PepQuant2

Developer Guide

Table of Contents

Step-by-step

Overview

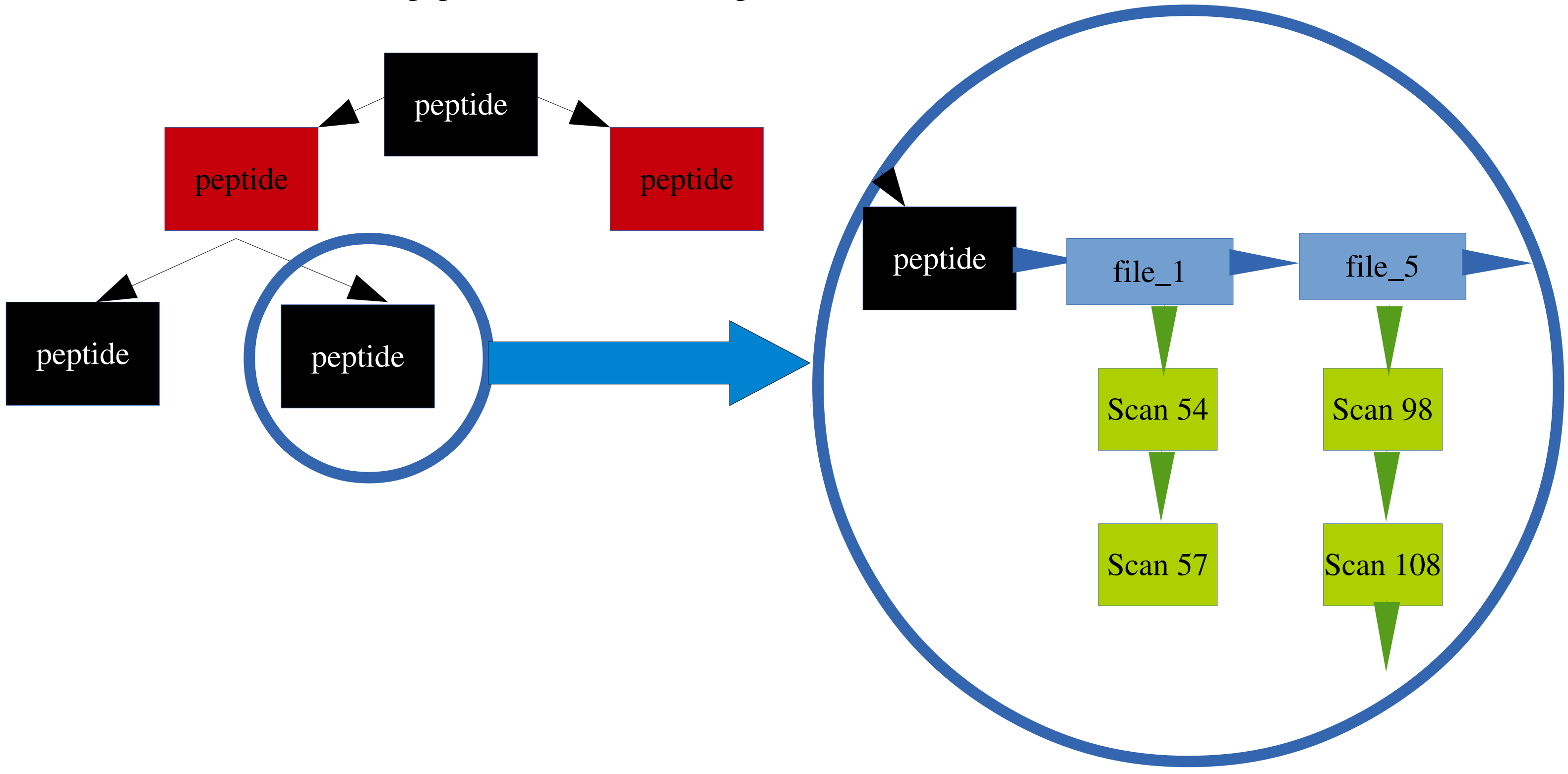
The main function is contained within pepquant2.c. It is an ugly function that controls the main logical flow of pepQuant2. The next sections document what is going on under the hood of PepQuant2 with references at a high level.

Parse arguments

Command line arguments are parsed in global.c and the values of these arguments are stored in global variables in pepquant2.c. Access to these global variables is provided by global.h.

Read input

Functions for parsing MaxQuant and StatQuest results are included in peptide.c, whereas functions for parsing pepXML files are in pepxml.c. PepQuant uses libxml2 for parsing xml files and dirent.h for manipulating directories. While reading these files PepQuant2 is really looking for (peptide, file(run), scanNumber) tuples. Since for large experiments this can be a sizable list and pepQuant wants to create a set of peptides it organizes the peptides as a red black tree. Every node in this red black tree represents a peptide and every peptide has two linked lists: an MS1 and MS2 linked list. The MS2 linked list is initialized while reading input files and for every input file (run) in which the peptide has an MS2 identification and node for that file is created. The file node itself contains a linked list of scan numbers in which the peptide was found. The figure below illustrates the overall structure.



During initial development peptides were also stored as a linked list and a lot of code was written assuming peptides were in a linear structure. To reduce the amount of changes to existing code, and to prevent coupling with the red black tree representation, an ordered array of pointers to the peptides in the tree is created and used for most subsequent steps. The switch to red black trees was made because it was significantly more efficient for large data sets with tens of thousands of peptides. However, making this switch did complicate the source code significantly, especially since the red black tree code has been coupled with the representation of peptides, etc.

Generate Isotopic Patterns

The peptide nodes shown in the previous figure is a little bit of a simplification. The MS1 linked list is similar in style to the MS2 list but will be populated at a later step. There is also a pointer to an isotopic pattern. For every peptide the theoretical isotopic pattern must be generated in order to accurately interrogate MS1 spectra.

PepQuant uses the polynomial expansion method for generating isotopes. It begins by constructing a library isotopic patterns for individual amino acids and common modifications for their constituent atoms. Heavy atoms are used if labeled peptides are to be used. After this library is constructed, isotopic profiles for peptides are created by combining the appropriate profiles from the constituent amino acids and mods. The profile generated is for a neutral peptide. If the peptide parser is unable to identify a character in the peptide string, e.g. an X, it will remove the peptide from the red black tree and continue to the next peptide.

This step does use semaphores and pthreads in order to generate multiple threads that can generate isotopic profiles for different peptides independently.

Search mzXMLs for isotopic patterns

This step is actually pretty intuitive. We open a mzXML and parse it. The parsing code is a little messy and could probably be cleaned up using Xpaths, it does depend on the libxml library. All it really does transform the mzXML into a struct of similar information. With this in hand we look at every MS1 spectra in the struct for every isotopic profile generated previously. As mentioned before the profile is for a neutral peptide. The profiles need to be converted to a charged state (charge states 1+ through 4+ by default) and we use a binary search to find something with accepted tolerances of each isotope. If we do find something, we do look at adjacent peaks that might also be within tolerances. After all the binning is done, we check to see if at least one charge state satisfied all required conditions:

- all isotopic state were observed (none had a zero intensity)
- the correlation between the observed and theoretical patterns is equal or greater than the specified cutoff (0.99 by default)
- the sum of all intensities in the pattern is greater than or equal to the specified intensity cutoff (1E6 by default)

If the above conditions are satisfied we record the hit in a manner similar to that used for recording MS2 hits. Since the mzXML has just been parsed and is taking up memory, we also update retention time information for both MS1 and MS2 hits present.

Calculate retention time tables

Pretty simple, refer to userGuide and source code.

Calibrate retention time tables

First we determine the median retention times for every peptide for both MS1 and MS2. We then align every run to the median retention times (once for MS1 and once for MS2). We can think of the non-zero retention times of for a given run as a vector x, and the corresponding median retention times as a vector y. We then fit the equation $y = -(a)x + b$ using least squares. We use MINPACK-1 Least Squares Fitting Library in order to do this. MINPACK is available online. PepQuant2 sort of steals the mpfit.c and mpfit.h files and nothing else. From these files it uses only the functions for fitting a linear equation. The parameters are recorded.

We now have two uncalibrated retention time tables, and two sets of alignment parameters. Refer to the userGuide and source code for how these combined to form the final calibrated retention time table.

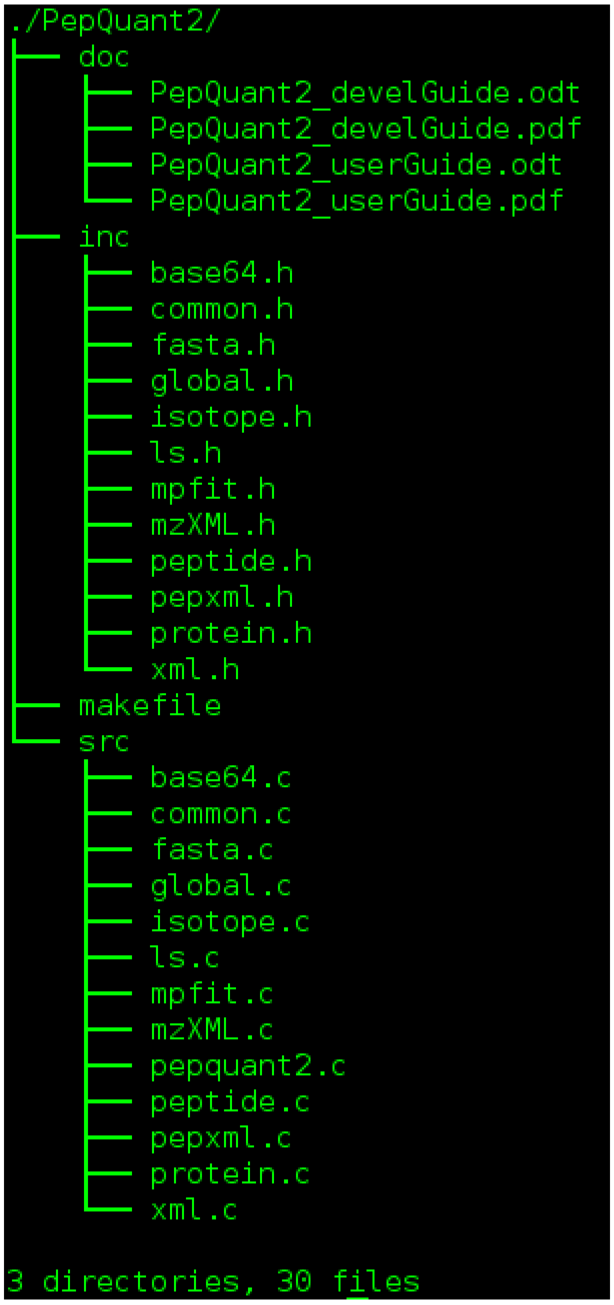
Quantify peptides and proteins

Again, pretty simple, refer to userGuide and source code. Lots of string manipulating to reduce peptides to their most basic form (stripping mods, etc) and checking if the peptides map to a protein in the fasta file. Labeled peptides get a bit messy but really you're dealing with three different peptide categories, heavy, light, and light by default (i.e. those that don't have K or R).

Issues and conclusions

PepQuant is good generating theoretical isotopic patterns and finding them. Areas for improvement include deciding what is a valid hit and what is not. Retention time alignment can also be improved.

Appendix 1: PepQuant2 Directory Structure



```
./PepQuant2/
├── doc
│   ├── PepQuant2_develGuide.odt
│   ├── PepQuant2_develGuide.pdf
│   ├── PepQuant2_userGuide.odt
│   └── PepQuant2_userGuide.pdf
├── inc
│   ├── base64.h
│   ├── common.h
│   ├── fasta.h
│   ├── global.h
│   ├── isotope.h
│   ├── ls.h
│   ├── mpfit.h
│   ├── mzXML.h
│   ├── peptide.h
│   ├── pepxml.h
│   ├── protein.h
│   └── xml.h
├── makefile
└── src
    ├── base64.c
    ├── common.c
    ├── fasta.c
    ├── global.c
    ├── isotope.c
    ├── ls.c
    ├── mpfit.c
    ├── mzXML.c
    ├── pepquant2.c
    ├── peptide.c
    ├── pepxml.c
    ├── protein.c
    └── xml.c

3 directories, 30 files
```

Appendix 2: Source Code Summary

peptide.c/peptide.h

Peptide TNILL
STATQUEST_EXT

-enum nodeColour
-struct scanNode
-struct spectraFileNode
-struct peptide
+delSpectraFileList
+printPeptides
+delPeptideList
+parseMaxQuant
+addPeptide
+stripMods
+initIsotopicPatterns
+printSearchResults
+printTable
+searchMzXMLs
+initFilelist
+getCount
+inOrder
+parseStatQuestdir

+struct spectraPackage
+newScanNode
+delScanNode
+addScanNode
+delScanNodeList
+newSpectraFileNode
+addSpectraFileNode
+delSpectraFileNode
+rbLeftRotate
+rbRightRotate
+rbInsertFixup
+rbDeleteFixup
+rbDelete
+rbFree
+rbTransplant
+treeMin
+rbPrintInOrder
+newPeptide
+fillPeptides
+searchSpectra
+checkChargeStates
+binSearch
+checkHit
+makePeptideThreadFunc
+sendModsLeft

global.c/global.h

+parseArgs

protein.c/protein.h

-struct proteinNode
+pep2prot
+prot2table
+delProteinList
+printProtTable

+newProteinNode
+delProteinNode
+addProteinNode

Pepquant2.c

external settings vars

+main

+genMS2rtTable
+genMS1rtTable
+medianRTtimes
+quant
+countSpectra

pepxml.c/pepxml.h

+parsePepXMLdir

+newMod
+addMod
+delMod
+delMods
+getMods
+convertMod
+modSequence
+getSearchResults
+readPepXML

common.c/common.h

MIN_CHARGE

+comparedouble
+median
+pearson
+new2Darray
+del2Darray

fasta.c/fasta.h

MAX_LINE
INIT_DB_SIZE

-struct entry
-struct fasta
+readFASTA
+delFasta
+findProtein
+trackCoverage
+printCoverage

+newEntry
+delEntry
+addEntry
+newFasta
+trimFastaDB
+appendSequence
+initCoverage

mzXML.c/mzXML.h

MIN_PEAK_COUNT
EMPTY_PEAK_LIST

-struct scan
-struct mzxml
+delMZXML
+readMZXML

+newScan
+delScan
+newMZXML
+getProperties
+parseScans
+getScanAttributes
+getPeaksAttributes
+checkCompatibility

ls.c/ls.h

-struct vars_struct
+fit
+linfunc

+leastSquares
+align

base64.c/base64.h

+decodeQuartet

xml.c/xml.h

+openXML
+searchForXPath
+downTo
+getAttribute

isotope.c/isotope.h

PROTON
ISOTOPE_COUNT
COLLECTION_SIZE
AMINO_ACIDS

-struct isotopicPattern
+delIsotopicPattern
+printIsotopicPattern
+newIPCollection
+delIPCollection
+makePeptide

+newIsotopicPattern
+combineIsotopicPatterns
+makeMolecule
+merge_sort
+merge
+swapEntries

Legend

🟦 FILE
🟪 EXTERN VARIABLES & MACROS
🟩 PUBLIC INTERFACE
🟥 PRIVATE FUNCTIONS