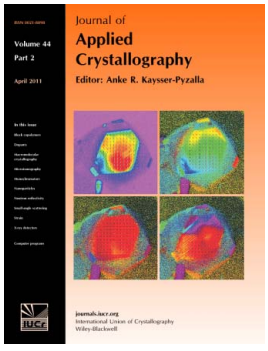
Journal of
Applied Crystallography
ISSN 0021-8898
Editor: **Anke R. Kaysser-Pyzalla**

Abstractions, algorithms and data structures for structural bioinformatics in *PyCogent*

Marcin Cieślik, Zygmunt S. Derewenda and Cameron Mura

J. Appl. Cryst. (2011). **44**, 424–428



Journal of
Applied Crystallography
ISSN 0021-8898

Received 17 June 2010
Accepted 7 February 2011

Abstractions, algorithms and data structures for structural bioinformatics in *PyCogent*

Marcin Cieślik,[a] Zygmunt S. Derewenda[b] and Cameron Mura[a]*

[a]Department of Chemistry, University of Virginia, Charlottesville, VA 22904, USA, and [b]Department of Molecular Physiology and Biological Physics, University of Virginia Health Sciences Center, Charlottesville, VA 22908, USA. Correspondence e-mail: cmura@virginia.edu

To facilitate flexible and efficient structural bioinformatics analyses, new functionality for three-dimensional structure processing and analysis has been introduced into *PyCogent* – a popular feature-rich framework for sequence-based bioinformatics, but one which has lacked equally powerful tools for handling stuctural/coordinate-based data. Extensible Python modules have been developed, which provide object-oriented abstractions (based on a hierarchical representation of macromolecules), efficient data structures (*e.g.* *k*D-trees), fast implementations of common algorithms (*e.g.* surface-area calculations), read/write support for Protein Data Bank-related file formats and wrappers for external command-line applications (*e.g.* *Stride*). Integration of this code into *PyCogent* is symbiotic, allowing sequence-based work to benefit from structure-derived data and, reciprocally, enabling structural studies to leverage *PyCogent*'s versatile tools for phylogenetic and evolutionary analyses.

1. Introduction and motivation

Structural biology continues to enjoy unprecedented rates of data accumulation. Together with structural genomics, advances in macromolecular crystallography have contributed to $>7 \times 10^4$ known three-dimensional structures, at recent rates of several thousand new structures per year (Terwilliger *et al.*, 2009). This deluge of coordinate data impacts fields ranging from molecular biophysics (*e.g.* elucidating relationships between conformational dynamics and protein function) to structural bioinformatics (*e.g.* determining fractional coverage of fold space) to more applied fields such as drug design (*e.g.* incorporating new structural knowledge in designing receptor–ligand complexes). Much of the effort in these areas now involves grappling with the sheer volume and heterogeneity of data. For instance, a biochemist seeking to analyse a 'representative' set of structural data may need to embark upon the tedious task of constructing a non-redundant subset of coordinates by, perhaps, removing structures exceeding some sequence (or structural) similarity threshold; similarly, in structural bioinformatics, sufficient statistical sampling entails calculations over as large a data set as possible, therefore necessitating robust automatable data processing.

Robust processing of coordinate data places stringent demands on both the data and the data-processing tools: data sets must be structured in some organized, well defined (parsable) manner, and the applied tools must be *scalable* (to cope with data volume), *robust* (to handle data heterogeneity, formatting errors) and *flexible* (to permit custom analyses). These requirements arise regardless of the sophistication or computational complexity of the structure analysis being undertaken.

1.1. Data format difficulties

In terms of data format, the Protein Data Bank (PDB) file (Bernstein *et al.*, 1977) is the standard exchange and archival system for macromolecular structures (atoms and three-dimensional coordinates) and associated structure-determination data (biopolymer sequence, details of the diffraction or NMR experiment, refinement steps *etc.*). More flexible data exchange formats, such as mmCIF (Westbrook & Bourne, 2000), have been devised, as have application-specific derivatives of the PDB format – *e.g.* 'PQR' files (Dolinsky *et al.*, 2004) replace the *B*-factor and occupancy fields with atomic charges ('*Q*') and radii ('*R*'), for use in electrostatics calculations. However, structure analysis schemes based on replacing standard PDB fields are inherently limited: (i) custom scripts are generally required to apply user-defined (or computed) values on a per-atom basis (*i.e.* mapping onto '`ATOM`' fields), rather than at an arbitrary level in the structural hierarchy (per-residue, per-chain *etc.*); and (ii) this approach is limited to storage of only a couple of different types of (scalar) data fields (*e.g.* *Q*, *R*). Nevertheless, the PDB file format remains the *de facto* standard for biopolymer three-dimensional structures, and therefore must be dealt with both flexibly and robustly in order to automate structural analysis workflows.

By providing high-level abstractions and robust data structures for coordinate (and coordinate-derived) data, the software introduced here transcends the difficulties and limitations of such approaches as manipulating fixed-width predefined PDB fields.

1.2. Data-processing challenges

Typical structure analysis pipelines begin with three-dimensional coordinates as primary data and involve computing a wide variety of secondary/derived properties. Calculated quantities may include (i) physics-based characteristics, such as electrostatic potentials; (ii) geometric features, such as concavity and curvature, shape complementarity indices (Lawrence & Colman, 1993), and surface areas (buried, solvent accessible, van der Waals envelope *etc.*); (iii) protein structural descriptors (secondary structure assignments); and (iv) a host of 'hybrid' physicochemical measures, such as hydrogen-bonding patterns, geometric distributions of residues (*e.g.* radial distribution

functions of atom types), hydrophobic clusters, interaction patches, hydrophobic moments and so on. Many separate tools for performing one or other of these sorts of calculations exist, but these are often in the form of machine/architecture-specific binaries that take some input files and command-line arguments and yield the resulting data as output files in a format highly specific to that particular program. [Indeed, one of the early motivations for software suites such as *CCP4* (Collaborative Computational Project, Number 4, 1994) was to unify the input/output syntax across a set of individually useful programs.] However, performing structure analysis in this manner is brittle, error-prone and almost always necessitates development of a set of post-processing scripts ('glue code') to finesse the results into a form usable for the next stage of data analysis. This approach becomes further untenable if multiple streams of output must be combined (extensive dataflow interdependencies) or if the structure analysis steps require the chaining together of numerous output values $N_{i-1} \rightarrow N_i \rightarrow N_{i+1}$. In short, 'one-off' tools/scripts are generally quite idiosyncratic in nature, with each new set of analyses requiring a new set of scripts.

The software presented here provides general-purpose libraries and tools – easily accessible and manipulable data structures, algorithms for coordinate-based calculations – for both routine and sophisticated (user-configurable) structural analyses. Integration of this functionality into the versatile, well established *PyCogent* bioinformatics framework (Knight *et al.*, 2007) effectively supplies a highly extensible toolkit for structural bioinformatics workflows, obviating many of the data-processing challenges described above.

2. Overview of the developed software

After outlining the architecture, design considerations and implementation of our code, this section summarizes the general software capabilities. Thorough descriptions of our abstractions and data structures (illustrated with concrete examples), as well as notes on highly specific details such as our code's handling of sequence and structural heterogeneity in PDB files (atomic alternate location identifiers, insertion codes), can be found in the supplementary material[1] (§§1 and 2).

2.1. Architecture and design

We have employed a hierarchical internal representation of macromolecular structures, reflecting the implicit *Structure* $\supset$ *Model* $\supset$ *Chain* $\supset$ *Residue* $\supset$ *Atom* ('SMCRA') organization of a PDB file (Hamelryck & Manderick, 2003). Such a hierarchy is most naturally programmed in the object-oriented paradigm. Elements of our hierarchical design pattern are termed 'entities', with base classes *Entity* and *MultiEntity*. At the top tier of the hierarchy is a *Structure*, which is a container for *Models*. Each *Model* can hold multiple *Chains*, which in turn contain *Residues*. The bottom of the hierarchy consists of *Atoms*, which are themselves individual entities (not higher order; childless). Thus, any atom stored in our data structure is uniquely identified by its location within the hierarchy, effectively providing a per-atom *composite identifier*. Each entity between the (top) level of *Structure* and the (bottom) level of *Atom* is linked to potentially multiple children (entities lower in the hierarchy tree) and to only a single parent (higher in the hierarchy tree). In other words, there are no cycles or closed loops in the graph of this tree-like hierarchy. As a concrete example, note that a particular *Residue* has a single *Chain* parent and multiple *Atom*ic children.

We extend this SMCRA schema in *PyCogent via* an additional group of virtual 'holder entities' lying outside the hierarchy. These *holders* are used to define groups of entities irrespective of their location within the hierarchy, thereby enabling one to combine structural units (residues, atoms) into application-specific clusters, patches or any other type of arbitrary (user-definable) collection of structural elements. Each *holder* is technically an unrestricted grouping of entities of the same hierarchical level (*e.g.* atoms), and holder children are guaranteed to have at most one parent. Holders, which may be (loosely) thought of as 'atom selections' in the macro language of programs such as *PyMOL* (DeLano, 2002), are intended to be temporary in nature, and can be created or destroyed without affecting the underlying macromolecular structure. In the parlance of object-oriented programming, *Entity* is a 'base class' (or a 'superclass'), and the relationship between the two complementary hierarchies – SMCRA and holders – is most definitively seen by inspecting the *Entity* inheritance diagram.

PyCogent abstracts a PDB file as a top-level *Structure* entity in the hierarchical data structure described above, and allows for its easy and arbitrary manipulation without compromising the uniqueness of the corresponding structure. As explicitly shown in some of the examples in the supplementary material (§2), modifications to the *Structure* can involve such operations as changing the attributes drawn from the PDB file (*e.g.* altering coordinates or residue numbers), or annotating with additional attributes that are generated either within *PyCogent* proper or by external libraries (*e.g.* computing electrostatic potentials and mapping the values as attributes of the nearest atom to a voxel). Such modifications are typically on per-atom or per-residue bases, and are therefore initially mapped onto entities at those levels. However, *PyCogent* also provides functionality to propagate these values across the hierarchy tree. The 'type' of propagation that is performed depends on the nature of the data items – (i) scalar values can be summed or averaged [*e.g.* the accessible surface area (ASA) of a residue is the sum of ASA values of its constituent atoms, whereas residue depth could be computed as mean atomic depth], while (ii) ordinal data types are generally propagated using set-theoretic (union or intersection) operations (*e.g.* defining the list of atoms contacting residue i as the union of all the pairwise contacts of atoms in residue i with non-i atoms).

2.2. Implementation and availability

Implemented in standard (CPython) platform-independent Python, our software makes extensive use of object-oriented (OO) programming. The benefits of scientific software development in the high-level interpreted Python language, using an OO approach, have been reviewed elsewhere (Knight *et al.*, 2007; Grosse-Kunstleve *et al.*, 2002). Key OO concepts in our codebase – the *Entity* base class, attribute dictionaries, *holders etc.* – are outlined above and further described in §§1 and 2 of the supplementary material. In addition, §3 of the supplementary material provides further information on code performance (benchmarking and timing statistics; §3.1) as well as notes on platform independence and Cython extensions (§3.2). The code development described here has been integrated into the latest stable release (version 1.4.1) of the open-source *PyCogent* project and is freely available at http://pycogent.sourceforge.net. Extensive documentation can be found at that URL and in the supplementary material.

[1] Supplementary material discussed in this paper is available from the IUCr electronic archives (Reference: KK5078). Services for accessing this material are described at the back of the journal.

2.3. Capabilities and features

The software introduced herein extends *PyCogent*'s capabilities into the realm of crystal structure processing and analysis (Table 1). The application programming interface aims to strike a balance between flexibility/extensibility (advanced users, more complex analyses) *versus* robust/simpler default behavior for handling of PDB files and PDB-derived structural data. Thus, various capabilities are provided by generalized classes and methods, which expose that particular functionality (*e.g.* surface-area calculations) to advanced users with as few limitations as possible. For example, the default parameter set of atomic radii that we provide, drawn from the *Areaimol* program (Collaborative Computational Project, Number 4, 1994), can be easily user-adjusted for different applications and purposes.

In addition to reading/writing standards-compliant PDB files (and close relatives, such as PQR), we provide facilities for (1) arbitrary entity selections (atoms, residues *etc.*); (2) flexible data manipulation and propagation (described above); (3) protein structure cleanup; (4) fast calculation of ASAs and molecular surfaces; (5) unit-cell- and lattice-related calculations (finding crystal contacts, computing coordination numbers); (6) nearest-neighbor searches; and (7) superimposing structures. Some of these functionalities are more computationally expensive, be it due to numerical reasons (*e.g.* surface area calculations) or combinatorial complexity (*e.g.* constructing nearest-neighbor and contact lists); computationally intensive portions of the code were implemented using C extensions for Python, *via* the Pyrex-based Cython compiler (Behnel *et al.*, 2008). Further details, particularly with respect to issues of cross-platform compatibility and C extensions, are provided in §3 of the supplementary material.

Most of the aforementioned features are intrinsic (*i.e.* free of external dependencies), including surface area calculations and contact searching (Table 1). Parameters for each intrinsic algorithm are user-accessible and adjustable – one can modify atomic and probe radii, source and target atom collections for contacts *etc.* The algorithms properly handle crystal lattices and space-group symmetry such that, for instance, the correct neighborhoods of symmetry mates are generated in crystal contact calculations. Other functionality is provided by wrapping existing, freely available software – for instance, application controllers and utility functions are provided for molecular surface calculations using *MSMS* (Sanner *et al.*, 1996) and secondary structure assignment *via Stride* (Frishman & Argos, 1995). In such cases, all input files required by the external binary are auto-generated from our hierarchical representation, and users can modify command-line switches and parameters; finally, we provide parsers for the resulting output, which can then be used to annotate the structure under study. The following subsections describe implementation details for the numerically intensive functionalities – neighbor search, surface calculations and contact calculations.

2.3.1. Nearest-neighbor searches. Nearest-neighbor search (NNS) algorithms are used in many fields, ranging from data compression to DNA sequencing and genome assembly. The k-nearest-neighbors (k-NN) problem is to locate the k closest points to a query point in some well defined (*e.g.* Euclidean, Manhattan) metric space. By employing specialized data structures and space-partitioning methods [such as kD-trees (k-dimensional trees)], the computational complexity of NNS can be reduced from the brute-force $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$. Our code supplies built-in k-NN search procedures for arbitrary structural entities (atoms, residues, user-defined groups of residues *etc.*) *via* a custom kD-tree implementation; this provides the basis for our software's higher-level functionality, such as calculation of entity coordination numbers, surface patches *etc.* The kD-tree functionality was written as compiled Cython code (the 'ckd3' module in the codebase).

Table 1
Major features of the new additions to *PyCogent*.

Major functionality and specific utilities provided by our code are indicated as 'intrinsic'; tools provided *via* external packages are referenced in the text.

Read, write and manipulate PDB files (standardize, renumber, modify fields *etc.*)	Intrinsic
Miscellaneous geometric analyses (*e.g.* compute geometric centers)	Intrinsic
Nearest-neighbor search, kD-trees	Intrinsic
Interatomic contact calculations	Intrinsic
Molecular surfaces	*MSMS*
ASA calculations	Intrinsic, *Stride*
Atom/residue depth calculations (under development)	Intrinsic
Secondary structure calculation and residue-based ASA	*Stride*
Proper handling of space-group symmetry	Intrinsic

2.3.2. Surface calculations. Of the many physicochemical features of a protein structure, the ASA plays a key role in defining biochemical activity. The new code reported here now provides the *PyCogent* package with many tools (again, intrinsic and external) for computing and handling surface area data. The relative accessible surface area (rASA) can also be computed; this parameter is normalized by the maximum possible ASA (*i.e.* in an extended peptide) for each amino acid, thereby offering a useful criterion for comparison between different residue types (Rost & Sander, 1994). Residue-based ASAs can be computed using our interface to *Stride*; though primarily included for purposes of secondary structure assignment, *Stride*-derived ASA values can be used for cross-validation and comparison, *versus* ASAs computed using our built-in (Cython-encoded) ASA facility. (Thus, users are provided with multiple options for a single type of task.) An interface is also provided to the *MSMS* program, which exists as a stand-alone binary that takes as input a file of coordinates and associated atomic radii, and generates an output file in the form of an array of surface dots. Given a PDB file our software (i) produces the necessary input files for an *MSMS* calculation, (ii) allows user modification of such parameters as atomic radii and run-time options (surface accuracy), and, finally, (iii) provides data structures to access the output array as a matrix of floating point numbers.

2.3.3. Intermolecular contacts. Several types of intermolecular contacts may be important in biochemical and structural studies of physiological functions (protein–protein, protein–ligand interactions); in characterization of biophysical properties, such as hydration (protein–water); and in understanding the crystallization process itself (homologous and possibly heterologous protein–protein contacts within an asymmetric unit and across a lattice). The problem of computing chain or residue coordination numbers is an essentially identical task to contact analysis. The coordinates in a PDB file correspond to a single asymmetric unit, posing a potential problem for crystal contact calculations. Our *PyCogent* implementation provides functionality for inter-entity (inter-{atom, residue, chain}) calculations, as detailed in §4 of the supplementary material; notably, lattices and space-group symmetry are properly handled such that, for instance, the correct neighborhood of symmetry mates is generated in *PyCogent*-based crystal contact calculations.

3. Case studies

The first case study below (§3.1, Fig. 1) shows the ease with which routine structural tasks can now be performed in *PyCogent*. Then, to demonstrate how our new software can leverage *PyCogent*'s existing repertoire of sequence-based functionality to perform more complex

structural analyses, an advanced example (§3.2, Fig. 2) computes sequence conservation scores (as profile entropies) and maps the resulting information onto a structure. Additional examples are provided in the supplementary material, including (i) further details on this second use case (§5.1); (ii) calculation of interatomic contacts (§4); and (iii) a use case that expands upon the §3.2 sample shown below by demonstrating how coordinate-derived data (*e.g.* secondary structural information) can be incorporated into a relatively sophisticated statistical sequence/structure calculation. Notably, such a calculation cannot be performed as efficiently(/compactly) in existing structural analysis software packages of which we are aware.

3.1. Display the quaternary structure of an oligomer

To demonstrate the ease of accessing our new structural tools in *PyCogent*, the following block of code shows, in conjunction with a Python-aware molecular graphics system, how one can efficiently (∼10 lines) go from a raw PDB file to a visual representation of the quaternary architecture of a protein complex. In this example, the center of geometry is automatically computed (as implicit attributes of *Chain*-level entities) by *PyCogent*'s structure-handling functionality. *PyMOL*'s compiled graphics object (CGO) facility is utilized to render the centers as spheres (Fig. 1), the final CGO object having been built-up *via* successive iterations over the chains of the assembly (line 4 loops at level 'C' of SMCRA). Note that the following block of code is entered directly at the *PyMOL* command prompt (a further note on setting up *PyMOL* to be *PyCogent*-aware can be found in §3.3 of the supplementary material):

```
from cogent.parse.pdb import PDBParser # PyCogent
myPDBfile = open('./1i8f.pdb')
myNewStruc = PDBParser(myPDBfile)
for chain in myNewStruc.table['C']:
  myChainID = myNewStruc.table['C'][chain].getName()
  # initialize a PyMOL CGO list:
  cgoString = [COLOR, 0.7, 0.7, 0.7]
  if myChainID ! = ' ': # only non-empty chains
   # fetch center of geometry as numpy array:
   center = myNewStruc.table['C'][chain].coords
   # build-up CGO:
   cgoString.extend ([SPHERE, center[0], center[1], \
    center[2], 3.0]) # 3 A-radius sphere
   # and load into PyMOL as uniquely id'd spheres:
   cmd.load_cgo(cgoString,'cent_'+myChainID)
```

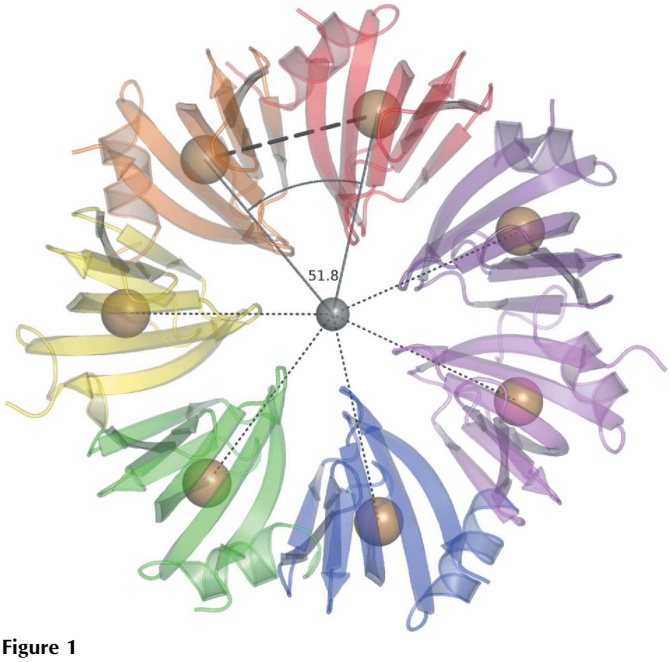


Figure 1
Architecture of a heptameric assembly (PDB code 1i8f; Mura *et al.*, 2001), illustrated in terms of the geometric centers of the entire complex (gray sphere) and its individual subunits (tan spheres). Automatically computed center positions were retrieved while looping over protein chains (see code in text).

3.2. Compute sequence entropy, map onto structure

This case study illustrates how sequence-derived information can be mapped onto a structure, *e.g.* for purposes of visualization. The specific task is to identify, and then illustrate, fragments of a protein structure that are conserved at the level of sequence, with 'conservation' measured as bits of entropy (a quantity readily computable from an alignment; see, *e.g.*, Durbin *et al.*, 1998). This analysis proceeds through several stages, involving (i) loading an alignment and computing its profile-based Shannon entropy at each residue position; (ii) loading a three-dimensional structure representative of the sequences in the above alignment, fetching (*e.g.* from UniProtKB; http://www.uniprot.org/) sequences similar to that in the PDB file and performing the many manipulations required to properly align it; and (iii) computing a valid mapping of entropy scores to residues in the PDB file, in order to be able to (iv) annotate the three-dimensional structure with entropy scores and store the structure for downstream visualization (*e.g.* by color-ramping residues based on bit scores).

Perhaps the most difficult step in the above workflow is correctly mapping residues from the three-dimensional structure to positions in the sequence alignment. As detailed in the supplementary material (§5.1), global sequence alignment *via* the Needleman–Wunsch algorithm was used to establish a correspondence between three-dimensional structure (PDB-derived sequence) and sequence profile position. Uncertainty bits (entropy scores) for alignment positions that correspond to gaps in the three-dimensional structure-derived sequence (*i.e.* missing residues in the PDB file) were simply discarded, as they cannot be visualized. The final sequence entropy →

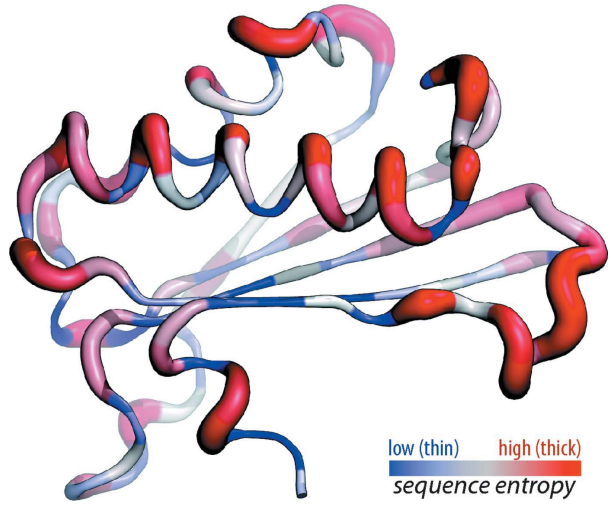


Figure 2
In this example (§3.2), sequence entropy was computed for the chorismate mutase family and mapped to PDB structure 1ui9 (E. Inagaki, S. Kuramitsu, S. Yokoyama, M. Miyano & T. H. Tahirov, in preparation). Note that higher-entropy segments (red/thicker tubes) correspond to loops or exposed regions, whereas the β-sheet core is characterized by lower overall entropy values (blue/thin tubes).

structure mapping is shown in Fig. 2. Notably, integrating our structure-handling software with *PyCogent*'s bioinformatics utilities enabled this relatively involved structural analysis to be achieved fairly efficiently (<50 lines of code; see §5.1 in the supplementary material).

4. Summary

We have developed software for the processing and analysis of three-dimensional structural data, utilizing abstractions and hierarchical data structures that are well suited to representing macromolecular structures. The object-oriented code represents biopolymer structure *via* multiple classes and methods that together can be used to generate, process and store structural data; the software also provides efficient algorithms for manipulating coordinate-related data structures. Functionality for common structure analysis tasks is provided, including (i) file input/output (PDB parsers, writers); (ii) more advanced data structures, suitable for analysis of three-dimensional coordinate sets (*k*D-trees); (iii) structure analysis algorithms (surface areas, interatomic contacts); and (iv) a host of utility functions (structure cleanup, superposition, entity selection, data propagation). The software can also be interfaced to external libraries or used to 'wrap' binaries (*e.g. CCP4*), thereby combining the efficiency of compiled languages like C or Fortran with the high level of abstraction and readability of an interpreted language such as Python. This latter aspect – using an interpreted language with an enhanced, interactive shell (Pérez & Granger, 2007) – facilitates exploratory data analysis and rapid prototyping of structural analysis workflows.

Originally motivated by needs arising in specific structural analysis projects (Cieślik & Derewenda, 2009), our software is not the only toolkit of this type [see, *e.g.*, *p3d* (Fufezan & Specht, 2009) or *Bio3d* (Grant *et al.*, 2006)]. However, the code presented here is highly extensible, which is crucial for development of automated processing and structure analysis pipelines; in addition, relative to currently available packages, some aspects of the software (such as the *k*D-tree implementation) provide rich functionality for neighbor search and related tasks (such as protein contact analysis). The emphasis on robust fault-tolerant file processing and flexible data manipulation makes the code lightweight and concise when compared with more ambitious Python packages that focus, for instance, on molecular simulations (*e.g. MMTK*; Hinsen, 2000). Perhaps the most useful feature of our software is not the code itself but rather its context: in order to leverage the power of both sequence- and structure-based analysis, the software was fully integrated into the modern feature-rich *PyCogent* bioinformatics framework. As illustrated above (§3.2), this symbiotic relationship enables sequence/structure analyses – *i.e.* structural bioinformatics – to be pursued in a highly integrated, flexible and efficient manner.

We gratefully acknowledge support from University of Virginia start-up funds (to CM) and the Jeffress Memorial Trust (J-971; CM), as well as the NIH's NIGMS *via* the PSI2 Program (U54 GM074946-01; ZSD).

References

Behnel, S., Bradshaw, R. & Sverre Seljebotn, D. (2008). *Cython: C-Extensions for Python*, http://www.cython.org.
Bernstein, F. C., Koetzle, T. F., Williams, G. J., Meyer, E. F., Brice, M. D., Rodgers, J. R., Kennard, O., Shimanouchi, T. & Tasumi, M. (1977). *J. Mol. Biol.* **112**, 535–542.
Cieślik, M. & Derewenda, Z. S. (2009). *Acta Cryst.* D**65**, 500–509.
Collaborative Computational Project, Number 4 (1994). *Acta Cryst.* D**50**, 760–763.
DeLano, W. L. (2002). *The PyMOL Molecular Graphics System*, http://pymol.org.
Dolinsky, T. J., Nielsen, J. E., McCammon, J. A. & Baker, N. A. (2004). *Nucleic Acids Res.* **32**, W665–W667.
Durbin, R., Eddy, S., Krogh, A. & Mitchison, G. (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press.
Frishman, D. & Argos, P. (1995). *Proteins*, **23**, 566–579.
Fufezan, C. & Specht, M. (2009). *BMC Bioinformatics*, **10**, 258.
Grant, B. J., Rodrigues, A. P. C., ElSawy, K. M., McCammon, J. A. & Caves, L. S. D. (2006). *Bioinformatics*, **22**, 2695–2696.
Grosse-Kunstleve, R. W., Sauter, N. K., Moriarty, N. W. & Adams, P. D. (2002). *J. Appl. Cryst.* **35**, 126–136.
Hamelryck, T. & Manderick, B. (2003). *Bioinformatics*, **19**, 2308–2310.
Hinsen, K. (2000). *J. Comput. Chem.* **21**, 79–85.
Knight, R. *et al.* (2007). *Genome Biol.* **8**, R171.
Lawrence, M. C. & Colman, P. M. (1993). *J. Mol. Biol.* **234**, 946–950.
Mura, C., Cascio, D., Sawaya, M. R. & Eisenberg, D. S. (2001). *Proc. Natl Acad. Sci. USA*, **98**, 5532–5537.
Pérez, F. & Granger, B. E. (2007). *Comput. Sci. Eng.* **9**, 21–29.
Rost, B. & Sander, C. (1994). *Proteins*, **20**, 216–226.
Sanner, M. F., Olson, A. J. & Spehner, J. C. (1996). *Biopolymers*, **38**, 305–320.
Terwilliger, T. C., Stuart, D. & Yokoyama, S. (2009). *Annu. Rev. Biophys.* **38**, 371–383.
Westbrook, J. D. & Bourne, P. E. (2000). *Bioinformatics*, **16**, 159–168.