Finding Gaps Between Mass Peaks - Analysis and Design Decisions

BELOW STILL WIP

We have a list of mass spectrometry readings, with a mass value and an intensity value. First, we want to remove all low-intensity readings. Then, we want to find gaps in the masses that correspond to the masses in some table of compounds (with some tolerance, either percentile or absolute). Then we want to find all sequences of masses where one mass begins on a peak as another ends, with no subsequences. This is just a matter of comparing masses then checking if they lie within some threshold.

This problem can be modelled as a DAG (directed acyclic graph), where each vertex is enumerated and vertices can only have edges to vertices with higher numbers, edges have a label, which is a list of all compounds that could be between the peaks the vertices represent, and these edges and their labels are unknown, but discoverable.

In this case, vertices are individual mass spectrometry readings, and edges are potential compounds from gaps between mass peaks.

The total number of sequence tags in the worst-case is:

$$\sum_{s=2}^{n}\sum_{r=1}^{s-1}((s-1)C(r)).a^r$$

for input size of vertices n, subgraph size s, sequence tag length r and alphabet size a. To demonstrate this, consider adding another nth node at the first position in the graph. To construct a sequence tag, we must choose up to r of the following $n-1$ nodes. Since there the vertices are ordered, there is only one valid ordering and therefore this is $(n-1)C(r)$. To find sequence tags of all lengths for this new node, we sum over all sequence tag lengths r, so $\sum_{r=1}^{n-1}(n-1)C(r)$. Then we need to repeat this for each node... So we iteratively sum the results of all subgraphs from two nodes to n nodes. This is $\sum_{s=2}^{n}\sum_{r=1}^{s-1}(s-1)C(r)$. This so far concerns all combinations of peaks (i.e. vertices), but we also need to consider the actual tag itself. There are a possibilities at each position in the tag, meaning there are a^r possibilities total for each tag of length r. So we have (finally):

$$\sum_{s=2}^{n}\sum_{r=1}^{s-1}((s-1)C(r)).a^r$$

This should be $O(e^n)$ (to verify), but we don't intend to exhaustively list every potential sequence tag. Rather than dealing with each sequence tag individually for a sequence of peaks, we can choose to represent each edge once

and have multiple possibilities represented at one point in the tag. This cuts us back down to:

$$\sum_{s=2}^{n}\sum_{r=1}^{s-1}(s-1)Cr$$

$$=\sum_{s=2}^{n}2^{(s-1)}-1$$

...

There are multiple design decisions to be considered here both in order to cut down the potential solutions and to achieve better running time within the algorithm. The following statements only concern the case where $r = 1$ or $r = n$, so be aware that they'll change.

Firstly, the most naive algorithm treats any vertex as a potential beginning point for a sequence tag. This means for any vertex k, in the worst-case (which is very unlikely) we must check $(k-1)+(k-2)+...+2+1$ vertices. This is $O(n^2)$.

However if we visit a later vertex as part of discovering an earlier one, we know that any sequence tag that we visit as part of another one must be a subsequence. Then we can ignore these vertices as starting points. Then each new kth input can only be visited at most $c(k-c-1)$ times, where c is the number of previously visited nodes s.t. $0 < c < k-2$.
...
This resolves to $O(n)$.

Furthermore, we can search this graph either by first compiling all discoverable edges into some auxiliary data structure and then walking across it, or we can construct walks as we discover edges. While the auxiliary data structure method can be done by either breadth-first search or depth-first, it is slightly more naturally predisposed to breadth-first search, and the latter almost exclusively favours depth-first. The auxiliary data structure of course has the disadvantage of having to traverse it, so is slightly slower in best-cases, but prevents recalculation so should be significantly faster in worst-cases (and it also lets us maintain a more concise stack and fewer tables of mass differences). Traversing this auxiliary data-structure is also $O(n)$, by the same logic as above (it is essentially a model of the same graph with the edges discovered) but it should be significantly sparser than the data itself.

Thirdly, we can either discover edges by naive comparison with a Python for loop (saving a list of differences to prevent recalculation for each compound) or by mass-calculation with NumPy arrays.

Combinatorial possibilities