Chapter 7

BLAST

Hey, everybody loves BLAST right? I mean, geez, how can it get any easier to do comparisons between one of your sequences and every other sequence in the known world? But, of course, this section isn't about how cool BLAST is, since we already know that. It is about the problem with BLAST – it can be really difficult to deal with the volume of data generated by large runs, and to automate BLAST runs in general.

Fortunately, the Biopython folks know this only too well, so they've developed lots of tools for dealing with BLAST and making things much easier. This section details how to use these tools and do useful things with them.

Dealing with BLAST can be split up into two steps, both of which can be done from within Biopython. Firstly, running BLAST for your query sequence(s), and getting some output. Secondly, parsing the BLAST output in Python for further analysis.

Your first introduction to running BLAST was probably via the NCBI web-service. In fact, there are lots of ways you can run BLAST, which can be categorised in several ways. The most important distinction is running BLAST locally (on your own machine), and running BLAST remotely (on another machine, typically the NCBI servers). We're going to start this chapter by invoking the NCBI online BLAST service from within a Python script.

NOTE: The following Chapter 8 describes `Bio.SearchIO`, an *experimental* module in Biopython. We intend this to ultimately replace the older `Bio.Blast` module, as it provides a more general framework handling other related sequence searching tools as well. However, until that is declared stable, for production code please continue to use the `Bio.Blast` module for dealing with NCBI BLAST.

7.1 Running BLAST over the Internet

We use the function `qblast()` in the `Bio.Blast.NCBIWWW` module to call the online version of BLAST. This has three non-optional arguments:

- The first argument is the blast program to use for the search, as a lower case string. The options and descriptions of the programs are available at https://blast.ncbi.nlm.nih.gov/Blast.cgi. Currently `qblast` only works with blastn, blastp, blastx, tblast and tblastx.
- The second argument specifies the databases to search against. Again, the options for this are available on the NCBI Guide to BLAST ftp://ftp.ncbi.nlm.nih.gov/pub/factsheets/HowTo_BLASTGuide.pdf.
- The third argument is a string containing your query sequence. This can either be the sequence itself, the sequence in fasta format, or an identifier like a GI number.

The `qblast` function also take a number of other option arguments which are basically analogous to the different parameters you can set on the BLAST web page. We'll just highlight a few of them here:

- The argument `url_base` sets the base URL for running BLAST over the internet. By default it connects to the NCBI, but one can use this to connect to an instance of NCBI BLAST running in the cloud. Please refer to the documentation for the `qblast` function for further details.
- The `qblast` function can return the BLAST results in various formats, which you can choose with the optional `format_type` keyword: `"HTML"`, `"Text"`, `"ASN.1"`, or `"XML"`. The default is `"XML"`, as that is the format expected by the parser, described in section 7.3 below.
- The argument `expect` sets the expectation or e-value threshold.

For more about the optional BLAST arguments, we refer you to the NCBI's own documentation, or that built into Biopython:

```
>>> from Bio.Blast import NCBIWWW
>>> help(NCBIWWW.qblast)
...
```

Note that the default settings on the NCBI BLAST website are not quite the same as the defaults on QBLAST. If you get different results, you'll need to check the parameters (e.g., the expectation value threshold and the gap values).

For example, if you have a nucleotide sequence you want to search against the nucleotide database (nt) using BLASTN, and you know the GI number of your query sequence, you can use:

```
>>> from Bio.Blast import NCBIWWW
>>> result_handle = NCBIWWW.qblast("blastn", "nt", "8332116")
```

Alternatively, if we have our query sequence already in a FASTA formatted file, we just need to open the file and read in this record as a string, and use that as the query argument:

```
>>> from Bio.Blast import NCBIWWW
>>> fasta_string = open("m_cold.fasta").read()
>>> result_handle = NCBIWWW.qblast("blastn", "nt", fasta_string)
```

We could also have read in the FASTA file as a `SeqRecord` and then supplied just the sequence itself:

```
>>> from Bio.Blast import NCBIWWW
>>> from Bio import SeqIO
>>> record = SeqIO.read("m_cold.fasta", format="fasta")
>>> result_handle = NCBIWWW.qblast("blastn", "nt", record.seq)
```

Supplying just the sequence means that BLAST will assign an identifier for your sequence automatically. You might prefer to use the `SeqRecord` object's format method to make a FASTA string (which will include the existing identifier):

```
>>> from Bio.Blast import NCBIWWW
>>> from Bio import SeqIO
>>> record = SeqIO.read("m_cold.fasta", format="fasta")
>>> result_handle = NCBIWWW.qblast("blastn", "nt", record.format("fasta"))
```

This approach makes more sense if you have your sequence(s) in a non-FASTA file format which you can extract using `Bio.SeqIO` (see Chapter 5).

Whatever arguments you give the `qblast()` function, you should get back your results in a handle object (by default in XML format). The next step would be to parse the XML output into Python objects representing the search results (Section 7.3), but you might want to save a local copy of the output file first. I find this especially useful when debugging my code that extracts info from the BLAST results (because re-running the online search is slow and wastes the NCBI computer time).

We need to be a bit careful since we can use `result_handle.read()` to read the BLAST output only once – calling `result_handle.read()` again returns an empty string.

```
>>> with open("my_blast.xml", "w") as out_handle:
...     out_handle.write(result_handle.read())
...
>>> result_handle.close()
```

After doing this, the results are in the file `my_blast.xml` and the original handle has had all its data extracted (so we closed it). However, the `parse` function of the BLAST parser (described in 7.3) takes a file-handle-like object, so we can just open the saved file for input:

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

Now that we've got the BLAST results back into a handle again, we are ready to do something with them, so this leads us right into the parsing section (see Section 7.3 below). You may want to jump ahead to that now ....

7.2 Running BLAST locally

7.2.1 Introduction

Running BLAST locally (as opposed to over the internet, see Section 7.1) has at least major two advantages:

- Local BLAST may be faster than BLAST over the internet;
- Local BLAST allows you to make your own database to search for sequences against.

Dealing with proprietary or unpublished sequence data can be another reason to run BLAST locally. You may not be allowed to redistribute the sequences, so submitting them to the NCBI as a BLAST query would not be an option.

Unfortunately, there are some major drawbacks too – installing all the bits and getting it setup right takes some effort:

- Local BLAST requires command line tools to be installed.
- Local BLAST requires (large) BLAST databases to be setup (and potentially kept up to date).

To further confuse matters there are several different BLAST packages available, and there are also other tools which can produce imitation BLAST output files, such as BLAT.

7.2.2 Standalone NCBI BLAST+

The "new" NCBI BLAST+ suite was released in 2009. This replaces the old NCBI "legacy" BLAST package (see below).

This section will show briefly how to use these tools from within Python. If you have already read or tried the alignment tool examples in Section 6.5 this should all seem quite straightforward. First, we construct a command line string (as you would type in at the command line prompt if running standalone BLAST by hand). Then we can execute this command from within Python.

For example, taking a FASTA file of gene nucleotide sequences, you might want to run a BLASTX (translation) search against the non-redundant (NR) protein database. Assuming you (or your systems administrator) has downloaded and installed the NR database, you might run:

```
$ blastx -query opuntia.fasta -db nr -out opuntia.xml -evalue 0.001 -outfmt 5
```

This should run BLASTX against the NR database, using an expectation cut-off value of 0.001 and produce XML output to the specified file (which we can then parse). On my computer this takes about six minutes - a good reason to save the output to a file so you can repeat any analysis as needed.

From within Biopython we can use the NCBI BLASTX wrapper from the `Bio.Blast.Applications` module to build the command line string, and run it: