Biopython Tutorial and Cookbook

Je Chang, Brad Chapman, Iddo Friedberg, Thomas Hamelryck, Michiel de Hoon, Peter Cock

Contents

Chapter 1

Introduction

1.1 What is Biopython?

The Biopython Project is an international association of developers of freely available Python (http://www.python.org) tools for computational molecular biology. The web site http://www.biopython.org

Interfaces to common bioinformatics programs such as:

- Standalone Blast from NCBI
- Clustalw alignment program.

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

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

Chapter 2

Quick Start { What can you do with Biopython?

2.4 Parsing sequence le formats

2.4.2 Simple GenBank parsing example

Now let's load the GenBank le instead - notice that the code to do this is almost identical to the snippet

SCOP

The code is these modules basically makes it easy to write python code that interact with the CGI scripts on these pages, so that you can get results in an easy to deal with format. In some cases, the results can be tightly integrated with the Biopython parsers to make it even easier to extract information.

Here we'll show a simple example of performing a remote Entrez query. More information on the other services is available in the Cookbook, which begins on page 41.

In section 2.3 of the parsing examples, we talked about using Entrez website to search the NCBI nucleotide databases for info on Cypripedioideae, our friends the lady slipper orchids. Now, we'll look at how to automate that process using a python script. For Entrez searching, this is more useful for displaying results then as a tool for getting seque24results

Snazzy! We can fetch things and display them automatically { you could use this to quickly set up searches that you want to repeat on a daily basis and check by hand, or to set up a small CGI script to do queries and locally save the results before displaying them (as a kind of lab notebook of our search results). Hopefully whatever your task, the database connectivity code will make things lots easier for you!

2.6 What to do next

Now that you've made it this far, you hopefully have a good understanding of the basics of Biopython and

Chapter 3

Sequence objects

```
Seq('AGTACACTGGT', Alphabet())
>>> my_seq.alphabet
Alphabet()
```

However, where possible you should specify the alphabet explicitly when creating your sequence objects - in this case an unambiguous DNA alphabet object:

```
>>> my_seq = Seq("GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA", IUPAC.unambiguous_dna)
>>> standard_translator.translate(my_seq)
Seq('AIVMGR*KGAR', IUPACProtein())
>>> mito_translator.translate(my_seq)
Seq('AIVMGRWKGAR', IUPACProtein())
```

Notice that the default translation will just go ahead and proceed blindly through a stop codon. If you are aware that you are translating some kind of open reading frame and want to just see everything up until the stop codon, this can be easily done with the `translate_to_stop` function:

```
>>> standard_translator.translate_to_stop(my_seq)
Seq('AIVMGR', IUPACProtein())
```

Chapter 4

Sequence Input/Output

In this chapter we'll discuss the `Bio.SeqIO` module introduced earlier in more detail. This is a new interface added to Biopython 1.43, which aims to provide a simple interface for working with assorted sequence le

```
    print len(seq_record.seq)
handle.close()
```

Similarly, if you wanted to read in a file in another file format, then assuming `Bio.SeqIO.parse()`

4.1.5 Getting a list of the records in a sequence le

In the previous section we talked about the fact that `Bio.SeqIO`

This time the keys are:

```
['gi|2765596|emb|Z78471.1|PDZ78471', 'gi|2765646|emb|Z78521.1|CCZ78521', ...
 ..., 'gi|2765613|emb|Z78488.1|PTZ78488', 'gi|2765583|emb|Z78458.1|PHZ78458']
```

You should recognise these strings from when we parsed the FASTA file earlier in Section 2.4.1. Suppose you would rather have something else as the keys - like the accesion numbers. This brings us nicely to `SeqIO.to_dict()`'s optional argument `key_function`, which lets you define what to use as the dictionary key for your records.

First you must write your own function to return the key you want (as a string) when given a SeqRecord

```
from Bio import SeqIO
handle = open("my_example.faa", "w")
SeqIO.write(my_records, handle, "fasta")
handle.close()
```

And if you open this le in your favourite text editor it should look like this:

```
>gi|14150838|gb|AAK54648.1|AF376133_1 chalcone synthase [Cucumis sativus]
MMYQQGCFAGGTVLRLAK6rK
```

```
from Bio import SeqIO
in_handle = open("ls_orchid.gbk")
for record in SeqIO.parse(in_handle1
```

```
from Bio import SeqIO
SeqIO.write((make_rc_record(rec) for rec in \
            SeqIO.parse(open("ls_orchid.fasta", "r"), "fasta") if len(rec.seq) < 700), \
            open("rev_comp.fasta", "w"), "fasta")
```

Personally, I think the above snippe21.9w8(e21.conipp)dhetseeeabe21.compact,er014h,wingucboeasier1,

Chapter 5

BLAST

Hey, everybody loves BLAST right? I mean, geez, how can get it get any easier to do comparisons between one of your sequences and every other sequence in the known world? Heck, if I was writing the code to do

First, we need to get the info in the FASTA le. The easiest way to do this is to use the Bio.SeqIO module. In this example, we'll use Bio.SeqIO.parse to parse the FASTA le and store the rst FASTA record in the le in a SeqRecord object (section 2.4.1 explains Bio.SeqIO.parse in more detail).

```
>>> from Bio import 626Tf5048-11.955Td[(>>>)-525ile_h8(a14le)-52=le"m_cold.fasta"ilTf5048-11.955Td[(>>>
```

```
>>> result_handle = open("my_blast.xml")
```

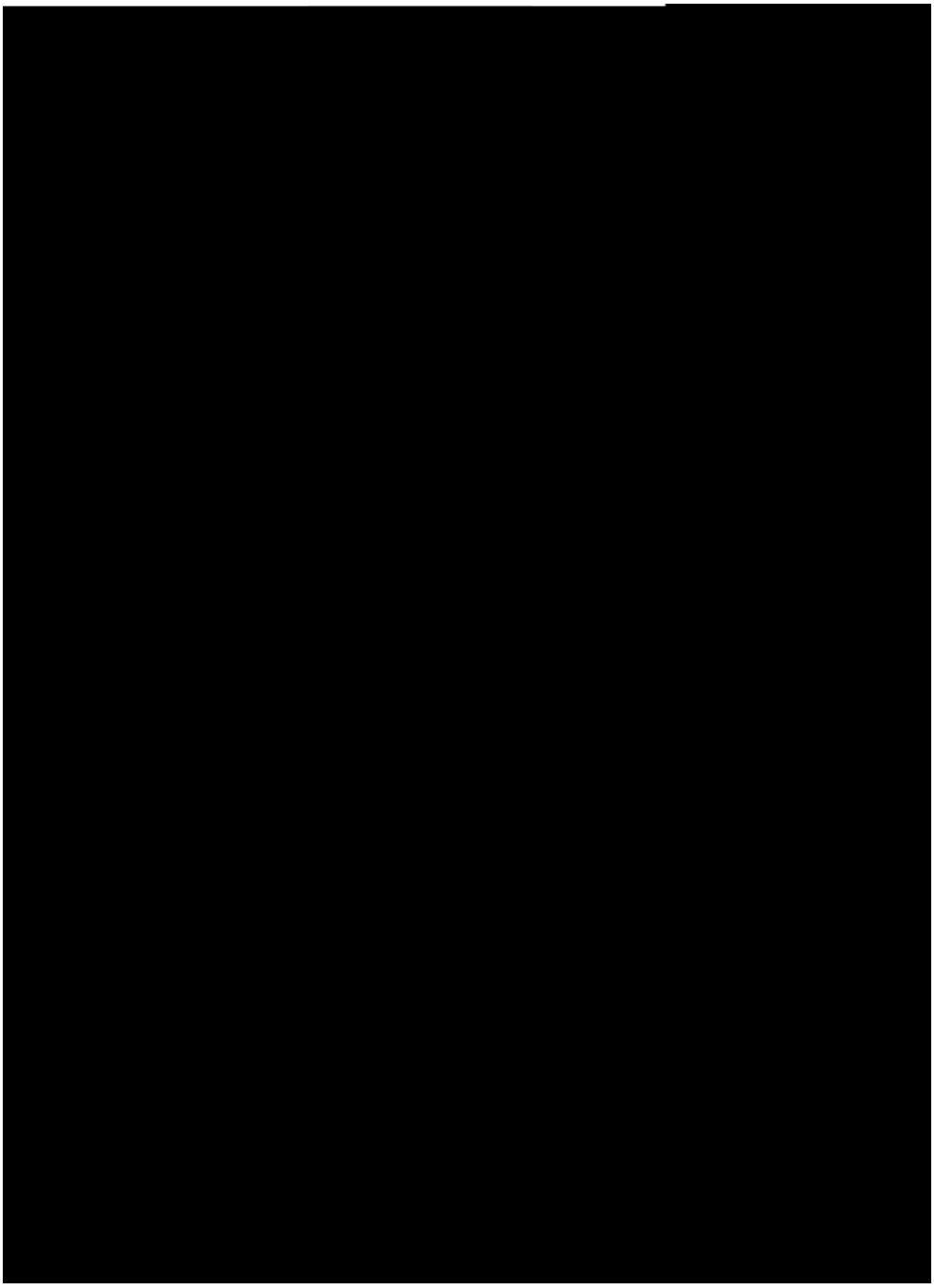

Figure 5.1: Class diagram for the Blast Record class representing all of the info in a BLAST report

Then we will assume we have a handle to a bunch of blast records, which we'll call `result_handle`. Getting a handle is described in full detail above in the blast parsing sections.

Now that we've got a parser and a handle, we are ready to set up the iterator with the following command:

5.7 Dealing with PSIBlast

Chapter 6

Cookbook { Cool things to do with it

6.1 SWISS-PROT

6.1.1 Retrieving a SWISS-PROT record

SwissProt (http://www.expasy.org/sprot/sprot-top.html) is a hand-curated database of protein se-

Note that we convert `all_results`, which is a string, into a handle before passing it. The iterator requires a handle to be passed so that it can read in everything one line at a time. The `Bio.File` module has a nice StringHandle, which conveniently will convert a string into a handle. Very nice! Now we are ready to start extracting information.

To get out the information, we'll go through everything record by record using the iterator. For each record, we'll just print out some summary information:

```
while 1:
    cur_record = s_iterator.next()

    if cur_record is None:
        break

    print "description:", cur_record.description
    for ref in cur_record.references:
        print "authors:", ref.authors
        print "title:", ref.title

    print "classification:", cur_record.organism_classification
    print
```

This prints out a summary like the following:

```
description: CHALCONE SYNTHASE 8 (EC 2.3.1.74) (NARINGENIN-CHALCONE SYNTHASE 8)
authors: Liew C.F., Lim S.H., Loh C.S., Goh C.J.;
title: "Molecular cloning and sequence analysis of chalcone synthase cDNAs of
Bromheadia finlaysoniana.";
classification: ['Eukaryota', 'Viridiplantae', 'Embryophyta', 'Tracheophyta',
'Spermatophyta', 'Magnoliophyta', 'Liliopsida', 'Asparagales', 'Orchidaceae',
'Bromheadia']
```

```
from Bio.WWW import ExPASy
import re

handle = ExPASy.sprot_search_de("Orchid Chalcone Synthase")
# or:
# handle = ExPASy.sprot_search_ful("Orchid and {Chalcone Synthase}")
html_results = handle.read()
if "Number of sequences found" in html_results:
    ids = re.findall(r'HREF="/uniprot/(\w+)"', html_results)
else:
    ids = re.findall(r'href="/cgi-bin/niceprot\.pl\?(\w+)"', html_results)
```

6.2 PubMed

6.2.1 Sending a query to PubMed

If you are in the Medical eld or interested in human issues (and many times even if you are not,/,are

Pub/Med

Querying PubMed using Biopython extremely To42330(get)-3312/ll of articlefor articlesving

rom Bio import PubMed

earch_term = 'orchid'

Now let's look at how to use this nice dictionary to print out some information a-324(d)1(ictionary)-3ctieinformation ads

2.

6.4.1 Clustalw

As the name suggests, this is a really simple consensus calculator, and will just add up all of the residues at each point in the consensus, and if the most common value is higher than some threshold value (the default is .3) will add the common residue to the consensus. If it doesn't reach the threshold, it adds an ambiguity

2. The sequence passed to be displayed along the left side of the axis does not need to be the consensus.

Well, now that we have an idea what information content is being calculated in Biopython, let's look at

```
from Bio import Clustalw
from Bio.Alphabet import IUPAC
from Bio.Align import AlignInfo

# get an alignment object from a Clustalw alignment output
c_align = Clustalw.parse_file("protein.aln", IUPAC.protein)
```

Once you've got your log odds matrix, you can display it prettily using the function `print_mat`. Doing this on our created matrix gives:

strand

If you don't want to deal with fuzzy p wantananpt85454(y)28(ou)-4juson'tt tot

This section deals with the speci cs of setting up and using this sysFcs of

Figure 6.1: UML diagram of the SMCRA data structure used to represent a macromolecular structure.

Disordered atoms and residues are represented by DisorderedAtom and DisorderedResidue classes, which are both subclasses of the DisorderedEntityWrapper base class. They hide the complexity associated with

6.10.1.1 Structure

The second field in the Residue id is the sequence identifier, an integer describing the position of the residue in the chain.

The third field is a string, consisting of the insertion code. The insertion code is sometimes used to

```
a.get_bfactor()    # B factor

a.get_occupancy()  # occupancy

a.get_altloc()     # alternative location specifie

a.get_sigatm()     # std. dev. of atomic parameters

a.get_siguij()     # std. dev. of anisotropic B factor

a.get_anisou()     # anisotropic B factor

a.get_fullname()   # atom name (with spaces, e.g. ".CA.")
```

To represent the atom coordinates, siguij, anisotropic B factor and sigatm Numpy arrays are used.

6.10.2 Disorder

6.10.2.1 General approach

Disorder should be dealt with from two points of view: the atom and the residue points of view. In general,

storing the DisorderedAtom object in a Residue object just like ordinary Atom objects. The DisorderedAtom

```
residue_id=("H_GLC", 10, " ")
residue=chain[residue_id]
```

Print all hetero residues in chain.

```
for residue in chain.get_list():
residue_id=residue.get_id()
hetfield=residue_id[0]
if hetfield[0]=="H":
print residue_id
```

Print out the coordinates of all CA atoms in a structure with B factor greater than 50.

```
for model in structure.get_list():
  for chain in model.get_list():
    for residue in chain.get_list():
      if residue.has_id("CA"):
        ca=residue["CA"]
        if ca.get_bfactor()>50.0:
          print ca.get_coord()
```

Print out all the residues that contain disordered atoms.

```
for model in structure.get_list()
  for chain in model.get_list():
    for residue in chain.get_list():
      if residue.is_disordered():
```

6.10.5 Common problems in PDB files

6.10.5.1 Examples

The PDBParser/Structure class was tested on about 800 structures (each belonging to a unique SCOP superfamily). This takes about 20 minutes, or on average 1.5 seconds per structure. Parsing the structure of the large ribosomal subunit (1FKK), which contains about 64000 atoms, takes 10 seconds on a 1000 MHz PC.

Three exceptions were generated in cases where an unambiguous data structure could not be built. In all three cases, the likely cause is an error in the PDB file that should be corrected. Generating an exception in these cases is much better than running the chance of incorrectly describing the structure in a data structure.

6.10.5.1.1 Duplicate residues

The residue names of the residues in the case of point mutations (to store the Residue objects in a DisorderedResidue object).

```
#  The value is the GenePop record.
#  rec is not altered.

rec_pops =  rec.split_in_pops(pop_names)
#Splits a record in populations, that is, for each population, it creates
#  a new record, with a single population and all loci.
#  The result is returned in a dictionary, being each key
#  the population name. As population names are not available in GenePop,
#  they are passed in array (pop_names).
#  The value of each dictionary entry is the GenePop record.
#  rec is not altered.
```

You can only call cplot after having run fdist.
This will calculate the con dence intervals (99% in this case) for a previous fdist run. A list of quadruples

6.12 Miscellaneous

6.12.1 Translating a DNA sequence to Protein

Chapter 7

Advanced

Scanner:

feed(self, handle, consumer):

should implement a method named 'feed' that takes a file handle and a consumer. Th

7.3.7 Enzyme

The Enzyme.py module works with the enzyme.dat le included with the Enzyme distribution. The Enzyme Scanner produces the following events:

```
record
    iyfificabutime
```

```
dblinks_id
record_end
```

```
english_abstract
entry_month
gene_symbol
identification
issue_part_supplement
issn
journal_title_code
language
special_list
last_revision_date
mesh_heading
mesh_tree_number
major_revision_date
no_author
substance_name
pagination
personal_name_as_subject
publication_type
number_of_references
cas_registry_number
record_originator
journal_subset
subheadings
secondary_source_id
source
title_abbreviation
title
transliterated_title
unique_identifier
volume_issue
year
pubmed_id
```

```
    documentation
    terminator
```

The PRODOC scanner produces the following events:

```
record
    accession
    prosite_reference
    text
    reference
```

7.3.12 SWISS-PROT

```
sequence_type
sequence_name
comment
```

```
enzyme
matrix_row
sum_is_constant_line
end_stochiometric
end_kernel
end_subsets
end_reduced_system
end_convex_basis
end_conservation_relations
end_elementary_modes
```

7.4 Substitution Matrices

7.4.1 SubsMat

v. `build_later`: default false. If true, user may supply only alphabet and empty dictionary, if intending to build the matrix later. this skips the sanity check of alphabet size vs. matrix size.

(b) `entropy(self,obs_freq_mat)`

i. `obs_freq_mat`

(b) Generating the observed frequency matrix (OFM)

Use:

```
OFM = SubsMat._build_obs_freq_mat(ARM)
```

Chapter 8

Where to go from here { contributing to Biopython

8.1 Maintaining a distribution for a platform

Macintosh { We would love to find someone who wants to maintain a Macintosh distribution, and make it available in a Macintosh friendly format like bin-hex. This would basically include finding a way to compile everything on the Mac, making sure all of the code written by us UNIX-based developers works well on the Mac, and providing any Mac-friendly hints for us.

Chapter 9

Appendix: Useful stu about Python

If you haven't spent a lot of time programming in python, many questions and problems that come up in