BIOINFORMATICS APPLICATIONS NOTE Vol. 21 no. 20 2005, pages 3931–3932
doi:10.1093/bioinformatics/bti630

Structural bioinformatics

MESHI: a new library of Java classes for molecular modeling

Nir Kalisman[1], Ami Levi[1], Tetyana Maximova[1], Dan Reshef[2], Sharon Zafriri-Lynn[1], Yan Gleyzer[1] and Chen Keasar[1,2,*]

[1]Department of Computer Science and [2]Department of Life Science, Ben-Gurion University, Beer-Sheva 84105, Israel

Received on May 8, 2005; revised on July 25, 2005; accepted on August 12, 2005
Advance Access publication August 16, 2005

ABSTRACT

Summary: Adapting a modular and object-oriented approach in the design of molecular modeling packages may reduce the software development barrier between ideas and their programed applications. Towards this goal we developed MESHI, a new, strictly object-oriented, molecular modeling suite written in Java. MESHI provides a comprehensive library of extendable classes for all the essential components of molecular modeling: molecular and geometry elements, energy functions and optimization methods.

Availability: MESHI and its related documentation are freely available at http://www.cs.bgu.ac.il/~meshi; the MESHI API is available at http://www.cs.bgu.ac.il/~meshi/API

Contact: keasar@cs.bgu.ac.il

Supplementary information: The Supplementary information includes (1) a detailed description of several key packages and classes, and (2) a brief presentation of results achieved by using the MESHI application—Beautify—in the CASP6 experiment.

INTRODUCTION

We present here, MESHI, a new molecular modeling package, which is built around one concept—developers first. Somewhat selfishly, we believe that the most precious resource in molecular modeling is the developer's time, as Moore's law does not apply to it. In practice, we interpret this postulate as 'above all—maximize code-reuse and minimize debugging'. We believe that these goals may be most easily achieved with object-oriented design.

The current set of classes in MESHI is very much biased towards protein structure prediction, reflecting our major research interest. We do try, however, to write the classes in a general way that would make it easy to utilize them in other aspects of computational structural biology.

ARCHITECTURE

A major decision that one needs to make when designing a new software package is that of the language. We have chosen Java, which enforces object-oriented design more vigorously than the major alternative—C++, has a built-in garbage-collection utility and is platform independent. Indeed, this is not an obvious decision. Molecular modeling is a computationally intensive field and Java is known to be slower than Fortran, C or C++. Recent benchmarks, however, suggest that the performance sacrifice is less severe than what might have been expected (Perchelt, 1999; Bull *et al*., 2001; Vivanco and Pizzi, 2005)

The object-oriented design implies that every aspect of molecular modeling is represented by either a class or an interface. Thus, MESHI includes classes not only for molecular elements, such as atoms, residues and proteins, but also for geometrical concepts, such as distances and angles, for energy terms and for algorithmic procedures, such as line-search. We take full advantage of the inheritance mechanism provided by Java to maximize code reuse on the one hand and clarity of the program on the otherhand. For example, the low-level class 'AbstractEnergy' is extended by a variety of higher-level classes representing different energy terms (Fig. S4). These classes serve as building blocks for applications.

The MESHI classes are arranged in a hierarchy of packages. A brief summary of the five major packages is presented below. For more details see the supplementary information and the MESHI API.

Molecular elements. This package includes general purpose classes for atoms, residues and proteins (Figs S2 and S3). It also includes specialized lists. Specific molecular models (e.g. All-atom and Cα-only proteins) are represented by sub-packages, which include extensions of the basic classes.

Geometry. This package includes classes that represent coordinates, distances, angles and torsion angles, as well as specialized containers (lists and the DistanceMatrix are described below). The geometry objects may be shared by any number of energy functions.

Energy. This package includes general purpose abstract classes (Fig. S4) that represent different aspects of energy terms: reading parameters from files, binding of atoms and geometry elements to their different roles in the energy function, and the actual evaluation of the energy. Specific energy terms (e.g. bond and angle) are sub-packages containing extensions of the basic abstract classes. The TotalEnergy class is a container that can store and handle any number of energy terms and present a simple interface to the optimization classes.

Optimizers. Includes several classes that implement optimization and conformational search algorithms (Fig. S5). Currently, only algorithms that rely on energy function derivability are implemented. The most useful ones are LBFGS (Liu and Nocendal, 1989) and MCM (Li and Scheraga, 1987).

Util. This class includes several utility classes that handle files, lists and command interpretation.

*To whom correspondence should be addressed.

SPECIAL FEATURES

The Distance Matrix class. Inter-atomic distances are required by almost all energy terms. Typically the distances are calculated by the procedures that implement the energy terms. It is very common, in molecular modeling software, to merge the van der Waals and electrostatic calculations into one procedure. This way one avoids the duplication of both code and computer time. However, it is hard to generalize this approach. If one wants to experiment with quite a few energy terms, as we do, the resulting code may be rather cumbersome and bug prone. The DistanceMatrix class was designed to solve this problem. It calculates and stores all the inter-atomic distances that are required for any energy term. This central handling guarantees that each inter-atomic distance is calculated (at most) once in each energy evaluation step, no matter how many energy terms require it.

The DistanceMatrix class is also responsible for the major computational bottleneck, the updating of the non-bonded-list. Intensive code-optimization efforts have been applied to this class, with significant success.

Built-in energy function testing. All the currently implemented energy terms are derivable. From the developer's point of view, derivable energy functions have a major advantage: they include a built-in quality assurance mechanism. A derivable energy function is practically coded twice, the function itself and its derivative. A mistake may occur in each part of the code, but it is extremely implausible that two complementary mistakes will occur. Such errors manifest themselves by simulation breakdown, as the force-driven algorithms (e.g. steepest descent) rely on the accurate derivation of the energy function.

While it is fairly easy to Figure out that an error has occurred, locating the bug may be a rather difficult task. Especially so, when the bug has to do with some unanticipated edge case, occurring in the middle of the simulation. An old 'trick of the trade' is to compare the analytical derivative of the energy function with the numerical one. MESHI energy functions come with a built-in test utility that locates the specific energy term and the specific coordinate at which the analytic and numeric derivatives diverge.

Command file interpretation. Molecular modeling programs typically have numerous user-defined parameters: number of steps, convergence criteria, weights of different energy terms, input file names, etc. These parameters need to be read, interpreted, stored and provided to the program components that require them. Handling user-defined parameters tends to clutter the programs. It also makes it hard to write new programs that use the existing building blocks differently.

In MESHI, moving the responsibility to the classes solves this problem. The program uses a single CommandsList object that builds itself from a command file. Each component of the program extracts the parameters it needs from this object. The CommandsList object provides various safeguards to ensure that the components of the program get all the parameters that the user intended them to have.

IMPLEMENTED ENERGY TERMS

The current distribution includes the following energy terms: bond, angle, plane, out-of-plane (chirality), distance constraints, Lennard-Jones, excluded volume, electrostatics, implicit solvation, torsion pair (Ramachandran+rotamers), hydrogen bonding and cooperative (patterned) hydrogen bonding.

The parameters for most of these terms are knowledge based, reflecting our focus on protein structure prediction. The current release does not include any implementation of an established force field. An exception is the electrostatic energy term implementing the OPLS force field (Jorgensen and Tiradorive, 1988) as used in the MOIL package (Elber *et al*., 1995). This implementation was successfully completed as a short undergraduate project. The electrostatic term demonstrates that MESHI is flexible enough to allow easy implementation of any reasonable established force field.

CURRENT APPLICATIONS

MESHI is a platform for development of novel algorithms and energy functions. As such, its main deliverables are classes and methods, not programs. Notwithstanding this, the current distribution does include two applications.

MinimizeProtein is a 'getting started program' where generality and usefulness were sacrificed for simplicity sake. It minimizes a protein structure according to standard energy terms, such as covalent bonding and VDW. The *Beautify* program demonstrates the usability of MESHI even in its current stage. This program completes and refines fragmented Cα models generated by fold-recognition methods. Beautify produces all-atom models, minimized according to selected MESHI energy functions, with all the missing residues of the original models completed. A preliminary version of Beautify was tested in the CASP6 experiment. For a brief summary of the results see the Supplementary information.

ACKNOWLEDGEMENTS

While most of the code is original, it was written under the inspiration of the packages written by Ron Elber, Michael Levitt, Jeffery Skolnick and Andrzej Kolinski that C.K. was advantageous to study, use and manipulate. Eitan Domany's support was invaluable. C.K. is a Ralph Selig Career Development Chair in information theory, N.K. is supported by the Kreitman Foundation. The EU 6th Framework Program is gratefully acknowledged for support to the GeneFun project, contract No: LSHG-CT-2004-503567. S.Z.L. was supported by the Israeli Ministry of Science.

Conflict of Interest: none declared.

REFERENCES

Bull,M., Smith,L., Pottage,L. and Freeman,R. (2001) Benchmarking Java against C and Fortran for scientific applications. *Proceedings of the 2001 joint ACM-SCOPE conference on Java Grade*. ACM Press, CA, pp. 97–105.

Elber,R. *et al*. (1995) MOIL: a program for simulations of macromolecules. *Comput. Phys. Commun*., **91**, 159–189.

Li,Z. and Scheraga,H.A. (1987) Monte Carlo-minimization approach to the multiple-minima problem in protein folding. *Proc. Natl Acad. Sci. USA*, **84**, 6611–6615.

Liu,D.C. and Nocedal,J. (1989) On the limited-memory BFGS method for large scale optimization. *Math. Program.*, **45**, 503–528.

Jorgensen,W.L. and Tiradorive,J. (1988) The OPLS potential function for proteins—energy minimization for crystals of cyclic-peptides and crambin. *J. Am. Chem. Soc*., **110**, 1657–1666.

Prechelt,L. (1999) Comparing Java vs. C/C++ efficiency differences to interpersonal differences. *Commun. ACM*, **42**, 109–112.

Vivanco,R.A. and Pizzi,N.J. (2005) Scientific computing with Java and C++: a case study using functional magnetic resonance neuroimages. *Softw. Pract. Exper*., **35**, 237–254.