for a whole molecule or may vary on a per-atom basis, Ensemble::Legacy::Corma allows the correlation time to be supplied as either a scalar or a vector. Ensemble::Legacy::Corma then generates a CORMA command input file and runs CORMA as a child process. CORMA computes the R factor of the PDB files written by Ensemble:Legacy::Corma. When control returns to Ensemble::Legacy::Corma, it determines if CORMA executed successfully and if so, parses the CORMA output file to determine the R factor value(s). Although the Corma legacy program is designed to produce an R factor for a single PDB file, the Ensemble::Legacy::Corma interface is designed to automate the process of calculating R factors for each structure in an ensemble of structures. In the case of a single structure, a scalar value is returned, while in the case of an ensemble of structures, a vector of R factors is returned. The CORMA application can also take a collection of PDB files and compute an "ensemble" R factor; Ensemble::Legacy::Corma supports this as well. This demonstrates how Ensemble/Legacy is able to handle "ensemble" data in a natural way: as scalar data when a single structure is considered, vector data when an ensemble of structures is considered, and possibly scalar data produced from a reduction of vector data (such as the mean R factor from an ensemble of structures). Here we use the CORMA legacy interface to compute the R factor for each of the structures generated by molecular dynamics above. Then, the Python module "Statistics" is used to compute the mean and standard deviation for the R factors computed by CORMA.

```
c = Ensemble.Legacy.Corma(dynamicStructures)
##Run CORMA on the ensemble
data = c()
##Average the R factors returned from CORMA
averageR = Statistics.average(data.r)
standardDeviation = Statistics.standardDeviation(data.r)
```

4 Conclusion

Modelling biomolecular structures is a fruitful but challenging endeavor. As experimental techniques make rapid advances in the ability to probe molecular flexibility and dynamics, greater demands are being placed on analysis programs to reduce this data to easily presented and understood information. Ensemble/Legacy was designed to address the issues associated with molecular ensembles; in

particular, to handle various forms of ensembles in a consistent manner, and to provide services for analyzing ensemble data by providing an object oriented interface to legacy programs. We have chosen a design strategy which allows for the greatest flexibility for the user: while all the necessary tools are made available in the form of an extensible programming library, there is also a graphical user interface which provides this functionality in a user-friendly format. Furthermore, by building on top of the extensible architecture built into the OTF and Chimera, we are able to take advantage of work done by others rather than having to "reinvent the wheel again and again". This library is already being used by several users in the Computer Graphics Lab to analyze real experimental data and produce publication-quality plots, demonstrating the utility of the library in real-life situations.

Acknowledgements

David Konerding gratefully acknowledges financial support from his advisor (NIH grant GM39247, Thomas James, PI). Additionally, he would like to thank the UCSF Computer Graphics Laboratory (NIH NCRR P41-RR01081, Thomas Ferrin, PI), for access to computational resources and technical information which made the development of Ensemble/Legacy possible. Furthermore, he would like to show his gratitude to Conrad Huang, Eric Petterson, Greg Couch and Thomas Ferrin of the Computer Graphics Lab for advice and assistance during the development of this program.

References

1. C. C. Huang, G. S. Couch, E. F. Pettersen, T. E. Ferrin, A. E. Howard and T. E. Klein, Pac Symp Biocomput.), 349-61.(1998)
2. M. F. Sanner, Spehner, J.-C., and Olson, A.J., Biopolymers. **38**(3), 305-320.(1996)
3. D. A. Pearlman, D. A. Case, J. C. Caldwell, W. S. Ross, T. E. Cheatham III, D. N. Ferguson, G. L. Seibel, U. C. Singh, P. K. Weiner and P. A. Kollman, *AMBER version 4.1*, . 1995, University of San Francisco: San Francisco.
4. A. T. Breunger, P. D. Adams and L. M. Rice, Curr Opin Struct Biol. **8**(5), 606-11.(1998)
5. H. Liu, H. P. Spielmann, N. B. Ulyanov, D. E. Wemmer and T. L. James, J Biomol NMR. **6**(4), 390-402.(1995)
6. G. Ravishanker, S. Swaminathan, D. L. Beveridge, R. Lavery and H. Sklenar, Journal of Biomolecular Structure and Dynamics. **6**(4.(1989)
7. R. Lavery and H. Sklenar, Journal of Biomolecular Structure and Dynamics. **6**(4), 655-667.(1989)
8. C. Huang, G. Couch, E. Petterson and T. E. Ferrin, unpublished.
9. G. Rossum, *http://www.python.org*, . 1991.

10. F. C. Bernstein, T. F. Koetzle, G. J. Williams, E. E. J. Meyer, M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi and M. Tasumi, J. Mol. Biol. **112**), 535.(1977)
11. T. E. Ferrin, G. S. Couch, C. C. Huang, E. F. Pettersen and R. Langridge, J. Mol. Graphics. **9**(1), 27-32.(1991)
12. M. F. Sanner, B. S. Duncan, C. J. Carillo and A. J. Olson, PSB. **1999**), 400-412.(1999)