Massifquant Parameter Exploration

Christopher Conley

March 31, 2014

Contents

1 Introduction

Often the default parameters of algorithms for isotope trace (IT) detection can be improved upon for each new data set analyzed. In many multi-parameter algorithms, interest in parameter optimization almost always exceeds patience to do a thorough investigation. A thorough optimization may require hundreds of parameter settings accompanied with (i) an annotation of true IT on a very small subset of the data and (ii) a very clear criteria of evaluation (e.g. f-score). Most analysts cannot afford time to actually conduct this rigorous optimization because it is simply not the end goal.

By understanding where parameters may be improved and algorithmic tendencies, a practical and efficient optimization is attainable. Massifquant is prone to reporting too many false positives, especially at lower intensities. So simple filters of minimum intensity and length for each IT can eliminate many of these at little cost of true positives with the appropriate tools. **The tools & strategy presented here are very intuitive and generally may be applied to other IT detection algorithms like centWave and matchedFilter (with varying success).**

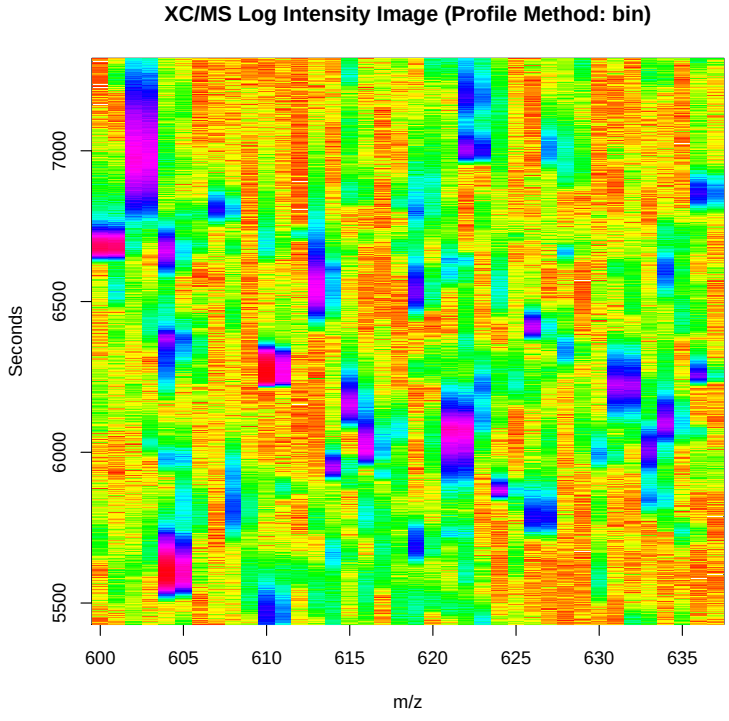
2 Software & MOUSE Data Input

The analysis needs the XCMS software suite to be installed from bioconductor. For further instruction see `http://bioconductor.org/packages/release/bioc/html/xcms.html`. The package *xtable* is a nice utility for displaying tables in dynamically generated documents. The *signal* package is useful for doing Savitsky-Golay smoothing to visualize ITs. The *pdist* package efficiently computes the pairwise euclidean distance between two matrices containing different ITs stored in m/z and time coordinates.

```
library(xcms)  #massifquant (make sure mzR and Rcpp are under same version build)
library(xtable)  #latex table generation
library(signal)  #savitsky-golay filtering
library(pdist)  #euclidean distance between two matrices
```

There are two ways one may access the MOUSE subsample data files. The easiest is to simply clone the *optimize-it* repository on GitHub (see `https://github.com/topherconley/optimize-it`. Alternativly, one may download the MOUSE subsample from the Brigham Young University Scholars' Archive (see `http://hdl.lib.byu.edu/1877/3232`). The Scholars' Archive includes annotation information for those interested. We illustrate the first option below, while the second is commented out. It is important to note that this is merely a subset of the original sample. The exact provenance of the sample is detailed in the Massifquant publication. This highly complex sample is good for illustrating optimization because the behavior of the parameters has been well characterized. Hence, the techniques employed below are well-validated. Once downloaded, unzip the contents to a convenient directory. We import the subsample in mzML format and visually confirm it read the data correctly.

```
# OPTION 1: GitHub
file_mouse <- list.files(path = "mouse_data", pattern = "mzML", full.names = TRUE)
# OPTION 2: scholars' archive archive_path <- 'mouse_annot_complete/dataSourceFile/'
# file_mouse <- list.files(path = archive_path, pattern = 'mzML', full.names = TRUE)
data_mouse <- xcmsRaw(filename = file_mouse)
image(data_mouse)
```



From this simplified rendering of the MOUSE subsample, one can see general locations of very large ITs.

3 Simple Filters

Simple filters are those that post-process the ITs detected by Massifquant's trackers. Minimum length and minimum of the maximum intensity are good examples. With these types of filters, its effective to choose

parameters that will tend to report more false positives and then raise the stringency standard as needed. For instance, one might choose low minimum IT intensity and length thresholds.

	prefilter[2]	peakwidth[1]	ppm	criticalValue	consecMissedLimit
1	65000.00	15.00	25.00	1.50	2.00

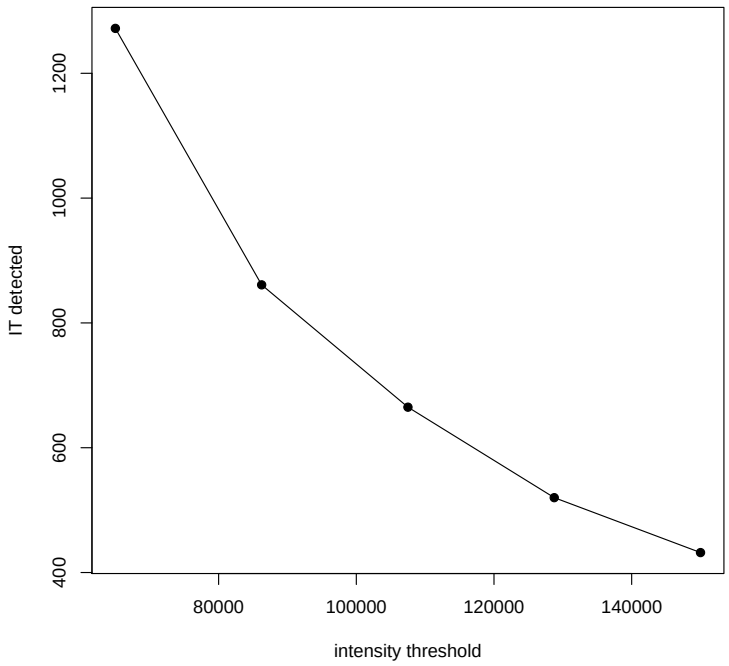
Table 1: First Pass Leniency

Next run Massifquant under these parameters inducing high false positives and with paramter values that are intentionally not exactly the same as what was found to be best in the formal evaluation of MOUSE. It is valuable to see how visually-driven optimization methods presented in this analysis compare with a formulaic f-score approach.

```
ixs <- xcmsSet(files = file_mouse, method = "massifquant", prefilter = c(1, 65000), peakwidth = c(15,
    100), ppm = 25, criticalValue = 1.5, consecMissedLimit = 2, withWave = 0)
```

3.1 Minimum Intensity Filter

The reported IT of this first pass are then subjected to increasing intensity thresholds to see the elimination effect each threshold has on reported IT. As seen in the figure below, the chosen intensity thresholds have a non-negligle effect on the number of peaks reported in the algorithm.



3.2 Relaxed IT Set Contribution (RISC)

For any two intensity thresholds generating IT sets A, B, we can investigate the effect of the more stringent threshold that generated the set B. Specifically, we can evaluate the importance of the IT set $A \setminus B = \{x \in A : x \notin B\}$, denoted as the relaxed IT set contribution (RISC). Often, for simple post-processing parameters like intensity or length, we have a special RISC, where $B \subset A$. Under this subset condition, if the RISC contains important features, then we should keep the more relaxed threshold generating A. Plotting and

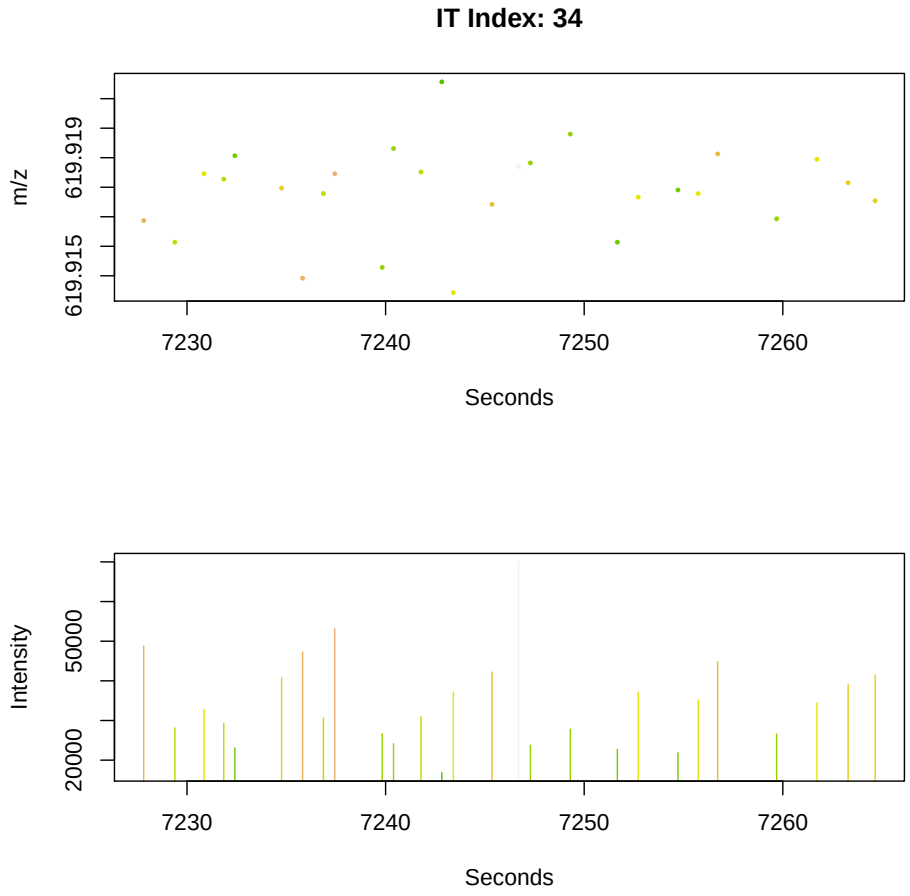
saving arbitrary IT sets (e.g. RISC) to file allows for a relatively quick determination of how to choose between two thresholds, which is what the following "plot IT set" function achieves. In cases where there are sufficient centroids, a Savitsky-Golay smoother overlays a solid black line over the spectra.

```
plot_it_set <- function(xr, xs, filename = NULL, it_set_idx) {
    write_to_file <- !is.null(filename)
    if (write_to_file)
        pdf(filename, height = 8, width = 8, onefile = TRUE)
    ncentroids <- vector(mode = "integer", length = length(it_set_idx))
    cnt <- 1
    for (idx in it_set_idx) {
        par(mfrow = c(2, 1))
        mzr <- xs@peaks[idx, c("mzmin", "mzmax")]
        rtr <- xs@peaks[idx, c("rtmin", "rtmax")]
        it_data <- plotRaw(xr, mzr, rtr, title = paste("IT Index:", idx))
        ymax <- max(it_data[, "intensity"])
        colorlut <- terrain.colors(16)
        col <- colorlut[it_data[, "intensity"]/ymax * 15 + 1]
        plot(it_data[, 1], it_data[, 3], pch = 19, type = "h", col = col, xlab = "Seconds",
            ylab = "Intensity")
        ncentroids[cnt] <- nrow(it_data)
        if (ncentroids[cnt] > 30) {
            nwindow <- round(ncentroids[cnt]/3)
            if (nwindow%%2 == 0)
                nwindow <- nwindow - 1
            sg <- sgolay(p = 2, n = nwindow, m = 0, ts = mean(diff(it_data[, "time"])))
            ysg <- filter(sg, x = it_data[, "intensity"])
            lines(x = it_data[, "time"], y = ysg, lwd = 3, col = "black")
        }
        cnt <- cnt + 1
    }
    if (write_to_file)
        dev.off()
    return(ncentroids)
}
```

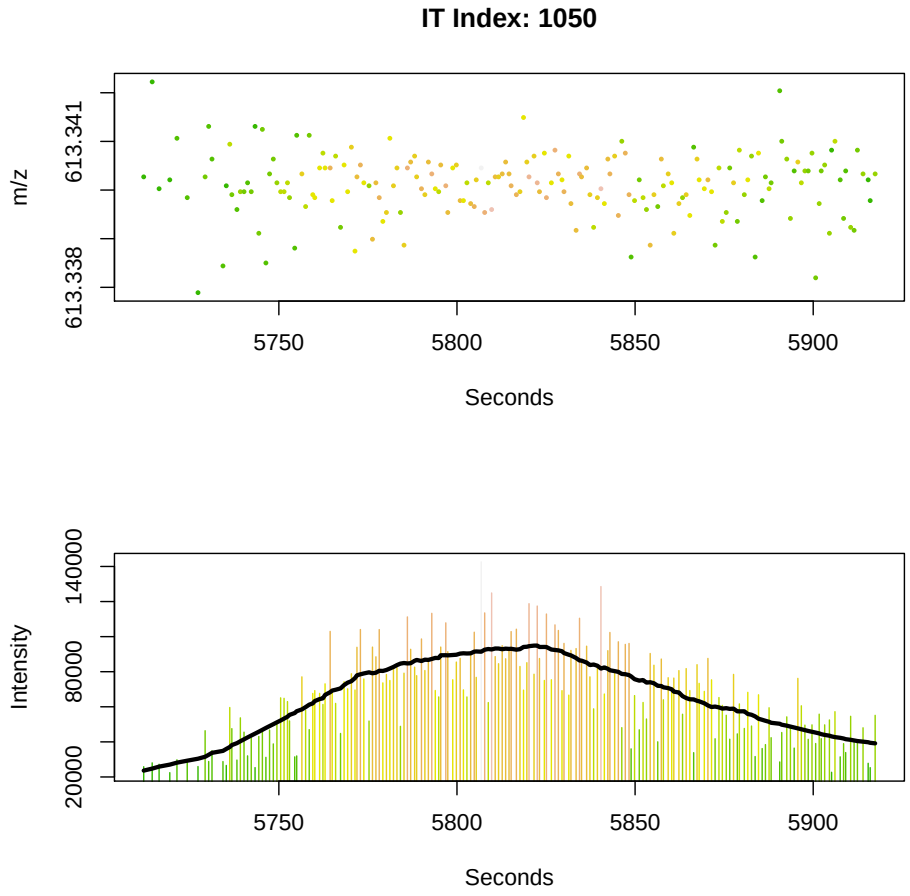
For example, with the five thresholds chosen we will investigate adjacent threshold contributions through the sets $\{RISC_{i,i+1} : i = 1,2,3,4\}$. Below we plot the RISC and save the results to a separate file because the size of each set is respectively (411, 196, 145, 88).

```
iRISC <- sapply(1:(length(idetect) - 1), function(j) {
    f <- paste("figs/iRISC", j, j + 1, ".pdf", sep = "-")
    idx <- setdiff(idetect[[j]], idetect[[j + 1]])
    plot_it_set(xr = data_mouse, xs = ixs, filename = f, it_set_idx = idx)
})
```

Visual inspection shows $RISC_{1,2}, RISC_{2,3}$ are almost pure noise, meaning the first two thresholds are too lenient. For example, the figure below shows a false postive IT with very low intensity.



$RISC_{3,4}$ shows some legitimate ITs that will be lost if we strictly use the 3rd threshold. $RISC_{4,5}$ has about 50% potential true positives (just eye-balling feature shape). Globally this is unacceptable, but remember, most of the false positives come from the low intensity features, which is the case in this set. Depending on the project's aims, the analyst might vary the the tolerance of low-intensity false positives. The next figure comes from the $RISC_{4,5}$ to justify using the 4th threshold instead of the 5th. Further, the manual annotation recorded 589 true ITs, which is in the same neighborhood as the number of features (520) under the 4th threshold. Of course, one can further tune these filter parameters, but for the purposes of this exercise we are satisfied.



	prefilter[2]	peakwidth[1]	ppm	criticalValue	consecMissedLimit
1	65000.00	15.00	25.00	1.50	2.00
2	128750.00	15.00	25.00	1.50	2.00

Table 2: Second Parameter Pass

3.3 Minimum Length Filter

Now we can potentially improve upon the filter for the minimum length in the same fashion applied to the minimum intensity. Run Massifquant with the updated min intensity setting (prefilter[2]) and the result is slightly different than 520. The reason is that the segmentation correction was applied under a more stringent intensity than the original parameter setting. The difference should not be significant.

```
lxs <- xcmsSet(files = file_mouse, method = "massifquant", prefilter = c(1, 128750), peakwidth = c(15,
    100), ppm = 25, criticalValue = 1.5, consecMissedLimit = 2, withWave = 0)
```

To get the length of each feature, we could simply subtract the retention time minimum from the maximum, but this is slightly inaccurate because of time warping that has not been corrected for.

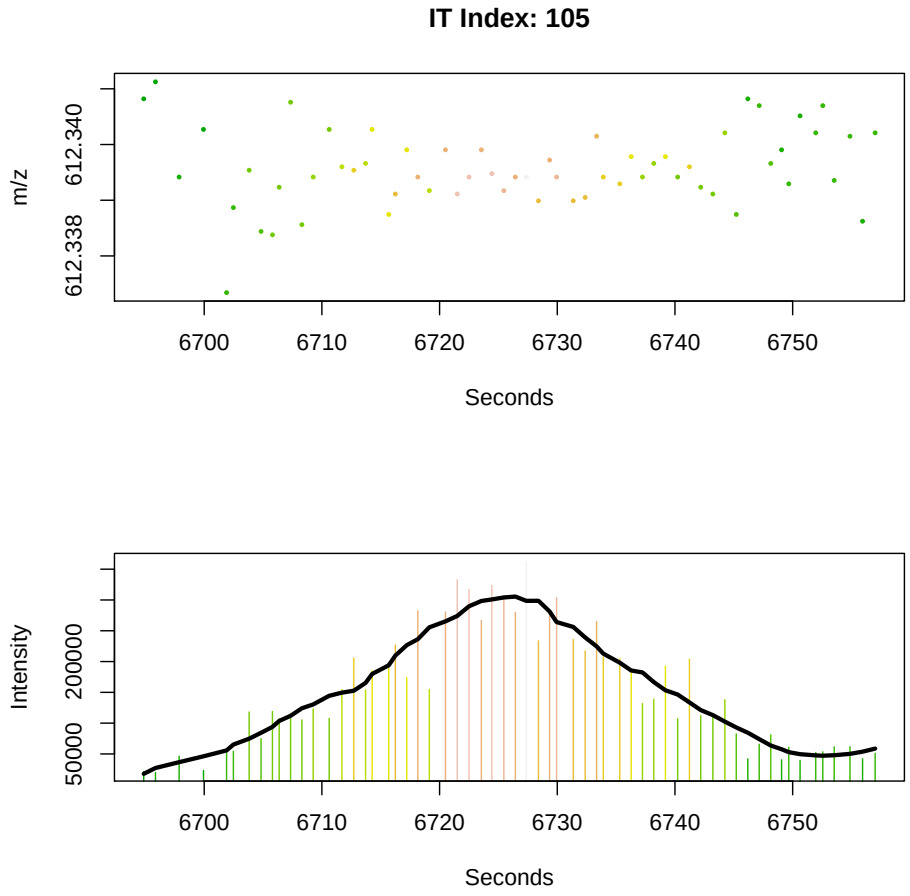
```
lidxs <- seq_len(nrow(lxs@peaks))
len_lxs <- plot_it_set(data_mouse, lxs, "figs/tmp.pdf", lidxs)
rm_tmpfig <- system("rm figs/tmp.pdf")
```

```
lthreshold <- seq(from = 15, to = 80, by = 15)
ldetect <- lapply(lthreshold, function(l) which(len_lxs >= l))
lnumdetect <- sapply(ldetect, length)
```

The RISCs are generated under the varying length thresholds.

```
lenRISC <- sapply(1:(length(ldetect) - 1), function(j) {
    f <- paste("figs/lenRISC", j, j + 1, ".pdf", sep = "-")
    idx <- setdiff(ldetect[[j]], ldetect[[j + 1]])
    plot_it_set(xr = data_mouse, xs = lxs, filename = f, it_set_idx = idx)
})
```

For the minimum length filter, it is not until about $RISC_{4,5}$ do we start to see many legitimate features that would be lost if the stringency exceeded the 4th length threshold. Below is a an example of a true postive IT that would be unreported under the 5th length threshold. Losing true positive ITs is the cost of being too stringent.



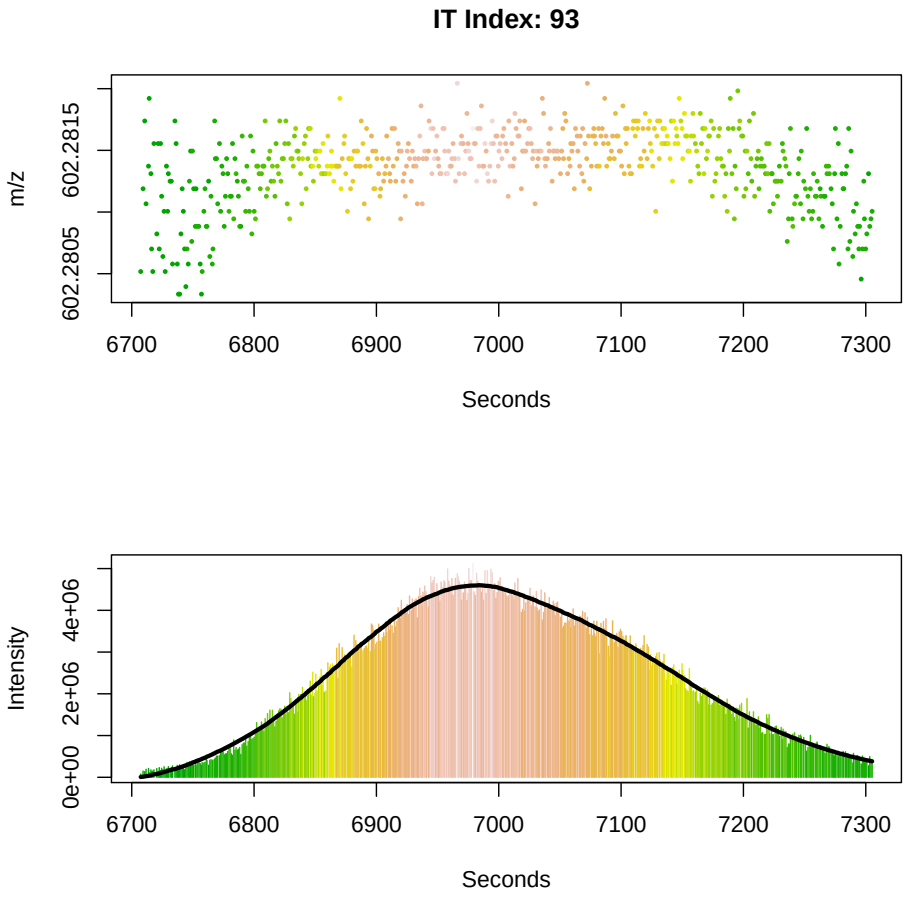
The simple RISC filter strategy affords predictable optimization outcomes. The filters eliminate a lot of false positives in the subsample that would likely be encountered on the whole sample. It also happens to preserve many prominent IT.

3.4 Comparing visual vs. formula-driven optimization outcomes

How does a visual RISC optimization strategy compare to a greedy search cross referencing reported ITs to an annotation with precise metrics (e.g. f-score) as presented in the Massifquant publication? The

former is quick and simple, while the latter is time-consuming and more sophisticated. Can the simplicity of exploratory data analysis be justified? Here Massifquant is run once more with the final length and intensity filters and the IT set generated is visually confirmed in the pdf file,"fxs-out-ITs". The same visualization could be done for the greedy-search optimized parameters. The figure below illustrates a prominent IT from the MOUSE subsample.

```
fxs <- xcmsSet(files = file_mouse, method = "massifquant", prefilter = c(1, 128750), peakwidth = c(60,
    100), ppm = 25, criticalValue = 1.5, consecMissedLimit = 2, withWave = 0)
all_idxs <- seq_len(nrow(fxs@peaks))
plot_it_set(data_mouse, fxs, "figs/fxs-out-ITs.pdf", all_idxs)
plot_it_set(xr = data_mouse, xs = fxs, it_set_idx = 93)
```



The IT set generated under the RISC-driven approach is then formally evaluated based on the annotation of MOUSE. Of course, this comparison has the limitation that we chose the starting parameters cognizant of the greedy-search optimized ones. The strength of comparison would be much greater if we were blind to the formal optimization.

	f-score(1)	IT sensitivity	IT precision	sample sensitivity	sample specificity
Greedy Search	0.7952	0.7284	0.8755	0.9334	0.7458
Visual	0.8563	0.9355	0.7894	0.9610	0.7232

Table 3: Greedy Search vs. Visual Optimization Results

The quick visual optimization compares well to a greedy search and happens to report less false positives at the cost of reduced sensitivity. The similarity boosts confidence in both methods' validity, despite

partial evaluation dependence.

4 More Complicated Parameter Exploration

Other parameters not limited to, but including *criticalValue*, *ppm*, and *consecMissedLim* can be optimized in Massifquant. For *criticalValue* and *consecMissedLim*, one parameter setting may not be a subset of another, and the tools to diagnose exact RISCs is not as precise. It should be noted that *criticalValue* has been shown to vary little in performance for a wide range of values on both MOUSE and MM14 data sets (see Figures S5, S12 of Massifquant publication). It is probably one of the last parameters that should try to be optimized. An approximate RISC can be generated by evaluating the euclidean distance on the vectors [min(m/z), max(m/z), min(time), max(time)] between ITs of the respective parameter settings. The function "dset" reports the respective indices of the approximately unique ITs contributed under different parameters. We say the RISC is "approximate" because the euclidean distance sometimes identifies the same IT in two different parameter settings unique to both.

Massifquant currently hides several of the Kalman Filter model parameters (e.g. m/z estimation noise) from user manipulation that may yet still be improved. Those interested in further extending Massifquant, especially optimizing or integrating less intuitive parameters or IT filters may benefit from applying the evaluation methods contained in this section.

```
dset <- function(xs1, xs2, tol = 0.001) {
    it1 <- xs1@peaks[, c("mzmin", "mzmax", "rtmin", "rtmax")]
    it2 <- xs2@peaks[, c("mzmin", "mzmax", "rtmin", "rtmax")]
    d <- as.matrix(pdist(X = it1, Y = it2))
    uniq_idx1 <- sapply(seq_len(nrow(d)), function(r) all(d[r, ] > tol))
    uniq_idx2 <- sapply(seq_len(ncol(d)), function(c) all(d[, c] > tol))
    list(uniq1 = which(uniq_idx1), uniq2 = which(uniq_idx2))
}
```

Now investigate the difference in unique IT produced by different settings of *ppm*. Massifquant must be run two different times to capture this information.

```
ppmthreshold <- c(2, 10)
# run Massifquant for different ppm tolerance
ppm_xs_gen <- function(p) {
    xcmsSet(files = file_mouse, method = "massifquant", prefilter = c(1, 128750), peakwidth = c(60,
        100), ppm = p, criticalValue = 1.5, consecMissedLimit = 2, withWave = 0)
}
ppm_xs <- lapply(ppmthreshold, ppm_xs_gen)
```

In the case of these two ppm settings, there is absolutely no difference in the outcome (no unique indices reported.)

```
dppm <- dset(ppm_xs[[1]], ppm_xs[[2]])
dppm

## $uniq1
## integer(0)
##
## $uniq2
## integer(0)
```

4.1 Effect of Segmentation Correction

Sometimes Massifquant does not fully track an IT, and reports the partitioned segments. The *union* parameter turns on a small corrective procedure to combine contiguous m/z IT tracks. While not all segmentation-correction is favorable (e.g. combining consecutive peaks), it does help in some instances; so an analyst can quickly determine the cost or benefit with minimal effort. The code that follows illustrates how.

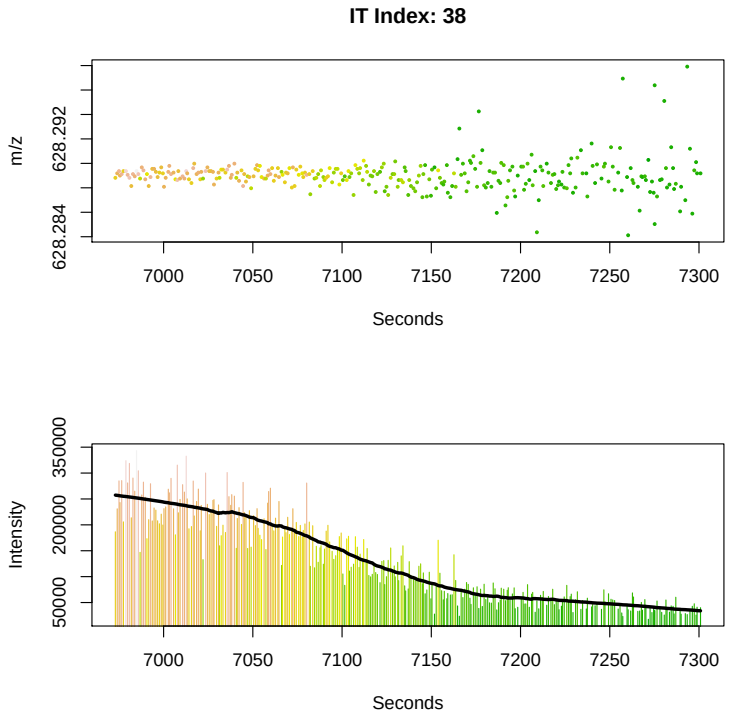
```
union_status <- c(0, 1)
# run Massifquant under different setting
union_xs_gen <- function(u) {
    xcmsSet(files = file_mouse, method = "massifquant", prefilter = c(1, 128750), peakwidth = c(60,
        100), ppm = 10, criticalValue = 1.5, consecMissedLimit = 2, withWave = 0, unions = u)
}
union_xs <- lapply(union_status, union_xs_gen)
# RISCs determined
du <- dset(union_xs[[1]], union_xs[[2]])
du
```

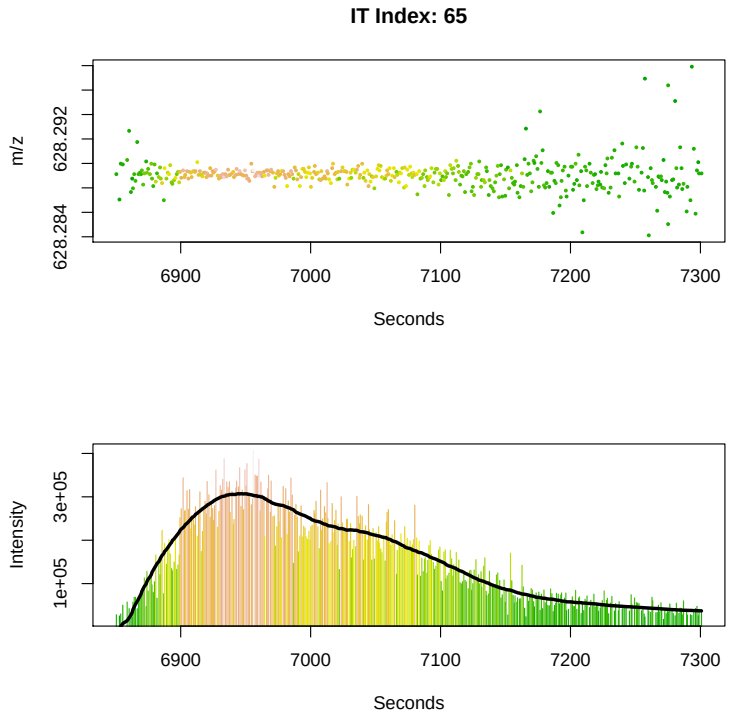
These RISC sets are easily saved to file for documentation.

```
plot_it_set(xr = data_mouse, xs = union_xs[[1]], filename = "figs/no_uRISC.pdf", it_set_idx = du$uniq1)
plot_it_set(xr = data_mouse, xs = union_xs[[2]], filename = "figs/uRISC.pdf", it_set_idx = du$uniq2)
```

We illustrate a useful segmentation correction example. IT index 38 has no segmentation correction, while index 65 shows the corrected version; the indices are different because they are different Massifquant runs.

```
plot_it_set(xr = data_mouse, xs = union_xs[[1]], it_set_idx = 38)
plot_it_set(xr = data_mouse, xs = union_xs[[2]], it_set_idx = 65)
```





5 Conclusions

This analysis has demonstrated the intuitive & visual ease of sufficiently optimizing Massifquant to a particular sample without requiring a manual annotation or complex evaluation criterion. The method is relatively quick, simple, and general enough to apply to other IT detection algorithms.

6 Session Info

```
sessionInfo()

## R version 3.0.2 (2013-09-25)
## Platform: x86_64-apple-darwin10.8.0 (64-bit)
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] codetools_0.2-8    pdist_1.2          signal_0.7-3       MASS_7.3-30
##  [5] xtable_1.7-3       xcms_1.39.5        Biobase_2.22.0     BiocGenerics_0.8.0
##  [9] mzR_1.8.1.1        Rcpp_0.11.0        knitr_1.5
##
## loaded via a namespace (and not attached):
## [1] evaluate_0.5.1 formatR_0.10   highr_0.3      stringr_0.6.2  tools_3.0.2
```