Project Practical

Fundamentals of Bioinformatics

September 11, 2012

1 Introduction

In this document you will find all the technical information and background you need to build and script the database queries you need to complete the assignment in the Fundamentals of Bioinformatics practical. The first part is about how to post to URLs in python. The second part is divided into 4 sections, which go parallel with the steps in the practical workflow.

1 Find matching sequences using BLAST and PSI-BLAST

2 Find GO terms for your proteins

3 Find the Pfam family for your proteins

4 Benchmarking BLAST and PSI-BLAST

2 Practical 1: BLAST/PSI-BLAST

You can either perform BLAST search by using it via their website (see appendix) or you can use the standalone version locally in your own computer. To run BLAST and PSI-BLAST locally you can use the blast2 package. You can install the Blast2 package in your Debian/Ubuntu linux machine by typing the following command (This package is already installed in the VU's linux workstations):

```
$ sudo apt-get install blast2
```

Before we can start performing BLAST/PSI-BLAST searches locally, we have to create directories to store our query sequences, database, and search results. You can create these directories using command line:

```
~$ mkdir queries
~$ mkdir db
~$ mkdir results
```

One of the main advantage of running the standalone version of BLAST is that you can create your own database. You can build your database out of any fasta formatted file containing multiple proteins. We have provided you with a list of 215 protein sequences, this file can be downloaded from the blackboard or wiki page (SCOP_selections.txt). in this practical we are going to do all against all BLAST/PSI-BLAST searches which means that we are going to use these sequences as both queries and database.

Q1.a. Create a python script to download the sequences (in fasta format) of all the proteins in the SCOP_selections.txt

Q1.b. Store these sequences into a single fasta file in the db directory that you have created and secondly store these sequences as separate fasta files in the queries directory.

Now that you have made your database fasta file, the next step is to format the database file using "formatdb" program which is also part of the blast2 package. To find out about all options that are available for this program you can do it in the following way:

```
~$ formatdb -
```

For more detailed explanation about formatdb please look at:
ftp://ftp.ncbi.nih.gov/blast/documents/formatdb.html

Q2. Add a new function in your python script to format your database file

Q3. How many entries are in your database after invoking formatdb? Which format does the database file(s) have?

The next step is to perform the actual BLAST/PSI-BLAST searches using blast2 (standard BLAST) and blastpgp (PSI-BLAST), to find out about all options that are available for these programs, type the following commands in your command-line interface:

```
~$ blast2 -
~$ blastpgp -
```

in the following steps we are going to use protein P00698 as a test case. Try to perform a default BLAST search of P00698, using the following command:

```
~$ blast2 -p blastp -i queries/P00698.fasta -d db/SCOP_selections.fasta
```

Now run the PSI-BLAST using the following command and compare the results. *Hint:try the option -m 9, this option produces a concise overview of BLAST/PSI-BLAST output*

```
~$ blastpgp  -j 3 -i queries/P00698.fasta -d db/SCOP_selections.fasta
```

Note that in order to run PSI-BLAST, the -j parameter must be set to something greater than 1. The default of -j 1 means that there are no iterations and that it's therefore the same as a single BLAST search.

Q4. what is the difference between e-value threshold parameters -e and -h in blastpgp. Try different e-value thresholds, explain your observation.

Q5. Is it possible that PSI-BLAST stops before it reached the number of iterations as determined by the -j parameter. If so, why? *Hint: Think in relationship with the -h parameter.*

2.1 input/output

tab separated
Input: list with unprot ids
output: 2 files with list with pairs of uniprot ids, and an e-value, with NA, if no match was found between proteins even at a high e-value cut-off.
One output file for BLAST, one for PSI-BLAST.

```
P000001  P000002  0.0001
P000001  P000003  10.0
P000002  P000003  NA
```

3 Practical 2: Gene Ontology

The goal of this practical is to get some hands-on experience with the Gene Ontology (GO) database (`http://www.geneontology.org/`). This database is a community effort to establish a standardized representation of gene and gene product attributes across different species.

As the name suggests, GO is an ontology. An *ontology* is an explicit specification of how terms in a specific domain are organized. In GO the domain consists of terms annotating genes and gene products. These terms are organized in a specific way.

Q1. How are the terms in the Gene Ontology database organized?

Q2. Can you have multiple GO terms per protein? Explain why or why not?

Biologists commonly use sequence alignment to infer whether two proteins are functionally similar. Typically, a sequence identity larger than 35% is considered to be a strong indication for functional similarity. On the other hand, a sequence identity between 20% and 35% corresponds to the so called twilight zone, for which there is not enough evidence to infer functional similarity.

Q3. It might be the case that two proteins share exactly the same function, but have different sequences. Give a biological reason for why this may be the case.

An alternative way to infer functional similarity is by comparing GO terms. For example, proteins LAP3 (UniProt ID: Q01532) and BLMH (UniProt ID: Q13867) are functionally related.

Q4. List the GO terms of these two proteins.

Q5. List the GO terms that the two proteins have in common.

The idea is that the more GO terms two protein share, the more likely it is that they have similar functions. We can design a score taking this aspect into account. Preferably the score should be normalized, e.g. between 0 and 1.

Q6. Why is it convenient that in general scores are normalized?

Given a protein pair (p_1, p_2) a possible scoring function is:

$$s(p_1, p_2) = \frac{g(p_1) \cap g(p_2)}{g(p_1) \cup g(p_2)}$$

where $g(p)$ is the set of GO terms associated to protein p.

Q7. Use the formula to compute the score between LAP3 and BLMH.

Q8. What would be the score of a protein when compared to itself?

Q9. Which aspect of the GO database is not taken into account by this scoring function? Why is this a problem?

The file '`SCOP_selections.txt`' contains 215 UniProt protein IDs. For each pair of proteins, we want to determine whether they are functionally similar using the above scoring function.

Q10. Excluding pairs consisting of the same protein, how many unique pairs are there?

The remainder of this assignment involves writing a script to classify pairs based on their scores as being either ambiguous, different or similar using two thresholds.

Q10. Fill in the missing parts in the script '`classify.py`'.

Q11. Determine suitable thresholds by plotting a distribution of the scores. Indicate the two thresholds in the plot.

4 Practical 3: Pfam

The URL access to Pfam by UniProt ID is very simple. For example, the url access for UniProtID P00698 is `http://pfam.sanger.ac.uk/protein?output=xml&acc=P00698`
The reply will look something like:

```
<!--
information on UniProt entry P00698 (LYSC_CHICK), generated: 09:24:32 11-Sep-2012
-->
<pfam xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://pfam.sanger.ac.uk/"
 xsi:schemaLocation="http://pfam.sanger.ac.uk/ http://pfam.sanger.ac.uk/static/documents/schemas/protein.xsd" release="26.0"
 release_date="2011-11-17">
<entry entry_type="sequence" db="uniprot" db_release="2011_06" accession="P00698" id="LYSC_CHICK">
<description>
<![CDATA[ Lysozyme C EC=3.2.1.17 ]]>
</description>
<taxonomy tax_id="9031" species_name="Gallus gallus (Chicken)">
Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi; Archosauria; Dinosauria; Saurischia;
Theropoda; Coelurosauria; Aves; Neognathae; Galliformes; Phasianidae; Phasianinae; Gallus.
</taxonomy>
<sequence length="147" md5="e485279d5bf21caa3b7967c566c68a5a" crc64="81E85743FF579468" version="1">
MRSLLILVLCFLPLAALGKVFGRCELAAAMKRHGLDNYRGYSLGNWVCAAKFESNFNTQATNRNTDGSTDYGILQINSRWWCNDGRTPGSRNLCNIPCS
ALLSSDITASVNCAKKIVSDGNGMNAWVAWRNRCKGTDVQAWIRGCRL
</sequence>
<matches>
<match accession="PF00062" id="Lys" type="Pfam-A">
<location start="19" end="145" ali_start="19" ali_end="145" hmm_start="1" hmm_end="125" evalue="2.8e-52" bitscore="185.70"/>
</match>
</matches>
</entry>
</pfam>
```

The most important lines are like these:

```
<match accession="PF00062" id="Lys" type="Pfam-A">
```

This holds the Pfam family accession (PF00062) and name (Lys). The final entry (type) indicates the Pfam section; either Pfam-A (curated and most reliable) or Pfam-B (automated). The full description of this interface (part of the 'RESTful' interface) can be found in the Pfam help pages: `http://pfam.sanger.ac.uk/help#tabview=tab10`. A nice extension of this part is to also look at the level of Pfam clans (groups of related families). The definition of the Pfam clans is given in a simple text-format file: `ftp://ftp.sanger.ac.uk/pub/databases/Pfam/current_release/Pfam-C.gz`.

5 Practical 4: SCOP

To retrieve SCOP family IDs for a protein, you need PDB identifiers. There are several ways to get them from your UniProt ID, but for this practical we have provided you with a UniProt Id - PDB Id lookup table (SCOP_selection_uni-pdb.tab). Once you have the PDB ids, you can look for the SCOP classification for your proteins using the flat-file 'parseable' version of the SCOP classification available for download from the SCOP website:
`http://scop.mrc-lmb.cam.ac.uk/scop/index.html`.
The full description of the SCOP search engine interface can be found at:
`http://scop.mrc-lmb.cam.ac.uk/common-to-all/release-notes.html#search-engine`.

6 Appendix

6.1 URL post in python

```
# Defaults
dbName = 'uniprot'
entryId = 'wap_rat'
format = None

# Construct URL
baseUrl = 'http://www.ebi.ac.uk/Tools/webservices/rest/dbfetch'
url = baseUrl + '/' + dbName + '/' + entryId
if format != None:
    url += '/' + format

# Get the entry
fh = urllib2.urlopen(url)
result = fh.read()
fh.close()

# Print the entry
print result
```

6.2 Using BLAST via URLAPI

There are three steps involved in performing BLAST searches and retrieving results using the URL. First, we have to supply a sequence via an accepted sequence ID or FASTA sequence. As we will use IDs only, it is possibly to use the entire BLAST system using URLs only. Second, we have to wait for the completion of BLAST. Third, we can retrieve the results in one of several formats. Below, an example BLAST via the URLAPI is described.
Full documentation is available at the NCBI:
`http://www.ncbi.nlm.nih.gov/staff/tao/URLAPI/new/index.html`

6.2.1 Step 1: Starting a protein BLAST

BLAST is started using a **PUT** command. PUT is a HTTP command to put data on a server. A web browser generally uses GET to retrieve a information (e.g. if you do a click on a link, you GET the file). As an example, we use APOE as a query sequence and swissprot as the database. First, we have to know how to provide the sequence. APOE has a SwissProtUniProt entry: APOE_HUMAN, a SwissProt ID (P02649), and a GI number (114039). GI numbers are provided by the NCBI. For the BLAST at NCBI we can use sequences (FASTA format) but also identifiers. The protein BLAST is started with this URL command. You can paste the URL above into your browser and see what happens:

```
http://blast.ncbi.nlm.nih.gov/Blast.cgi?
CMD=Put
&QUERY=P02649
&DATABASE=swissprot
```

```
&PROGRAM=blastp
&FILTER=T
&EXPECT=100
&FORMAT_TYPE=Text
```

The QUERY can be a SwissProt ID or GI number, but not a SwissProt entry name. It is also possible to use sequence (in fasta format) directly as query, for example:
If you want to perform BLAST search on the following fasta file:

```
>test_seq
HVLHAKHP
```

Then your query parameter is as follows:

```
&QUERY=%3Etest_seq%0D%0AHVLHAKHP
```

Note that the >and the new line characters are replaced by %3E and %0D%0A, respectively. This is necessary because many characters are unsafe if used within URL, therefore they have to be properly escaped. For more details see:
`http://www.ncbi.nlm.nih.gov/staff/tao/URLAPI/new/node101.html`
The DATABASE is swissprot, the program blastp to protein BLAST, the low-complexity FILTER is True (turned on) and the E-value threshold (EXPECT) 100.

6.2.2 Step 2: Checking your BLAST progress and retrieving the result

The HTML page that is returned contains a so-called Request ID (RID)and an indication when your BLAST might be finished (Request Time of Execution; RTOE):

```
<!--QBlastInfoBegin
    RID = 7WS3FMZW01P
    RTOE = 11
QBlastInfoEnd
-->
```

The RID uniquely identifies each individual search and is valid for 36 hours. It is a mandatory parameter for the second 'Get' or formatting step.The estimated 'Request Time of Execution' (RTOE) for the search is given in seconds. We should not just make a loop while loop until BLAST is finised, but should wait in the loop, preferably obeying the RTOE times! When we BLAST programmatically, we should get the info explained above from the HTML page. We now just GET the result, but have to check if it is finished too:

```
http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?
CMD=Get&
RID=7WU424SH01N&
FORMAT_OBJECT=Alignment&
ALIGNMENT_VIEW=Tabular&
FORMAT_TYPE=Text&
ALIGNMENTS=5
```

The URL to *get* our results clearly has to contain a Get command (CMD) and our identifier (RID). It may contain other parameters to indicate what we want and in which format (for details see the URLAPI docs at NCBI). Here, we select the BLAST Hittable output, also known as Tabular output, in plain text. Additionally, we only wish the see the first 5 ALIGNMENTS (hits). NOTE that parameters have DEFAULT values.
If the result is not ready we need to try to get the results again. The page might contain:

```
<!--
QBlastInfoBegin
    Status=WAITING
QBlastInfoEnd
-->
```

If the search is completed and the result is ready, the output will contain the formatted result with Status=READY contained in comment lines above. If your search result has the status of 'WAITING', it is advisable to wait for at least one minute before trying the retrieval again. Other possible values for the status are ERROR, FAILED, UNKNOWN, or empty.

6.2.3 Step 3: Read the result

The URL in step 2 will retrieve the first 5 hits in so-called Hit-table format. The first lines contain information on the BLAST run after #, including the BLAST program and the full description line of the query sequence. Note that you can also view these BLAST results on the web (http://blast.ncbi.nlm.nih.gov/) using your RID number. It is also possible to (re)format to the results to get other BLAST report formats, such as Pairwise.