**Generation of DIA/SWATH-MS spectral libraries to identify and quantify HLA-associated peptides**

## Introduction

## Materials

**Required Software**

Trans Proteomic Pipeline v5.2.0

NetMHCpan-4.1

Python 3.5 or greater

OpenSWATH (if converting SPECLIB files to OpenSWATH libraries)

**Software sources**

The Trans Proteomic Pipeline can be acquired from tppms.org.

NetMHCpan-4.1 must be acquired from DTU Health Tech at <https://services.healthtech.dtu.dk/software.php>  
NetMHC requires a Linux or Mac computer to run.

Most recent Linux distribution will come with Python3 installed. If this is not the case, installation instructions can be found at python.org. For Windows computers, we recommend installing the Anaconda Python distribution (https://www.anaconda.com/products/individual) or installing from python.org

OpenSWATH is distributed as a component of OpenMS. Installation instructions can be found at http://openswath.org/en/latest/index.html.

**Notes on data and software**

This protocol assumes all mass spectrometry data has been previously converted to mzXML format (note that mzXML is required for the version of X! Tandem distributed with the TPP). This can be accomplished for all major vendor formats using MSConvert on a Windows computer.

The majority of the protocol can be carried out either on a Windows or Linux computer, with the exception of running NetMHCpan, which must be done on a Linux computer. The examples given will be on a Linux computer.

The TPP steps (database search through validation) will be done using TPP’s graphical interface, Petunia. We will be using v5.2.0. Earlier versions should work fine, but might differ somewhat in interface.

**Data prerequisites**

DDA raw files for spectral library building - must contain iRT peptides

DIA raw files for DIA analysis - must contain iRT peptides

FASTA database must contain iRT peptides

## Methods

**Organization of data**

Before beginning, you will need to organize your data.

1. In a file browser, create a new folder for your experiment in a location of your choosing (we have called it MhcExperiment)
2. Create the following directory structure in your experiment folder:

-MhcExperiment

|-data

|-search\_results

|-fasta

|-params

|-comet

|-tandem

|-taxonomy

1. Copy your DDA mzML files into the data folders.
2. Copy or download a fresh FASTA file into the “fasta” folder.

**Add iRT peptides to FASTA**

1. Open the FASTA file in a text editor
2. Add the following lines to the end of the file:

>Biognosys|iRT-Kit\_WR\_fusion

LGGNEQVTRYILAGVENSKGTFIIDPGGVIRGTFIIDPAAVIRGAGSSEPVTGLDAKTPVISGGPYEYRVEATFGVDESNAKTPVITGAPYEYRDGLDAASYYAPVRADVTPADFSEWSKLFLQFGAQGSPFLK

**Generate decoy database**

1. Open a new terminal
2. Navigate to tpp installation folder on your computer
3. Create a concatenated target-decoy database (replace filenames and paths as appropriate):

>./bin/decoyFASTA /path/to/MhcExperiment/fasta/fasta.fasta /path/to/MhcExperiment/fasta/fasta\_concatenated\_target\_decoy.fasta

**Prepare search parameters**

1. Copy the default parameters for X! Tandem and Comet into your experiment folder (here we assume the tpp data folder is inside the tpp installation folder):

>cp ./data/params/comet.params /path/to/MhcExperiment/params/comet

>cp ./data/params/tandem\_params.xml /path/to/MhcExperiment/params/tandem

>cp ./data/params/taxonomy.xml /path/to/MhcExperiment/params/tandem/taxonomy

1. Open the new comet.params files and edit as follows:

database\_name = /path/to/MhcExperiment/fasta\_concatenated\_target\_decoy.fasta

peptide\_mass\_tolerance = **10.0 [or another appropriate value in ppm]**

peptide\_mass\_units = 2

search\_enzyme\_number = 0

fragment\_bin\_tol = **[1.0005 for low-res, 0.02 for hi-res]**

fragment\_bin\_offset = **[0.4 for low-res, 0.0 for hi-res]**

sample\_enzyme\_number = 0

peptide\_length\_range = 8 15

output\_suffix = .comet

add\_C\_cysteine = 0.0000 (leave as 57.021464 if CAM is expected)

1. Open the new taxonomy.xml file and replace the contents with the following (replacing the path to the FASTA file):

<?xml version="1.0"?>

<bioml label="x! taxon-to-file matching list">

<taxon label="**mhc\_database**">

<file format="peptide" URL="**/path/to/MhcExperiment/fasta/fasta\_concatenated\_target\_decoy.fasta**" />

</taxon>

</bioml>

1. Make a copy of the new tandem\_params.xml file for each mzXML file (data\_file.mzXML below) and edit as follows:

<note type="input" label="spectrum, path">**/path/to/MhcExperiment/data/data\_file.mzXML**</note>

<note type="input" label="output, path">**/path/to/MhcExperiment/data/search\_results/data\_file.tandem.pep.xml**</note>

<note type="input" label="list path, taxonomy information">**/path/to/MhcExperiment/params/tandem/taxonomy/taxonomy.xml**</note>

<note type="input" label="protein, taxon">**mhc\_database**</note>

<note type="input" label="spectrum, parent monoisotopic mass error minus">**5.0**</note> **(or another appropriate value in ppm)**

<note type="input" label="spectrum, parent monoisotopic mass error plus">**5.0**</note> **(or another appropriate value in ppm)**

<note type="input" label="spectrum, parent monoisotopic mass error units">**ppm**</note>

<note type="input" label="residue, modification mass">**57.021464@C**</note> **(delete if CAM not expected)**

<note type="input" label="protein, cleavage semi">**no**</note>

<note type="input" label="scoring, maximum missed cleavage sites">**15**</note>

In addition, add the following content to each file:

<note type="input" label="protein, cleavage site">[X]|[X]</note>

**Run database searches**

1. In the open teminal, run Comet searches on all the data files:

>./bin/comet /path/to/MhcExperiment/params/comet/comet.params /path/to/MhcExperiment/data/\*.mzXML

1. Move the pepXML results files into the search\_results folder:

>mv /path/to/MhcExperiment/data/\*.comet.pep.xml /path/to/MhcExperiment/data/search\_results

1. Run X! Tandem searches on all the data files:

>for i in /path/to/MhcExperiment/params/tandem/\*; do ./bin/tandem $i; done

**Convert X! Tandem output files**

1. Convert X! Tandem output files into pepXML files

>for i in /path/to/MhcExperiment/data/search\_results/\*.tandem.pep.xml; do ./bin/Tandem2XML $i > “$i.pepXML”; done

**Validation with PeptideProphet**

1. Run PeptideProphet on pepXML files. “xinteract” combines pepXML files into one file and then runs PeptideProphet on the result. It is run twice, once for Comet files and once for X! Tandem files:

>./bin/xinteract -OAPd -drev\_ -THREADS=**12** -Ninteract.tandem.pepXML /path/to/MhcExperiment/data/search\_results/\*.pepXML

>./bin/xinteract -OAPd -drev\_ -THREADS=**12** -Ninteract.comet.pepXML /path/to/MhcExperiment/data/search\_results/\*.comet.pep.xml

**Integration with iProphet**

1. Integrate the search results from Comet and X! Tandem using iProphet (change THREADS=12 to whatever is appropriate for your system):

>./bin/InterProphetParser THREADS=12 DECOY=rev\_ MINPROB=0.7 /path/to/MhcExperiment/data/search\_results/interact.comet.pepXML /path/to/MhcExperiment/data/search\_results/interact.tandem.pepXML /path/to/MhcExperiment/data/search\_results/interact.iproph.pepXML

**Convert iProphet output to TSV and False Discovery Rate Filtering**

1. Extract a feature subset of the pepXML output file from iProphet to a TSV file and apply a 1% FDR filter using pepXML2tsv.py (provided in appendix):

>python3 pepXML2tsv.py -f /path/to/MhcExperiment/data/search\_results/interact.iproph.pepXML -q -c 0.01 -o interact.iproph.tsv

**HLA allele annotation and building SpectraST inclusion lists**

1. Annotated “interact.iproph.tsv” with NetMHCpan\_annotate\_file.py (provided in appendix) (replace alleles as appropriate):

>python3 ./NetMHCpan\_annotate\_file.py ./interact.iproph.tsv HLA-A0201 HLA-B0702 HLA-C0702

1. Build allele-specific inclusion lists for SpectraST at a NetMHCpan eluted ligand %rank cutoff of 2.0 using build\_spectrast\_incl\_lists.py (provided in appendix):

>python3 ./build\_spectrast\_incl\_lists.py ./interact.iproph\_annotated.tsv 2.0

**Generation of HLA Allele-Specific Peptide Spectral Libraries**

Generate allele-specific and merged consensus libraries. **Bold** sections of commands indicate sections which must be changed to reflect the analysis (e.g. allele name, instrument type, etc).

1. Use SpectraST in library generation mode to generate a raw library for each allele.

>./bin/spectrast -cNSpecLib\_allele -cICID-QTOF -cTallele\_inclusion\_list.tsv -c\_IRTiRT.txt -c\_IRR interact.ipro.pepXML

The -cI parameter indicates the instrument/detector architecture. For QTOF instruments use -cICID-QTOF, for Orbitrap dectectors (i.e. Orbi-Orbi method) use -cIHCD, and for ion trap detectors (e.g. Orbi-Trap method), use -cICID.

-c\_IRTiRT.txt and -c\_IRR are for normalization of retention times. -c\_IRTiRT.txt refers to a file (here named iRT.txt) which contains the retention time reference peptides. An example containing the iRT peptides from Biognosys is provided in appendix.

-cT specifies the file containing the peptide inclusion list for the respective allele.

Courier New

-cN indicates the output filename.

1. Use SpectraST again to create consensus libraries for each allele (again indicate the correct -cI parameter). These libraries will be allele-specific libraries for the experiment:

>./bin/spectrast -cNSpecLib\_allele\_cons -cICID-QTOF -cAC SpecLib\_allele.splib

1. To make a library containing all allele-specific spectra, merge the consensus libraries:

./bin/spectrast -cNSpecLib\_alleles\_cons -cJU -cAC /path/to/MhcExperiment/data/search\_results/\*\_cons.splib

**Conversion to SWATH assay libraries**

The SPLIB files created in the previous section are the final spectral libraries and can be used with SpectraST or converted into formats usable by DIA analysis tools. Here we convert the library into a tabular format usable by Spectronaut and Skyline, which is further converted into a TraML file usable by OpenMS.

- “swaths.txt” is a file which contains the window limits used in the respective DIA experiment (one window per line, window edges tab-separated). An example is provided in appendix.

- “SpecLib\_alleles\_cons.tsv” is the name of the output assay library

1. Convert the desired SPLIB file into a TSV file using spectrast2tsv.py:

>spectrast2tsv.py -l 350,2000 -s b,y -x 1,2 -o 6 -n 6 -p 0.05 -d -e -w swaths.txt -k openswath -a SpecLib\_alleles\_cons.tsv

SpecLib\_alleles\_cons.sptxt

This assay library is usable by Spectronaut and Skyline. To produce a TraML file for use in OpenSWATH, the following commands can be used.

1. Convert the “SpecLib\_alleles\_cons\_openswath.tsv” file to a TraML file using the OpenSWATH tool ConvertTSVToTraML:

>ConvertTSVToTraML -in SpecLib\_alleles\_cons.tsv -out SpecLib\_alleles\_cons.TraML

1. Append decoys to the TraML assay library with the OpenSWATH tool OpenSwathDecoyGenerator:

>OpenSwathDecoyGenerator -in SpecLib\_alleles\_cons.TraML -out SpecLib\_alleles\_cons\_decoy.TraML -method shuffle -append -exclude\_similar

Appendices

None of the examples have appendices, so not sure here. Perhaps these will go on the CaronLab github repository and we can reference that.

Appendix A. iRT.txt

LGGNEQVTR -28.308

GAGSSEPVTGLDAK 0.227

VEATFGVDESNAK 13.1078

YILAGVENSK 22.3798

TPVISGGPYEYR 28.9999

TPVITGAPYEYR 33.6311

DGLDAASYYAPVR 43.2819

ADVTPADFSEWSK 54.969

GTFIIDPGGVIR 71.3819

GTFIIDPAAVIR 86.7152

LFLQFGAQGSPFLK 98.0897

Appendix B. swaths.txt

400 425

424 450

449 475

474 500

499 525

524 550

549 575

574 600

599 625

624 650

649 675

674 700

699 725

724 750

749 775

774 800

799 825

824 850

849 875

874 900

899 925

924 950

949 975

974 1000

999 1025

1024 1050

1049 1075

1074 1100

1099 1125

1124 1150

1149 1175

1174 1200

**Appendix C. pepXML2tsv.py**

#/bin/python3

import xml.etree.ElementTree as ET

from sys import argv

from os import path

import re

import argparse

parser = argparse.ArgumentParser(description='Parse one pepXML file(s) and convet to tsv, optionally applying an FDR filter.')

parser.add\_argument('-f', '--file', type=str, required=True, help='Path to one pepXML files.')

parser.add\_argument('-q', '--qvalue', action='store\_true', help='Caculcate q-values.')

parser.add\_argument('-c', '--fdr\_cutoff', type=float, help='Apply a specified FDR cutoff.')

parser.add\_argument('-d', '--decoy\_prefix', type=str, default='rev\_', help='The decoy prefix, used for calculating q-values.')

parser.add\_argument('-o', '--output\_file', type=str, required=True, help='The output filename.')

args = parser.parse\_args()

def get\_namespace(element):

m = re.match(r'\{.\*\}', element.tag)

return m.group(0) if m else ''

pepxml\_file = args.file

print('Loading pepXML file')

ET.register\_namespace('', 'http://regis-web.systemsbiology.net/pepXML')

tree = ET.parse(pepxml\_file)

root = tree.getroot()

ns = get\_namespace(root)

f\_out = args.output\_file

print('Parsing pepXML file')

with open(f\_out, 'w') as f:

f.write('Spectrum\tScan\tRT\tPeptide\tSpectraST\_Peptide\tProtein\tLabel\tiProphet\_prob\n')

for spectrum\_query in root.iter('{}spectrum\_query'.format(ns)):

spectrum = spectrum\_query.attrib['spectrum']

scan = spectrum\_query.attrib['start\_scan']

rt = spectrum\_query.attrib['retention\_time\_sec']

hit = spectrum\_query.find('{}search\_result'.format(ns)).find('{}search\_hit'.format(ns))

mod\_info = hit.find('{}modification\_info'.format(ns))

pep = hit.attrib['peptide']

protein = hit.attrib['protein']

if protein.startswith(args.decoy\_prefix):

label = 'decoy'

else:

label = 'target'

if mod\_info is not None:

spectrast\_pep = mod\_info.attrib['modified\_peptide'] + '/' + spectrum\_query.attrib['assumed\_charge']

else:

spectrast\_pep = hit.attrib['peptide'] + '/' + spectrum\_query.attrib['assumed\_charge']

for analysis in hit.findall('{}analysis\_result'.format(ns)):

if analysis.attrib['analysis'] == 'peptideprophet':

peptideprophet\_prob = analysis.find('{}peptideprophet\_result'.format(ns)).attrib['probability']

else:

iprophet\_prob = analysis.find('{}interprophet\_result'.format(ns)).attrib['probability']

f.write('\t'.join([spectrum, scan, rt, pep, spectrast\_pep, protein, label, iprophet\_prob]) + '\n')

if args.qvalue:

print('Adding q-values')

with open(f\_out, 'r') as f:

header = f.readline().strip().split()

contents = [x.strip().split() for x in f.readlines()]

prob\_index = header.index('iProphet\_prob')

label\_index = header.index('Label')

target\_probs = [float(x[prob\_index]) for x in contents if x[label\_index] == 'target']

decoy\_probs = [float(x[prob\_index]) for x in contents if x[label\_index] == 'decoy']

for spec in contents:

prob = float(spec[prob\_index])

n\_target = len([1 for x in target\_probs if x >= prob])

n\_decoy = len([1 for x in decoy\_probs if x >= prob])

q = float(n\_decoy) / (n\_decoy + n\_target)

spec.append(str(q))

with open(f\_out, 'w') as f:

header.append('q\_value')

f.write('\t'.join(header) + '\n')

if args.fdr\_cutoff:

for line in [x for x in contents if float(x[-1]) <= args.fdr\_cutoff]:

f.write('\t'.join(line) + '\n')

else:

for line in contents:

f.write('\t'.join(line) + '\n')

**Appendix D.**