Lecture Note 3 for MBG 404

Formats

Not all results are equal and thus the output of different programs will look significantly different as well. The redirection that was introduced above created a plain text file representing the information as text.

Binary

Files that only display cryptic information when viewed in a text editor such as notepad or wordpad (try notepad++ and see if you like it more than other text editors, http://notepad-plus-plus.org/) are probably binary files. Such files to not store characters, they may store information in different format although they could store characters but in such a different way that other text editors will not understand. Microsoft Word's doc format is an example for that. Although text is stored, the information contained in the file cannot be read with a plain text editor. This is due to a large amount of formatting controls in MS Word which need to be stored in the file, if they were applied. Another issue here is that MS is not adhering to standards like the open document format which could be read in a plain text editor.

Text

Plain text files are easy to handle and there are many editors to edit them as well. They are universally recognized except for different line terminators which can lead to formatting errors if you use the same files on different operating systems with different editors.

FASTA

FASTA is a special format of a plain text file which introduces a piece of information by a right angle bracket '>' followed by the identifier for that piece of information. The actual data starts on the following line in the text file and continues to the end of the file or to the next right angle bracket.

```
>Gene1
CAGCACTGACTGCGCATCGATAGTCAGTAGCTAGCTAGCTAGCT
ACACGTAGCTGCAGTACTGACTGATCGATCGATGCTAGCTAGCT
CAGCGAT
>Gene2
GTGEGTCTGACTGATCGATCGATGCCTGACTGATCGATCGATGC
ACGTCTGACTGATCGATCGATGCCTGACTGATCGATCGATGCGT
GGGGGTGT
```

This is an example for a FASTA file containing two genes. This approach adds some structure to the text file and that enables programs to accept FASTA as input. Unstructured data is impossible to handle with a program not specifically designed for that type of data. Structured data is easy to handle which is why the extensible markup language was introduced.

XML

e**X**tensible **M**arkup **L**anguage (XML) is an approach to completely structure the information contained in a text file. Each bit of information needs to be described in order to produce valid XML.

```
<sequence id=”Gene1”>
CAGCACTGACTGCGCATCGATAGTCAGTAGCTAGCTAGCTAGCT
ACACGTAGCTGCAGTACTGACTGATCGATCGATGCTAGCTAGCT
CAGCGAT
</sequence>
```

In this example the general idea of XML can be seen. Each piece of information is named. The sequence is within a so called element which consists of an opening and a closing tag.

```
<sequence id=”Gene1”>    ← Opening tag
CAGCACTGACTGCGCATCGAT                      }
ACACGTAGCTGCAGTACTGAC                      }  Element
CAGCGAT                                    }
</sequence>              ← Closing tag
```

The opening tag can also contain attributes which come as name, value pairs. Here the name of the attribute is id and the value is Gene1. Both files, FASTA and XML, can contain the same information but the second one can easily be extended with additional data in a well structured manner. Whereas the FASTA cannot easily be extended.
Clearly XML is more complex than displayed here and some information needs to be provided in each XML file. Furthermore, XML is hierarchical and therefore the sequence element cannot stand by itself as shown in the example. It needs to be enclosed in at least the root element.

```
<?xml version=”1.0”>
<MySeqs>
     <some_info info=”42”/>
     <sequence id=”nogene”>THE SEQUENCE</sequence>
</MySeqs>
```

The structure above shows how an XML file could be organized. Left angle brackets followed by question marks are used to introduce commands to the interpreter of the XML file (e.g. some web browser). This is used to specify the version of XML that this file conforms to in this case. Character sets and other information such as validation files can be presented in that way as well.
Following is the root tag (which can take any name, of course). When the closing counterpart of the root tag is encountered, all information has been processed. This also means that all data need to be enclosed within the root tag. The some_info element does not have a matching closing tag. Since it does not contain other elements and no text information, the closing tag can be abbreviated as /> instead of </some_info>. The sequence element is the same as used earlier.
This is a complete working XML file that you could display with any web browser for example.

ASN.1

This format is again storing information in plain text but the information is structured as in the case of XML but completely different. NCBI uses this format to present information as for example gene bank records. An example is seen below.

1: NC_003622. Reports Grapevine chrome ...[gi:16117]

```
Seq-entry ::= set {
  level 1 ,
  class nuc-prot ,
  release "" ,
  descr {
    source {
      genome genomic ,
      org {
        taxname "Grapevine chrome mosaic virus" ,
        db {
          {
            db "taxon" ,
            tag
              id 12273 } } ,
        orgname {
          name
            virus "Grapevine chrome mosaic virus" ,
          lineage "Viruses; ssRNA positive-strand viruses, no DNA stage;
 Picornavirales; Comoviridae; Nepovirus; Subgroup B" ,
          gcode 1 ,
          div "VRL" } } ,
      subtype {
        {
          subtype segment ,
          name "RNA 1" } } } ,
    user {
      class "Genomes" ,
      type
        str "info" ,
      data {
        {
          label
            id 1 ,
          data
            int 16117 } } } ,
    user {
      type
        str "GenomesInfo" ,
      data {
        {
          label
```

Figure 1: Section of an ASN.1 file containing information about Grapevine chrome mosaic virus from NCBI.

JSON

JSON is an alternative to XML which you will see frequently in the future. Below you see some data concerning a person. In the case of JSON there is no concern of opening and closing tags. This is modeled using curly braces "}{". Following person the data enclosed in the opening curly brace until the end, the closing curly brace, is the information about this person. When comparing JSON and ASN.1 it can be seen that they are related.

```
person = {
  "name": "Some Willie",
  "age": 45,
  "height": 1.85,
  "urls": [
    "http://someone.net/",
    "http://www.flickr.com/photos/someone/",
    "http://someone.incognito.com/"
  ]
}
```

I will not delve into discussions trying to determine which of the formats may be better or worse instead I propose that each of them will eventually find their niche. This is evolution at its best. May the fittest survive.

Other

File formats are numerous and many programs have their own specific format. Many programs can however accept different types of input and can also produce different types of output. As long at the format is based on plain text it can be handled easily. If the file is in binary format, there may be problems when the program is no longer available. Therefore it is always a good idea to have the results in a format that is supported by a variety of programs.

Console Applications

Many programs in biology are programmed by biologists to achieve a specific task. These programs have not been developed with user friendliness in mind. They were rather developed task centered with the developer in mind (needs to finish the PhD). Developing console applications without graphical user interfaces can be achieved much quicker with a focus on the correctness of the program instead of with a focus on what the user could do wrong. Due to the overhead incurred by producing graphical user interfaces and the follow-up problematic of validating the extended input, many programs are delivered as command line interfaces. Often the documentation for these programs is reduced to a bare minimum which may be difficult to understand. Remember that the purpose of the program usually is to processes some collected data in a highly specific manner. This may initially only be interesting for a very small number of users which can be instructed by the developer so there is no need for large amount of documentation and fancy GUIs. Once the field expands this changes but often the programs remain unchanged.
Console applications can also have an advantage for us. They can be automated and thus be used to batch process large amounts of data and can further be assembled to form computational pipelines as we will see in the next chapter.

File Addressing

Whenever you need to specify the location of a file there are several options that you have. One option is to copy all files necessary into the same folder as the program and then specify (call) the files by only using their names and extension. For larger files that are used frequently, this may not be a good choice. There are two ways commonly used to avoid copying files around and potentially cluttering your computer with many unused copies of the same file.

Absolute Addressing

The first type of specifying file paths provides the **complete path** of the file to the program.

Windows

A path starts with the drive letter followed by a colon ':' and a backslash '\' or a forward slash '/'. Following all directories that the file is contained in are listed separated by back- or forward slashes. Finally, the file name consisting of **file title** and **file extension** needs to be specified.

```
C:\windows\temp\test.txt
```

C: is the drive letter here. The file with the title text and the extension txt (**file name**: test.txt) is located in the temp folder which is a folder in the windows directory. One more problem arises when the file path contains spaces. Remember that parameters and switches are separated by spaces. Therefore, a file path which contains spaces, 'Documents and Settings' for instance, would produce a problem since it would be resolved to three different parameters, one with the value *Documents*, one with and *and* the final one with the value *Settings*. To avoid this problem, such file paths can be enclosed in quotation marks.

```
"C:\Documents and Settings\jens\grades.xls"
```

This is a file located in my documents containing the grades of the students currently enrolled in computational biology. This way of specifying files may not be very convenient since file paths can become rather long quickly. Relative addressing can solve the problem in some cases.

Relative Addressing

A relative address is like directions you would give to someone that needs the way to the cinema. You start explaining from the current location until the destination is sufficiently described. The same is true for **relative addressing** in the console.

Work Directory/ Current Directory

The **current directory** is specified in front of the command prompt (here: >)[a].

```
C:\windows\temp>
```

Here for instance the current directory is a subdirectory of the windows directory called temp. Both located on drive C. The work directory may or may not be the same as the current directory. The first assumption can be however that the current directory is also the directory where you will find output of programs that you will run in the console. If that is not the case

then either the work directory was changed by the program using a parameter that you passed to the program or the program simply set its directory to the work directory. In that case check the parameters and switches you set for the program and check whether output was generated in the directory enclosing the program that you used. The current directory will in the following often be abbreviated to '?>' where the question mark stands for the absolute path to your current directory.

Directions

There are only two possible operations:

1. Go into a child directory
2. Go into the parent directory.

To direct the path into a child directory its name needs to be specified. To direct the path into the parent directory '..\' or '../' can be used. In the following examples we will assume to be in the directory

```
C:\programs\blast\bin\
```

The file we want to execute is *BLAST.EXE* which needs the path to the database that is too be searched. The database is located in:

```
C:\databases\
```

And here is how we tell blast to use that database using relative addressing:

```
blast.exe ../../../databases/nr.fasta
```

Since we are in the bin directory and since it contains *BLAST.EXE* we do not need to specify a path to *BLAST.EXE*. The only parameter we are concerned with in this simplified example is the path to the database. This path can be accessed by going into bin's parent directory (../) then going into blast's and finally into programs' parent directory. Now we need to progress into sub directories. Here we can find the nr.fasta file in the databases directory.

Essential Console Commands

There are too many console commands and in the context of this course we will mostly use the ones listed below. Of those dir and cd will be used most often.

help	Lists all available console commands in case you forgot … or never new them.
dir	Lists files and folders in the current directory
cd	Changes current directory by using relative or absolute addressing May not be used to switch between drives
del filepath	Deletes the specified file
mkdir name	Creates a new folder entitled name in the current directory
copy src dest	Copies a file from a source to a destination (several sources may be combined by using '+')
rename org new ren org new	Renames a file or folder from its original name to a new name that you provide. Can also be used to change file extensions. Batch modes are possible. Rename *.txt *.rtf for instance changes all file extension from txt to rtf in the current directory.
find	Can find text in files e.g.: find "test" c:*.txt will look at all files on the C drive and show the lines in text files that contain 'test'
fc	Compares two files and displays the differences

Example Programs

The list of console applications is rather long so it is difficult to pick out programs as examples. PepNovo was chosen since the number or parameters is rather short and it can thus quickly be successfully executed. Blast was chosen to show that it can be much more complex and that it often needs several programs that need to be used to achieve a result.

PepNovo

PepNovo [1] is a program which performs de novo sequencing of tandem mass spectra. which aims to assign a sequence to an MS/MS spectrum without using information apart from the measured spectrum such as databases. PepNovo, by Frank and coworkers, is an example for a console application which echoes the results to screen. As we can use redirection of pipes it is possible to save or append the results to a text file.
PepNovo needs an MS/MS spectrum as input (the path to it of course) which should be introduced with the –dta switch. It needs another file as input which contains the mathematic model for the mass spectrometer which produced the MS/MS spectrum. There are further options which you can review by starting the program from the console without giving any parameters.

Parameters and Switches

-dta [path to dta file]
-list [path to text file containing a list of paths to dta files]
-model [path to a file containing the mathematical model for the specific environment]
-num_tags [How many tags should be generated]

-non_tryptic [The peptides submitted to the mass spectrometer were not tryptically digested]
-tag_length [Number of amino acids that should make up a tag (3-6)]

Example

```
?>PepNovo_W32.exe -dta MSMSSpectrum.dta -model tryp_model.txt
```

Download

http://www-cse.ucsd.edu/groups/bioinformatics/software.html#pepnovo
http://mbg403.allmer.de/tools/pn.zip[b]

BLAST

The basic local alignment search tool [2] by Altschul and coworkers is an algorithm to find a sequence in another sequence. It is one of the tools with the highest impact on biology and bioinformatics. It basically points out all exact and those approximate matches which pass a user set threshold for a query in a library of sequences. It is mostly used online where there are numerous servers that provide computational resources and a convenient graphical user interface (web page). These interfaces are however limited in several regards which makes it sometimes necessary to run *BLAST* locally. Downloading *BLAST* reveals that it is not just one program but a collection of programs. A list of the programs contained in the *BLAST* package is seen on the right. Not all programs are needed in all circumstances. For instance, if sequences are available in the correct format, then *formatdb.exe* is not needed.

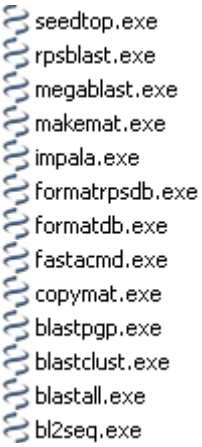

Figure 2: blast package

Download

http://www.ncbi.nlm.nih.gov/BLAST/download.shtml

Make BLAST DB

This program (makeblastdb.exe) comes as part of the *BLAST* package and shall be examined first. Its purpose is to format a sequence database such that they are compatible with the format that *BLAST* expects. Some inputs are expected such as the actual file that needs to be converted. More information is in the documentation of the blast package which came with the download.

Help output (makeblastdb.exe –help)

USAGE
makeblastdb.exe [-h] [-help] [-in input_file] [-dbtype molecule_type]
[-title database_title] [-parse_seqids] [-hash_index]
[-mask_data mask_data_files] [-gi_mask]
[-gi_mask_name gi_based_mask_names] [-out database_name]
[-max_file_sz number_of_bytes] [-taxid TaxID] [-taxid_map TaxIDMapFile]
[-logfile File_Name] [-version]

```
DESCRIPTION
   Application to create BLAST databases, version 2.2.24+

OPTIONAL ARGUMENTS
 -h
   Print USAGE and DESCRIPTION;  ignore other arguments
 -help
   Print USAGE, DESCRIPTION and ARGUMENTS description;  ignore other arguments
 -version
   Print version number;  ignore other arguments

 *** Input options
 -in <File_In>
   Input file/database name; the data type is automatically detected, it may
   be any of the following:
         FASTA file(s) and/or
         BLAST database(s)
   Default = `-'
 -dbtype <String, `nucl', `prot'>
   Molecule type of input
   Default = `prot'

 *** Configuration options
 -title <String>
   Title for BLAST database
   Default = input file name provided to -in argument
 -parse_seqids
   Parse Seq-ids in FASTA input
 -hash_index
   Create index of sequence hash values.

 *** Sequence masking options
 -mask_data <String>
   Comma-separated list of input files containing masking data as produced by
   NCBI masking applications (e.g. dustmasker, segmasker, windowmasker)
 -gi_mask
   Create GI indexed masking data.
    * Requires:  parse_seqids
 -gi_mask_name <String>
   Comma-separated list of masking data output files.
    * Requires:  mask_data, gi_mask

 *** Output options
 -out <String>
   Name of BLAST database to be created
   Default = input file name provided to -in argumentRequired if multiple
   file(s)/database(s) are provided as input
 -max_file_sz <String>
   Maximum file size for BLAST database files
   Default = `1GB'
```

*** Taxonomy options
-taxid <Integer, >=0>
Taxonomy ID to assign to all sequences
* Incompatible with: taxid_map
-taxid_map <File_In>
Text file mapping sequence IDs to taxonomy IDs.
Format:<SequenceId> <TaxonomyId><newline>
* Incompatible with: taxid
-logfile <File_Out>
File to which the program log should be redirected

Documentation information

This application serves as a replacement for formatdb.
4.6.8.1 in: Input file or BLAST database name to use as source; the data type is automatically detected. Note that multiple input files/BLAST databases can be provided, each must be separated by white space in a string quoted with single quotation marks. Multiple input files/ BLAST databases which contain white space in them should be quoted with double quotation marks inside the white space-separated, single quoted string (e.g.: `-in '"C:\My Documents \seqs.fsa" "E:\Users\Joe Smith\myfasta.fsa"'`).
Page 12
BLAST Command Line Applications User Manual
BLAST Help BLAST Help BLAST Help BLAST Help
4.6.8.2 title: Title for the BLAST database to create
4.6.8.3 parse_seqids: Parse the Seq-id(s) in the FASTA input provided. Please note that this option should be provided consistently among the various applications involved in creating BLAST databases. For instance, the filtering applications as well as convert2blastmask should use this option if makeblastdb uses it also.
4.6.8.4 hash_index: Enables the creation of sequence hash values. These hash values can then be used to quickly determine if a given sequence data exists in this BLAST database.
4.6.8.5 mask_data: Comma-separated list of input files containing masking data to apply to the sequences being added to the BLAST database being created. For more information, see Masking in BLAST databases and the examples.
4.6.8.6 out: Name of the BLAST database to create.
4.6.8.7 max_file_sz: Maximum file size for any of the BLAST database files created.
4.6.8.8 logfile: Name of the file to which the program log should be redirected (stdout by default).
4.6.8.9 taxid: Taxonomy ID to assign to all sequences.
4.6.8.10 taxid_map: Name of file which provides a mapping of sequence IDs to taxonomy IDs.

FormatDB

This program comes as part of an older *BLAST* package and serves the same purpose as makeblastdb.exe.

Parameters and Switches

-t Title for database file [String] Optional
-i Input file(s) for formatting [File In] Optional
-l Logfile name: [File Out] Optional
default = formatdb.log
-p Type of file [T/F] Optional
default = T
-o Parse options [T/F] Optional
T - True: Parse SeqId and create indexes.
F - False: Do not parse SeqId. Do not create indexes.
default = F

-a Input file is database in ASN.1 format (otherwise FASTA is expected)
[T/F] Optional
default = F
-b ASN.1 database in binary mode [T/F] Optional
T - binary,
F - text mode.
default = F
-e Input is a Seq-entry [T/F] Optional
default = F
-n Base name for BLAST files [String] Optional
-v Database volume size in millions of letters [Integer] Optional
default = 4000
-s Create indexes limited only to accessions - sparse [T/F] Optional
default = F
-V Verbose: check for non-unique string ids in the database [T/F] Optional
default = F
-L Create an alias file with this name
use the gifile arg (below) if set to calculate db size
use the BLAST db specified with -i (above) [File Out] Optional
-F Gifile (file containing list of gi's) [File In] Optional
-B Binary Gifile produced from the Gifile specified above [File Out] Optional
-T Taxid file to set the taxonomy ids in ASN.1 deflines [File In] Optional

Example

```
?>formatdb –i proteins.fasta –n chlre3
```

References

[1] A. Frank and P. Pevzner, “PepNovo: de novo peptide sequencing via probabilistic network modeling,” *Anal Chem*, vol. 77, Feb. 2005, pp. 964-73.
[2] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, “Basic local alignment search tool,” *J Mol Biol*, vol. 215, Oct. 1990, pp. 403-10.

Notes

a. The command prompt can also be different or can even be changed. Check the *PROMPT* command in your console to find out how.
b. For the version of PepNovo mentioned in this text you need to download from here.