ABSTRACT

Summary: Computational Structural Biology Toolbox (CSB) is a cross-platform Python class library for reading, storing and analyzing biomolecular structures with rich support for statistical analyses. CSB is designed for reusability and extensibility and comes with a clean, well-documented API following good object-oriented engineering practice.

Availability: Stable release packages are available for download from the Python Package Index (PyPI) as well as from the project’s website http://csb.codeplex.com.

Contacts: ivan.kalev@gmail.com or michael.habeck@tuebingen.mpg.de

Received on July 10, 2012; revised on August 6, 2012; accepted on August 24, 2012

1 INTRODUCTION

The Python programming language is becoming an increasingly popular choice in research. With its comprehensive numerical libraries and dynamic type system, Python facilitates rapid application development. But although rapid prototyping is very practical for experimenting with new techniques or features, systematic use of ad hoc scripting often turns into a burden preventing efficient code reuse. This problem is solved with the use of continuously developed, well-abstracted and tested software libraries. Productivity in building solid, reliable and extensible bioinformatics applications could therefore significantly benefit from the practice of using library code.

Here, we introduce the public release of CSB, a Python library designed for solving problems in the field of computational structural biology. CSB improves over existing libraries such as Biopython (Cock et al., 2009) with its granular, consistent and extensible object model and also provides new features like a comprehensive statistical API and support for new abstractions and file formats. This project is a quickly growing class library designed for reusability and extensibility and comes with a clean, well-documented API following good object-oriented engineering practice.

2 CORE LIBRARY

CSB consists of several highly branched, hierarchical Python packages. The core library is roughly divided into bioinformatics (csb.bio.*) and statistics (csb.statistics*).

The csb.bio namespace hosts a number of packages that define all fundamental biological abstractions of the library. For example, csb.bio.sequence defines the AbstractSequence and AbstractAlignment objects and also provides some standard implementations of these abstractions, such as Sequence, SequenceAlignment and StructureAlignment. As suggested by its name, csb.bio.hmm deals with HHpred and its profile HMMs (Söding, 2005), while csb.bio.fragments contains objects describing protein structure fragment libraries. The package csb.bio.structure implements essential CSB objects like Structure, Chain, Residue and Atom.

The Structure class illustrates examples of CSB’s design philosophy. Structure instances are hierarchical objects, implementing the composite pattern. Each level in this hierarchy is represented by a class, derived from the base AbstractEntity. Every entity thus exposes a standard set of operations such as AbstractEntity.transform(), which automatically propagate down the tree when invoked at arbitrary level. Users are free to define their own, pluggable AbstractEntity implementations.

Another aspect of the core library is our I/O API for a broad variety of biological file formats (csb.bio.io*). For example, csb.bio.io.hhpred is the first publicly available Python module to date for working with HHpred’s HMM and result files (Söding, 2005). Another module, csb.bio.io.mrc, implements objects for processing cryo-electron microscopy maps, where csb.bio.io.wwpdb provides I/O for CLANS (Frickey and Lupas, 2004). Reading and writing PDB files is done using our PDB API, which is part of csb.bio.io.wwpdb. It is worth mentioning that the default PDB parser in CSB differs significantly from existing solutions such as Biopython. StructureParser reads and initializes all residues from SEQRES (if available), rather than the ATOM fields in the file. ATOM records are subsequently mapped to the residue objects using a simple and fast alignment algorithm. Therefore, Chain objects in CSB always contain the complete primary structure of the PDB chain, as defined by the SEQRES fields. This feature eliminates the need to relate the PDB atoms back to the real sequence of the protein in question—a process which is often difficult and error-prone. When benchmarked over the complete PDB database, our SEQRES mapping algorithm fails for about 250 structures. This is frequently an indication of a PDB format issue.

*To whom correspondence should be addressed.
case, the parser would raise a characteristic exception upon which the client can switch to an ATOM-based parsing mode.

We compared the performance of RegularStructureParser with PDB I/O modules from alternative libraries: Biopython, PyCogent (Cielick et al., 2011) and the C++ based Open Structure (Biasini et al., 2010). As expected, OpenStructure was the fastest and parsed 4000 PDB entries with 0.09 s per structure. CSB is positioned between Biopython (0.19 s) and PyCogent (0.43 s) with 0.32 s per structure, which suggests that the SEQRES mapping feature comes with an acceptable performance overhead.

Our library also hosts a collection of statistical models in the csb.statistics namespace. Among these models are standard univariate and multivariate probability distributions such as the Normal and the Gamma distribution and also more exotic distributions such as the multivariate normal inverse Gaussian distribution used to model multivariate heavy-tailed data. Several estimators based on maximum likelihood and Gibbs sampling are implemented. Moreover, we provide a general framework for Markov chain Monte Carlo simulation and implementation of standard schemes such as random walk Metropolis Hastings, Hamiltonian Monte Carlo (Duane et al., 1987) and replica-exchange Monte Carlo (Swendsen and Wang, 1986). Methods to analyze Monte Carlo output are also provided such as, for example, a non-parametric histogram reweighting scheme for the estimation of free energy differences (Habeck, 2012).

4 DEVELOPMENT

One of the key design goals of CSB is providing clean, extensible, object-oriented APIs with accompanying API documentation. This project puts a strong emphasis on quality, achieved through systematic use of abstraction, strong encapsulation, separation of responsibilities and refactoring with classic design patterns.

Our development team has adopted a continuous integration model. The reliability of the production code is controlled by CSB’s built-in high-coverage unit test framework. Stable builds will be gradually released to the public domain, and nightly builds can be obtained upon request. Portability is also a design goal, so CSB works without modification on every major platform (Windows, Linux and Mac) and any modern Python interpreter (version 2.6 or higher, including Python 3).

Funding: Contract research ‘Methoden in den Lebenswissenschaften’ of the Baden-Württemberg Stiftung, by Deutsche Forschungsgemeinschaft (DFG) grant HA 5918/1-1 and by the Max Planck Society.

Conflict of Interest: none declared.

REFERENCES