R2DGC

Contents

1 Introduction

1.1 Overview

This is supplementary text designed to expand upon methods implemented in the R2DGC package. It is not intended to be a comprehensive resource describing all possible flags or outputs for each function, rather this document seeks to provide an overview of the functionality of the package. Please refer to each individual function's help page using the ?[function name] command in R after installation. For clarity, function names are bolded and function arguments or flags are italicized.

This package is designed to allow users to align 2D-GCMS metabolite peaks common to multiple samples from exported peak files generated by Chromatof and identify metabolites based on a standard reference library. Metabolomics data processing can be roughly binned into three parts (Figure 1).

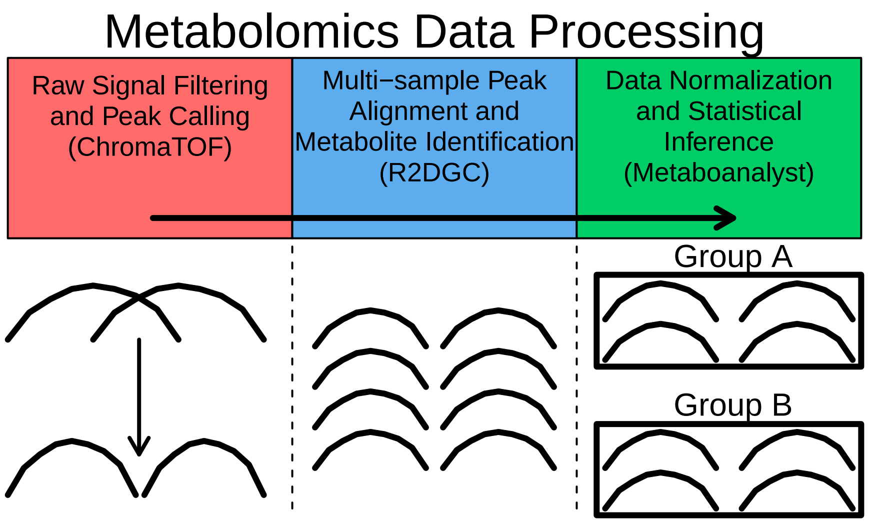
Metabolomics Data Processing
Raw Signal Filtering and Peak Calling (ChromaTOF)
Multi−sample Peak Alignment and Metabolite Identification (R2DGC)
Data Normalization and Statistical Inference (Metaboanalyst)
Group A
Group B

Figure 1: Overview of metabolomics data processing

The first is base level filtering of intensity signals via signal-to-noise thresholds and subsequent peak localization, deconvolution and integration. The second is alignment of peaks common to multiple sample files and identification of the metabolites from which each peak was likely derived. The final component of data processing is missing value imputation, normalization and statistical comparison of metabolite levels between groups of samples. The first step is usually handled by vendor software such as Chromatof (http://www.leco.com/products/separation-science/software-accessories/chromatof-software) that is installed with an institution's 2D-GCMS equipment. Well-polished solutions, such as Metaboanalyst (http://www.metaboanalyst.ca) also exist for the third data processing step. This package is focused on providing an easy to use tool for conducting the second step of the data processing pipeline.

1.2 Basic workflow

The basic workflow for metabolite alignment consists of two optional pre-processing functions and a function for performing the final alignment (Figure 2).

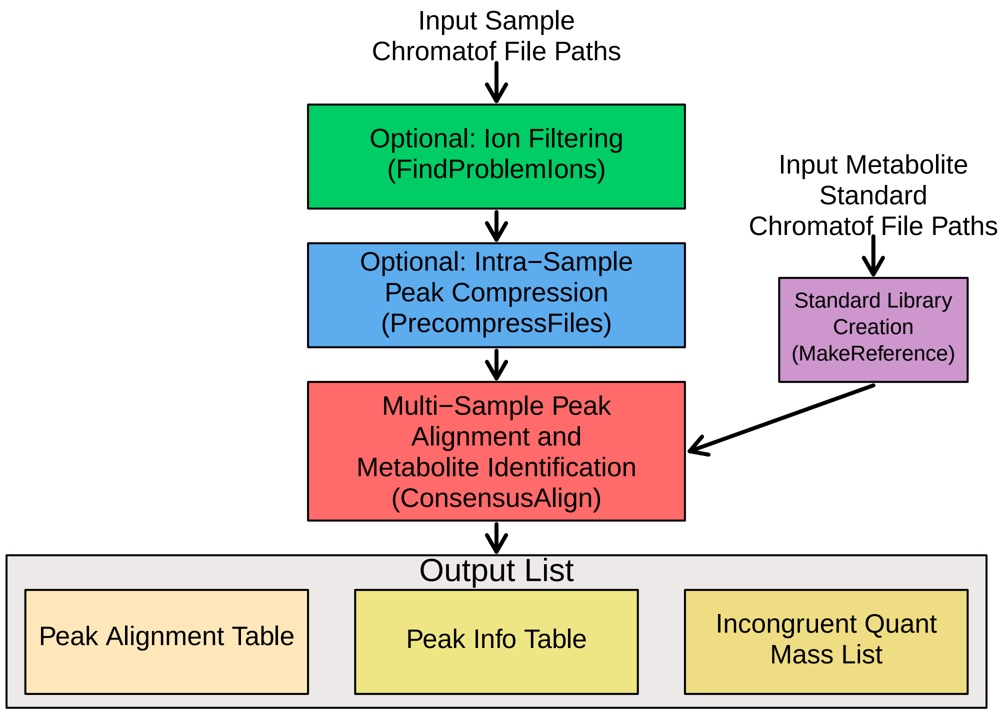
Input Sample Chromatof File Paths
Optional: Ion Filtering (FindProblemIons)
Optional: Intra−Sample Peak Compression (PrecompressFiles)
Multi−Sample Peak Alignment and Metabolite Identification (ConsensusAlign)
Input Metabolite Standard Chromatof File Paths
Standard Library Creation (MakeReference)
Output List
Peak Alignment Table
Peak Info Table
Incongruent Quant Mass List

Figure 2: Basic Workflow. Functions shown in parenthesis

Paths to raw Chromatof peak lists for each sample are provided as input. To improve the speed and accuracy of alignment, mass spectral ions that are absent from all peaks or are common to all peaks (derivatization artifacts) can be identified with the **FindProblemIons** function. Next, peaks that were unnecessarily split by Chromatof and likely represent the same analyte, which we have found to cause problematic splitting in downstream alignments, can be combined with the **PrecompressFiles** function. Admittedly, parameters within the peak calling software can be optimized to improve peak calling as well, but given the time consuming nature of this step, we found it beneificial to add this functionality as post-peak calling. Lastly, the **ConsensusAlign function** will perform the final alignment of peaks across samples, identify likely metabolites represented by each peak, if a metabolite standard reference library is provided, and output: (1) a peak alignment table with the intensities for aligned peaks in each sample, (2) a peak information table (containing peak retention times, mass spectral and standard library hits) and, (3) depending on the peak quantification method used, a list of peaks that were aligned, but did not have matching quant or apexing masses. Another optional function, **MakeReference**, is used to generate metabolite standard reference libraries for use in the **ConsensusAlign** function.

2 Detailed Workflow

2.1 Installation

There are multiple ways to install this package. The package can be installed after downloading the binary source file from github (https://github.com/rramaker/R2DGC) and installing the package manually:

```
install.packages("/PathToDownloadedFile/R2DGC_1.0.1.tgz")
```

The development version package can be directly installed from github with the devtools package

```
library(devtools)
install_github("rramaker/R2DGC")
```

Once installed the functions from R2DGC can be made available in your current working environment with the library() command:

```
library(R2DGC)
```

2.2 Formatting input data

The basic R2DGC input is a list of file paths to tab delimited sample files containing retention time, peak area, mass spectra, and peak quantification method information for each peak as shown below:

Name	Retention Times	Area	Quant Mass/Apexing Masses	Spectra
Alanine	540 , 2.810	1798077	T	116:44973 73:387722 . . .
FAME_8	560 , 3.220	33713901	T	74:522236 87:195537 . . .
Valine	700 , 2.830	1050266	T	73:26438 144:18832 . . .

As shown above, the five required columns (the order shown is required) are the peak name, retention time (outputted by Chromatof as first retention time, second retention time), peak area, quant mass, and mass spectra (cutoff after the first two masses to save space). All of these columns can be specified as output in the data processing method used by the Chromatof software and do not need to be altered post-output. The fourth column is the only column that can change depending on the data processing method used. Chromatof allows for three different methods for quantifying peak areas. The simplest is the total ion chromatogram (TIC), which uses all ions to compute the peak area. This method has no unique quant mass, thus Chromatof outputs "T" for the quant mass as shown above. The second is the unique mass method, which uses an ion relatively unique to a peak to compute the area for a peak in an effort to improve resolution of overlapping peaks. Chromatof will output the unique mass used to compute peak area in the quant mass column if this method is selected as shown below:

Name	Retention Times	Area	Quant Mass/Apexing Masses	Spectra
Alanine	540 , 2.810	1798077	116	116:44973 73:387722 . . .
FAME_8	560 , 3.220	33713901	74	74:522236 87:195537 . . .
Valine	700 , 2.830	1050266	73	73:26438 144:18832 . . .

The final method, apexing masses, is a compromise of the first two. This approach uses several ions that apex at similar retention times to compute the peak area. If this method is use "Apexing Masses" should be outputed as the fourth column instead of "Quant Mass". Chromatof will output each appexing mass separated by a plus sign as shown below:

Name	Retention Times	Area	Quant Mass/Apexing Masses	Spectra
Alanine	540 , 2.810	1798077	116+73	116:44973 73:387722 . . .
FAME_8	560 , 3.220	33713901	74+87+75+101	74:522236 87:195537 . . .
Valine	700 , 2.830	1050266	73+144+75	73:26438 144:18832 . . .

Sample input files for all R2DGC functions should look identical to the SampleA.txt or SampleB.txt example files provided with the package in the inst/extdata/ folder with the exception of the quant method used as described above. Chromatof allows for several other output columns that are not required for R2DGC. We recommend against including these in the input files to reduce memory usage during alignment. The example files have also been reduced to just 13 amino acids and 9 (Fatty Acid Methyl Esters) FAME standards for simplicity, however peak files with several hundred peaks are acceptable. Notice a consistent naming scheme was used for each of the FAME standards. This is required if the user wishes to use these peaks for retention time indexing in downstream processing. Other than retention time index standards the peak names are irrelevant and not used in downstream processing.

2.3 Ion filtering with FindProblemIons

2.3.1 FindProblemIons overview

This is an optional step allowing the user to identify ions across all peak spectras to exclude in downstream processing steps. There are at least two potential reasons to exclude ions. The first is that ions may be absent or present at such a low level in all peaks such that removing it does not affect peak alignment quality, but improves alignment efficiency. By Chromatof includes ion information over a continuous range of masses (we export 70-600) and we've found that at least half of these are present at such low levels (< 1% of all spectras) that they can be excluded from downstream processing without a significant effect. The second reason to exclude an ion is because it may be so ubiquitious that does not contribute to distinguishing different peak spectras. This is common for masses that are derived from the derivitization process used in sample preparation (e.g. m/z 73 for TMS). The **FindProblemIons** function will identify ions that fall into both of these categories. Typically a representative input file is provided for this function and the output is used in downstream processing for all files. Below is an example calling this function on one of the example files:

```
ProblemIons<-FindProblemIons(inputFile=system.file("extdata", "SampleA.txt",
                                                      package="R2DGC"))
```

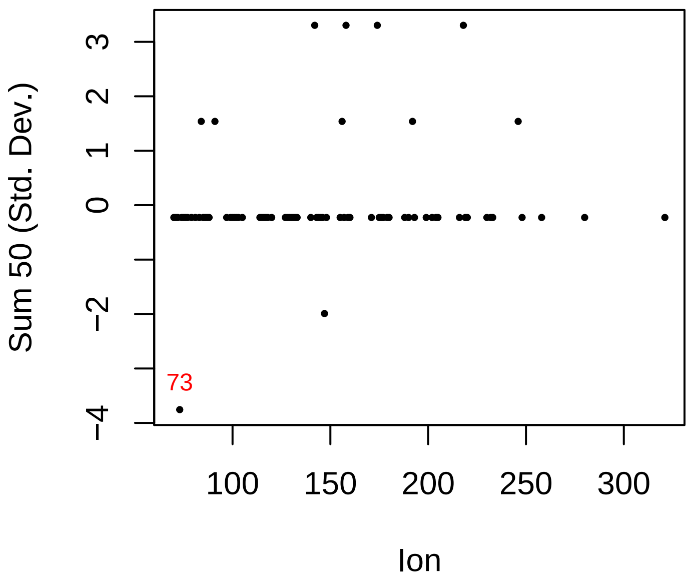

Figure 3: Problem ion scores

As shown in Figure 3, this function outputs (unless *plotData*=FALSE) a plot showing each ion on the X-axis and a score for each ion on the Y-axis. We have defined this score as "Sum50" or the number of peak spectras within a file that have a dot product similarity score greater than 50 after removing a given ion from each spectra. The dot product similarity score is calculated as follows:

First, A and B are each defined as two peaks with spectra that can be represented as vectors $A = [a_{70}, a_{71}, ..., a_{600}]$ and $B = [b_{70}, b_{71}, ..., b_{600}]$ with each element representing an ion intensity. Than the similarity of two spectras, S, can be assessed as the inner product of the two vectors divided by the product of the length of each vector:

$$S = ((A \bullet B)/(||A|| \times ||B||)) \times 100$$

Thus, ions that show a low “Sum50” should likely be excluded from downstream processing because our ability to resolve peaks within each sample or the degree to which each peak has a unique spectra is inhibited by including these ions. This function plots a z-scored Sum50 score for each ion specified by *possibleIons*. We have found that ions with Sum50 scores greater than 2 standard deviations below the mean should be excluded from downstream processing. This threshold can be changed using the *commonIonThreshold* flag. Any ion with a Sum50 score below the specified threshold will be labeled in red on the plot.

This function also outputs a data frame containing information on each ion that should be excluded and whether it was classified as an “absent” or “common” ion:

```
head(ProblemIons)
```

```
##    Ions Status
## 4    73 Common
## 9    78 Absent
## 11   80 Absent
## 13   82 Absent
## 20   89 Absent
## 21   90 Absent
```

The first column of this output is the ion and the second is the reason each ion was outputted. By default absent ions are those that represent less than 1% of the total ion intensity of each peak within the sample. This threshold can also be modified with the *absentIonThreshold* flag. This function defaults to using 1 core for processing, but we recommend increasing this by setting the *numCores* flag to as many cores as available. Increasing the number of cores used exponentially increases processing time. This goes for all functions in the R2DGC package except for the **MakeReference** function, which is less computationally intensive. The number of cores available can be determined using the parallel package’s **detectCores** function:

```
library(parallel)
detectCores()
```

```
## [1] 8
```

2.4 Intra-sample peak compression with PrecompressFiles

2.4.1 PrecompressFiles overview

This is another optional function that allows users to input a list of sample peak files and combine peaks that were that were determined during peak calling to be multiple peaks, but likely represent the same analyte. We have found that a small number of peaks in each sample are split for a variety of reasons including saturation of large peaks. This causes problems in downstream alignments because peaks will semi-randomly be assigned to each of the split peaks resulting in gaps in the final alignment table. This can be avoided by scanning each sample file for peaks that should be combined prior to alignment. An example of this command is shown below with a sample file that has had its Alanine peak split:

```
#Read in file containing split peak
SampleC<-system.file("extdata", "SampleC.txt", package="R2DGC")

CompressionInfo<-PrecompressFiles(inputFileList=SampleC)
```

This function returns a 15 column data frame (5 original columns + 2 parsed retention times from each peak and the sample file path) listing all peak pairs that were combined for each sample.

2.4.2 Computing peak similarity scores

This function uses peak retention times and the dot product of the peak spectra to assess peak similarity as described above (Section 2.3). If we assume A and B are two peaks with spectra that can be represented as vectors $A = [a_{70}, a_{71}, ..., a_{600}]$ and $B = [b_{70}, b_{71}, ..., b_{600}]$ with each element representing an ion intensity. Than the similarity of two spectras, S, can be assessed as the dot product of the two vectors divided by the product of the length of each vector to normalize comparisons of all peaks such that:

$$S = ((A \bullet B)/(||A|| \times ||B||)) \times 100$$

To calculate the final peak similarity, we subtract absolute value of the retention time differences multiplied by their specified penalty weights (P_1 and P_2). Thus, if A and B each have two retention times A_p, A_q and B_p, B_q, the final similarity score between the two peaks can be computed as:

$$S = (((A \bullet B)/(||A|| \times ||B||)) \times 100) - (|A_p - B_p| \times P_1) - (|A_q - B_q| \times P_2)$$

This results in a range of possible peak similarity scores, S, from a perfect score of 100 to $-\infty$. The retention time penalties are set with the *RT1Penalty* and *RT2Penalty* flags and default to 1 and 10 respectively. The second retention receives a higher penalty by default because it is typically a shorter column, thus smaller retention time differences are more important. However, the user should adjust these based on the stability of each retention time. In other words, if the second retention time is highly variable, reduce *RT2Penalty* relative to *RT1Penalty.* The absolute values of *RT1Penalty* and *RT2Penalty* will dictate the overall retention time penalty relative to the spectra similarity. We recommend using a relatively stringent spectra similarity threshold (*similarityCutoff* defaults to 95) as only peaks that are exceptionally similar should be combined. A pairwise comparison of all peaks for each sample is made in this step. Problem ions previously identified by the **FindProblemIons** function (section 2.3) can be filtered prior to performing pair wise comparisons by setting commonIons flag to the first column of the FindProblemIons output dataframe:

```
CompressionInfo<-PrecompressFiles(inputFileList=SampleC, commonIons = ProblemIons[,1])
```

If the *outputFiles* flag is set to TRUE, this function will peform a putative summation of peaks to be combined and write new sample files to the input file path with "_Processed.txt" appended to the end of the file path. It is almost always preferable to go back and combine peaks with the primary peak calling software to ensure proper peak area integration, however these putatively combined peaks can be used for preliminary analysis.

2.4.3 Handling different quant masses

One important thing to note is that the *quantMethod* flag should be set to the quant method used in the Chromatof data processing used to generate the sample files. If TIC or apexing masses are used (quantMethod = "T" or "A") then peaks that are nominated for combining will simply be summed together. However, if the unique mass method is used (quantMethod="U") and peaks nominated have different unique masses, proportional conversion will be performed to convert the peak masses to what they would be if they had the same unique mass. In other words, if a peak area, A, is computed with a unique mass of 73 and a second peak area B, is computed with a unique mass of 74, B is converted to roughly what the mass would have been if its unique mass would have been 73 by multiplying it by the ratio of its ion intensities b_{73}/b_{74}. Thus, in this scenario with differing unique masses the final combined peak area, C, would be computed as follows:

$$C = A + (B \times b_{73}/b_{74})$$

We have found this proportional conversion to be servicable when the unique mass ion intensities are relatively comprable, however, this relationship does break down the more disparate the intensities. This should be a relatively rare occurance because samples with vastly differing unique masses are unlikely to meet the peak similarity threshold for compression or alignment. The spectra of the peak with the larger area is retained for the final output.

2.5 Generating a metabolite standard library with MakeReference

2.5.1 MakeReference overview

The third and final optional function in the package is used to generate metabolite standard libraries for reference in the **ConsensusAlign** function for identifying metabolites from which peaks are observed. A list of file paths to Chromatof files derived from metabolite standards are used as inputs and the output is a dataframe that can be used as input for the standardLibrary flag in the **ConsensusAlign** function. I've provided two example metabolite standard Chromatof files so users can be sure their formatting is correct in the inst/extdata package folder.

These are tab separated files generated by running standards locally with a series of FAME standards as retention indices. There is a header describing each column as is standard output from Chromatof software. Notice these files only require 3 columns and do not need the peak area or quant mass/apexing mass column used for sample files. Below is a table showing the first three lines of the first standard file to demonstrate each of the 3 required columns in the input file:

Name	Retention Times	Spectra
Alanine	545 , 2.880	73:55154 116:51493
FAME_8	565 , 3.310	74:425579 87:225021
FAME_10	845 , 3.240	74:453713 87:208511

As shown above, the only three columns that should be present in the metabolite standard files are the peak name, the retention times and the mass spectra. The rows consist of the metabolite standard peak of interest (Alanine) and 9 FAME standards. Standards that result in multiple peaks should be split up into separate metabolite standard input files. This is the first instance in which we have encountered using retention time standards. We highly recommend using retention time standards such as FAMEs to index your standards. This is recommended because retention times naturally drift as GC columns age or are replaced and retention time standards provides a form of universal retention indexing such that any sample that also has similar retention time standards spiked into it can be compared. It is important to note that a standard naming scheme is applied to the FAME standards in the example. This is required for the aligner to identify the standards in each sample. To make use of retention indexing, be sure each retention time standard is named consistantly across all standards and sample files. Once metabolite standard files are formatted correctly, the standard reference library can be generated easily:

```
Standard1<-system.file("extdata", "Alanine_150226_1.txt", package="R2DGC")
Standard2<-system.file("extdata", "Serine_022715_1.txt", package="R2DGC")
StandardLibrary<-MakeReference(inputFileList = c(Standard1, Standard2),
                               RT1_Standards=paste0("FAME_", seq(8,24,2)))
```

Notice, in addition to providing the file paths to the standards we provided a vector with the names of each of the FAME standards in the *RT1_Standards* flag. A separate set of standards can be provided for use indexing the second retention time with the *RT2_Standards* flag as well. It is important to note, that retention time standards are only helpful if they span a large portion of possible retention times. The FAME standards indicated (C:8, C:10, C:12, C:14, C:16, C:18, C:20, C:22, C:24) are distributed across the first retention time, but have similar second retention times so we have only used them to index the first retention time.

The output is a data frame with two rows representing the two metabolite standards we ran the function on. The first three columns are the original peak name, retention time, and spectra columns from the input files. The following columns represent the two retention times followed by the first retention time relative to each FAME standard to allow for retention time indexing while performing alignments with the **ConsensusAlign** function (see section 2.5.3 below for more details). Though recommended, retention time standards are not required for alignment or standard library construction. If retention time standards are not desired, the user should just submit a metabolite standard Chromatof file formatted identically to the examples above except with only one row describing the metabolite standard of interest.

2.5.2 Finding the pre-formatted standard library that comes with the package

We have provided a pre-formatted standard library with ~300 peaks with rentention indexes calculated with 9 FAME standards (C:8, C:10, C:12, C:14, C:16, C:18, C:20, C:22, C:24). We have used the naming convention described above (Section 2.2), FAME_[number of carbons]. Any user can use this library regardless of whether they have FAME standards spiked into their samples if they perform a retention time standard-free alignment (see section 2.6 below). Users who have spiked these FAME standards into their samples and named the FAME peaks according to the convention described can take advantage of retention indexing. If only a subset of the FAME standards have been used with a user's samples, this library can still be used if the columns corresponding to the missing FAME peaks are deleted from the standard library data frame. The standard library can be accessed easily with the data() command:

```r
data("StandardLibrary_030117")
```

This creates a new variable named "StandardLibrary_030117" in your R environment, which can be used as input for the *standardLibrary* flag in the **ConsensusAlign** function (section 2.6).

2.5.3 Computing and using retention time indexes

The R2DGC package uses a very simple approach to incorporating retention time standards for universal retention time indexing. The maximum amount of retention time the user's retention time standards cover is computed as a total RT length L. Next, the difference between a peak's retention time and each retention time standard is computed as a vector of differences, $D = [d_1, d_2, ..., d_n]$, where d representes the difference in retention time between a peak and a given retention time standard and n is the total number of retention time standards used. Next each element in D is divided by L resulting in a retention index vector, R, with each retention time standard voting on the location of a peak relative to total retention time covered by all retention standards (Figure 4).

$$R = D/L = [r_1, r_2, ..., r_n]$$

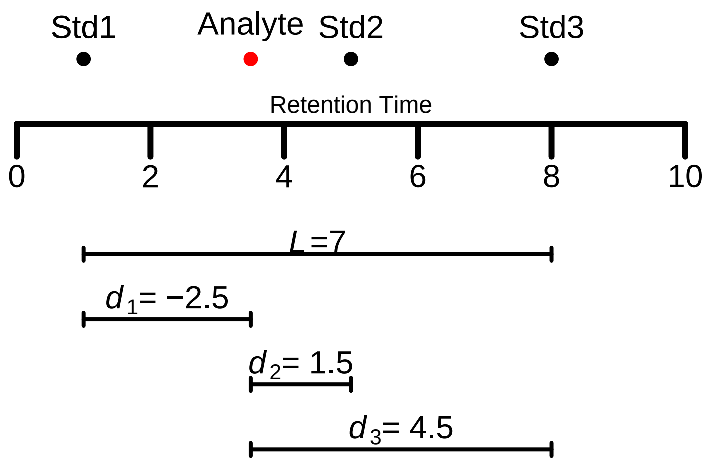

Figure 4: Computing retention index values

Retention index vectors can be compared similarly to simple retention times as described in section 2.4.2 above when computing similarity scores. Thus, retention index vectors R can be relatively easily substituted for retention times. As described in 2.4.2, if two peaks are represented as A and B, then instead of having a primary retention time A_p and B_p, the peaks can have retention index vectors A_R and B_R. The retention difference, W, is then computed as the sum of the absolute value difference between each element of the two peaks respective retention indices:

$$W = \sum_{i=1}^{n} |A_R - B_R|$$

Where n represents the total number of retention standards used. Substituting W into our original similarity score, S, equation gives us:

$$S = (((A \bullet B)/(||A|| \times ||B||)) \times 100) - (W \times P_1) - (|A_q - B_q| \times P_2)$$

The accuracy of this approach increases with the number of retention time standards used as increasing the number of retention time standards provides a finer representation of retention time shifts. Second, the accuracy of this approach breaks down with peaks that occur outside the range of the retention time standards, so the user should use retention time standards that cover the range of retention times of the metabolite peaks of interest. The advantages of this approach include its simplicity, speed of computation, and ability to scale to multiple columns.

2.6 Multi-sample alignment and metabolite identification with ConsensusAlign

2.6.1 ConsensusAlign overview

This is the final and only mandatory function included in the package. It takes a list of input sample Chromatof file paths formatted as described in section 2.2 above as well as an optional metabolite standard library as described in section 2.5 and will align common peaks across the samples and identifies metabolites from which each aligned peak is likely derived from if a standard library is provided. Figure 5 diagrams the basic processed involved in the **ConsensusAlign** function:

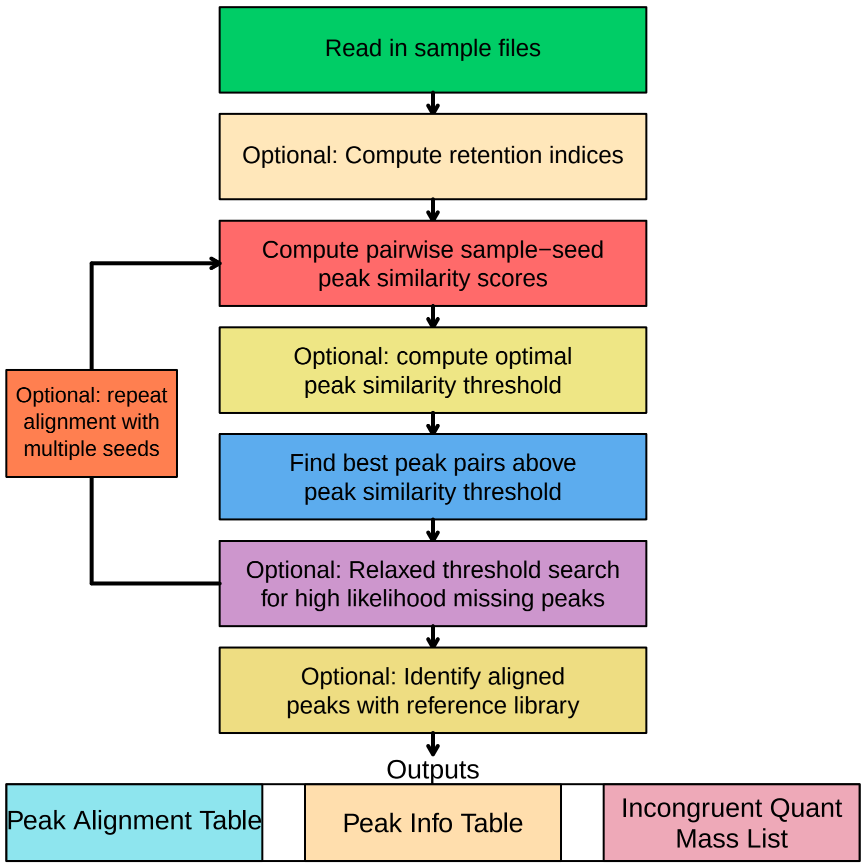
Read in sample files
Optional: Compute retention indices
Compute pairwise sample−seed peak similarity scores
Optional: compute optimal peak similarity threshold
Find best peak pairs above peak similarity threshold
Optional: Relaxed threshold search for high likelihood missing peaks
Optional: Identify aligned peaks with reference library
Optional: repeat alignment with multiple seeds
Outputs
Peak Alignment Table
Peak Info Table
Incongruent Quant Mass List

Figure 5: Overview of ConsensusAlign function

As described above, sample files are read in from file paths provided by the *inputFilePaths* flag. If retention time standards are specified with the *RT1_Standards* or *RT2_Standards* flags, retention indexes are computed for each peak in each sample file as described in section 2.5.2 above. Next, pairwise peak similarity scores are computed for each peak in each sample file and a designated seed file. The seed file defaults to the first file in the *inputFilePaths* argument, however this can be changed with the *seedFile argument*. Similarity scores are computed as described in section 2.4.2 above. Next, the best peak matches (between each sample file and the seed file) that are above the specified similarity score threshold are added to a final alignment table. The similarity score threshold dictates the stringency of the alignment and can be adjusted with the *similarityCutoff* flag - acceptable values usually range from 0 to 95 with higher values resulting in more missing values, but higher confidence alignments. By setting *autoTuneMatchStringency* to TRUE, the function will ignore the user inputted *similarityCutoff* and attempt to compute an optimal similarity score threshold (see section 2.6.2. below). Peaks that are present in sample files, but not present in the seed file, will be iteratively added to the alignment table if they have a similarity score less than the specified *dissimilarityCutoff* flag (defaults 90 less than the *similarityCutoff*) for all other peaks already present in the alignment table. It's important that the *dissimilarityCutoff* is significantly less than the *similarityCutoff* so the same peaks are not split over multiple rows in the alignment table. We have found this is a major source of missing data in other aligners.

After all alignments are complete, the alignment table will be trimmed to only peaks that were successfulled found in a greater or equal fraction of the samples than that specified by the *missingValueLimit* flag (defaults to 0.75). If the *missingPeakFinderSimilarityLax* flag is less than 1 (typical range 0.75-1), the function will perform a second pass alignment on the *missingValueLimit* trimmed alignment table looking for the missing peaks in samples at a relaxed fraction of the initial *similarityCutoff*. These peaks are usually not truly missing, rather they simply had a similarity score that fell just below the initial *similarityCutoff* threshold.

Specifying a seed file to which we compare all other sample files greatly reduces the number of pairwise comparisons necessary for alignment, allowing us to conduct threshold free alignments instead of specifying retention time or spectral match thresholds like many other aligners. However, a disadvantage to this approach is that alignments have the potential to be biased based on the specified seed file. To address this, the user can provide multiple seed files to the *seedFile* flag as a vector (3 is usually sufficient depending on the number of total sample files). If more than one seed file is provided, the alignment will be performed with each seed file and only aligned peaks with greater 50% of peaks consistent across all alignments will be returned. Variable sample values are assigned a median value from all alignments.

After the alignment(s) are complete, if the user provides a metabolite standard library data frame to the *standardLibrary* flag as described in section 2.5 above, the function will compute similarity scores between each peak in the final alignment table and each metabolite standard in the library. The top three hits in the library and their respective similarity scores are reported in the peak information table. For peaks not present in the standard library, the similarity scores will be very low (<0) and these matches can probably ignored. Based on our experience, similarity scores greater than 50 warrant attention as a likely hit.

The last thing to note about this function is that it is compatible with all three Chromatof quant methods, so the user should specify the method used for data processing with the *quantMethod* flag. If the unique mass method is used (quantMethod="U"), any peaks that are aligned, but have different quant masses than the seed file will have their peak area proportionally converted as described in section 2.4.3 above and will be outputted in the incongruent quant mass list. If the apexing mass method is used (quantMethod="A"), if a sample peak has less than 50% apexing masses in common with the seed file and is aligned, the peak and sample will be returned in the incrongruent quant mass list. These occurences should be rare in peaks that pass similarity score thresholds, however, we recommend examing the incongruent quant mass list if using the unique mass or apexing mass quant methods, manually changing the quant mass of the peaks, and reprocessing for final analyses.

Here's a brief example to examine the outputs using the sample input files described in section 2.2, the standard library generated in section 2.5, and filtering problem ions identified in section 2.3 above:

```r
#Find sample input file paths
SampleA<-system.file("extdata", "SampleA.txt", package="R2DGC")
SampleB<-system.file("extdata", "SampleB.txt", package="R2DGC")

#Perform alignment
Alignment<-ConsensusAlign(c(SampleA,SampleB), standardLibrary = StandardLibrary,
                          commonIons = ProblemIons)
```

```
## seed is 1

## Computing peak similarity threshold

## Searching for missing peaks

## Matching peaks to standard library
```

The output consists of a list of three objects. The first is the alignment matrix. This matrix contains all aligned peak areas with the same number of rows as aligned peaks and columns as number of samples.

```r
colnames(Alignment$Alignment_Matrix)<-gsub("^.+/","",colnames(Alignment$Alignment_Matrix))
head(Alignment$Alignment_Matrix, n=3)
```

```
##                    SampleA.txt SampleB.txt
## Alanine_Standard       1798077     2041511
## FAME_8                33713901    42451556
## Valine_Standard        1050266     1045578
```

The next output is the peak info table. This is a dataframe that contains the same number of rows as aligned peaks and at least 7 columns containing information describing each peak. If a standard library is used, as we did above, the first three columns are the top three library hits provided as [standard name]_[similarity score]. The next 7 columns are the core columns always exported and consist of the peak name, retention time, area, spectra quant mass (from seed file) and parsed retention times as they appear in the first seed file or the first sample file a peak was observed in. The final columns are the retention indexes for each peak with the same number of columns returned as retention time standards specified.

The last object returned is the Unmatched_Quant_Masses dataframe. If unique mass or apexing mass quant methods are used, this contains information on peaks that were aligned, but had incongruent quant masses as described above. It contains 12 columns, the original 5 columns from the Chromatof file for each of the two aligned peaks and the file path of each sample file similar to the output described in section 2.4 above. In this example we have used the default TIC quant method so we do not have an UnmatchedQuantMasses dataframe.

```
Alignment$Unmatched_Quant_Masses
```

```
## NULL
```

2.6.2 Computing the optimal similarity score threshold

As described above, the user has the option of manually providing a similarity score threshold for identifying a successful alignment via the *similarityCutoff* flag of in the **ConsensusAlign** function, however, user imposed thresholds are often arbitrary and can be difficult to optimize. Thus, we havve provided the option of allowing the function to find a semi-optimized threshold by setting the *autoTuneMatchStringency* flag to TRUE. We have emprically developed an approach to do this that tests a range of potential similarity score thresholds to maximize the number peaks with at least one successful match, but minimize the total number of successful matches for each peak. More specifically, this approach can be defined as:

$$\forall i \in [1, 2, 3, ..., 100],\ argmax(Y_i/\sqrt{Z_i})$$

where i is the putative similarity score threshold, Y_i is the number of sample file peaks with at least one successful match to a seed file peak, and Z_i is the total number of sample file peaks with a successful match to a seed file peak at a give threshold, i. A successful match is defined as a seed file peak and sample file peak with a similarity score, S, greater than the putative similarity score threshold, i. After testing several data sets we've found this approach typically converges at a maximum similarity score threshold, i, between 1 and 100.

3 Examples use cases

3.1 Use case #1. Align peaks from multiple samples without retention time standards

```
#Find sample input file paths
SampleA<-system.file("extdata", "SampleA.txt", package="R2DGC")
SampleB<-system.file("extdata", "SampleB.txt", package="R2DGC")

#Perform alignment
Alignment<-ConsensusAlign(inputFileList = c(SampleA,SampleB),
                          RT1_Standards = c(), numCores = 4)
```

3.2 Use case #2. Align peaks from multiple samples with retention time standards

```r
#Find sample input file paths
SampleA<-system.file("extdata", "SampleA.txt", package="R2DGC")
SampleB<-system.file("extdata", "SampleB.txt", package="R2DGC")

#Perform alignment
Alignment<-ConsensusAlign(inputFileList = c(SampleA,SampleB),
                          RT1_Standards = paste0("FAME_", seq(8,24,2)),
                          numCores = 4)
```

3.3 Use case #3. Align peaks from multiple samples and identify metabolites from a standard library

```r
#Find reference example standards
Standard1<-system.file("extdata", "Alanine_150226_1.txt", package="R2DGC")
Standard2<-system.file("extdata", "Serine_022715_1.txt", package="R2DGC")

#Make standard library
StandardLibrary<-MakeReference(inputFileList = c(Standard1, Standard2),
                               RT1_Standards=paste0("FAME_", seq(8,24,2)))

#Find sample input file paths
SampleA<-system.file("extdata", "SampleA.txt", package="R2DGC")
SampleB<-system.file("extdata", "SampleB.txt", package="R2DGC")

#Perform alignment
Alignment<-ConsensusAlign(inputFileList = c(SampleA,SampleB),
                          RT1_Standards = paste("FAME_", seq(8,24,2)),
                          standardLibrary = standardLibrary, numCores = 4)
```

3.4 Use case #4. Find problem ions, compress peaks, align peaks from multiple samples with retention time standards, and identify metabolites from a standard library

```r
#Find problem ions
ProblemIons<-FindProblemIons(
  inputFile=system.file("extdata", "SampleA.txt", package="R2DGC"),
  possibleIons = 70:78, numCores = 4)

#Find sample input file paths
SampleA<-system.file("extdata", "SampleA.txt", package="R2DGC")
SampleB<-system.file("extdata", "SampleB.txt", package="R2DGC")
SampleC<-system.file("extdata", "SampleC.txt", package="R2DGC")

#Compress sample peaks (files outputted with _Processed.txt extention)
PrecompressFiles(inputFileList = c(SampleA,SampleB,SampleC),
                 outputFiles = T, commonIons = ProblemIons, numCores = 4)
```

```r
#Find reference example standards
Standard1<-system.file("extdata", "Alanine_150226_1.txt", package="R2DGC")
Standard2<-system.file("extdata", "Serine_022715_1.txt", package="R2DGC")

#Make standard library
StandardLibrary<-MakeReference(inputFileList = c(Standard1, Standard2),
                               RT1_Standards=paste0("FAME_", seq(8,24,2)))

#Perform alignment
Alignment<-ConsensusAlign(
  inputFileList = paste0(c(SampleA,SampleB,SampleC),"_Processed.txt"),
  RT1_Standards = paste("FAME_", seq(8,24,2)), standardLibrary = StandardLibrary,
  commonIons = ProblemIons, numCores = 4)
```

3.5 Use case #5. Align peaks from multiple samples with retention time standards with multiple seed files

```r
#Find sample input file paths
SampleA<-system.file("extdata", "SampleA.txt", package="R2DGC")
SampleB<-system.file("extdata", "SampleB.txt", package="R2DGC")
SampleC<-system.file("extdata", "SampleC.txt", package="R2DGC")

#Perform alignment
Alignment<-ConsensusAlign(inputFileList = c(SampleA,SampleB),
                          RT1_Standards = paste("FAME_", seq(8,24,2)),
                          numCores = 4, seedFile = c(1,2,3))
```