Biopython

Introduction

Biopython is a set of libraries to provide the ability to deal with "things" of interest to biologists working on the computer. In general this means that you will need to have at least some programming experience (in python, of course) or at least an interest in learning to program. Biopython's job is to make your job easier as a programmer by supplying reusable libraries so that you can focus on answering your specific question of interest, instead of focusing on the internals of parsing a particular file format. One thing to note about Biopython is that it often provides multiple ways of “doing the same thing."

About Biopython

The Biopython Project is an international association of developers of freely available Python (*http://www.python.org*) tools for computational molecular biology. Biopython was established in the year 1999. The web site of Biopython *http://www.biopython.org* provides an online resource for modules, scripts, and web links for developers of Python-based software for life science research. Basically, it is aimed to program in python and want to make it as easy as possible to use python for bioinformatics by creating high quality, reusable modules and scripts.

Obtaining Biopython

Biopython's internet home is at, naturally enough, *http://www.biopython.org*. This is the home of all things Biopython, so it is the best place to start looking around if you are interested. When you feel ready to dive in and start working with the code, you have two choices:

1. **Release code** – This is made available in both stable and developer's releases on the download page (*http://www.biopython.org/Download/*). The stable releases are likely to be more well tested, while the development releases are closer to what is in CVS, and so will probably have more features. The releases are also available both as source and as installers (rpms and windows installers, right now), so you have some choices to pick from on releases if you prefer not to deal with source code directly.

2. **CVS** - The current working copy of the Biopython sources is always available via CVS (Concurrent Versions Systems - *http://www.cvshome.org/*). Concise instructions for accessing this copy are available at *http://cvs.biopython.org*.

Biopython development philosophies

Basically the philosophy is of all of the various open Bio-Projects. The major approach is to focus on the development of libraries, as opposed to ready-to-run programs. The alternative approaches are: (i) Design one program which does something well (BLAST, ClustalW) and (ii) Design a suite of programs that can work together (EMBOSS, PHYLIP, HMMER).

Advantages of this approach

- Flexible set of “small as possible” modules that can be used in numerous ways.
- Different parts can interoperate (no reading and writing of file formats).
- Can utilize the power of a programming language.
- Automation – a single program (written by you) can accomplish several different tasks.
- Allows you to take advantage of existing programs by executing them in your program.

Disadvantages of this approach

- Need to be able to program.
- Division of labor – multiple projects for different programming languages (BioPerl, BioJava, Biopython, BioRuby. . .)
- Requires more activation energy on the part of users
    - Tough to get started and get something happening without thinking.
    - Need to learn the code base and the way things are done.

Great problem in Biopython

1. *Data items distributed in multiple locations, which change over time to time*
    - Open Bioinformatics Database Access (OBDA) – generalized access to retrieving information into standard representations
    - EUtils at NCBI – connect with specifically designed retrieval interfaces (instead of scraping web pages).
2. *Massaging and changing data in potentially simple ways* (*translation of DNA sequences to protein*)
    - Simple sequence manipulation interfaces
    - Changes which can be made through a programming language (changing titles, manipulating strings, ugly things).

Biopython Package

The main Biopython releases have lots of functionality, including:

1. The ability to parse bioinformatics files into python utilizable data structures, including support for the following formats:
    - Blast output - both from standalone and WWW Blast
    - ClustalW
    - FASTA
    - GenBank
    - PubMed and Medline
    - Expasy files, like Enzyme, Prodoc and Prosite
    - SCOP, including 'dom' and 'lin' files
    - Rebase
    - UniGene
    - SwissProt

2. Files in the supported formats can be iterated over record by record or indexed and accessed via a Dictionary interface.

3. Code to deal with popular on-line bioinformatics destinations such as:
    - NCBI - Blast, Entrez and PubMed services
    - Expasy - Prodoc and Prosite entries

4. Interfaces to common bioinformatics programs such as:
    - Standalone Blast from NCBI
    - ClustalW alignment program.

5. A standard sequence class that deals with sequences, ids on sequences, and sequence features.

6. Tools for performing common operations on sequences, such as translation, transcription and weight calculations.

7. Code to perform classification of data using k Nearest Neighbors, Naive Bayes or Support Vector Machines.

8. Code for dealing with alignments, including a standard way to create and deal with substitution matrices.

9. Code making it easy to split up parallelizable tasks into separate processes.

10. GUI-based programs to do basic sequence manipulations, translations, BLASTing, etc.

11. Extensive documentation and help with using the modules, including this file, on-line wiki documentation, the web site, and the mailing list.

12. Integration with other languages, including the Bioperl and Biojava projects, using the BioCorba interface standard (available with the biopython-corba module).

Module Hierarchy

In python, modules are encapsulated program entities. Each single python file can be considered to be module. The python modules must be declared before using the files. There are two ways to declare the modules:

1. Import a "pythonfile.py"
   ***import* pythonfile (or)**
   ***from* pythondirectory *import* pythonfile**

2. Execute a function in "pythonfile.py":
   pythonfile.pfunction(arg)

Currently there are many python modules are available to solve bioinformatics problems. The main modules included in Biopython are listed below.

- **Bio**: *Collection of modules for dealing with biological data in Python.*
  - **Bio.Affy**: *Deal with Affymetrix related data such as cel files.*
  - **Bio.Ais**: *Immune system simulation based on ideas from Immunocomputing: a survey.*
  - **Bio.Align**: *Code for dealing with sequence alignments.*
  - **Bio.AlignAce**: *Parser and code for dealing with the standalone version of AlignAce, a motif search program.*
  - **Bio.AlignIO**: *Multiple sequence alignment input/output as Alignment objects.*
  - **Bio.Alphabet**: *Alphabets used in Seq objects etc to declare sequence type and letters.*
  - **Bio.Application**: *General mechanisms to access applications in biopython.*
  - **Bio.Blast**: *Code for dealing with BLAST programs and output.*
  - **Bio.CAPS**: *This module deals with CAPS markers.*
  - **Bio.CDD**: *Deal with Conserved Domain Database* (*CDD*) *entries from NCBI.*
  - **Bio.ClustalW**: *A set of classes to interact with the multiple alignment command line program ClustalW.*
  - **Bio.Cluster**: *Support for commonly used data clustering methods.*
  - **Bio.Compass**: *Code to deal with COMPASS output, a program for profile/profile comparison.*
  - **Bio.Crystal**: *Hetero, Crystal and Chain exist to represent the NDB Atlas structure.*
  - **Bio.DBXRef**

- **Bio.Data**: *Collections of various bits of useful biological data.*
- **Bio.Decode**
- **Bio.DocSQL**
- **Bio.ECell**: *For reading the ECell spreadsheet format from Ecell2* (*DEPRECATED*).
- **Bio.EUtils**: *A client-side library for the Entrez databases at NCBI* (*DEPRECATED*).
- **Bio.EZRetrieve**: *This module contains code to access EZRetrieve.*
- **Bio.Emboss**: *Code to interact with the ever-so-useful EMBOSS programs.*
- **Bio.Encodings**: *Properties for functionality such as transcription and translation.*
- **Bio.Entrez**: *Provides code to access NCBI over the WWW.*
- **Bio.Enzyme**: *This module provides code to work with the enzyme.dat file from Enzyme.*
- **Bio.ExPASy**: *This module provides code to access resources at ExPASy over the WWW.*
- **Bio.FSSP**: *Parser for FSSP files, used in a database of protein fold classifications.*
- **Bio.Fasta**: *Utilities for working with FASTA-formatted sequences* (*OBSOLETE*).
- **Bio.File**: *Code for more fancy file handles.*
- **Bio.FilteredReader**: *Code for more fancy file handles.*
- **Bio.GA**: *A selection of genetic algorithm code.*
- **Bio.GFF**: *Access to General Feature Format databases created with Bio::DB:GFF*
- **Bio.GenBank**: *Code to work with GenBank formatted files.*
- **Bio.Geo**: *Parser for files from NCBI's Gene Expression Omnibus* (*GEO*).
- **Bio.Gobase**: *This module provides code to work with files from Gobase.*
- **Bio.Graphics**
- **Bio.HMM**: *A selection of Hidden Markov Model code.*
- **Bio.HotRand**: *handles true random numbers supplied from the the web server of fourmilab.*
- **Bio.Index**: *Index.py*
- **Bio.IntelliGenetics**: *Parser for the MASE/IntelliGenetics alignment file format* (*DEPRECATED*).
- **Bio.InterPro**: *This module provides code to work with html files from InterPro, and code to access resources at InterPro over the WWW.*
- **Bio.KDTree**: *The KD tree data structure can be used for all kinds of searches that involve N-dimensional vectors.*
- **Bio.KEGG**: *This module provides code to work with data from the KEGG database.*
- **Bio.LocusLink**
- **Bio.LogisticRegression**: *This module provides code for doing logistic regressions.*
- **Bio.MEME**: *Parser for dealing with text output from the MEME motif search program*
- **Bio.MarkovModel**: *This is an implementation of a state-emitting MarkovModel.*
- **Bio.MaxEntropy**: *Maximum Entropy code.*
- **Bio.Medline**: *This module provides code to work with Medline.*
- **Bio.MetaTool**: *Parser for output from MetaTool 3.5* (*DEPRECATED*).
- **Bio.Mindy**
- **Bio.NBRF**: *Parser for the NBRF/PIR file format* (*DEPRECATED*).
- **Bio.NMR**: *Code for working with NMR data*
- **Bio.NaiveBayes**: *This provides code for a general Naive Bayes learner.*
- **Bio.Ndb**: *This module provides code to work with html files from NDB.*
- **Bio.NetCatch**: *NetCatch enables the user to scan a list of labelled urls and select a subset to read into a file.*
- **Bio.NeuralNetwork**: *Represent Neural Networks*
- **Bio.Nexus**: *The Bio.Nexus contains a NEXUS file parser and objects to model this data.*
- **Bio.PDB**: *Classes that deal with macromolecular crystal structures.*
- **Bio.ParserSupport**: *Code to support writing parsers.*
- **Bio.Parsers**: *Third party and other parsers useful internally to Biopython.*

- **Bio.Pathway**: *BioPython Pathway module.*
- **Bio.PopGen**: *PopGen: Population Genetics and Genomics library in Python*
- **Bio.Prosite**: *This module provides code to work with the prosite dat file from Prosite.*
- **Bio.PubMed**: *This module provides code to work with PubMed from the NCBI* (*OBSOLETE*).
- **Bio.Rebase**: *This module provides code to work with files from Rebase.*
- **Bio.Restriction**: *About the diverses class of the restriction enzyme implementation*
- **Bio.SCOP**: *SCOP: Structural Classification of Proteins.*
- **Bio.SGMLExtractor**: *Code for more fancy file handles.*
- **Bio.SVDSuperimposer**: *SVDSuperimposer finds the best rotation and translation to put two point sets on top of each other* (*minimizing the RMSD*).
- **Bio.Saf**: *Parser for SAF* (*Simple Alignment Format*).
- **Bio.Search**
- **Bio.Seq**: *Represent a sequence or mutable sequence, with an alphabet.*
- **Bio.SeqFeature**: *Represent a Sequence Feature holding info about a part of a sequence.*
- **Bio.SeqIO**: *Sequence input/output as SeqRecord objects.*
- **Bio.SeqRecord**: *Represent a Sequence Record, a sequence with annotation.*
- **Bio.SeqUtils**
- **Bio.Sequencing**: *Code to deal with various programs for sequencing and assembly.*
- **Bio.Statistics**
- **Bio.Std**
- **Bio.StdHandler**
- **Bio.SubsMat**: *Substitution matrices, log odds matrices, and operations on them.*
- **Bio.SwissProt**: *Parsers for file formats from the SwissProt database.*
- **Bio.Transcribe**: *Code to transcribe DNA into RNA or back* (*OBSOLETE*).
- **Bio.Translate**: *Code to translate DNA or RNA into proteins.*
- **Bio.UniGene**: *Parse Unigene flat file format files such as the Hs.data file.*
- **Bio.WWW**: *Deal with various biological databases and services on the web* (*DEPRECATED*).
- **Bio.Wise**
- **Bio.Writer**
- **Bio.builders**
- **Bio.config**
- **Bio.dbdefs**: *Martel based file format defintions* (*DEPRECATED*).
- **Bio.distance**: *This module provides code for various distance measures.*
- **Bio.expressions**
- **Bio.formatdefs**: *Martel based file format defintions* (*DEPRECATED*).
- **Bio.kNN**: *This module provides code for doing k-nearest-neighbors classification.*
- **Bio.listfns**: *This provides useful general functions for working with lists.*
- **Bio.mathfns**: *This provides useful general math tools.*
- **Bio.pairwise2**: *This package implements pairwise sequence alignment using a dynamic programming algorithm.*
- **Bio.stringfns**: *This provides useful general functions for working with strings.*
- **Bio.triefind**: *Given a trie, find all occurrences of a word in the trie in a string.*
- **Bio.utils**
- **Bio.writers**

- **BioSQL**: *Code for storing and retrieving biological sequences from a BioSQL relational database.*
- **Martel**: *Martel is a 'regular expressions on steroids' parser generator* (*OBSOLETE*).

For detailed literature: *http://biopython.open-bio.org/DIST/docs/api/module-tree.html*

An example Biopython program:

*Open **NOTEPAD** and type the following script and save as **CentralDogma.py**. Then open **PythonWin** and load **CentralDogma.py** and run.*

CentralDogma.py

```
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC
sequence = raw_input("Enter the DNA sequence: ")
seq = Seq(sequence, IUPAC.unambiguous_dna)
seq_list = list(sequence)
seq_list.reverse()
rev_seq = "".join(seq_list)
length =len(seq)
seq_comp = seq.complement()
rev_comp = seq.reverse_complement()
from Bio.Seq import reverse_complement, transcribe, back_transcribe, translate
seq_rna = transcribe(seq)
pro_seq = translate(seq)
back_comp = back_transcribe(seq_rna)
print "Given DNA sequence: ", seq
print "Length of the sequence: ", length
print "Reverse of the sequence: ", rev_seq
print "Compliment of the DNA sequence: ", seq_comp
print "Reverse Compliment of the DNA sequence: ", rev_comp
print "Transcription of the DNA sequence: ", seq_rna
print "Back Transcription of the RNA sequence: ", back_comp
print "Translation of DNA sequence: ", pro_seq
```

Output

```
>>> Given DNA sequence:  atcgtcgtatatgctgtcgat
Length of the sequence:  21
Reverse of the sequence:  tagctgtcgtatatgctgcta
Compliment of the DNA sequence:  tagcagcatatacgacagcta
Reverse Compliment of the DNA sequence:  atcgacagcatatacgacgat
Transcription of the DNA sequence:  aucgucguauaugcugucgau
Back Transcription of the RNA sequence:  atcgtcgtatatgctgtcgat
Translation of DNA sequence:  IVVYAVD
```