Biological Object Based Software (BOS): An Integrative Biological Programming Environment

Burra V L S Prasad*
Professor, K L University, Green Fields,
Vaddeswaram, Guntur, Andhra Pradesh -522502
*pburra@kluniversity.in

Received: *08.05.2019,* ***Accepted:*** *20.05.2019*

Abstract

The state of biological research is evolving and transitioning from static data/information-based modelling approaches toward knowledge-based modelling of bio-dynamics. The biological complexity involves various interlinked processes and products of evolution. The further layers of complexity overlaid can be attributed to the dynamics and varied rates of changes to the known and emerging parameters. Computational advancements and technologies are now inextricable part of today's biological research helping the challenge of de-convoluting discovery and characterization of the complex parameters. As a strategic step, last two decades has seen creation of huge data repositories capturing as much data and information as possible. The next strategic step would be to develop comprehensive and integrative methods to understand and re-construct the complexity in the most reliable way which eventually will lead to biological applications.

Biological software has been co-evolving in parallel to the discoveries of the biological complexity. Hence, there is huge scope and demand for more efficient biological software applications and tools, especially post human genome sequencing. Biological Object based Software (BOS) - a *biological programming environment*, is one such attempt to equip the researchers with seamless integration, efficient extraction and effortless analysis of the data from various biological databases and algorithms, be it genomics, structure analysis, literature mining or simultaneous use of all. Reusability, flexibility, extensibility and integration have been the primary principles on which the software solution is designed and developed. BOS has features that enable automation in implementing novel bio-computational workflows, while reducing the time significantly.

Key words - Next generation sequencing, Biological Object based Software, *Genomics,* BioADT

Introduction

Today, on careful examination, based on the type of biological data the biological space is naturally getting sub-classified into *genomics* (genome sequence data), *transcriptomics* (mRNA, EST, expression data), *proteomics* (protein sequence, data), *structural genomics* (3D structure data), *interactomics* (biological molecule interaction data), *metabolomics* (metabolite, substrate, and small molecule data), *fluxomics* (data of rates of metabolic reactions in cell) and *phenomics* (organism phenotype data). Since inception of bioinformatics, every attempt has been and is still being made to develop necessary technologies in order to capture all the data and information pertaining to each of the above sub-classes. This resulted in many technological innovations which led to huge databases and repositories and many

software applications / solutions (Galperin and Fernández-Suárez, 2012).

Next generation sequencing (NGS) technologies will continue to add sequence data at an unprecedented rate challenging the existing computer infrastructure as well as software applications (Richter and Sexton, 2009). It is the fastest growing and the most lucrative segment in the genomics space. These technologies will revolutionize the medical research in academic laboratories, bio-pharma and the applied markets influencing various fields of science such as cancer research, bio-fuels, marine sciences, livestock research, agricultural and veterinary research. The global NGS market was valued at $842.5 million in the year 2011, growing at a compound annual growth rate of 22.7% from 2012 to 2016 (http://www.marketsandmarkets.com/Market-Reports/next-generation-sequencing-ngs-technologies-market-546.html). The current size of biological data is approximated to be in Petabytes (10^6 Gigabytes). With NGS in the offing, the total size is expected to surpass Zettabytes of data (10^{12} GigaBytes) in next five years. The solution that would address the issue discussed and other requirements seem to be to design and develop an efficient biology specific data and knowledge sourcing platform which should be comprehensive, flexible and inclusive.

Over the period of time, many software applications are developed covering various tasks, ranging from simple automation scripts, data mining, curation, annotation, analysis, representation to automated scientific publications and technical documentations. Further, these software applications range from handling single/simple task to complex workflows/data flows. Based on the number of tasks and the way the tasks being served, the software applications are being called by different names – a) the applications which handle single task are generally referred as tools. For example: CLUSTALX (Larkin *et al.,* 2007), STAMP (Russell and Barton, 1992), MOLSCRIPT (Kraulis, 1991) and so on. b) The applications which provide a common interface and enable the user to perform multiple tasks are being referred as Integrative Analytic Platforms or embedded systems. For example: Tripos-Sybyl (www.tripos.com), Accelrys-InsightII (www.accelrys.com), EMBOSS (Rice *et al.,* 2000), CCP4 (Winn *et al.,* 2011) and others. c) There are other applications, generally referred as application programming interfaces (APIs), such as BioJava (Holland *et al.,* 2008), BioPerl (Stajich *et al.,* 2002), BioPython (Cock *et al.,* 2009), Bio++ (Dutheil *et al.,* 2006) and others. These are applications which encourage exploratory research that gives researcher additional flexibility for analysis in addition to the existing predefined functions or access to third party tools. In recent times, web based analytical platforms such as Mobyle (Neron *et al.,* 2009) and graphical workflow design systems are also being developed such as TAVERNA (Hull *et al.,* 2006), UGENE (Okonechnikov *et al.,* 2012) and others. However, these attempts may be considered to be at infancy with huge potential.

In the domain of computer science, object-oriented application frameworks (OOAFs) and domain specific programming environments are considered as the cornerstones of modern software engineering (Fayad *et al.,* 1999). *An OOAF is a domain specific, 'semi-complete" application that can be specialized to produce custom applications*. The primary benefits of an OOAF stems from the modularity, reusability, extensibility, and inversion of control they provide to developers. DENZO (Crystallography, http://www.hkl-xray.com/), MATLAB (Mathematics, http://www.mathworks.in), GNUPLOT (graph plotting, http://www.gnuplot.info/), R programming Environment (Statistics, http://www.r-project.org/) are some of the highly successful programming environments in respective domains which take advantage of these advanced software engineering designs/concepts.

BOS is an attempt to design and develop a biology specific comprehensive OOAF and a biological

object-based programming environment. Using BOS, the end user i.e., biologist will now be able to organize and / or instruct computational queries with biological vocabulary (objects), in addition to various other features, discussed below.

Materials and Methods

Current BOS architecture is two tiered. The core, referred as "BOS Kernel" executes the programming instructions passed to it. The user interface layer (tier) wrapped around the Kernel is referred as "BOS Editor".

BOS Kernel

Object Oriented Paradigm (OOP) revolutionized the design and development of compilers and computer programming languages hence improves the efficiency of software application development. Biology is inherently Object Oriented. The concepts of OOP especially modularity, reusability and extensibility are followed at almost all levels of life systems i.e., from molecules such as amino acids, nucleotides, DNA, RNA, proteins through cells, tissues, organs to vast ecosystems. *Each of these biological entities holds certain data/information and a unique associated behaviour. This natural pairing of data with associated behavior is considered a necessary condition to define an object in OOP*. This principle, in essence, enabled us to define various biological entities as biological objects, referred as Biological Abstract Data types (BioADTs) and differentiates our design from the other applications in the segment. The BioADTs are designed in order to be used as independent objects or '*is-like'* / *'contained in'* other pertinent ADTs. This resulted in a collection of large number of BioADTs implemented as classes. The collection of all the classes resulted in a comprehensive biological OOAF (http://www.biobhasha.org/docs/classes.html). Using CINT (http://root.cern.ch/drupal/content/cint) and the framework, a biology specific interpreter was developed which is referred as BOS Kernel henceforth.

BOS Kernel is developed in C/C++ language with ~150,000 lines of code. Functionality and utility-based abstraction led us to group all BioADTs into 8 major modules as described in Table 1.

Table 1: Core modules of BOS Kernel

Module	Description
Inputs	Input Module contains various BioADTs used for data sourcing from various biological data formats such as Fasta, Genbank, PDB, Swissprot, Pubmed, EMBL, DDBJ, SCOP, EST, ClustalW, MSF and others. Example BioADTs: BioFasta, BioGenBank, BioEmbl, BioPdb, BioHkl, BioEst, BioScop, BioBlast and so on.
Sequence	Sequence Module contains various sequence / string based BioADTs which use DNA / Protein sequences. OOP principles such as abstraction, inheritance, overloading have been extensively used to create relevant BioADTs. Example BioADTs: BioSequence, BioDnaSequence, BioProteinSequence and soon.

Module	Description
Structure	Structure Module contains various structure specific BioADTs which use DNA / RNA / Protein 3D structure information. Example BioADTs: BioPoint, BioAtom, BioResidue, BioChain, BioProtein, BioWater and so on.
Algorithms	Algorithms Module contains various algorithm implementations commonly used in biological analysis. The effort has been to make the algorithms reusable and generic which gives flexibility. The constructors are overloaded to accept various formats. GOOBA is an acronym which stands for 'generic object-oriented biological algorithms' is one of the sections found in 'BioBox' in 'BOS Editor'. The current version the following algorithms - dotplot, global/local sequence alignment, protein structure alignment, secondary structure prediction, secondary structure alignment, gene prediction, restriction map analysis, overlap and repeat identification. Example BioADTs: BioProteinSequenceGlobalAlignment, BioRestrictionMap, BioProteinStructureAlignment, BioDnaSequenceLocalAlignment and so on.
Library	Library Module is a unique set of BioADTs which are aimed at giving the researcher an ability to write novel algorithms from first principles. Example BioADTs: BioAminoAcidLibrary, BioNucleicAcidLibrary, BioSpaceGroupLibrary, BioRestrictionEnzymeLibrary, BioElementLibrary, BioAtomLibrary, BioStdCodonLibrary and so on.
System	System Module contains BioADTs that enable the researcher to do system level programming, especially the automation tasks such as searching, scanning the local directories and files, connecting over the available network and downloading the relevant files and other such tasks. The most commonly used BioADT is BioDatabase which has methods such as getFiles(), get Sub Directories WithFullPath(), removeDirectory() and so on.
Utilities	Utility Module contains many general utility classes and functions. Example BioADTs: BioMatrix, BioStatistics and so on.
Output	Output Module contains BioADTs which are used for presentation such as formating of data and the analysis results. Three output stream formats supported currently are text, postscript and HTML. Example BioADTs: BioOutputStream, BioHtml and BioPostScript.

This design enables systematic simulation of complex biological systems and their behaviour reusing available BioADTs and / or defining new BioADTs. For example UML diagram (Figure 1) shows inheritance pattern in Sequence module for different sequence data types.

BOS Editor

Most of the available biological libraries such as BioPERL, BioPython, are console based with the exception of R/BioConductor which has graphical user interface (GUI). GUI provides several advantages over console-based usage. BOS Editor is a GUI which helps the researcher to learn / write and 'run' the biological instructions rapidly with minimal errors. The features of the editor are described in results section.

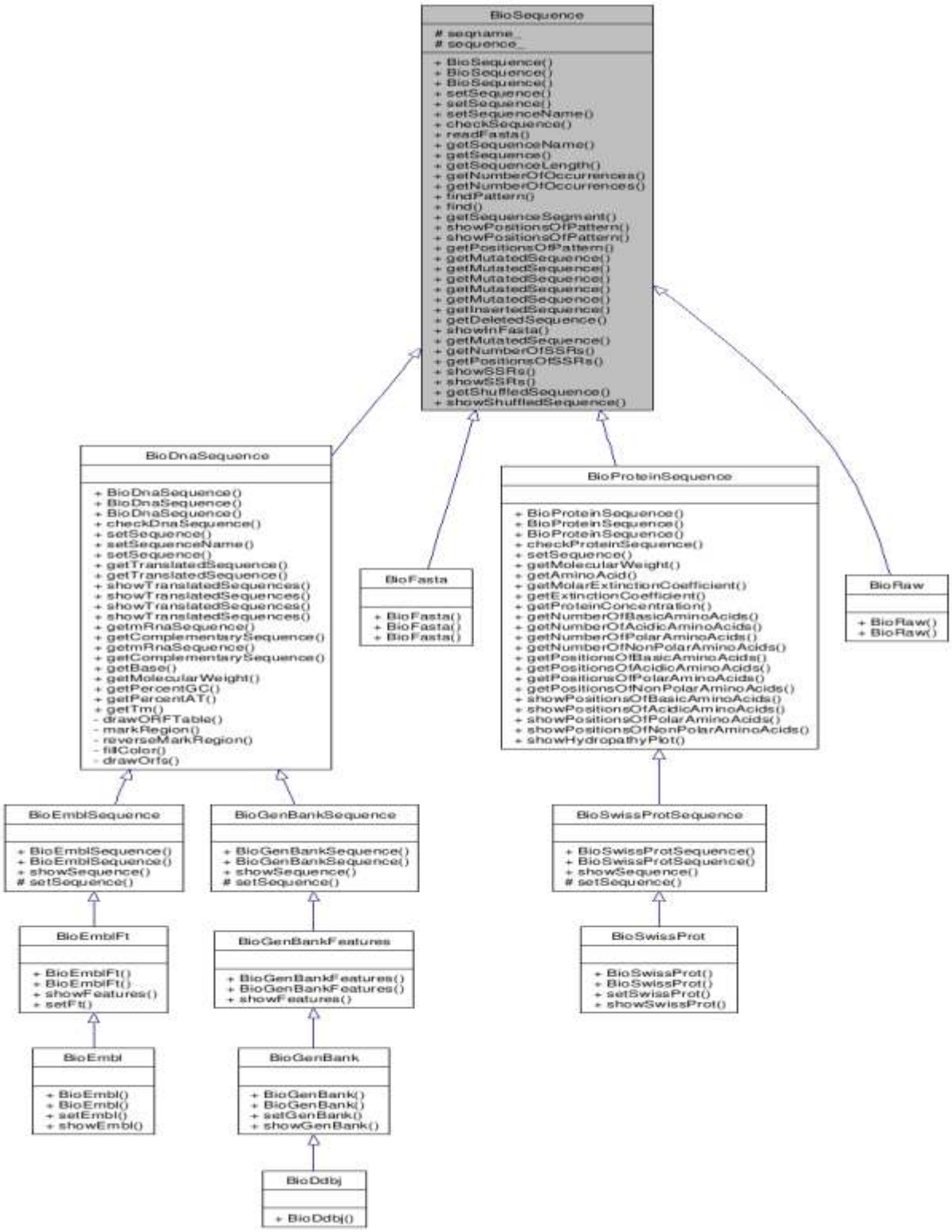

Figure 1: Inheritance UML diagram of BioSequence BioADT.

BioSequence is the parent sequence specific BioADT that has all the getters, setters, show and find methods common to both DNA and protein sequence data such as getMutatedSequence(), getNumberOfOccurrences(), findPattern() and others. BioDnaSequence and BioProteinSequence BioADTs are derived from BioSequence. Since Fasta format can have both types of data, BioFasta is derived from BioSequence unlike BioGenBank and other formats. BioGenBank, BioEmbl, BioDdbj formats are derived from BioDnaSequence and BioSwissprot ADT is derived from BioProteinSequence.

Results and Discussion

As mentioned above there is a subtle difference between *using OOP for biological data processing* and *abstracting biology into relevant BioADT's using OOP*. Our careful and in-depth study of this subtle difference enabled us to design Bio-*logical* entities using their inter-relationships. This design synchronizes with the natural thought process of a researcher and the instructions to be *typed* for getting the results.

For example consider the pseudo-code:

BioProtein(). BioProteinChain().BioResidue().BioAtom().getBfactor()

BioMultipleGenBank(). BioGenBank().BioFTKMatPeptide().BioFTQEvidence().getEvidence()

This convergence of thought process and scripting reduces digression, providing more productive time for analysis for the researcher.

Biological Abstraction

Since C++ supports procedural and OO paradigms, multiple inheritance, strict data typing and generic programming, we could create the data model that almost followed the naturally observed biological entity relationships. Biojava implemented similar syntax by using 'interfaces' to overcome the limitation of multiple inheritance and other features inherent with Java programming language. The comparison between BioJava, BioPython, BioPerl and others were discussed in earlier publications (Mangalam, 2002; Fourment and Gillings, 2008). The detailed comparison of the design principles between BOS and other applications is beyond the scope of the current article. We highlight few features (described in Table 2) to emphasize, a) deeper layers of abstraction, b) biologically more relevant abstraction and c) alternative methods of abstraction.

BOS Kernel

Careful and systematic abstraction resulted in the successful implementation of current version of BOS. The current version of the Kernel has: a) Around 400 Biological Objects, b) Around 14000 methods/functions, c) 12 Algorithm implementations, d) 17 Database input formats, e) 3 Output formats, f) 5 Biological Libraries g) Several sequence and structure specific objects, h) Statistics and matrix functions and other miscellaneous classes and functions.

Table 2 : Feature comparison between Biojava and BOS-Kernel

Design features	BioJava	BOS-Kernel
Abstraction of a. protein structure domain	• *PdbFileReader* is a class which reads a PDB file. • *PdbFileReader* is composed of Structure object. • The primary mode of access to information in a PDB is via methods of *PdbFileReader* and *Structure* object such as *getStructure(..), getId(..), findChain(..)* etc. User can access data fields of PDB records (like header, dbref, compounds etc.) and entity information and coordinate data types like chain, residue (group), atom etc.	• Protein domain is abstracted into general, structure specific and coordinates information. • BioPdb is the derived class which multiply inherits all classes which are independent and represent records/fields in PDB file (HEADER, COMPND, CRYST1, SOURCE, REMARK and others). • A biologically meaningful derived class, *BioSecondaryStructure class*, is implemented which is composed of *BioPdbHelix, BioPdbTurn, BioPdbSheet. BioSecondaryStructure* class is one of the parent classes of BioPdb class, giving access to secondary structure elements via getHelixCoordinates(..), getStrandSequence(…), getTurn(..) etc., methods. • The coordinate information is parsed and populates BioProtein composed of BioProteinChains and BioWater.
Independent usage of BioADTs	• The user depends primarily on the methods provided by *PdbFileReader* such as *PdbFileReader().getStructure() .getChain().getSeqRes Sequence (),* etc. To use *getId()*, *PdbFileReader* has to store the entire PDB file in memory making it memory intensive.	• All the BioADTs can be independently used for analysis. For example, *BioPdbHeader* can be used to parse the PDB file. Only the HEADER line of PDB is read into Memory and this enables us to access PDB Id using method getPdbId() which is both faster and memory efficient. • Similarly, *BioSecondaryStructure* or *BioPdbSeqres* can be used independently in case user intends to work only on the secondary structure or sequence information of the PDB file respectively.
Accessing various biological data formats	The primary mode of parsing sequences from various sequence data formats seems to be through a common sequence utility class, *SeqIOTools*, with various methods such as *readEmbl(..),read Swissprot (..), readGenbank(..)* and so on.	All biological formats are provided as independent classes like *BioGenbank, BioSwissProt, BioEmbl*. Separate BioADTs are provided for multiple entry formats as well, such as *BioMultipleFasta, BioMultipleGenBank, BioMultipleEmbl, BioClustal, BioMsf* etc.

BOS Editor

Since C++/ Java/ Javascript/ PHP syntax is the most accepted/ adopted syntax and closely resembles biological abstraction, it was a strategic decision to retain the same syntax that enabled using C++ in bio-scripting for the first time, alternative to PERL and Python based scripting.

The researcher need not digress from his/ her core competence and invest resources to learn about source code compilation, linking and other details specific to software build system configurations. The researcher is forced to digress as it is essential to have working knowledge about compilation and linking in order to use Biological APIs and libraries. However, BOS expects the user/researcher to write the instructions and 'run' (F5) the code to instantaneously see the results. This substantially reduces the time spent by researcher.

The programming environment provides the common features of an editor (Figure 2). For example, File options such as 'New project', 'New file', 'Open', 'Close', 'Quit' and others; 'Edit' options such as 'Cut', 'Copy' & 'Paste', 'Find & Replace' and others. Table 3 shows the different components of the editor. Besides the common editor features, there are bioinformatics specific features such as Substitution matrix dialog box, in *View Options,* that provides the researcher with an easy way of creating and using one's own customized substitution matrices which are known to significantly influence the pair wise alignment of proteins as well as DNA sequences.

Figure 2: BOS Editor illustrating various components

The researcher can also use the short-cut key combinations to write faster code. A 'dot' operator and 'cntrl'&'alt' key combination is associated with 'dynamic intelligent' help. On using the short-cut, list of all the associated methods for particular BioADT under use pops up, enables selection and auto-completes any partial instruction at the cursor position. This greatly reduces the learning curve for BOS which eliminates the need to memorize the various objects, methods and functions.

A project can be sub-divided into smaller tasks and tested independently. The researcher can run one task at a time or all at once. Similarly, one can create many projects and run them simultaneously also.

Table 3: Features provided by BOS Editor

S. No	Editor Compartment	Feature Description
1	Menu Bar	New project, new file, Open, Cut, Copy, Paste, Find, replace, run, help, view
2	Tool Bar	Common icons for creating new or opening files
3	Message Box	This box shows the error messages if any and shows method prototypes to avoid syntax errors
4	Status Bar	Gives the cursor position and scope of variable and instructions being written
5	Working Area	The editor where instructions are written for execution
6	Bio-Box	Provides access to the various BioADTs presented in a modular way. The user interface is intuitive and self-explanatory dialogue popup boxes.
7	BOS-Box	A interface for learning generic C++/Java programming syntax for beginners
8	Bio-Global Functions	A set of globally available biologically useful functions such as functions to calculate torsion angle, angle, distance, direction cosines and others
9	Math & miscellaneous	A set of commonly used mathematical and system wide functions

Example illustrations

The following simple BOS scripts provide the necessary code to understand the biological programming paradigm and implement one's own query using BOS.

Object based Biological Scripting

A script to access structural information from a PDB file by using biological entities like protein, chain, residue, atom etc.

```
BioProtein biop("c:/BOS_Scripts/BOS_Inputs/pdb/pdb3APP.ent");
cout << "Number of protein chains: "<< biop.getNumberOfProteinChains()<< endl;
//Displays the number of chains

BioProteinChain bcp = biop.getProteinChain(0); //get first protein chain from PDB file
for(int i = 0; i < bpc.getNumberOfResidues(); i++)
{
  cout << i+1 << "\t" << bpc.getResidue(i).getAtom("CA").getBfactor() << endl;
  //Prints Bfactor of each alpha carbon atom in all residues of first protein chain
}
BioGetChar();
```

Biologically Intuitive

Example of intuitive programming, demonstrated with a code to extract gene locus names and sequence of human entries from multi-entry GenBank file.

```
BioMultipleGenBank b("c:/BOS_Scripts/BOS_Inputs/GBK/p53.mgbk");
for(int i =0; i < b.getNumberOfEntries(); i++)
{
   //Function getEntry(index) returns a single gbk entry from multiple gbk file
   if( b.getEntry(i).findOrganism("Homo sapiens")) //check the organism name
   {       cout << "Locus: "<< b.getEntry(i).getLocusName()<<endl;
           cout << b.getEntry(i).getSequence() << endl;
   }
}
BioGetChar();
```

Multiple Data Sourcing

A script demonstrating use of many data types (formats) in a single program.

```
BioFasta p53gene("p53gene.txt");    // fasta format file as input
BioFasta newGene("putative gene","GCTAGCATGCGTGATCGGATCGGTGTAC");
//Overloaded constructor of BioFasta, taking gene name and sequence as parameters

BioGenBank p53gene2("p53gene.gbk");  //genbank format file as input

BioPdb p53protein_str("p53str.pdb"); //PDB structure format file as input

BioSwissProt p53protein_seq("p53pro_seq.swiss");//swissprot Sequence file

BioPubmed p53_literature("p53_lit.txt"); //Pubmed or Medine format file as input

BioPoint p1(2.3, 3.4, 5.4), p2(x,y,z); //Point with 3 co-ordinates in space

BioWater wat("pdb2apr.ent"); //'wat' holds the water information from a PDB file
```



```
BioMultipleEmbl all_seq_env("hiv_env_gene.txt"); // multiple Embl format file
BioGetChar();
```

Using Biological Libraries

Amino acid library stores basic information about each of the 20 amino acids, such as molecular weight, number of atoms, accessible surface area, hydrophobicity, dc volume and so on.

```
string am = "CYS"; //Amino acid cystein

BioAminoAcidLibrary::initialised(); //AminoAcid Library is ready for use
cout << "AA code" << "\t" << "SurfArea" << "\t" << "CFsheetProp" << endl;
cout << BioAminoAcidLibrary::AminoAcid[am].getSingleLetterCode() << "\t"
<< BioAminoAcidLibrary::AminoAcid[am].getAccessibleSurfaceArea() << "\t"
<< BioAminoAcidLibrary::AminoAcid[am].getCFSheetPropensity() << endl;
//Prints a single letter code (i.e. 'C'), accessible surface is and chou-fasman sheet propensity of Cystein
BioGetChar();
```

Using Algorithms

One of the most common tasks in Bioinformatics is to do an all-to-all alignment of complete gene sequences or protein sequences provided, in general, in multi-entry FASTA format. This example demonstrates how to use an algorithm BioADT such as BioProteinSequenceGlobalAlignment in conjunction with other BioADTs and unique methods such as getIdentity().

```
BioMultipleFasta spfas("asp.mfaa");
cout<<"The Number of Sequences : "<<aspfas.getNumberOfEntries()<<endl;
for ( int i = 0; j < aspfas.getNumberOfEntries(); i++)
{
  BioFasta entry1 = aspfas.getEntry(i),
  for( int j = 0; j < aspfas.getNumberOfEntries(); j++)
  {
        if(i!=j)
        {
              BioFasta entry2 = aspfas.getEntry(j);
              BioProteinSequenceGlobalAlignment pgl(entry1, entry2,"BLOSUM62");
              if( pgl.getIdentity() > 80.0 ) //Checks if %Identity greater than 80
              {
                    cout<<"Percent Identity: "<< pgl.getIdentity() << endl;
              pgl.showAlignment(); //Display alignment on standard output
              }
          }
        }
}
 BioGetChar();
```

Data Representation

Currently BOS supports text, HTML and PostScript Outputs. The code below displays Ramachandran Plot of a given PDB file in the PostScript format (shown in Figure 3).

```
BioPostScript ps("RamachandranPlot.PS");

BioProteinChain bc("c:/BOS_Scripts/BOS_Inputs/pdb/pdb3APP.ent");

bc.setRamaPlotColorSpec(255,0,0,250,250,0,0,0,250);

bc.showRamaChandranPlot(ps,FILLED_TRIANGLE);

ps.setFontSize(15);

string str1 = toString(int(bc.getNumberOfResidues()));

ps.setText("Number of Residues = ",80,140);

ps.setText(str1,300,140);

string str2 = toString(int(bc.getNumberOfAtoms()));

ps.setText("Number  of  Atoms = ",80,120);

ps.setText(str2,300,120);

ps.setAlign("CENTER");

ps.setText("RamaChandranPlot of 3APP Molecule",300,175);
BioGetChar();
```

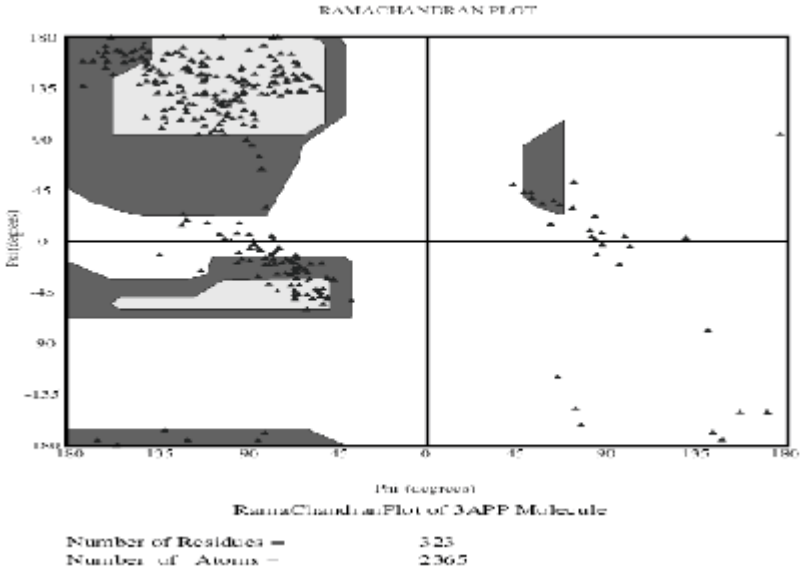

Figure 3: Post Script format output (Ramachandran plot) of the script

Conclusion and future scope

Current version of BOS attempts to provide a comprehensive platform for analysis which includes sequence domain, structure domain and literature search capabilities without losing the data/ workflow features. With these above-mentioned features, BOS provides an efficient biologist friendly programming environment for biological data analysis.

The download version (available at http://www.biobhasha.org) supports sequence alignment (global, local, repeat, overlap), structure alignment, dot-plot, restriction map, protein secondary structure prediction and literature mining. It supports Fasta, Genbank, Embl, DDBJ, Swissprot, PDB, Pubmed and many such database formats and strictly implemented the controlled vocabulary recommendations of International Nucleotide Sequence Database Collaboration (INSDC) (http://www.insdc.org).

As can be seen the development roadmap of BOS is complex, multi-directional, resource intensive and is work-in-progress. The current design enables future versions to become more biologist-friendly and integrative. The design implementation makes flexibility, customizability and extensibility, inherent, independent and inclusive of futuristic as well as unforeseen design requirements. Various microarray (affymetrix, illumina, agilent), next generation sequencing (NGS) data formats (FASTQ, SCARF) and BioHMM - generic BioADT are currently being implemented and tested to become part of next major release. In addition, efforts are being put to make the current interface (vocabulary [functionalities]) standards compliant with Gene Ontology (GO) (Ashburner *et al.,* 2000), Systems Biology Mark-up Language (SBML) (Hucka *et al.,* 2018) and others which will enable seamless integration. The design is being developed to port it and take advantage of cloud computing.

References

Néron, B., Ménager, H., Maufrais, C., Joly, N., Maupetit, J., Letort, S. et al. 2009. Mobyle: a new full web bioinformatics framework. *Bioinformatics,* 25(22): 3005-3011.

Richter, B. and Sexton, D. 2009. Managing and Analyzing Next-Generation Sequence Data. *PLoS Comput Biol.* 5(6), e1000369.

Hull, D., Wolstencroft, K., Stevens, R. et al. 2006. Taverna: a tool for building and running workflows of services. *Nucleic Acids Res.* 34,Web Server issue, 729-732.

Winn, M.D. et al. 2011. Overview of the CCP4 suite and current developments. *Acta Crystallogr D Biol Crystallogr.* D67, 235-242.

Mangalam. H. 2002. The Bio-* toolkits – a brief overview. *Briefings in Bioinformatics*. 3(3) 296–302.

Hucka, M., Bergmann, F.T., Dräger, A., Hoops, S., Keating, S.M., Le Novère, N., Myers, C.J., Olivier, B.G., Sahle, S., Schaff, J.C., Smith, L.P., Waltemath, D., Wilkinson, D.J. 2018. The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 2 Core. *J Integr Bioinform,* 15(1):266.

Dutheil, J., Gaillard, S., Bazin, E. et al. 2006. Bio++: a set of C++ libraries for sequence analysis, phylogenetics, molecular evolution and population genetics. *BMC Bioinformatics*, 7:188.

Stajich, J. et al. 2002. The Bioperl Toolkit: Perl Modules for the Life Sciences. *Genome Res.* 12, 1611-1618.

Okonechnikov, K., Golosova, O., Fursov, M. et al. 2012. Unipro UGENE: a unified bioinformatics toolkit. *Bioinformatics.* 28 (8);1166-1167.

Ashburner, M. et al. 2000. Gene Ontology: tool for the unification of biology. *Nat Genet.* 25(1), 25–29.

Fayad, M. et al. 1999. Building Application Frameworks. Addison-Wesley Pub Co, 1st edition.

Fourment M. and Gillings. M. 2008. A comparison of common programming languages used in bioinformatics. *BMC Bioinformatics,* 9:82.

Galperin, M. and Fernández-Suárez, X. 2012. The 2012 Nucleic Acids Research Database Issue and the online Molecular Biology Database Collection. *Nucleic Acids Res.* 40 (D1), D1-D8.

Larkin, et al. M. 2007. ClustalW and ClustalX version 2. *Bioinformatics,* 23, 2947-2948.

Cock, P., Antao, T., Chang, J. et al. 2009. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics,* 25(11), 1422-1423.

Kraulis, P. 1991. MOLSCRIPT: A program to produce both detailed and schematic plots of protein structures. *J. Appl. Crystallogr.,* 24, 946-950.

Rice, P., Longden, I., Bleasby, A. 2000. EMBOSS: The European Molecular Biology Open Software Suite. *Trends Genet.* 16, (6) pp 276-277.

Holland, R., Down, T., Pocock, M. et al. 2008. BioJava: an open-source framework for bioinformatics. *Bioinformatics,* 24(18), 2096-2097.

Russell R. and Barton. G. 1992. Multiple protein sequence alignment from tertiary structure comparison: assignment of global and residue confidence levels. *Proteins.* 14,309-23.

