Inside version 3, we find that test\_1.py in the ClintBestMassFragmentChooser directory gives the following calculated results, compared to the following (old) expected results:

New Calculated Results:

[(26.0, 43.0, 45.0), (26.0, 45.0, 46.0), (15.0, 26.0, 31.0), (25.0, 26.0, 31.0), (26.0, 27.0, 31.0), (26.0, 29.0, 31.0), (26.0, 30.0, 31.0), (26.0, 31.0, 43.0), (26.0, 31.0, 45.0), (26.0, 31.0, 46.0)]

Old Expected Results:

[(26.0, 27.0, 28.0), (26.0, 27.0, 31.0), (26.0, 27.0, 45.0), (26.0, 27.0, 29.0), (26.0, 27.0, 46.0), (25.0, 26.0, 27.0), (26.0, 27.0, 43.0), (15.0, 26.0, 27.0), (26.0, 27.0, 30.0), (26.0, 28.0, 31.0)]

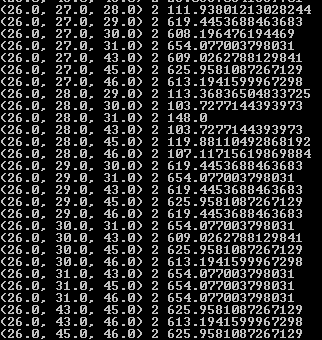
they are different, so the 1st question is: are the new choices better? The answer to that question is yes. All of these solve for both molecules, but the old expected results do not choose as unique fragments. Look at the file AcetaldehydeNISTRefMixed2someMassesHighlighted.xlsx

we see that the choice of (26.0, 27.0, 28.0) does not even give complete uniqueness, and that a bunch of these choices do not even include mass 31, for example.

With the new calculated results, we see a bunch of mass 31 choices.

However, there is still a problem, because we see 26, 27, 31. And that is not a good choice relative to 26,28,31 yet no choice with 28 appears.

Printing out the summed significances, we get the following, which shows that 26,27,31 solves for both molecules and has a summed significance of 654.077, while 26,28,31 as a summed significance of 148.0. Clearly there’s some kind of problem, and we need to figure out why.



Looking into the code, we see that summedSignificance is returned as a ZERO by ExtentOfSLSUniqueSolvable if things are not further solvable, and that it is only returned as a non-zero when things are (at least) partially solvable.

So for example, with 26, 31, 43, it looks like we get the summedSignificance for 26,31, and that the summedSignificance is neglected when 43 comes along. Which is why 26, 31, 46 and 26, 31, 45, have the exact same value of 654.077

So maybe I just need to add what is not yet solved to the else statement? … so extent solvable dominates, and the other ones are still accounted for? No! That’s not the problem! The problem is… why does 26,27,31 appear over 26,28,31? Let’s look at the outputs… (changed buffers under properties and options of the command prompt, had to right click on the bar at top to do that, turned on quick edit so can copy out of it).

(26.0, 28.0, 31.0)

1 [1. 0.] 99.0 [28.0]

(26.0, 28.0, 31.0)

2 [1. 1.] 148.0 [28.0, 31.0]

2 [1. 1.] 148.0 [28.0, 31.0]

(26.0, 28.0, 31.0) 2 148.0

(26.0, 27.0, 31.0)

1 [0. 1.] 49.0 [31.0]

(26.0, 27.0, 31.0)

2 [1. 1.] 654.077003798031 [31.0, 26.0]

2 [1. 1.] 654.077003798031 [31.0, 26.0]

(26.0, 27.0, 31.0) 2 654.077003798031

Apparently the real difference is that 31 got chosen initially in the second case, but not the first. Why not? I think there’s a 5% filter on, so m28 may be considered unique and better. Or more correctly, equal as a “unique” at 100.

Trying to print out the significance matrices also…

significance matrix:

[[605.0770038 4.72771444]

[ 99. 0. ]

[ 0. 49. ]]

(26.0, 28.0, 31.0) 2 148.0

significance matrix:

[[605.0770038 4.72771444]

[394.19943829 12.93801213]

[ 0. 49. ]]

(26.0, 27.0, 31.0) 2 654.077003798031

We see that 28 is indeed considered a unique mass. Also, we see that for some reason the 26 value is more significant than the 28 value. Need to figure out how this is calculated. Is it giving 0 for infinity?

This is apparently calculated in generateSignificanceMatrix and is a function that Clint made inside ExtentOfSLSUniqueSolvable .py. It’s the function that I looked at knowing it needed to be tested!

But it does use MSRESOLVE.ElemSignificanceCalculator

After further thought and looking at things at home, I realized the problem is that we use a value of 0 for the significance when a value is 0. The real issue is that the feature was not designed for cases when SLSUnique is already solved and we just want to find out what other masses to use. So now the plan is to give significances that are high (like 100, for example) for cases that are 0. The way am going to implement this involves putting a “cap” on some of the terms inside the significance values at the element level.

As described on Andrea’s poster (and in \2018WattCharlesHERESummer2018\MSRESOLVE Development\0-ProcDoc.docx):

Equation 1. Equation used to calculate the significance of a given mass fragment (for a given molecule, the c is fixing the molecule). yc is the intensity of the mass fragment for that given molecule, yi is the intensity of that same mass fragment when looped across all molecules (i is the molecule index). The sum of the significance across all of the mass fragments served significance pre-check.

Note: I have added some clarity to the sentences above relative to what was on Andrea’s poster.

Right now, if there is anything in the summation term with a denominator of 0, we will remove that point entirely from the summation because we cannot have divided by infinity. That occurs inside IndElemSignificanceCalculator, if rowDataArray[moleculecounter] != 0

At present, the summation term simply does not add anything when the number is 0.

Inside version 5 I have fixed things inside IndElemSignificanceCalculator so that there is now

indSummationTermCap = maxIntensityPossible/minThreshold – 1

where maxIntensityPossible has a default of 1.

I then propagated these variables through bestMassFragChooser.py and ExtentOfSLSUniqueSolvabl.py in the ClintBestMassFragmentChooser directory.

Now, the outputs are different:

(26.0, 28.0, 31.0)

1 [1. 0.] 1998.9999999999998 [28.0]

(26.0, 28.0, 31.0)

2 [1. 1.] 2998.0 [28.0, 31.0]

2 [1. 1.] 2998.0 [28.0, 31.0]

significance matrix:

[[ 605.0770038 4.72771444]

[1999. 0. ]

[ 0. 999. ]]

(26.0, 28.0, 31.0) 2 2998.0

(26.0, 27.0, 31.0)

1 [0. 1.] 999.0 [31.0]

(26.0, 27.0, 31.0)

2 [1. 1.] 1604.077003798031 [31.0, 26.0]

2 [1. 1.] 1604.077003798031 [31.0, 26.0]

significance matrix:

[[605.0770038 4.72771444]

[394.19943829 12.93801213]

[ 0. 999. ]]

(26.0, 27.0, 31.0) 2 1604.077003798031

So now it does say that 26 28 31 is a good combination.

The calculated best ones are now:

[(28.0, 45.0, 46.0), (15.0, 28.0, 31.0), (25.0, 28.0, 31.0), (26.0, 28.0, 31.0), (27.0, 28.0, 31.0), (28.0, 29.0, 31.0), (28.0, 30.0, 31.0), (28.0, 31.0, 43.0), (28.0, 31.0, 45.0), (28.0, 31.0, 46.0)]

This seems like a very reasonable set! I think that the feature is now ready. In version 6 I will probably do a test with a bigger