Parsers, Data Structures and Algorithms for Macromolecular Analysis Toolkit (MAT): Design and Implementation

Gazal Kalyan[a], Vivek Junghare[a], S John S[b], Anupam Chattopadhyay[c,*], Pralay Mitra[d,*], Saugata Hazra[a,e,*]

[a] *Department of Biotechnology, Indian Institute of Technology Roorkee, Roorkee, 247667, India*
[b] *Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Roorkee, 247667, India*
[c] *School of Computer Engineering, Nanyang Technological University, 639798, Singapore*
[d] *Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur , Kharagpur, 721302, India*
[e] *Centre of Nanotechnology, Indian Institute of Technology Roorkee, Roorkee, 247667, India*

Abstract

The structural information of biological macromolecules are stored in .pdb, .mmcif and lately mmtf files and thus it requires accurate and efficient biological tools for various utilities. Here, we describe Macromolecular Analysis Toolkit (MAT) that parses .pdb, .mmcif and .mmtf files; and builds data structures from the input. This original program is written in C++ programming language to ensure efficiency and consistency to organize structural information in an integral way. The novelty of the program lies in the addition of new structure-based biological algorithms and applications. This package also stands out from other similar libraries by being 1) faster and 2) accurate. We also provide detailed comparison of available parsers on the whole PDB database. The parser of MAT is designed in such a way that it allows quick extraction and organized loading of the core data structure. The same data structure is extended to accommodate information from the .mmcif and .mmtf file parsers. Tokenization of the data allows the extraction of information from disordered text, making it compatible for accurate identification of the entities present in the .pdb file. Additionally, we add a new approach of performance optimization by creating a few derived data structures, namely kD-Tree, Octree and graphs, for certain applications that need spatial coordinate calculations. MAT provides advanced data structure which is time efficient and is designed to avail reusability and consistency in

*Corresponding Authors at: Indian Institute of Technology Roorkee, Roorkee (Saugata Hazra) and Nanyang Technological University, Singapore (Anupam Chattopadhyay)

Email addresses: gazal@bt.iitr.ac.in (Gazal Kalyan), vivek.junghare@gmail.com (Vivek Junghare), s@cs.iitr.ac.in (S John S), anupam@ntu.edu.sg (Anupam Chattopadhyay), pralay@cse.iitkgp.ernet.in (Pralay Mitra), saugata.iitk@gmail.com (Saugata Hazra)

a systematic framework. MAT parser can be accessed online through bitbucket at https://bitbucket.org/gazalk/pdb_parser/.

Keywords: , MAT, PDB, Data structure, kD-Tree, Octree

---

1. Introduction

In the past decades, there has been a boom in the rate of growth of experimentally determined macromolecular structures. The three dimensional (3D) coordinates along with biological, molecular and experimental information of those experimentally solved biological macromolecules (protein, DNA, RNA) are stored in an online database known as Protein Data Bank (PDB) [1]. Each .pdb file is broadly divided into two parts: descriptive content and coordinate information. While coordinate information is well structured and easy to infer, most of the descriptive content is hidden as a running text. Looking into details, we find that every .pdb file of PDB is presented as an ASCII text file where each line consists of 80 columns and is terminated by an end-of-line indicator. In the header section, the information about protein, experimental details and additional structural attributes are mentioned. Every record starts with specific predefined keywords, however the information contained in them are predominantly textual in nature. Some records like ATOM, HETATM, MODEL, etc. are structured and follow strict rules facilitating easy parsing of its data. The file format itself is very primitive and with lot of limitations for automatic inferences. An attempt has been made to overcome some of the limitations in the .pdb file format by introducing the .mmcif format which is again very bulky and exhaustive. A recent advance has been made to introduce a binary file format as .mmtf [2] along with its parsers, which has also been integrated in this package. Our focus is to parse and load the .pdb, .mmcif or .mmtf file in our abstract data structure such that automatic information retrieval becomes effective and efficient.

A number of programs were developed as PDB Parsers and PDB based tools. Some early parsers are outdated due to either poor design or lack of maintenance such as BioC++[3], dsr-pdb[4] and PDBlib [5]. Python based programs like Biopython PDB Parser [6] does not provide the speed as needed for the present set of problems due to dynamic types and the use of an interpreter. With C++, we have much control on the code optimization to achieve high performance and speed. Many PDB parsers provide only basic data types and representations for a protein while others focus on specific applications. Victor C++ library [7] provides large-scale methods for structure manipulation which makes the code and data structure extra-detailed and heavy. ESBTL [8] is a library which allows handling of PDB data and also provides computational geometry methods. BALL [9] is another extensive library for algorithms. BioLib is a C programming library for loading and manipulating macromolecular structures [10]. Finally, most of the parsers focus on the coordinate data explicitly and none describe handling of the header information especially REMARK

section present in the .pdb file. However BiopLib [10] does provide substantial parsing of the header information which is generally neglected by other parsers. Haskell based library hPDB

The need of analysing protein structures i.e. their .pdb files is a high-end requirement of present era for making new drugs, analysing protein interactions, studying MD simulation intermediates and such other structure based studies. Now, .pdb files for macromolecular structures especially big ones are already disabled by RCSB and in this context it is very critical to have software platforms which could also read .mmcif and .mmtf files with good efficiency. Nevertheless, the existing structural techniques are still heavily based only on the .pdb file format.

2. Materials and Methods

2.1. Data set

For testing the features of our program, molecular structures for the test data set were selected from multiple sources:
1. The experimentally derived macromolecular structural data as available on RCSB-PDB was downloaded in the format .pdb and .mmCIF.
2. Intermediate MD simulation .pdb files.
3. Computationally modelled protein structures.
4. Macromolecular complexes resulting after molecular docking.

2.2. Tokenization of input file

Despite having format specifications for .pdb files, formatting deviations are often observed, as the .pdb file format is merely a plain ASCII .txt file. The lack of rigidity during creation and modification of .pdb files thus allows illegal token inclusions. This hinders the application of any automated software for data extraction based upon the guidelines specified for pdb format. Hence, the first step of our software is the tokenization. The process of tokenization is where the .pdb file data is broken into so called tokens or fragments. The lexical patterns in the file are identified using Flex [11] and the rule section of flex code is used to define the tokens. We utilize start conditions in Flex where we can conditionally switch between rules to tokenize the sections like REMARK, ATOM, HETATM, CONNECT, etc. In our program, we have two modes of tokenization: strict and casual. In casual mode, each record is read as one line per token and fed to next program for column-wise processing. It is difficult for computer programs to differentiate the identity of the atom without proper naming. Using strict mode in the MAT parser we have identified all the atoms that use non-standard atom names or disallowed characters in the file. Due to column wise distribution of the attributes in the .pdb file, there may be lack of delimiters and the values are compelled to merge. Hence, we use regular expressions instead of any delimiter for each column of the ATOM and HETATM record. As there are many sections in the .pdb files, we require various set of grammar rules for tokenization of each section which are elaborated in

the Supplementary Tables S1 and S2.
The parser can be called in the following way, where the variable `optarg` is the file name for any of the .pdb/.mmcif/.mmtf file:

```
File::m_get_instance()->m_parse_file(optarg);
```

When you want to use the PDB ID of the structure and download it through the program, you can call the following function; here `optarg` being PDB ID:

```
File::m_get_instance()->m_download_pdb(optarg);
File::m_get_instance()->m_parse_file();
```

2.3. Syntax Tree formation

The individual tokens generated by the program are analysed and used by Bison (a parser generator) to identify the variables and add them to the internal data structure. To load data from PDB file in the data structure, we divide the file into various sections, start conditions from Flex program categorizes the section of the PDB file. PDB basically contain static representation of atoms in terms of X, Y and Z coordinates and header information related to that macromolecular structure. A set of coordinates specifies the location of a point as an analogy for atom. However, we need to know the complete description of a single atom to describe its features that are mentioned in the .pdb file. The complete file is parsed by MAT PDB parser and the information (including atoms and header information) is stored in our data structure designed specifically for atoms. In our program, we avail the *Qt* library (*https://www.qt.io/*) for its container classes. The individual tokens generated by the program are analysed and used by Bison to identify the variables and add them to the internal data structure. Figure 1 explains the syntax tree formation and how components of the PDB are read and stored in our program.

2.4. Data structures

2.4.1. Atom bucket

Atom bucket is constructed as an elementary abstract data structure using QList container. This data structure is simplistic in nature, all the atom(s)/ hetatm(s) are deposited in this primary bucket. The atom bucket class uses singleton design pattern which has only one instance. It has a global scope of accessibility. Atom is the smallest representation of protein structure in the PDB file; all the information and the properties that are associated to them are given for individual atom. Lastly, all the functions are which are available for a QList can also be used directly with the Atom bucket. On the whole, the atom bucket contains information on each of the protein points described in the .pdb file.

2.4.2. Structural Hierarchy

For each level of structural hierarchy: atom/hetatm, residue and chain; we have a different bucket data structure which provides ease of accessibility to the information as they provide iterators for each level. Furthermore, every element of the level is represented as an object typed with its respective class. Since

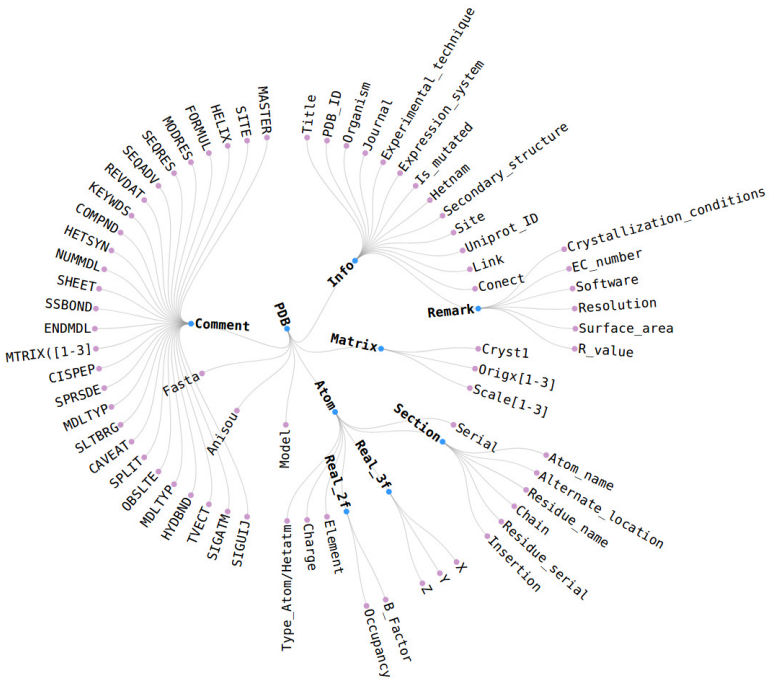

Fig 1: *Syntax tree. This shows various components of the .pdb file. These are broken and identified into tokens and the macromolecular structure is maintained in our data structure for accessibility of atom/hetatm and accompanying information. The .pdb file attributes, info, remark, matrix, atom, model, fasta, anisou can be accessed using member functions provided. Some of data which is read but commented out is shown in comment section*

every level-element has been treated as a class, their properties, members and methods are also grouped separately. In doing so, we take in mind that all the groups at each level points to the same data. Therefore we can access, modify and use the information in atom-wise, residue-wise, chain-wise fashion. These designed buckets together make up the core data structure to handle the entire information of a protein system. Internally data structure looked like:

```
QList<Atom*> m_Atom_Bucket;
QList<Residue*> m_Residue_Bucket;
QList<Chain*> m_Chain_Bucket;
```

And these are the iterator types to access these information

```
Atom_iterator
Residue_iterator
Chain_iterator
```

2.4.3. Bonded Graph

A graph G(V,E) is defined to store the molecular structures, where V is the set of atoms and E is the set of covalent bonds among the corresponding atoms. The covalent bond information is stored as adjacency lists [12]. The interpretation of a chemical molecular structure is best represented by this data structure. It features identification of missing atoms in the structure and visualization of an atom in a structure even with multiple occupancies. For example, we can create a graph for all atoms in Atom Bucket as follows:

```
BondedGraph *m_Bonds
  = new BondedGraph((AtomRoot::m_get_instance())->m_get_atom_bucket());
```

2.5. Protein space partitioning

2.5.1. KD-tree

k-dimensional (kD) tree is a binary tree with each node having maximum two children [13]. In case of protein structure, it is a 3D spatial tree which stores the 3D atom coordinates of a protein. It is used for searching the atomic nearest neighbors (NN) with respect to a query atom [14]. NN searches are performed very often while analyzing the structural aspects of the protein. kD-tree has been extensively used in related software such as ESBTL, BALL, Victor C++ library, etc.

We have iterated over the atom buckets to divide the protein space and store in kD tree. The root node represents the whole data space, at each level of the tree, the data is partitioned into two halves by the median point. This step is performed recursively for each dimensions X, Y and Z in the order of alternate turns. The median is selected from the list considering one dimension at a time by using quickselect algorithm which splits the dimension into two, for instance when a median is selected from X-axis the space is divided by the YZ plane. The left subtree would have values of the corresponding dimension greater than the median and the right subtree would have lesser values. This method creates a balanced tree which makes the search related to nearest neighbors fast. Figure 2 shows a simplistic example of a small kD-Tree. A few methods for NN search include k-nearest-neighbor, orthogonal search, circular search helping in the variety of biological problems. Algorithm 1 is for spherical search for kD-Tree radius search in atomic structures. It can be used and called in the steps mentioned below:

```
KD_Tree* kd_tree1 = new KD_Tree(true);
```

Here, the variable `query_node` is the query point of type `Atom` and `range` is the radial range for the neighbour search to be given in Å.

```
Atom* query_node = new Atom(88457, "C", ".", "TYR", "A", 188, ".",
  138.719, 33.258, 41.444, 1.00, 13.43, "C");
```

```
kd_tree1->m_set_querynode(query_node);
QList<KD_Node*> KD_neighbors = kd_tree1->m_get_sphere_nn(query, range);
kd_tree1->m_print_list( KD_neighbors );
```

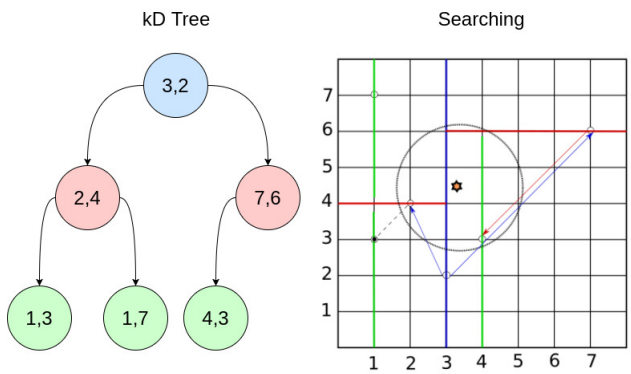

Fig 2: *kD-Tree. The example of constructed kD-tree is shown with example of a 2-dimensional dataset. Searching for nearest neighbors of a query point(yellow star) is started at the root node of kD-Tree which is shown in blue. A recursive approach is then followed to check all the nodes and prune unnecessary sections.*

2.5.2. Octree

Octree is a space partitioning tree with each node having exactly eight children. The root node contains the whole dataset and represents the orthogonal cuboid bounding box (voxel) for the data points as represented in Figure 3. These trees have been popularly used in the dynamic data sets and hence will be useful to deal with MD simulation data. The advantage of this tree is that insertion of the new data point does not need the tree rebalancing as each point finds its positions in this hierarchical spatial tree. For its construction, we first get the bounding box specifications for the data set i.e. the *min* and *max* values for X, Y and Z defining the enclosure for all the atom points. Then the space is divided recursively until it reaches the smallest node dimension possible i.e. voxel, as specified or until it contains single point in the cell. For n points, the insert function is called n times for insertion of each point in the octree data structure as explained in the Algorithm 2.

We have implemented spherical search which is mostly used for molecular structures. The neighbor search is explained in Algorithm 3.

Octree contruction and Nearest-Neighbor searching can be used in following way. First we are giving an example of calling a function to ignore alternate locations of same atom:

```
Rule::m_get_instance()->m_keep_highest_occ();
Octree* octree1 = new Octree();
```

Algorithm 1: kD-Tree spherical search

Input: *root*: root node , *query*: point , *r*: radius , *axis*: splitting dimension
Output: *neighborlist*: list of all neighboring atoms

```
function SPHERICAL_SEARCH(root, query, r, axis)
    if root = NULL then
        return
    end if
    if IS_INSIDE(root,query,r) ← true then
        neighborlist ← root
    end if
    if IS_LEAF(root) ← true then
        return
    end if
    Let left and right be children of root
    if left[axis] > query[axis] then
        SPHERICAL_SEARCH(left, query, r, axis + 1)
        if AXIS_DISTANCE(root, query, axis) ≤ r then
            SPHERICAL_SEARCH(right, query, r, axis + 1)
        end if
    end if
    else
    if left[axis] < query[axis] then
        SPHERICAL_SEARCH(right, query, r, axis + 1)
        if AXIS_DISTANCE(root, query, axis) ≤ r then
            SPHERICAL_SEARCH(left, query, r, axis + 1)
        end if
    end if
    return neighborlist
end function
```

Algorithm 2: Octree construction

```
Input: root: root voxel , a: atom
function INSERT_ATOM(root, a)
    if IS_LEAF(root) ← true and COUNT_ATOMS(root) < capacity then
        root ← a
        return
    end if
    if IS_LEAF(root) ← true and COUNT_ATOMS(root) = capacity then
        SUBDIVIDE(root)
        for all a atoms of root do
            INSERT_ATOM(root, a)
        end for
    end if
    for all voxel of root do
        if CONTAINS(voxel, a) ← true then
            INSERT_ATOM(voxel, a)
        end if
    end for
end function
```

Algorithm 3: Octree spherical search

```
Input: root: root voxel , query: point , r: radius
Output: neighborlist: list of all neighboring atoms
function SPHERICAL_SEARCH(root, query, r)
    if IS_LEAF(root) ← true then
        for all atom ∈ root do
            if DISTANCE(atom,query) < r ← true then
                neighborlist ← root
            end if
        end for
    end if
    for all voxel of root do                                  ▷ 8 voxels
        if OVERLAPS(voxel, query, r) ← true then
            SPHERICAL_SEARCH(voxel, query, r)
        end if
    end for
    return neighborlist
end function
```

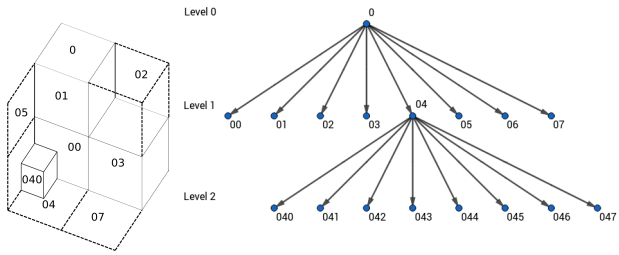

Fig 3: *Octree nodes. The outer bounding box is set as the root node shown. Each spatial division splits the volume into its constituent eight separate volumes. 1st division is shown in and in this level, we see the volume for 6th node (06) is not visible in the 3D cubic display. Each voxel is also uniquely represented and encoded as shown to identify its location.*

Here again, the variable the variable `query_node` is the query point of type `Atom` and `range` is the radial range for the neighbour search to be given in Å.

```
QList<Atom*> octree_neighbors =
  octree1->m_get_radius_neighbors(query, double(range));
octree1->m_print_list( octree_neighbors );
```

3. Results

3.1. Computational efficiency and speed comparison

For evaluating the performance and speed of our parser we have used the whole PDB database. Some files were incompatible due to bad formatting of the files (see section 3.2); however with regression testing and improvements in the code we are able to load all of the .pdb files successfully in our data structure with maximum accuracy. We also reported files which were non-standard with the pdb format. We have compared the runtime performance and efficiency of our software shown in Figure 4 with popular parsers. The benchmarks were made for MAT parser with programs: Pymol, BioJulia, BioPython, BioPerl, Victor, hPDB and ESBTL. It was observed to be fastest among all other packages that we tested for. The accuracy is based on the number of error occurrences while parsing the test data set as well as the ability to detect total number of residues in the input file correctly plotted in the graph Figure 5. This graph shows the number of files where the programs were either not able to read all the residues in the file or they were not able to read the whole file.

The algorithms of nearest neighbor search for our data structures were compared and tested for their accuracy and speed using Google Test [15]. It was found that both the kD-tree and Octree are fairly faster than simple linear search. The neighbors were exactly accurate in all the test cases. Furthermore, Figure 6 shows speed comparison of different voxel capacities of Octree and

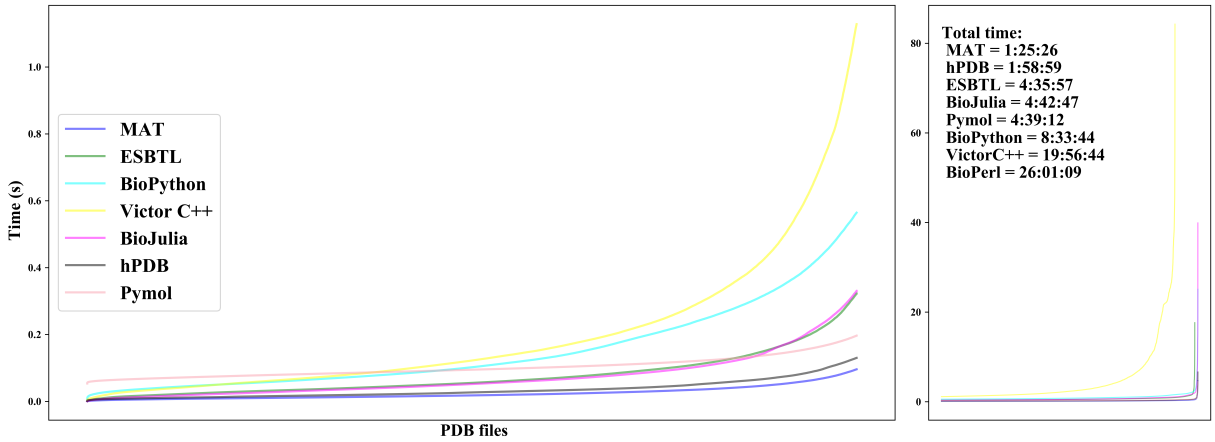

Fig 4: *Speed comparison. The chart illustrates the speed of related software showing their relative performances on a UNIX based 64-bit system with Intel ® Core^{TM} i7-3770 CPU @ 3.40GHz × 8 with 8GB RAM. Total time taken for parsing the whole database is given in the format HH:MM:SS.*

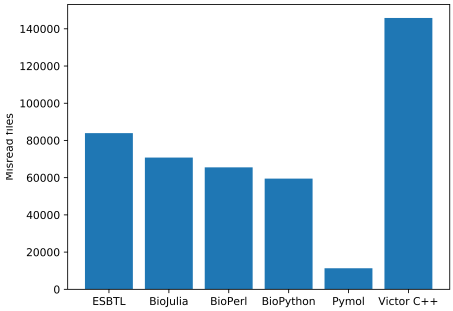

Fig 5: *Parsing total atoms in a file. This graph shows the number of instances of misread files. MAT Parser and hPDB were completely accurate and did not show any errors or incomplete parsing.*

kD-Tree, the latter being fastest. We found that voxel size of 3 was optimum and if we start to increase the voxel capacity more than 32, we see a noticeable difference in the speed. We have used kD-Tree extensively in finding networks of non-covalent interactions such as salt bridge and hydrogen bonds. It is advantageous to use this data structure especially where there is a requirement of frequent neighborhood search queries multiple times, for example, interaction studies. Secondly, it is stated that kD-Tree can only be used with static data set. If we consider semi-dynamic or dynamic data sets, it is advantageous to use Octree. Therefore for molecular dynamics and energy minimization, Octree can be used. It is so because the updation in this data structure does not involve

altering the whole data set but it only refer to positional changes inside the local reference frame of the bounding box. The benchmarks are given in the supplementary material.

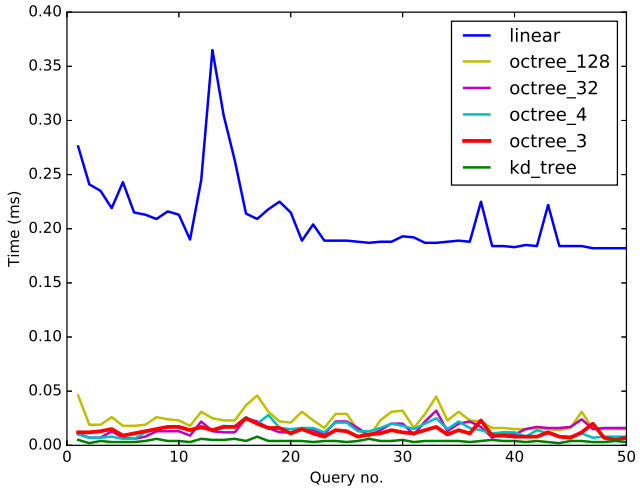

Fig 6: *Comparison of kD-Tree and Octree. kD-Tree is most efficient for neighbor searching. Octree is slightly slower, for dynamic or semi dynamic data sets of macromolecules voxel capacity of 3 to 32 is best but it could be improvised based on the nature of data set.*

3.2. Exception handling

The casual-parse mode in our parser does not check any passing criteria for handling individual attributes of ATOM/HETATM. However, strict-parse mode ensures only those attributes which fulfil the PDB format guidelines for naming and representation of the values. Many faulty .pdb files can be validated using this method at parsing stage. We were able to detect a few such .pdb files using our parser, some of them are also reported to the RCSB database administrators. There are other faults which we have tried to overcome; labeled as PDB format faults, structural faults and uncorrectable faults.

1. We have attempted to correct and overcome some of the PDB format limitations in our program. We have parsed all the important header section data and stored in our data structure which is easily accessible. Using various ligand names for single ligand in PDB file highly discouraged, for example water molecule is found as H2O, WAT, HOH, etc but it is simplified into a single name as HOH.
2. The structural faults are identified and reported; showing un-natural dihedral angles, giving contradictory UNIPROT and structural sequence alignments, removing solvents which are present because of experimental techniques. Graph data structure specially designed to represent covalent bonds of only highest occupancy atoms and reports missing atoms.

3. We are still left with many faults which are uncorrectable due to errors that stem from experimental data, for instance, the electron density might not match with the coordinates. Plenty of PDB format limitations are intrinsic, these have been addressed in newer formats like mmCIF [17] and mmTF.

4. Discussion

4.1. Extension for biochemical applications

The developed parser can be used in different aspects of macromolecular system analysis such as: sequence and structure comparison, surface atoms detection, active water molecule identification and salt bridge determination etc. In addition to this, the program is also able to detect the active site in a protein and identify the presence of any metal and its interactions.

We have used this parser in our lab for analyzing numerous macromolecular structures, virtual screening and docking results, non-covalent interaction studies, .pdb files from molecular dynamics simulations and other structure based data-set studies. In a case study (see supplementary material), we are able to identify stabilizing and destabilizing salt bridges from two homologous mesophilic and thermophilic α-carbonic anhydrase[18]. Studying these types of structural factors like salt bridges, surface residues, and active site water molecules for any macromolecular system is much more simple, fast and error-free.

The protein's coordinate, bonding; dihedral and other such structural data can be used to study different structural and functional aspects of it. The new parser, which as discussed so far, provides an innovative, faster and necessary platform to prepare the data in structured form for further analysis. Using which we have incorporated various macromolecular analytical tools at one place. The motivation of our work is to provide a integrative tool for in-silico studies that range from getting FASTA sequence to getting active site information.

4.2. Applicability and scope

The user interface provides physical abstractions (e.g. atoms, bonds, molecules) of the data that could be easily manipulated by the user.Having active and growing international developer community, ongoing and future developments will improve performance further, introduce transparent parallelization schemes to utilize multi-core and GPU systems efficiently, and interface with high performance data analytics algorithms [19]. We think that this will be a major step in bringing forward C++ language for biology and it being used for open source for its performance and therefore contributing in improving the popularity and current lack of any ongoing projects.

We, strongly recommend that the format of the .pdb file should be reconsidered for making it software-friendly so as to improve the performance of the software as well as to improve the digital readability of the format. A more detailed and strict organization of the attributes is called for. These attributes should be separated by something like space aiding in the distinction of the values which are currently merged in many cases and, therefore, become vague. Now, for solving this problem, we have developed a efficient parser which can arrange all the important data in a well-mannered form as being a fastest program among various others. In addition to the task of parsing of the .pdb, .cif and .mmtf files for working with static structures for molecular analysis; we also wish to add dynamic molecular structural analysis in future.

4.3. Availability

The source is available on bitbucket at https://bitbucket.org/gazalk/pdb_parser/. It is based on C++11 and requires Qt >5.6 and MsgPack for C to be pre-installed. MAT is available as a web-service at http:/mat.iitr.ac.in/.

5. Supplementary Material

See Supplementary material for grammar rules and grammar table in Tables S1 and S2 for tokenization of each section in the parser. The benchmarks for hydrogen bond finding are shown in Figure 1. Construction and searching of the data structures, kD-Tree and Octree is given in Figure 2 and Figure 3 of the Supplementary data. The case study of α-carbonic anhydrase to mutation design in Figure 4 and Figure 5 while showing non-covalent interactions in Table S3.

6. Funding

This research was supported by the Department of Biotechnology (DBT), Ministry of Science and Technology, Government of India [Grant number DBT/2015/IIT-R/325] to G.K.

7. Acknowledgements

The work was partially conducted in the context of the Bioinformatics Resources and Applications Facility (BRAF), C-DAC, Pune, India. Institute Computer Center, IIT-Roorkee and Bioinformatics Facility, Department of Biotechnology, IIT-Roorkee granted the provision of computational facilities and support.

[1] F. C. Bernstein, T. F. Koetzle, G. J. Williams, E. F. Meyer, M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi, M. Tasumi, The Protein Data Bank: a computer-based archival file for macromolecular structures., Journal of molecular biology 112 (3) (1977) 535–542.

[2] A. R. Bradley, A. S. Rose, A. Pavelka, Y. Valasatava, J. M. Duarte, A. Prlić, P. W. Rose, Mmtf—an efficient file format for the transmission, visualization, and analysis of macromolecular structures, PLOS Computational Biology 13 (6) (2017) 1–16. doi:10.1371/journal.pcbi.1005575.
URL https://doi.org/10.1371/journal.pcbi.1005575

[3] Z. Honguy, J. Michael, M. Parag, C++ computational libraries for bioinformatics, version 0.3 (2006).
URL http://biocpp.sourceforge.net/

[4] R. Daniel, A simple c++ pdb reader (2004).
URL http://graphics.stanford.edu/~drussel/pdb/index.html

[5] W. Chang, I. Shindyalov, C. Pu, P. Bourne, Design and application of pdblib, a c++ macromolecular class library, Computer applications in the biosciences : CABIOS 10 (6) (1994) 575–586. doi:10.1093/bioinformatics/10.6.575.

[6] T. Hamelryck, B. Manderick, Pdb file parser and structure class implemented in python, Bioinformatics 19 (17) (2003) 2308–2310. doi:10.1093/bioinformatics/btg299.

[7] L. Hirsh, D. Piovesan, M. Giollo, C. Ferrari, S. C. Tosatto, The victor c++ library for protein representation and advanced manipulation, Bioinformaticsdoi:10.1093/bioinformatics/btu773.

[8] S. Loriot, F. Cazals, J. Bernauer, Esbtl: efficient pdb parser and data structure for the structural and geometric analysis of biological macromolecules, Bioinformatics 26 (8) (2010) 1127–1128. doi:10.1093/bioinformatics/btq083.

[9] A. Hildebrandt, A. K. Dehof, A. Rurainski, A. Bertsch, M. Schumann, N. C. Toussaint, A. Moll, D. Stöckel, S. Nickels, S. C. Mueller, H.-P. Lenhof, O. Kohlbacher, BALL–biochemical algorithms library 1.3., BMC bioinformatics 11 (1) (2010) 531.

[10] C. T. Porter, A. C. Martin, Bioplib and bioptools—a c programming library and toolset for manipulating protein structure, Bioinformatics 31 (24) (2015) 4017–4019. doi:10.1093/bioinformatics/btv482.

[11] A. V. Aho, R. Sethi, J. D. Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[12] D. Bonchev, Chemical Graph Theory: Introduction and Fundamentals, Chemical Graph Theory, Taylor & Francis, 1991.
URL https://books.google.co.in/books?id=X0AG7HhiccoC

[13] J. L. Bentley, Multidimensional binary search trees used for associative searching, Commun. ACM 18 (9) (1975) 509–517. doi:10.1145/361002.361007.

[14] J. H. Friedman, J. L. Bentley, R. A. Finkel, An algorithm for finding best matches in logarithmic expected time, ACM Trans. Math. Softw. 3 (3) (1977) 209–226. doi:10.1145/355744.355745.

[15] Google, Google's c++ test framework (August 2016).
URL https://github.com/google/googletest

[16] M. Heinig, D. Frishman, Stride: a web server for secondary structure assignment from known atomic coordinates of proteins, Nucleic Acids Research 32 (suppl 2) (2004) W500–W502. doi:10.1093/nar/gkh429.
URL http://dx.doi.org/10.1093/nar/gkh429

[17] S. R. Hall, F. H. Allen, I. D. Brown, The crystallographic information file (cif): a new standard archive file for crystallography, Acta Crystallographica Section A 47 (6) (1991) 655–685. doi:10.1107/S010876739101067X.
URL https://doi.org/10.1107/S010876739101067X

[18] S. K. Bharatiy, M. Hazra, M. Paul, S. Mohapatra, D. Samantaray, R. C. Dubey, S. Sanyal, S. Datta, S. Hazra, In silico designing of an industrially sustainable carbonic anhydrase using molecular dynamics simulation, ACS Omega 1 (6) (2016) 1081–1103. doi:10.1021/acsomega.6b00041.
URL http://dx.doi.org/10.1021/acsomega.6b00041

[19] J. Qiu, S. Jha, A. Luckow, G. C.Fox, Towards hpc-abds: An initial high-performance big data stack (2014).

[20] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-oriented Software, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[21] M. B. Kennel, KDTREE 2: Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional Euclidean space, ArXiv Physics e-printsarXiv:physics/0408067.

[22] R. Chowdhury, D. Beglov, M. Moghadasi, I. C. Paschalidis, P. Vakili, S. Vajda, C. Bajaj, D. Kozakov, Efficient maintenance and update of nonbonded lists in macromolecular simulations, Journal of Chemical Theory and Computation 10 (10) (2014) 4449–4454, pMID: 25328494. doi:10.1021/ct400474w.

[23] M. Cieslik, Z. Derewenda, C. Mura, Abstractions, Algorithms and Data Structures for Structural Bioinformatics in PyCogent, ArXiv e-printsarXiv:1407.5218.

[24] S. Kumar, R. Nussinov, Salt bridge stability in monomeric proteins1, Journal of Molecular Biology 293 (5) (1999) 1241 – 1255. doi:http://dx.doi.org/10.1006/jmbi.1999.3218.

[25] S. Kumar, R. Nussinov, Relationship between ion pair geometries and electrostatic strengths in proteins, Biophysical Journal 83 (3) (2002) 1595 – 1612. doi:http://dx.doi.org/10.1016/S0006-3495(02)73929-5.

[26] J. E. Donald, D. W. Kulp, W. F. DeGrado, Salt bridges: Geometrically specific, designable interactions, Proteins: Structure, Function, and Bioinformatics 79 (3) (2011) 898–915. doi:10.1002/prot.22927.

[27] M. Paul, M. Hazra, A. Barman, S. Hazra, Comparative molecular dynamics simulation studies for determining factors contributing to the thermostability of chemotaxis protein "CheY", Journal of Biomolecular Structure and Dynamics 32 (6) (2014) 928–949. doi:10.1080/07391102.2013.799438.

[28] J. Levine, Flex & Bison, 1st Edition, O'Reilly Media, Inc., Sebastopol, CA, 2009.

[29] B. DJ, T. JM, Ion-pairs in proteins, Journal of Mol Biol. 168 (4) (1983) 865–885.

[30] R. W. W. Hooft, G. Vriend, C. Sander, E. E. Abola, Errors in protein structures, Nature 381 (1996) 272. arXiv:physics/0408067.

[31] S. Hazra, S. Ort, M. Konrad, A. Lavie, Structural and kinetic characterization of human deoxycytidine kinase variants able to phosphorylate 5-substituted deoxycytidine and thymidine analogues,, Biochemistry 49 (31) (2010) 6784–6790, pMID: 20614893. doi:10.1021/bi100839e.

[32] S. Kumar, R. Nussinov, Fluctuations in ion pairs and their stabilities in proteins, Proteins: Structure, Function, and Bioinformatics 43 (4) (2001) 433–454. doi:10.1002/prot.1056.

[33] J. Luo, J.-D. Maréchal, S. Wärmländer, A. Gräslund, A. Perálvarez-Marín, *In Silico* analysis of the apolipoprotein e and the amyloid β peptide interaction: Misfolding induced by frustration of the salt bridge network, PLoS Comput Biol 6 (2) (2010) 1–7. doi:10.1371/journal.pcbi.1000663.

[34] E. A. Mackenzie, L. S. Klig, Computational modeling and in silico analysis of differential regulation of myo-inositol catabolic enzymes in cryptococcus neoformans, BMC Molecular Biology 9 (1) (2008) 1–9. doi:10.1186/1471-2199-9-88.

[35] S. G. Kurz, K. A. Wolff, S. Hazra, C. R. Bethel, A. M. Hujer, K. M. Smith, Y. Xu, L. W. Tremblay, J. S. Blanchard, L. Nguyen, R. A. Bonomo, Can inhibitor-resistant substitutions in the mycobacterium tuberculosis β-lactamase blac lead to clavulanate resistance?: a biochemical

rationale for the use of β-lactam-β-lactamase inhibitor combinations, Antimicrobial Agents and Chemotherapy 57 (12) (2013) 6085–6096. doi:10.1128/AAC.01253-13.

[36] S. Costantini, G. Colonna, A. M. Facchiano, Esbri: a web server for evaluating salt bridges in proteins, Bioinformation 3 (2) (2008) 137–138.

[37] C. Jackins, S. Tanimoto, Oct-trees and their use in representing three-dimensional objects, Computer Graphics and Image Processing 14 (3) (1980) 249–270. doi:10.1016/0146-664X(80)90055-6.

[38] W. Wang, S. M. McKinnie, M. Farhan, M. Paul, T. McDonald, B. McLean, C. Llorens-Cortes, S. Hazra, A. G. Murray, J. C. Vederas, G. Y. Oudit, Angiotensin-converting enzyme 2 metabolizes and partially inactivates pyrapelin-13 and apelin-17novelty and significance, Hypertension 68 (2) (2016) 365–377. doi:10.1161/HYPERTENSIONAHA.115.06892.
URL http://hyper.ahajournals.org/content/68/2/365

[39] Schrödinger LLC, The PyMOL molecular graphics system, version 1.8 (November 2015).

[40] T. Nakane, Glmol – molecular viewer on webgl/javascript, version 0.47 (2012).
URL http://webglmol.sourceforge.jp/index-en.html

[41] B. Hess, C. Kutzner, D. van der Spoel, E. Lindahl, Gromacs 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation, Journal of Chemical Theory and Computation 4 (3) (2008) 435–447, pMID: 26620784. arXiv:http://dx.doi.org/10.1021/ct700301q, doi:10.1021/ct700301q.
URL http://dx.doi.org/10.1021/ct700301q

[42] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, M. Karplus, Charmm: A program for macromolecular energy, minimization, and dynamics calculations, Journal of Computational Chemistry 4 (2) (1983) 187–217. doi:10.1002/jcc.540040211.
URL http://dx.doi.org/10.1002/jcc.540040211

[43] D. A. Pearlman, D. A. Case, J. W. Caldwell, W. S. Ross, T. E. Cheatham, S. DeBolt, D. Ferguson, G. Seibel, P. Kollman, Amber, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules, Computer Physics Communications 91 (1) (1995) 1 – 41. doi:https://doi.org/10.1016/0010-4655(95)00041-D.
URL http://www.sciencedirect.com/science/article/pii/001046559500041D

[44] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, K. Schulten, Scalable molecular dynamics with namd, Journal of Computational Chemistry 26 (16) (2005) 1781–1802. doi:10.1002/jcc.20289.
URL http://dx.doi.org/10.1002/jcc.20289

[45] M. Karplus, J. A. McCammon, Molecular dynamics simulations of biomolecules, Nature Structural Biology 9 (2002) 646, review Article.
URL http://dx.doi.org/10.1038/nsb0902-646

[46] M. C. Zwier, L. T. Chong, Reaching biological timescales with all-atom molecular dynamics simulations, Current Opinion in Pharmacology 10 (6) (2010) 745 – 752, endocrine and metabolic diseases/New technologies - the importance of protein dynamics. doi:https://doi.org/10.1016/j.coph.2010.09.008.
URL http://www.sciencedirect.com/science/article/pii/S1471489210001463

[47] M. Cascella, M. Dal Peraro, Challenges and perspectives in biomolecular simulations: From the atomistic picture to multiscale modeling, CHIMIA International Journal for Chemistry 63 (1-2) (2009) 14–18. doi:doi:10.2533/chimia.2009.14.
URL http://www.ingentaconnect.com/content/scs/chimia/2009/00000063/F0020001/art00003

[48] F. R. Salsbury, Molecular dynamics simulations of protein dynamics and their relevance to drug discovery, Current Opinion in Pharmacology 10 (6) (2010) 738 – 744, endocrine and metabolic diseases/New technologies - the importance of protein dynamics. doi:https://doi.org/10.1016/j.coph.2010.09.016.
URL http://www.sciencedirect.com/science/article/pii/S1471489210001542

[49] J. Gosling, H. McGilton, The java language environment a white paper, Tech. rep. (1996).

[50] D. R. Roe, T. E. Cheatham, Ptraj and cpptraj: Software for processing and analysis of molecular dynamics trajectory data, Journal of Chemical Theory and Computation 9 (7) (2013) 3084–3095, pMID: 26583988. arXiv:http://dx.doi.org/10.1021/ct400341p, doi:10.1021/ct400341p.
URL http://dx.doi.org/10.1021/ct400341p

[51] M. Bostock, V. Ogievetsky, J. Heer, D3: Data-driven documents, IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis).
URL http://vis.stanford.edu/papers/d3

[52] J. D. Hunter, Matplotlib: A 2d graphics environment, Computing In Science & Engineering 9 (3) (2007) 90–95. doi:10.1109/MCSE.2007.55.

[53] Z. S. Hendsch, B. Tidor, Do salt bridges stabilize proteins? a continuum electrostatic analysis, Protein Science 3 (2) (1994) 211–226. doi:10.1002/pro.5560030206.
URL http://dx.doi.org/10.1002/pro.5560030206

[54] M. J. Gajda, hpdb–haskell library for processing atomic biomolecular structures in protein data bank format, BMC research notes 6 (1) (2013) 483.