Bioinformatics Advance Access published September 14, 2015

PyPDB: a Python API for the Protein Data Bank

William Gilpin [1,*]

[1]Department of Applied Physics, Stanford University, Stanford, CA 94305, USA

Associate Editor: Prof. Anna Tramontano

ABSTRACT

Summary: We have created a Python programming interface for the RCSB Protein Data Bank (PDB) that allows search and data retrieval for a wide range of result types, including BLAST and sequence motif queries. The API relies on the existing XML-based API and operates by creating custom XML requests from native Python types, allowing extensibility and straightforward modification. The package has the ability to perform many types of advanced search of the Protein Data Bank that are otherwise only available through the PDB website.

Availability and Implementation: PyPDB is implemented exclusively in Python 3 using standard libraries for maximal compatibility. The most up-to-date version, including iPython notebooks containing usage tutorials, is available free-of-charge under an open-source MIT license via GitHub at https://github.com/williamgilpin/pypdb, and the full API reference is at http://williamgilpin.github.io/pypdb_docs/html/. The latest stable release is also available on PyPI.

Contact: wgilpin@stanford.edu

1 INTRODUCTION

The RCSB Protein Data Bank (PDB) represents one of of the most comprehensive structural biology information databases openly available to genomics and proteomics researchers (Berman *et al.*, 2000). It provides an online interface for browsing amino acid and genetic sequences, as well as crystallographic structures aggregated from a large number of sources. It also provides sophisticated tools for visualizing protein structure and sequence lineages, aligning sequences and searching for homologies, and it provides links to relevant entries in related databases, such as Genbank and UniProt.

The Python scripting language has demonstrated its usefulness to the bioinformatics community as a means of unifying different data sources and analysis tools, allowing diverse data streams to be retrieved, analyzed, and summarized from within the same workflow. This function has been complemented by new interface tools such as the iPython notebook, as well as general-purpose analysis toolkits like Biopython and Biskit (Cock *et al.*, 2009; Grünberg *et al.*, 2007).

In this article, we describe an API for the Protein Data Bank that allows advanced querying of information on PDB entries. Similar utilities exist for programmatic querying of other large bioinformatics databases (including Ensembl, PubChem, and UniProt), but, to our knowledge, no such tool currently exists for the Protein Data Bank (Strozzi and Aerts, 2011; Southern and Griffin, 2011; Patient *et al.*, 2008). Our tool facilitates integration of automatic Protein Data Bank searches within existing Python bioinformatics workflows, and it simplifies the process of performing multiple searches based on the results of existing searches.

*to whom correspondence should be addressed

2 IMPLEMENTATION

The Protein Data Bank currently uses a RESTful API that allows retrieval of information via standard HTML verbs such as POST/GET/PUT/etc. All advanced query types can be represented by XML. PyPDB converts nested dict() objects into structured XML strings, performs searches using these strings, and then parses the XML search results back into nested dict() objects. Most query and search results are represented either as native Python dict() or list() objects depending on their type.

PyPDB supports retrieval but not local manipulation of raw data files (`.pdb`, `.cif`, `.mcif`, etc.) available in the Protein Data Bank, as there are already libraries for manipulating these files included in existing Python toolkits, such as the Biopython module Bio.PDB (Hamelryck and Manderick, 2003). Bio.PDB enables local `.pdb` analysis like secondary structure prediction and neighbor searching, as well as internet retrieval of the full Protein Data Bank and single files using its `get_all_entries` and `retrieve_pdb_file` methods. However, Bio.PDB does not currently support keyword querying or advanced search, making PyPDB complementary to Bio.PDB's extensive range of capabilities.

3 FEATURES

PyPDB can search the PDB by keyword, author, publication date, and experimental method, as well as by specific sequence motifs. For individual PDB IDs, retrievable information ranges from metadata (date of posting, authors, etc.) to the full `.pdb` data file stored with an entry. Standard information such as gene ontology, ligand information, and protein family information can be accessed. PyPDB can also perform BLAST searches.

A list of the most common functions is available in Table 1. A full listing of the available methods is given in the documentation. The arguments, keywords, and outputs for each function are also described in each function's docstring, which can be accessed with `help(name_of_function)`

Example Usage: Figure 1 shows an example script in an iPython notebook. The find_dates() method creates a query dict() and structures it for the Protein Data Bank format (by default, string arguments are treated as keyword searches), and it then sends this query to the Protein Data Bank by internally calling PyPDB's

Table 1. The primary PyPDB functions

Function	Description
make_query	Structure a search request into a dict()
do_search	Perform a search for PDB IDs
get_all	Get all active PDB IDs
describe_pdb	All metadata about PDB entry
get_all_info	All information deposited in PDB entry
get_pdb_file	Retrieve .pdb/.cif/.xml file for PDB entry
get_blast	BLAST search results for PDB entry
find_papers	Find papers associated with keyword
find_authors	Find authors associated with keyword

Additional functions provided in the documentation

Graph new CRISPR entries versus time

```
In [16]: # Perform search
         all_dates = find_dates('crispr', max_results=500)
         all_dates = array(all_dates)
         all_dates = array([int(depdate[:4]) for depdate in all_dates])
         subs_v_time = histogram(all_dates, max(all_dates)-min(all_dates))
         dates, num_entries = subs_v_time[1][1:], subs_v_time[0]
         popgraph = fill_between(dates, 0, num_entries)
```

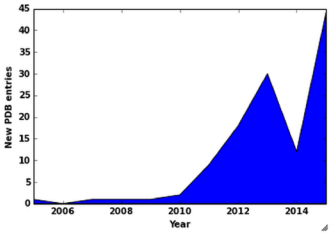

Fig. 1. Example PyPDB workflow in an iPython notebook.

make_query and do_search() methods. The search returns a list of strings representing four-character PDB IDs, and this function then tabulates the deposition dates associated with each PDB ID.The last code block plots the number of PDB IDs found for each year, representing popularity over time of the original search term (in this case, the CRISPR/Cas9 gene-editing method).

Extensibility: PyPDB includes most of the common types of operations involved when working with the Protein Data Bank. However, the depth and range of data represented in the Protein Data Bank may require eventual support for additional functions. Because PyPDB primarily operates by converting native Python dict() objects into valid XML strings, such operations are straightforward to implement. The full code is available on the GitHub website, and forks and pull requests are encouraged.

4 CONCLUSION

The library described here provides direct querying of the the Protein Data Bank using the Python programming language. This API complements the existing Protein Data Bank GUI and XML API by introducing the ability to directly retrieve information from the PDB from within existing Python bioinformatics workflows. The use of native Python datatypes to represent queries simplifies conducting multiple searches with similar queries, and it allows the individual PDB IDs returned in search results to be examined from within the same programming workflow as the original search.

The latest stable release of PyPDB is available on PyPI and can be installed with the command `pip install pypdb`. The source code of all versions of PyPDB is available at https://github.com/williamgilpin/pypdb, and full documentation is available at http://williamgilpin.github.io/pypdb_docs/html/. iPython notebooks containing usage examples also available on the GitHub website.

ACKNOWLEDGEMENTS

The author thanks Sebastian Doniach for his helpful suggestions.

Funding: This work was supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-114747. W.G. receives additional support from the Stanford EDGE-STEM Fellowship and the Stanford H&S Fellowship.

Conflict of Interest: none declared.

REFERENCES

Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T., Weissig, H., Shindyalov, I. N., and Bourne, P. E. (2000). The Protein Data Bank. *Nucleic Acids Research*, **28**(1), 235–242.

Cock, P. J., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., *et al.* (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, **25**(11), 1422–1423.

Grünberg, R., Nilges, M., and Leckner, J. (2007). Biskit–a software platform for structural bioinformatics. *Bioinformatics*, **23**(6), 769–770.

Hamelryck, T. and Manderick, B. (2003). PDB file parser and structure class implemented in Python. *Bioinformatics*, **19**(17), 2308–2310.

Patient, S., Wieser, D., Kleen, M., Kretschmann, E., Martin, M. J., and Apweiler, R. (2008). UniProtJAPI: a remote API for accessing UniProt data. *Bioinformatics*, **24**(10), 1321–1322.

Southern, M. R. and Griffin, P. R. (2011). A Java API for working with PubChem datasets. *Bioinformatics*, **27**(5), 741–742.

Strozzi, F. and Aerts, J. (2011). A Ruby API to query the Ensembl database for genomic features. *Bioinformatics*, **27**(7), 1013–1014.