rTANDEM: An R encapsulation of X!Tandem

Frederic Fournier[*], Charles Joly Beauparlant[†], Rene Paradis[‡], Arnaud Droit[§]

May 2, 2019

Introduction to rTANDEM

Contents

[*]frederic.fournier@crchuq.ulaval.ca
[†]charles.joly-beauparlant@crchul.ulaval.ca
[‡]rene.paradis@genome.ulaval.ca
[§]arnaud.droit@crchuq.ulaval.ca

1 Licensing

This package and the underlying X!Tandem code are distributed under the Artistic license 1.0. You are free to use and redistribute this software.

2 Introduction

X!Tandem is an open source software for protein identification by tandem mass spectrometry, and rTANDEM encapsulates this software in R. As of now, rTANDEM provides rather basic encapsulation of X!Tandem: it has a function that takes as an argument the path to an X!Tandem style parameter file and return the path to an X!tandem style output file. The package also presents some functions to transform parameters or results files into R objects and vice versa. Furthermore, some functions are available toexamine the results within R.

3 Manifesto

Most analytic pipelines for protein identification using tandem mass spectrometry experiments are proprietary. Those pipelines often have the advantage of being user-friendly (to a certain extent), but reliance on proprietary pipelines has many downfalls. Proprietary pipelines hinder collaboration between labs by making it more difficult to analyze data that have been obtained in a different pipeline. It prevents the development of new types of analyses or of new scientific questions by preventing the researchers from modifying the algorithms or inner settings of the analytic tools. It creates a 'black box' phenomenon where the results given by the analytic tools are to be trusted even though the process by which they have been obtained cannot be fully examined, tested or modified.

The encapsulation of X!Tandem in R is a fisrt step towards building a viable analytic pipeline for protein identification in R. The goal is to make R an interesting and powerful framework for ms/ms data analysis just like it is for NGS analysis.

4 rTANDEM input/output overview

The input of a standard X!Tandem analysis is composed of several files: three .xml files setting the parameters (usually named input.xml, default-input.xml and taxonomy.xml), one or more database files, and one or more

files containing the spectra to be analyzed. Detailed information is available on the X!Tandem website. rTANDEM can use the xml files just like X!Tandem, or R objects can be used in lieu of the .xml files. rTANDEM provides functions to create parameters or taxonomy objects from xml files, or to create the xml files from the R parameters or taxonomy objects. The output of a standard X!Tandem analysis is composed of a standard xml file. Since this output can be quite large, rTANDEM follows X!Tandem behaviour and writes the result of the analysis to an xml file. But the package provides a function to parse this file an create an R object from it.

4.1 The input file

The main input file is used to fix the major parameters of the search, namely

1. The path of the output file (the file that will be used to store the results).
2. The path of the spectra file (the file that contains the data to be analysed).
3. The taxon used for the search.
4. The path of the taxonomy file (the file that provides information on the databases to be used).
5. The path of a default-input file (a file that sets parameters that are common to many experiments).

4.2 The output file

rTANDEM output is an XML file. The format is based on the XML language BIOML (`www.bioml.com`) and a full description of the format is available at `http://www.thegpm.org/docs/X_series_output_form.pdf`. The function GetResultsFromXML() can be used to create a R object from the xml file.

4.3 The spectra file

The spectra file contains the data generated by the mass spectrometer. The supported formats for the data file are 'DTA', 'PKL' and 'MGF'.

4.4 The taxonomy file

The taxonomy file, or object, is used to link together a taxon and one or multiple fasta files. This file is not usually modified for every experiment, rather it provides the link between all taxa and their relative databases. IT should be modified only when databases are updated or replaced.

4.5 The database files

The database files contain the information that will be used to analyse the spectra files. The most common database files contain the protein sequences of a given taxon in ASCII '.fasta' format. X!Tandem also support the binary 'fasta.pro' format. (See `http://www.thegpm.org/TANDEM/api/fastapro.html`.) Lists of saps (single amino acid polymorphisms) or posttranslational modifications can also be used to complement the fasta files.

4.6 The default-parameter file

The default-parameter file is used to set fine-grained parameters. For example it is used to set the mass error tolerated, or to set which types of ions will be used for calculation. This file is not usually modified for every experiment, rather, a default-parameter can be created for every instrument used (the fine-grained parameters are often instrument-specific). The default-parameter file also provides links to formatting files (css and xsl) that are useful for consulting the output in a web browser.

5 rTANDEM typical usage

First of all, we must load rTANDEM into the session:

```
> library(rTANDEM)
```

To use rTANDEM and launch an analysis, we will need a couple of objects and files, including a taxonomy and a parameter set. The taxonomy is used to link taxa to fasta files. The rTANDEM package contains a partial fasta.pro file from the yeast proteome, so we can build a taxonomy object using it. (Note that we could have used a normal fasta file just as well):

```
> taxonomy <- rTTaxo(
+   taxon="yeast",
+   format="peptide",
+   URL=system.file("extdata/fasta/scd.fasta.pro", package="rTANDEM")
```

```
+ )
> taxonomy

  taxon  format
1 yeast peptide

1 C:/Users/biocbuild/bbs-3.9-bioc/tmpdir/RtmpYVhpgu/Rinst1998538f47fb/rTANDEM/extdata
```

Second we will need to determine the parameters for the analysis. There are usually two different sets of parameters: one set of fine-grained parameters (often related to the mass-spectrometer used to acquire the data); and a set of more basic parameters that specify which data file will be analysed and where should the output be written. A default 'fine-grained' parameter set is included in the package, as well as a small data file, so we can use them to create a complete parameter set. We will start by creating an empty parameter object and we will fill the information in:

```
> param <- rTParam()
> param <- setParamValue(param, 'protein', 'taxon', value="yeast")
> param <- setParamValue(param, 'list path', 'taxonomy information', taxonomy)
> param <- setParamValue(param, 'list path', 'default parameters',
+   value=system.file("extdata/default_input.xml", package="rTANDEM"))
> param <- setParamValue(param, 'spectrum', 'path',
+   value=system.file("extdata/test_spectra.mgf", package="rTANDEM"))
> param <- setParamValue(param, 'output', 'xsl path',
+   value=system.file("extdata/tandem-input-style.xsl", package="rTANDEM"))
> param <- setParamValue(param, 'output', 'path',
+   value=paste(getwd(), "output.xml", sep="/"))
```

Now that all the relevant parameters are recorded, we can launch the analysis using the param object:

```
> result.path <- tandem(param)

Loading spectra
 (mgf). loaded.
Spectra matching criteria = 242
Starting threads . started.
Computing models:
        testin
                sequences modelled = 5 ks
```

```
Model refinement:
        partial cleavage ..... done.
        unanticipated cleavage ..... done.
        modified N-terminus ..... done.
        finishing refinement ... done.
Creating report:
        initial calculations  ..... done.
        sorting  ..... done.
        finding repeats ..... done.
        evaluating results ..... done.
        calculating expectations ..... done.
        writing results ..... done.

Valid models = 40
Unique models = 41
Estimated false positives = 1 +/- 1
```

```
> result.path
```

```
[1] "C:/Users/biocbuild/bbs-3.9-bioc/tmpdir/RtmpYVhpgu/Rbuild19985f0d5322/rTANDEM/vig
```

The results are written in xml format to the directory specified. But we can load them in R for further processing.

```
> result.R <- GetResultsFromXML(result.path)
```

Having the results within R, we can now use all the power of R to explore the identified proteins and the peptides used to identify these proteins. For example, to get the list of the proteins identified from at least two peptides and with a X!tandem score corresponding to an expect value of 0.05 or better, we simply do:

```
> proteins <- GetProteins(result.R, log.expect=-1.3, min.peptides=2)
> proteins[, c(-4,-5), with=FALSE]  # columns were removed for better display
```

```
     uid expect.value   label description num.peptides
 1:  576        -27.2 YCR012W     YCR012W            5
 2: 1811        -14.5 YFR053C     YFR053C            3
 3: 2301        -12.8 YGR254W     YGR254W            3
 4:    4        -12.0 YAL005C     YAL005C            3
 5: 3517        -12.0 YLL024C     YLL024C            3
 6: 3328        -10.3 YKL152C     YKL152C            2
```

```
 7: 3386        -10.1 YKL216W     YKL216W           2
 8: 2281         -7.9 YGR234W     YGR234W           2
 9: 2568         -7.5 YHR174W     YHR174W           2
10: 2044         -7.1 YGL253W     YGL253W           2
```

GetProteins() returns a data.table, which is a kind of optimized data.frame. This object can be further manipulated to answer many common questions. For example, the first questions that are often asked about a new analysis are 'how many proteins have been identified with appropriate confidence level?', 'What are the top proteins identified?', and 'Were protein X, Y, and Z identified in the sample?' We can quickly answer those questions:

```
> # How many proteins have been identified with appropriate confidence?
> length(proteins[['uid']])

[1] 10

> # What are the top 5 proteins identified?
> proteins[1:5, c("label", "expect.value"), with=FALSE]

     label expect.value
1: YCR012W        -27.2
2: YFR053C        -14.5
3: YGR254W        -12.8
4: YAL005C        -12.0
5: YLL024C        -12.0

> # Were proteins YFR053C or P02267 identified in the sample?
> c("YFR053C", "P02267") %in% proteins[,"label", with=FALSE][[1]]

[1]  TRUE FALSE
```

At this point, we might want more information about protein YFR053C. For example, we might want to know which peptides from this protein were identified with appropriate confidence level. To get the list of the peptides from this protein identified with an expect value < 0.05, we can simply do:

```
> peptides <- GetPeptides(
+   protein.uid=subset(proteins, label=="YFR053C", uid)[[1]],
+   results    =result.R,
+   expect     =0.05
+   )
> peptides
```

```
    pep.id prot.uid spectrum.id spectrum.mh spectrum.z spectrum.sumI
1: 102.1.1     1811         102    942.5147          1          4.80
2: 250.1.1     1811         250   1212.5610          1          4.61
3:  60.1.1     1811          60    863.4933          1          4.26
   spectrum.maxI spectrum.fI expect.value tandem.score        mh   delta
1:       8764.25     87.6425      0.00660         31.9  942.5370 -0.0220
2:       8204.07     82.0407      0.00043         35.0 1212.5531  0.0079
3:       4485.89     44.8589      0.00870         21.7  863.4985 -0.0052
   peak.count missed.cleavages start.position end.position    sequence ptm.type
1:         NA                0            166          173    INEGILQR     <NA>
2:         NA                0            437          447 DIYGWTGDASK     <NA>
3:         NA                0            309          315     YLGELLR     <NA>
   ptm.at ptm.modified
1:     NA           NA
2:     NA           NA
3:     NA           NA
```

Sometimes, a peptide sequence will be found in many different proteins. If we are planning to focus on a particular peptide, we should make sure that is not the case. We can do this using the GetDegeneracy function. This function will return the list of the proteins in which a given peptide is found. Let's look at the first peptide identified for YFR053C:

```
> proteins.of.the.peptide <- GetDegeneracy(peptides[[1,"pep.id"]], result.R)
> proteins.of.the.peptide[,label]

[1] "YFR053C" "YGL253W"

> # Careful! This peptide belongs to 2 different proteins! It should not be
> # used for quantification, for MRM or as a biomarker.
```

With our results being in R, we can harness the power of other R and bioconductor packages to further analyze our data. For example, we can use biomaRt to get annotation information and cross-references. The reduced database that we used features ensembl peptide-id, but has no real description for the proteins. Let's use biomaRt to find a description of protein YFR053C, and while we are there, let's also get a cross-reference for this protein in uniprot, as well as the GO terms associated with the protein:

```
> library(biomaRt)
> ensembl.mart<- useMart(biomart="ensembl", dataset="scerevisiae_gene_ensembl")
```

```
> str(getBM(mart=ensembl.mart, filters="ensembl_peptide_id", values="YFR053C",
+           attributes="description"),
+     strict.width="wrap", nchar.max=500)
> getBM(mart=ensembl.mart, filters="ensembl_peptide_id", values="YFR053C",
+       attributes=c("ensembl_peptide_id", "uniprotswissprot"))
> getBM(mart=ensembl.mart, filters="ensembl_peptide_id", values="YFR053C",
+       attributes=c("go_id", "name_1006"))
```