This document contains the installation instructions for the Proteomics Pipeline developed by Ritesh Krishna and Andy Jones at University of Liverpool, UK.

Subject – Proteomics Pipeline Documentation

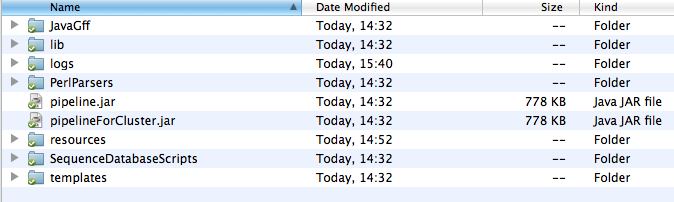
Date – Dec 1, 2011

Author – Ritesh Krishna

Email – [Ritesh.Krishna@liverpool.ac.uk](mailto:Ritesh.Krishna@liverpool.ac.uk)

**Directory structure of the pipeline**

The directory structure of a base distribution should have following directories and files. A screenshot is provided below-



* **JavaGFF** – Contains a jar file for summarizing the results in a single file. The pipeline outputs a separate directory for each input MGF file. The jar can be called after running the pipeline on the whole dataset to travel through each output directory and gather the results in a single output summary file.
* **lib** – Contains some dependent libraries for running of the pipeline
* **logs** – Contains log files for logging the events while running the pipeline. The logging for the pipeline has been done using the log4j mechanism, which allows the user to change the logging behaviour according to their need. The log4j properties file is present in the resources folder.
* **PerlParsers**  - Contains Perl source code for converting files to mzIdentML format.
* **resources** – Contains the information regarding the locations of external executables, log property file etc.
* **SequenceDatabaseSripts** – Contains Perl scripts for creating a Target-Decoy database for Proteomics search engines.
* **Templates** – Contains the templates with placeholders to create Omssa/X!Tandem search engine commands on the fly.
* **Pipeline.jar** – Jar file that will be called to execute the whole pipeline for a collection of input MGF files stored in a local directory.
* **PipelineForCluster.jar** – Jar file that will be called to execute the whole pipeline for a *single* MGF file. In a clustered environment, it is recommended to use this jar for executing the pipeline. It will be easier to provide each processing node an instance of the pipeline and a MGF for processing. After the processing is complete for all the MGF files, we can use the code in JavaGFF to create a final output summary file.

**Prerequisites**

The following software/libraries are required for running the pipeline.

* 1. OMSSA – The binaries can be downloaded from <http://pubchem.ncbi.nlm.nih.gov/omssa/download.htm>

The pipeline has been tested for

* + - omssa-2.1.9 version on Mac OS, and
    - omssa-2.1.7 version on Linux.

It should be noted that OMSSA has some kind of bug in interpreting

the description lines in Fasta files, so it’s safer to stick to 2.1.7

version on Linux. The installation simply requires unpacking of the

tar.gz file in a folder in a preferred location, as for example, in /opt

or /home/username.

* 1. X!Tandem – The binaries can be downloaded from

<ftp://ftp.thegpm.org/projects/tandem/>

The pipeline has been tested for

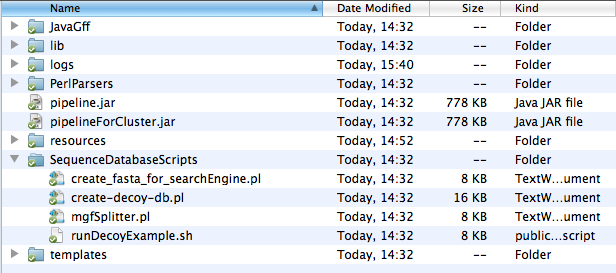
* tandem-10-01-01-4 version on Mac OS, and
* tandem-08-02-01-3 version on Linux

The installation simply requires unpacking of the compressed file in a preferred location.

* 1. NCBI Blast libraries – Omssa requires the tool ***formatdb*** (in older distributions)or ***makeblastdb*** (in newer distributions, formatdb is no more available, and has been replaced by makeblastdb). We used makeblastdb on Mac os, and formatdb on Linux.
  2. Java Runtime Version 1.6

**Setting up the pipeline**

* **Creation of the Fasta database** – In order to run the search engines, we need to setup a Fasta database. The database should contain the annotated protein sequences, and the fictional decoy sequences. The merging of decoy sequences with the actual sequences helps us in accessing the validity of the search results. There are various ways to create decoy sequences; we have used random sequence generators for this pipeline. The file to create decoy database is in ***SequenceDatabaseScripts*** folder.



The script create\_fasta\_for\_searchEngine.pl needs to be called at the command line in the following way

*$perl create\_fasta\_for\_searchEngine.pl ~/Ritesh\_Work/Toxo/ToxoDB/TgondiiME49\_ToxoDB-6\_2.fasta 3 ~/Ritesh\_Work/Toxo/ToxoDB/TgondiiME49\_ToxoDB-6\_2\_withdecoy.fasta*

where,

* 1. *~/Ritesh\_Work/Toxo/ToxoDB/TgondiiME49\_ToxoDB-6\_2.fasta* = The annotated Fasta file.
  2. *3* = The decoy ratio, i.e, how many random sequences to produce for each actual sequence.
  3. *~/Ritesh\_Work/Toxo/ToxoDB/TgondiiME49\_ToxoDB-6\_2\_withdecoy.fasta* = The output file produced
* **Blast formatting** - We need to blast-format the produced Fasta file using *formatdb* or *makeblastdb,* depending on the platform we are using.

When using makeblastdb –

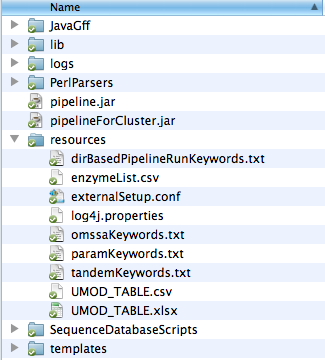
*$/usr/local/ncbi/blast/bin/makeblastdb -in TgondiiME49\_ToxoDB-6\_2\_withdecoy.fasta -dbtype prot*

When using formatdb –

*$/usr/local/ncbi/blast/bin/formatdb -i TgondiiME49\_ToxoDB-6\_2\_withdecoy.fasta -o*

This formatting should produce additional files with suffixes .pin,.psq etc. in the current directory.

* **Splitting the MGF files in smaller chunks (if needed)** – There is a Perl script MGFSplitter.pl to split the input MGF files into smaller sub-files. In cases where input MGF files are large and could lead to memory-intensive problems, this script should be a handy tool.
* **Specifying the path of the search engine executables** - Once the database has been created and blast formatted, we need to specify the location of OMSSA and X!Tandem executables in the pipeline. The locations of the executables can be specified in the file names **externalSetup.conf** in the resources directory (see the screenshot below)



The following lines in the **externalSetup.conf** need to be changed to reflect the actual location of the executables. The text in red indicates the fields to be changed –

omssa\_executable = /opt/omssa-2.1.9.macos/omssacl

tandem\_executable = /opt/tandem-osx-intel-10-01-01-4/bin/tandem

**Setting the search parameters**

At this point, we expect the pipeline to be set for running. Next, we need to specify the search criteria for running the pipeline. The directives for the search engines can be specified in the files inputFileTemplate.txt and mzIdentMLParser\_inputFile.txt in the inputFiles\_Example folder shown below.

These files can exist anywhere on the file system.



The contents of **mzIdentMLParser\_inputFile.txt** is here –

*user\_name = Ritesh*

*decoy\_regex = Rnd*

*decoy\_ratio = 3*

*rank\_threshold = 3*

* 1. decoy\_regex = Rnd, is the tag used for creating decoy sequences while creating the decoy database.
  2. decoy\_ratio = 3, indicates the ratio we used for creating number of decoy sequences for each actual sequence
  3. rank\_theshold = 3, indicates that we are interested in only top 3 hits for peptide spectrum matches reported by the search engines.
  4. user\_name = Ritesh, is a placeholder.

The content of **inputFileTemplate.txt** is presented here; the contents that can be changed to specify search criteria are highlighted.

*input\_file= -fm {{ input\_file\_path }}*

*fasta\_file= -d ~/ToxoDB/TgondiiME49\_ToxoDB-6\_2\_withdecoy.fasta*

*enzyme\_name = -e 0*

*product\_tolerance= -to 0.8*

*precursor\_tolerance= -te 1.5*

*fixed\_mod\_id= -mf 3*

*variable\_mod\_id= -mv 1*

*missed\_cleavages = -v 1*

*output\_file= -oc {{ output\_dir\_path }}/testrun\_osx.csv*

*species=Toxoplasma*

*default\_input\_file=/opt/tandem-osx-intel-10-01-01-4/bin/default\_input.xml*

*taxonomy\_file={{ output\_dir\_path }}/taxonomy.xml*

Description of each changeable line is –

* + - *fasta\_file* -- Name of the blast formatted fasta database
    - *enzyme\_name* – 0 represented Trypsin. The enzymes are specified by their numberic representation as specified by Omssa search engine. The numeric representation of other enzymes can be see by running - *$omssacl –el*
    - *product\_tolerance –* User specified number
    - *precursor\_tolerance -* User specified number
    - *fixed\_mod\_id –* Fixed modifications are represented by numeric representation of mods by Omssa. The list of the modifications and their respective numeric identifiers can be seen by running - *$omssacl –ml*
    - *variable\_mod\_id* – Variable modifications, same as above.
    - *missed\_cleavages* – Number of maximum missed cleavages allowed
    - *species –* Any name is allowed
    - *default\_input\_file –* The location of default\_input.xml for X!Tandem search engine. The file is usually present in the bin directory of the X!Tandem installation.

**Running the pipeline**

The pipeline can be run in two modes – i) Whole directory mode, where an input directory of multiple MGF files is provided and the pipeline can iteratively process each MGF file and produce output, ii) Single file mode, where the pipeline will be run for a single MGF file provided.

The pipeline can be run by changing to the root directory, or by declaring an environment variable for the installation folder of the pipeline.

**Whole directory mode** – The pipeline can be run in this mode by making a call to pipeline.jar with the following arguments –

*$ java -jar pipeline.jar inputFiles/inputFileTemplate.txt ~/Ritesh\_Work/Toxo/Toxo\_Test\_MSDataset ~/Ritesh\_Work/TestSpace/pipeline\_test/ = inputFiles/mzIdentMLParser\_inputFile.txt =*

The arguments are as following –

* *inputFiles/inputFileTemplate.txt* = location of the inputFileTemplate.txt which contains the criteria for search engines
* *~/Ritesh\_Work/Toxo/Toxo\_Test\_MSDataset*  = location of the folder which contains MGF files
* *~/Ritesh\_Work/TestSpace/pipeline\_test/* = location of the folder where the results will be stored
* = is a placeholder
* *inputFiles/mzIdentMLParser\_inputFile.txt* – location of the mzIdentMLParser\_inputFile.txt which contains the tags to identify random sequences, decoy ratio and rank threshold.
* = is a placeholder

**Single file mode** – This mode is most suited for parallel environments where each node can be independently provided with an instance of the pipeline and an input MGF file for processing. The single file mode can be run by making a call to pipelineForCluster.jar with the following arguments –

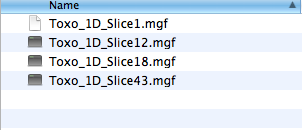
*$java –jar pipelineForCluster.jar inputFiles/inputFileTemplate.txt ~/Ritesh\_Work/Toxo/Toxo\_Test\_MSDataset/Toxo\_1D\_Slice43MGF ~/Ritesh\_Work/TestSpace/pipeline\_test/ inputFiles/mzIdentMLParser\_inputFile.txt*

The arguments are as following –

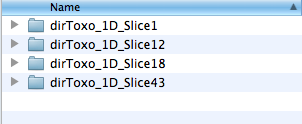
* *inputFiles/inputFileTemplate.txt* = location of the inputFileTemplate.txt which contains the criteria for search engines
* *~/Ritesh\_Work/Toxo/Toxo\_Test\_MSDataset/Toxo\_1D\_Slice43MGF*  = location of the input MGF file
* *~/Ritesh\_Work/TestSpace/pipeline\_test/* = location of the folder where the results will be stored
* *inputFiles/mzIdentMLParser\_inputFile.txt* = location of the mzIdentMLParser\_inputFile.txt which contains the tags to identify random sequences, decoy ratio and rank threshold.

**Output from the pipeline**

The pipeline produces a number of files for *each* input MGF file. In the *whole directory mode*, all the input MGF files can be put in a single directory, and the directory can be specified as input directory while calling the pipeline (in the above section we had the directory *~/Ritesh\_Work/Toxo/Toxo\_Test\_MSDataset* with all the input MGF files.). A screenshot of the input directory with four MGF files is given below –

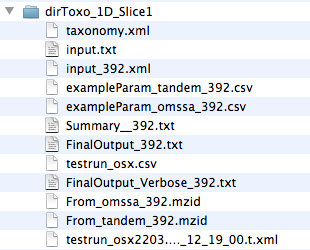


The pipeline will create four directories in the output directory specified (in the previous section - *~/Ritesh\_Work/TestSpace/pipeline\_test/*  was the output directory mentioned) as shown below. Each directory contains files specific to a given input MGF file –



Each directory contains the same number of files. Similarly, in the *single file mode*, there will be only one input file and a corresponding output directory after the execution of the pipeline.

The pipeline runs in different stages, and each stage produces a set of files. A typical structure of the output directory (dirToxo\_1D\_Slice1) is shown in the following screenshot –



**Description of the files in the above directory –** Each output directory should have 12 files as shown in the screenshot above. If we are missing some files, it means that the pipeline is failing at some stage and the logs must be checked to find the break point. The pipeline runs in four stages, and the output files are grouped according to the stages. We have appended a random number (in this example *392*) to most of the files produced, this mechanism was introduced as a identify the set of files while belonged together, but is redundant now and can be dropped in the later versions.

**Stage 1 - Call the search engines Omsaa and X!Tandem.**

The files used while calling the search engines are –

input\_392.xml

input.txt

taxonomy.xml

The files produced by Omssa is – testrun\_osx.csv

The file produced by X!Tandem is – testrun\_osx2203.2…….xml

**Stage 2 – Convert the outout files produced by search engines to a standard MzIdentML format**

The Omssa output file *testrun\_osx.csv*  is converted to From\_omssa\_392.mzid

The X!Tandem output file testrun\_osx2033.2…….xml is converted to From\_tandem\_392.mzid

The intermediate files used were - exampleParam\_omssa\_392.csv and exampleParam\_tandem\_392.csv.

The mzIdentML files can be used for extract other information as well using an independent parser developed in collaboration with European Bioinformatics Institute. The java based parser can be found at - (<http://code.google.com/p/jmzidentml/>)

**Stage 3 – Call the multiple search engine consensus algorithm to find the consensus between the search engines and correct the FDR score**

The files produced at this stage are –

FinalOutput\_392.txt

FinalOutput\_Verbose\_392.txt

**FinalOutput\_Verbose\_392.txt** – Contains two sections with headings, *X!Tandem***,** and, *Omssa***.** The **individual search results** are reported in this file.

All the peptides reported by X!Tandem and Omssa are listed under their respective headings. This is a CSV file where each row reports the information about a peptide and Protein found by the search engine. The columns in the file are –

* Line number
* Spectrum Identifier
* Peptide Sequence
* Estimated FDR score
* Calculated Mass
* Group-ID (this is a placeholder in this version, and can be used for Protein Ambiguity Grouping in the later version)
* Protein Accession
* Start location in the protein sequence
* End location in the protein sequence
* Simple FDR score
* Modifications used (multiple modifications are separated by ##)
* Charge
* Experimental Mass

An example row is shown below –

1 , 43.5500.5521.2.dta , ALLELQMDGEEIYQTFSR , 4.0362105E-10 , 2159.59 , GROUP\_ID , gb|TGME49\_005470 , 157, 174 , 0.0 , ## 15.9949\_M:7 , 2 , 2159.033

**FinalOutput\_392.txt**

The file contains information about the **consensus search result**, i.e., the spectrums which were identified by both the search engines. The results in this file are categorised under three headings –

* *Sequences in the container : [ot]* – The peptide sequences which were identified by both Omssa and X!Tandem
* *Sequences in the container : [o]* - The peptide sequences which were identified by both Omssa only
* *Sequences in the container : [t]* - The peptide sequences which were identified by X!Tandem only

The columns in this CSV file are same as the ones in FinalOutput\_Verbose\_392.txt, except the Estimated FDR column, which reports the new FDR values, computed after making the consensus among the search engine results.

**Stage 4 - Summarizing the search results**

The files produced at this stage is –

Summary\_\_392.txt

The Summary file is used for grouping the results in FinalOutput\_392.txt according to identified proteins. The peptide-spectrum matches are grouped according to proteins they belong to. Also, we compute a combined Protein Score for each identified protein. This is a tab-separated file with the following fields –

* Protein Accession (whenever this field is empty in a line, it means that the information in the current row is grouped with the previously seen Protein accession)
* Protein score
* Spectrum ID
* Peptide Sequence
* Estimated peptide FDR score
* Calculated mass
* Group-ID (Placeholder in this version)
* Start location of the peptide in the protein sequence
* End location of the peptide in the protein sequence
* Simple FDR score for the peptide
* Modifications used
* Experimental mass
* Search engine combination which found the peptide (ot = Omssa and X!Tandem, t = X!Tandem only, and o = Omssa only)

An example row is shown below –

**Note** – The Summary\_\_xxx.txt is the most important file in the whole output directory. Each input MGF file produces an output directory, and each output directory has one Summary\_\_xxx.txt file. In the end, when all the MGF files have been processed through the pipeline, we *walk through* each Summmary\_\_xxx.txt and combine their results in a *single final output file* for the whole dataset.

apidb|cds\_TGME49\_022860-1 7.802E-19 1.6718.6725.3.dta LSQMEPLLQMDYSFPSTIVLVGVPK 4.593E-11 2824.64 GROUP\_ID 43 67 1.123E-5 ## 15.9949\_M:4## 15.9949\_M:10 3 2824.451 ot

In case of any failure during the execution of the pipeline, we wouldn’t have the Summary\_\_xxx.txt for the corresponding input MGF file. The error will be reported in the log file. As a quick check if the pipeline ran properly for all the input MGF files, we can run a ***grep*** command in the main output directory (the parent directory of dirToxo\_1D\_Slice1 in this case) and count the number of Summary\_\_xxx.txt files produced, if this number matches with the total number of input MGFs given, we know that the all the input files were processed without failure.

**Summarizing results for the whole dataset**

At this stage, we have a number of output directories (each containing one Summary\_\_xxx.txt) for a given set of input MGF files. We can walk through all the Summary\_\_xxx.txt files and group the results for the entire dataset in a single tab-separated file. This operation will take place on a single node after the pipeline has been run for the whole dataset in single node mode or clustered mode. We will need to call createSummaryForWholeData.jar in JavaGFF folder with the following way –

*$java –jar createSummaryForWholeData.jar ~/Ritesh\_Work/TestSpace/pipeline\_test/ WholeDatasetSummary.txt*

where, *~/Ritesh\_Work/TestSpace/pipeline\_test/* is the main output folder with sub-directories dirToxo\_1D\_Slice1 etc., and the WholeDatasetSummary.txt is the entire dataset summary file produced. The code simply walks through all the Summary\_xxx.txt files and dumps all the proteins and respective records in the WholeDatasetSummary.txt file. It doesn’t filter out the proteins for redundancies across different Summary files, and the format of the columns remain the same as they were in the Summary files.