MATCH
Communications in Mathematical and in Computer Chemistry

MATCH Commun. Math. Comput. Chem. **88** (2022) 321–350
ISSN: 0340–6253
doi: 10.46793/match.88-2.321L

Enumerating Possible Molecular Formulae in Mass Spectrometry Using a Generating Function Based Method

Sean Li[a,*], **Björn Bohman**[a,b], **Dylan Jayatilaka**[a]

[a] *School of Molecular Sciences, The University of Western Australia, Perth*

[b] *Department of Plant Protection Biology, Swedish University of Agricultural Sciences, Lomma*

sean.li@research.uwa.edu.au, bjorn.bohman@slu.se, dylan.jayatilaka@uwa.edu.au

(Received January 28, 2022)

Abstract

What molecular formulae correspond to a given mass M? Here, we address this problem—the compomer problem of mass spectrometry. We describe, implement, and test a straightforward and practical *aufbau* algorithm which leads to the *molecular formula tree* from which all possible solutions may be obtained. The algorithm was inspired from the generating function method used in combinatorics, to count the number of compomers. The molecular formula tree can be generated in linear time with regards to the number of elements N, and in quadratic time with respect to the mass M; and we demonstrate that the memory requirements do not exceed that of a standard laptop for the formula of a small molecule, say about 10 elements and a mass of 1000 Da. Nevertheless, we still discuss and demonstrate how a range of heuristics can be used to mitigate the memory requirements.

*Corresponding author.

1 Introduction

Mass spectrometry (MS), often paired with gas or liquid chromatography, is a crucial tool in identifying compounds within complex chemical mixtures. This technique plays an important role in metabolomics [22], natural product chemistry [8], and environmental chemistry [24].

An important part of identifying unknown molecules using mass spectrometry is the assignment of a list of possible molecular formulae to a given fragment in the mass spectrum. Our motivation in pursuing this compomer problem is to advance the field of *de novo* structure determination for low-mass molecular compounds. The *de novo* problem of mass spectrometry—that is, the determination of unknown chemical compounds *only* from their mass spectra, without the use of spectral libraries—is a general chemical problem within many specific applications.

We are particularly interested in chemical ecology, where one is often trying to identify small-molecule semiochemicals. Moreover, several commercial tandem mass spectrometers are now available, which allow the collection of mass spectra not only for the target compound, but for its fragments. With this extra information, and with further advances in instrumentation, *de novo* structure determination is a welcome possibility, provided the compomer problem, among others, is addressed.

1.1 The compomer problem in more detail

Suppose the compound of interest may contain N possible elements,

$$\mathbf{E} = [E_1, E_2, \ldots, E_N] \tag{1}$$

referred to here as the *alphabet*, each with an associated mass

$$\mathbf{m} = [m_1, m_2, \ldots, m_N]. \tag{2}$$

Then, if the compound has mass M, we can write

$$M = \sum_{i=1}^{N} n_i m_i = \mathbf{n} \cdot \mathbf{m}, \tag{3}$$

where n_i, the multiplicity of element E_i, is a natural number equivalent to the coefficient of a particular element within the molecular formula [17]. For example, in the alphabet $\mathbf{E} = [\mathrm{C}, \mathrm{H}, \mathrm{N}, \mathrm{O}]$ with corresponding masses $\mathbf{m} = [12, 1, 14, 16]$ the molecular formula $\mathrm{C_2H_4O_2}$ with $M = 60$ may be expressed as $\mathbf{n} = [2, 4, 0, 2]$.

The compomer problem is then this: for a given alphabet with masses $\mathbf{m}$, and a given total M, compute all possible decompositions; multiplicities $\mathbf{n}$, which satisfies the above equation. For example, for $M = 60$ and with the alphabet and masses as given before, one possible solution to the above equation would be $\mathbf{n} = [2, 4, 0, 2]$, but another, not necessarily chemically relevant solution, would be $\mathbf{n} = [0, 60, 0, 0]$, corresponding to $\mathrm{H_{60}}$.

A key observation which motivates the method in Section 2.2 is that each decomposition (compomer) maps to an *integer partition*. For example, the compomer $\mathbf{n} = [2, 4, 0, 2]$ corresponds the integer partition $60 = 12 + 12 + 1 + 1 + 1 + 1 + 16 + 16$.

1.2 Previous work on the compomer problem

Since the compomer problem is well known, it is not surprising that there has been much work on it, which will be covered in this section. From a theoretical point of view, the time taken to list (enumerate) all compomers increases monotonically with respect to the number of compomers, which is approximately [5]

$$\gamma(M) \sim \frac{M^{N-1}}{(N-1)!\,\Pi_{i=1}^{N} m_i} \sim M^{N-1} \tag{4}$$

Thus, for larger masses and alphabets, or if the compomer problem needs to be repeatedly solved, the time taken to list all chemical formula, or to store all the solutions, quickly become prohibitive.

One way to increase the efficiency of solving the compomer problem is to implement heuristics that eliminate fruitless partial solutions. A very early example of this approach was implemented by Dromey and Foyster [10]. These authors split the compomer problem into two subproblems: for molecules involving the CHNOPS elements, solutions of the compomer

problem for the [N,O,P,S] elements with $m_1 \leq M$ were first found in the "naive" way, by iterating through all possible combinations of elements and eliminating those with $m_1 > M$. Then, for each compomer found in this step, all solutions of the compomer problem for [C, H] for $m_2 = M - m_1$ were found. Thus, combining some compomer $\mathbf{n}_1$ with mass m_1 with any of the C, H compomers $\mathbf{n}_2$ yielded a compomer within the alphabet [C,H,N,O,P,S] with mass equal to $m_1 + m_2 = M$.

Another heuristic involves expressing molecular formulae in terms of *formula components* (e.g CH_2), rather than in terms of the individual constituent elements. This reduces the number of combinations that need to be considered, and it even reduces the number of variables (i.e N). This technique was first utilised by Furst and coworkers to enumerate molecular formulae containing the elements C, H, N and O [13].

Later, Green and coworkers elaborated this technique introducing the concept of *low-mass moieties* (LMMs) [14, 20]. LMMs are formula components that have an integral mass of zero, such as CH_4O_{-1} or C_4O_{-3}. Given an arbitrary "seed" molecular formula of a fixed integral mass M, it is possible to generate more formulae possessing the same integral mass by adding or subtracting LMMs from the original formula. Of course, one has to check that the final result has non-negative multiplicities in order to be a valid solution. The LMM approach could reduce N by up to 3, and it was integrated into the CHOFIT software package [14].

It should be noted that the formula component methods still rely on searching over all possible combinations of elements or LMMs. Furthermore, depending on the LMMs selected, these methods are not guaranteed to list all valid molecular formulae for an arbitrary alphabet.

Böcker and coworkers have proposed and implemented a method which can enumerate all valid compomers using ideas from dynamic programming. In particular, an "extended residue table" for the alphabet is first constructed using a round-robin algorithm. Then, all possible compomers are found via a recursive divide-and-conquer approach, where the residue table is used to truncate the search space by allowing masses which cannot be decomposed with a given (sub)alphabet to be determined by table lookup, terminating any "fruitless" subproblems early on [5, 7]. An ad-

vantage of this method is that the time taken *per decomposition* is independent of M, however given that the number of decompositions increase with respect to M, ultimately the time taken to list all formulae will still increase.

Böcker and coworkers have also described an extension of their method to high-resolution mass spectrometry (HR-MS) which involves multiplying all masses by a constant factor and rounding to the nearest integer [6]. For example, the mass vector for CHNOPS with masses to 5 decimal places can be converted to integer form by multiplying by 10^5 to give $\mathbf{m}$ = [120000, 10078, 140031, 159949, 309738, 319721].

1.3 The use of chemical knowledge-based heuristics

Once one has a method that can produce a list of compomers, chemical knowledge-based heuristics can also be used; in contrast to the mathematical heuristics just discussed, these are rules which eliminate formulae that are not chemically relevant. Notably, Kind and Fiehn have proposed "seven golden rules" to filter out nonsensical molecular formula, by imposing standard valency rules, or the removal of molecular formulae which have excessively high (or low) hydrogen-to-carbon-atom ratios, as well as hetereoatom ratios [15]. In several very large databases, the correct molecular formula was assigned with a probability of 98%, and for "truly novel compounds" that were not present in the databases, the correct formula was found in the first three hits with a probability of 61-81% [15].

One can also filter out unlikely molecular formula by comparing a simulation of the abundances of isotopomer peaks relative to the monoisotopic which are then compared to experimentally observed ratios of peaks in order to determine the formulae which are most consistent with the observed mass fragments [27, 28].

Finally, there has been some work on utilising tandem mass spectrometry (MS^n) data to narrow down possible molecular formulae, as any daughter ions must logically be a *subformula* of the parent ion. See e.g [21] or [11] for more information.

1.4 This work

In this paper we present a simple algorithm for generating all chemical formula that could correspond to a fragment within the mass spectrum, which is much more compact than generating these formula one-by-one naively.

Our algorithm employs, and extends, the generating function method [16] for counting the number of molecular formulae with a given fragment of known total mass. Some of the earliest work based on generating function ideas was by Morikawa and coworkers [18, 19]. Our algorithm works with an arbitrary list of possible chemical elements from which the molecular formula may be composed of. Most important, the algorithm requires the production of what we call a *molecular formula tree* from which all possible molecular formula associated with a given ion obtained, as branches of this tree.

2 Theory

In this section we proceed directly to describe our algorithm, because we think an explanation by example provides the clearest exposition. Next, we justify our algorithm using the generating function method, which provides a powerful way to solve problems in combinatorics. The theoretical cost, pros and cons, and relationship of our method to other closely related methods are also discussed in this section.

2.1 An example of the algorithm

The algorithm involves constructing a molecular formula tree. The process is best illustrated by a concrete example. We take the particular case of $M = 7$ using a hypothetical alphabet

$$\begin{aligned} \mathbf{E} &= [\mathrm{H}, \mathrm{He}, \mathrm{Li}], \quad \text{with masses} \\ \mathbf{m} &= [1, 4, 7]. \end{aligned} \tag{5}$$

We first make the multiplicity-weighted mass vectors for each element:

$$\begin{aligned} \mathbf{m}_{\mathrm{H}} &= [0, 1, 2, 3, 4, 5, 6, 7], \\ \mathbf{m}_{\mathrm{He}} &= [0, 4], \\ \mathbf{m}_{\mathrm{Li}} &= [0, 7]. \end{aligned} \tag{6}$$

These vectors are just arithmetic sequences based on the mass of the element concerned, and importantly, they do not exceed the target value $M = 7$ of the mass spectrum ion of interest. For simplicity we refer to $\mathbf{m}_{\mathrm{H}}$ as, say, "the H vector".

We now describe the *aufbau* process for building up the solution

- To start the process we pick any vector, say the H vector, as shown in Figure 1(a).
- Next, we "convolute it" with the next vector, say the He vector. This means that we add each element in the He vector to every element in the H vector, and we record each pair as "branches" with the element in the He vector appended to those in the H vector, as shown in Figure 1(b). We do not keep any branches that exceed the value $M = 7$. We also keep track of the element multiplicites $\mathbf{n}_i$ by writing the partial chemical formula below its mass contribution. For example, we write He_1 below the mass contribution $4 \equiv n_2 m_2$.
- In the next step, we first sort the branches by the sum of the masses of the two elements, and then place branches possessing the same mass into groups. This is equivalent to merging together these branches into a tree, as shown in Figure 1(c).
- We then repeat the previous two steps for any other multiplicity-weighted mass vectors, sorting and grouping by the mass of the branches in the tree, up until the last vector. In this case, there is only the Li vector left to do, and the result of the convolution step, grouping and reordering is shown in Figure 1(d).

In the final step we may discard any values in the tree, which do not correspond to the target value of $M = 7$, but for this example, we have

shown all the possibilities for completeness, in Figure 1(e). The list of possible molecular formulae can be read off the tree by travelling along any one of the branches. It clearly does not matter = whether we go from top to bottom, or bottom to top, since it is only the sum $M = 7$ that matters. The algorithm also works no matter in which order in we choose the multiplicity-weighted mass vectors, for essentially the same reason. However, in general it may be more efficient to choose the larger mass vectors first (in the case the Li vector, then the He vector) because their sum is more likely to exceed the target M value earlier in the process. Also, at any stage in the procedure, branches may be pruned if the partial chemical formulae are nonsensical: this illustrates the use of chemical heuristics.

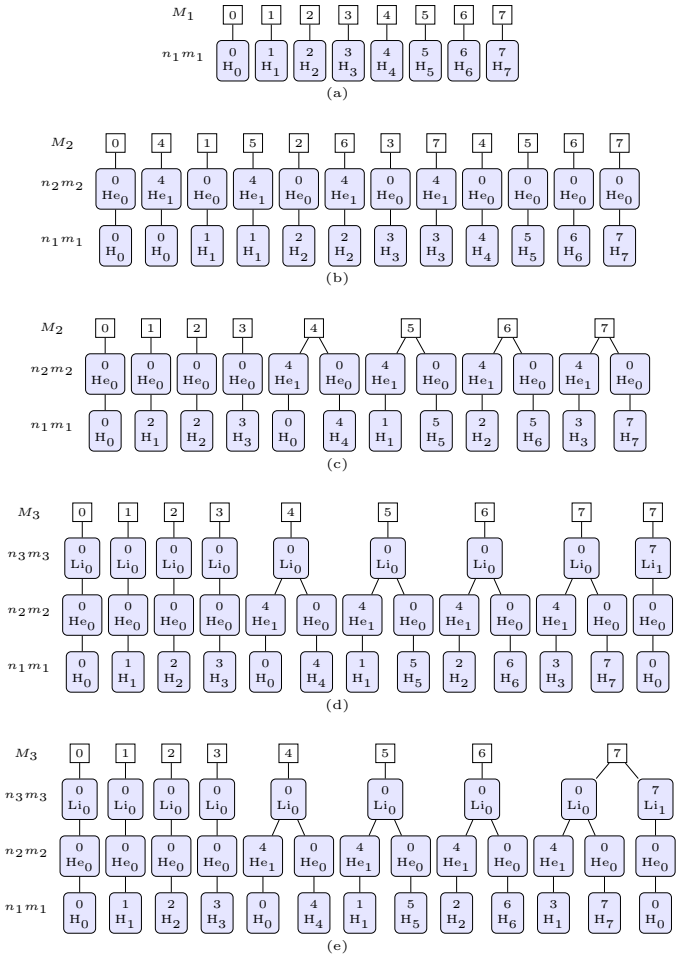

Figure 1. Simple example illustrating the iterative *aufbau* procedure for constructing the molecular formula tree for elements $\mathbf{E} = [\mathrm{H}, \mathrm{He}, \mathrm{Li}]$ with integer masses $\mathbf{m} = [1, 4, 7]$. Refer to text for full commentary. (a) Setting up the first layer of the tree for element H. (b)-(c) Adding all possible masses; the M_i are the masses of the partial molecular formulae of length i, where each element E_i occurs n_i times, $i = 1, \ldots, 3$. The final stage (d) includes all possible molecular formula of mass $0 \leq M \leq 7$. For example, the very last tree indicates the molecular formula Li_1, $\mathrm{He}_1\mathrm{H}_3$, and H_7, which are all possible solutions of mass $M = 7$ for this problem.

2.2 Generating functions

A *generating function*, $f(s)$, is a function, representable by an infinite power series, where each *coefficient* of said series corresponds to a term in a sequence [16]. If a sequence is finite, then $f(s)$ is also known as a *generating polynomial*. For example, the sequence $[1, 2, 3, 4, 5 \ldots]$ is represented by the generating function

$$f(s) = \sum_{n=0}(n+1)s^n = 1 + 2s + 3s^2 + 4s^3 + 5s^4 ... \tag{7}$$

where the power n of a term s^n indicates the position of the coefficient in the sequence.

We now show that by multiplying a particular kind of generating function, the powers of the variable s correspond naturally to a total mass, while the coefficients of the variable s correspond to the number of ways of generating a chemical formula with that mass.

Consider first the trivial case of an alphabet which contains only one element with the mass m. The number of integer partitions of M is consequently 1 if M is divisible by m, and 0 otherwise. For example, if $M = 8$ and $m = 2$, then there is only one way of decomposing M into parts equal to m, namely $8 = 2 + 2 + 2 + 2$, whereas if $M = 9$ then there is no way of decomposing M into parts equal to m. The generating function for this case is a sequence of 1's in the positions nm, where n is a non-negative integer, and its power series is

$$f(s; m) = \sum_{n=0} s^{nm} = 1 + s^m + s^{2m} + \ldots = \frac{1}{1 - s^m} \tag{8}$$

The expression on the right can be obtained by observing that the infinite series on the left is precisely its power series expansion.

Next, suppose that the alphabet contains two elements with masses m_1 and m_2, with generating functions $f_1 = f(s; m_1)$ and $f_2 = f(s; m_2)$ defined as above, for decomposing M into parts equal to m_1 or m_2 respectively. Then the number of ways to decompose a number M into a sum of m_1 or m_2 is obtained as the coefficients of s^M in the product of $f_1 f_2$.

For example, take $M = 12$, $m_1 = 2$, and $m_2 = 6$. Then there are only 3 possible ways to decompose 12 into a sum of terms involving 2 and 3, namely:

$$\begin{aligned} 12 &= 2+2+2+2+2+2, \\ 12 &= 2+2+2+6, \\ 12 &= 6+6. \end{aligned}$$

On the other hand,

$$\begin{aligned} f(s;2)\,f(s;6) &= (1+s^2+s^4+s^6+s^8+s^{10}+s^{12}+\ldots)(1+s^6+s^{12}+\ldots) \\ &= 1+s^2+s^4+s^6+s^8+s^{10}+s^{12} \\ &\quad +s^6+s^8+s^{10}+s^{12}\quad +s^{12} \\ &= 1+s^2+s^4+2s^6+2s^8+2s^{10}+3s^{12} \end{aligned}$$

In the above, to avoid unnecessary computation, terms are limited to the s^{12} term. The coefficient in front of s^{12} is indeed 3, which corresponds to the three possible products

$$(s^2)^6(s^0),\quad (s^2)^3(s^6),\quad \text{and}\quad (s^6)(s^6),$$

where the groupings correspond to the power series in $f(s;2)$ and $f(s;6)$ respectively.

It should now be obvious that the coefficient of s^M counts the number of ways s^M can be made from $s^{n_1 m_1 + n_2 m_2}$, where n_1 and n_2 are the multiplicities of the masses m_1 or m_2, because this is precisely the number of ways the individual monomials can multiply together so that the exponents sum in different ways to give s^M.

In the most general case, for a given list of possible elements with masses $\mathbf{m} = [m_1, m_2, \ldots, m_N]$, the number of possible integer partitions, and consequently the number of molecular formulae, of a compound with mass M, can simply be expressed in terms of the generating function

comprised of a product of generating simple unit sequences,

$$F(s;\mathbf{m}) = \prod_{i=1}^{N} f_i, \quad \text{with} \quad f_i = f(s;m_i) = \sum_{n_i=0} s^{n_i m_i}. \tag{9}$$

The coefficient in front of the s^M term in the resulting power series yields the number of molecular formulae.

2.3 Listing the chemical formulae

In the compomer problem, we are not interested in counting the *number* of molecular formulae that has a particular formula mass, but in explicitly *listing these formulae*. To list all molecular formulae instead of only counting them, all that needs to be done is to multiply together polynomials of *different* variables s_i, rather than the same variable s, where each variable s_i is associated with a different element E_i in the alphabet. Thus, for an alphabet of length N, the generating function can be written as:

$$F(\mathbf{s};\mathbf{m}) = \prod_{i=1}^{N} f_i, \quad \text{where now} \quad f_i = f(s_i;m_i) = \sum_{n_i=0} (s_i)^{n_i m_i}. \tag{10}$$

For example, using the previous case of $M = 12$, with masses $m_1 = 2$ $m_2 = 6$ correspond to elements A and B respectively, we can expand the product $f_1 f_2 = f(s_1;2)f(s_2;6)$ and keep only those terms with overall power $M = 12$. This results in

$$(s_1^2)^6(s_2^0), \quad (s_1^2)^3(s_2^6), \quad \text{and} \quad (s_1^6)(s_2^6),$$

which directly corresponds to the formulae

$$\mathrm{A}_6, \quad \mathrm{A}_3\mathrm{B}, \quad \text{and} \quad \mathrm{B}_2.$$

The multiplicities of the elements A and B are derived by dividing the power of the variables s_1 and s_2 by their respective masses, m_1 and m_2.

2.4 The molecular formula tree

In the previous section (Section 2.3), we showed that the enumeration of all possible molecular formulae can be achieved by the process of multiplying multivariable polynomials and choosing those monomials that have an overall power of M. A straightforward procedure for doing this involves successively multiplying polynomials f_1 and f_2, to get $F_2 = f_1 f_2$, then to get $F_3 = F_2 f_3$, and so on; or in general

$$F_j = F_{j-1} f_j, \quad \text{starting with} \quad F_1 = f_1 \quad \text{and finish with} \quad F_N = F, \tag{11}$$

with F defined in equation (10). Although there are efficient techniques for multiplying polynomials, in this case, we need only keep track of the multiplicities n_i of the variables s_i, because these multiplicities define the overall power

$$M_j = \sum_{i=1}^{j} n_i m_i \tag{12}$$

to which the monomial term is raised in each step j. Since the overall monomial power is also a partial mass sum, and in the final step, $j = N$, we must choose those monomials with the overall power $M_N = M$, the desired total mass.

Substituting equations (12) and (11) in equation (10) we obtain

$$M_j = M_{j-1} + n_j m_j. \tag{13}$$

Thus, as we have seen before, polynomial multiplication amounts only to addition of monomial powers. This suggests that at each step of the iterative process defined by equation (11), we only keep track of all the multiplicities $[n_i]_{i=1}^{j}$, and that we order these values according to their sum M_j. We may reject those values of M_j which exceed the target value M for $j < N$, and in the final step $j = N$ reject those values M_N that are not exactly equal to the target value of M.

The fact that the final sum M is made of successive intermediate sums in equation (13) suggests that a "tree" data structure is appropriate to record how the multiplicities at level $j-1$ are related to those at the next

level j; for example, one may recall the famous Pascal triangle, also made iteratively, but based on a similar two-term equation.

We thus arrive at the following algorithm.

1. *Initialization.*
   In the first step, $j = 1$, record an initial sequence of multiplicities $\mathbf{n}_1 = [1, 2, \ldots, k_1]$ and record the corresponding partial masses $\mathbf{M}_1 = [m_1, 2m_1, \ldots, k_1 m_1]$, where $k_1 m_1$ is the largest value that does not exceed M. The indices $\mathbf{n}_1$ form a sorted group of indices, the base of the tree structure.

2. *Convolution: building the tree by polynomial multiplication.*
   For the next steps $1 < j < N$, repeat the following:

   (a) Generate multiplicities $\mathbf{n}_j = [1, 2, \ldots, k_j]$ and record the corresponding mass values $\mathbf{m} = [m_j, 2m_j, \ldots, k_j m_j]$, where $k_j m_j$ does not exceed M. Add the elements of $\mathbf{m}$ to those in $\mathbf{M}_{j-1}$ in all possible ways to form $\mathbf{M}_j$; at the same time *append* each multiplicity in $\mathbf{n}_j$ to the previously formed and sorted *groups of multiplicities* associated with $\mathbf{M}_{j-1}$, to form a new layer of the tree.

   (b) Sort the branches of indices with the same partial mass sum $\mathbf{M}_j$ into groups, rejecting those where any is greater than M.

3. *Terminating the tree: choosing the final M value.*
   In the final step, $j = N$, perform step 2(a), but in step 2(b), keep only those values $\mathbf{M}_N$ which are exactly equal to M.

Note that the sorting part of step 2(b) may be combined into step 2(a), but they are kept separate here for clarity. Refer to Section 2.1 for more information.

2.5 Truncating element vectors

Suppose, within a particular class of chemical compounds, that the likely number of atoms of element E (with a mass of m) can be specified by a

range $[n_E^{\min}, n_E^{\max}]$. Then the generating function of E, $f(s_i)$, and thus the E vector, can be appropriately truncated:

$$f(s_i) = s_i^{n_E^{\min} m} + \ldots + s_i^{n_E^{\max} m} \tag{14}$$

The addition of these rules in order to truncate the element vectors can greatly improve both the run time of the program and reduce the memory required, since the size of the molecular formula tree and the time taken to convolute the vectors are both dependent on the size of the element vectors. This is especially notable in the case of hydrogen, as the chemically relevant upper bound to the number of hydrogen atoms in a molecule of mass M is much lower than the purely mathematical upper bound of M hydrogen atoms.

2.6 Analysis of algorithm

As already stated, the number of solutions to the compomer problem is asymptotically proportional to $\gamma(M)$ [4] and so the number of branches in the tree is also equal to this number. Consequently, the time needed to list all solutions is proportional to this number. However, as we show below, this is (asymptotically) much longer than the time to compute the molecular formula tree. Besides, the use of chemically-based heuristics may prune a significant number of branches from the molecular formula tree. Due to this pruning process it is more constructive to analyse the asymptotic time complexity of only computing the molecular formula tree.

In terms of time taken, the "bottleneck" in the algorithm is the sorting process, after polynomial multiplication, which amounts to addition of partial mass sequences. Thus, the time taken is related to the number of elements within the list to be sorted. The list to be sorted has a lower and an upper bound, and we consider only the upper bound.

For the upper bound, when the product list that is accumulating partial solutions contains approximately M elements, the length L of the list for variable s_i is the larger of M^2/m_i or Ml_i, where l_i represents the list length after pruning. After sorting a list, only another single pass through the sorted list is required to collect individual elements into groups. Thus, we

need only examine the time complexity of the sorting process.

For an arbitrary list, sorting can be done in a time that scales as $O(L \log L)$, where as explained

$$L = \max\left[(M^2/m_i), l_i\right] \tag{15}$$

is the list length (see e.g [9]). Since the process of polynomial multiplication and grouping needs to be to be done $N-1$ times for each variable s_i, the time complexity is therefore

$$t \sim \mathrm{O}\left(NL \log L\right).$$

Thus, the run time for the computation of the molecular formula tree is *linear* with respect to N, and roughly quadratic with respect to M, both which are significantly faster than M^{N-1}, ignoring prefactors.

Theoretically, the time complexity can be reduced to

$$t \sim \mathrm{O}\left(NL\right).$$

since the list to be sorted is a list of subtrees, with branches of positive integral mass $N \leq M$. Consequently, the sorting and grouping process can be done in one step using a bucket sort, without the subsequent merging of buckets. This was not done in the Python implementation of the script, because the factor $\log L$ is negligible in practice, however such an optimisation can be easily done.

The reason for the fast computation of the molecular formula tree is due to the simultaneous enumeration of all chemical formula in the form of a tree, rather than explicitly listing solutions one at a time. However, this comes at the cost of having to store all the solutions, in the more compressed form of a tree rather than an array of N-dimensional arrays.

A very similar method of enumerating an exponential number of terms (a power set of protein configurations) in sub-exponential time, using what is referred to as a *convolution tree*, was proposed by Serang and coworkers [26] in their analysis of protein tandem mass spectrometry data—albeit for a different purpose of evaluating the joint probabilities of all possible

combinations of proteins being present.

3 Materials and methods

3.1 Implementation of algorithm

The described algorithm was implemented as a "proof-of-concept" `python3` program, as shown in Listing 1. No chemical-knowledge based heuristics were used to limit the entries in any way. Only the built in `itertools` and `timeit` libraries was used.

```
import itertools

def is_leaf(t):
    return len(t) == 1

# Evaluate sum of tree by traversing doum ONE branch,
# since all branches have the same mass
def sum_tree(t):
    return sum(t) if is_leaf(t) else t[0] + sum_tree(t[1])

# Transforms the formula tree into a list by depth
# first traversal
def traverse_tree(tree):
    res = []
    def s(t,r):
        if is_leaf(t):
            #append formula to list, removes dummy root
            res.append(r[1:]+t)
        else:
            #recursive function call over each branch
            [s(branch,r+[t[0]]) for branch in t[1:]]
    s(tree,[])
    return res
```

```
# Multiplies two generating functions, builds a level
# in the formula tree
def multiply_gf(f1,f2,M):
    # Multiplication: throws away terms larger than M
    r = [j + i for i in f1 for j in f2 if sum_tree(j + i) <=
    ↪  M]
    r.sort(key=sum_tree)
    # Collect terms into groupings based on M of formulae
    return [list(g) for k,g in
    ↪  itertools.groupby(r,key=sum_tree)]

# Produce generating functions based on the alphabet
def produce_gfs(alphabet, M):
    return [ [ [j] for j in range(0,M+1,i)] for i in
    ↪  alphabet]

# Computes formula tree
def formula_tree(alphabet,M):
    l = produce_gfs(alphabet,M) # generates list of GFs
    prod = [ [i] for i in l[-1]]    # "groups" the base case
    # Multiplies the GFs in a loop and
    # return all formulae with m = M
    for i in range(len(l)-2,-1,-1):
        prod = multiply_gf(prod,l[i],M)
    return prod[-1]

# Flattens list of formula trees with m = M,
# resulting in a list of monomials corresponding to each
↪  formula
def generate_lst(t):
    # Merge list of trees into one
    # formula tree with a dummy root,
    # then traverse the tree  ...
```

```
    return traverse_tree([0] + t)

# Given a monomial "mon", produce a compomer,
# then produce a molecular formula from the compomer
def produce_MF(mon,alphabet):
    # Can be any list of symbols corresponding to elements
    lst = ["C","H","N","O","P","S"]
    # Obtain c_i of compomer by n_i/m_i
    compomer = [int(mon[i]/alphabet[i]) for i in
    ↪  range(len(mon))]
    # Swap carbons and hydrogens
    compomer[0],compomer[1] = compomer[1],compomer[0]
    # Remove element if coeff is 0 and remove the "1" if
    ↪  coeff is 1
    res = [lst[i]+str(compomer[i]) for i in
    ↪  range(len(compomer)) if compomer[i] != 0]
    return  "".join([i if len(i) != 2 or i[1] != "1" else
    ↪  i[0] for i in res])

# Generates a list of molecular formulae given an alphabet
# (list of integral m_i's, ascending order) and a mass M
def enumerate_MF(alphabet, M):
    return [produce_MF(i,alphabet) for i in
    ↪  generate_lst(formula_tree(alphabet,M))]

#this will print out all CHNOPS MF with integer mass 100
print(enumerate_MF([1,12,14,16,31,32],100))
```

Listing 1. Listing of the `python3` program used to implement the molecular formula tree. Calling the routine `enumerate_MF` with the given `alphabet` and integer mass `M` produces a list of molecular formulae with that mass

3.2 Details of the computer used

A standard Lenovo 81GC laptop with 8Gb of physical memory installed, an Intel(R) Core(TM) i7-8550U CPU, 4 cores, 8 logical processors, and a clock speed of 1.8GHz was used for all calculations.

4 Results and discussion

In this section we demonstrate our algorithm by performing several numerical experiments, and tests, intended to give an idea of the scope and ease of application of our method. The tests and their results are discussed together.

4.1 Time to construct the formula tree as a function of alphabet length

The time needed to compute the molecular formula tree as a function of the alphabet length was established for $N \leq 10$ for molecular formula trees with $M = 500$ and for the integer masses associated with the most common isotopes of the elements $\mathbf{E} = \{\mathrm{H}, \mathrm{He}, \mathrm{Li}, \ldots, \mathrm{Ne}\}$, or subsets comprising, successively, the first elements of this set. We used the lowest mass elements to ensure that we would get the longest generating functions and the largest trees. The results are shown in Figure 2.

We observe that the time scales linearly with respect to alphabet length N which agrees with the theoretical analysis.

Figure 3 illustrates a more detailed analysis, and shows a log-log plot of time required for $M \leq 500$ for $N \leq 10$. We observe that as N becomes larger all plots possess a similar gradient; the slope being approximately 2.06, indicating a scaling of $t \sim M^{2.06}$ (the point $N = 1$ and $N = 2$ was ignored in obtaining the least-squares fit for the slope). This again supports that the algorithm scales linearly with respect to N, and slightly slower than quadratically with respect to M. In particular, the essentially identical gradient from $N = 3$ to $N = 10$ supports the claim that run time scaling for molecular formula tree computations do not vary substantially with regard to alphabet length N.

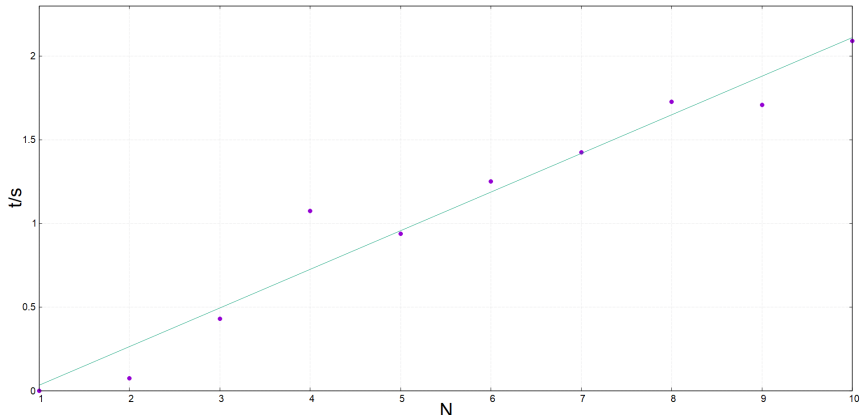

Figure 2. Time required to make the molecular formula tree as a linear function ($R^2 = 0.9557$) of alphabet length N for total mass $M = 500$, for the masses of the first ten elements $\mathbf{E} = \{\mathrm{H}, \mathrm{He}, \mathrm{Li}, \ldots, \mathrm{N}, \mathrm{O}, \mathrm{F}, \mathrm{Ne}\}$.

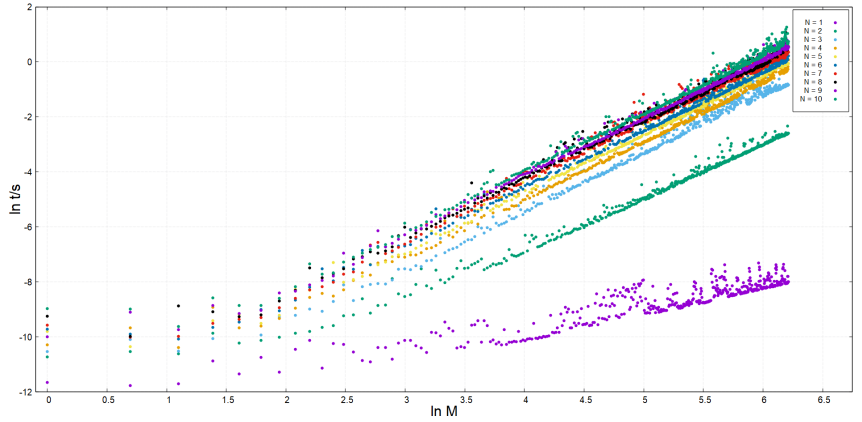

Figure 3. Log-log plot of time taken to compute the formula tree using our `python` program for masses $M \leq 500$ and for alphabets of length $N \leq 10$; different colours used for different N. Visually, the gradient does not vary substantially for $N \geq 3$; and a numerical fit to for these N has a slope 2.06 ± 0.02 with R^2 values in the range $[0.981, 0.992]$.

4.2 Counting molecular formulae

Using this method, the maximum number of molecular formulae that only contain some or all of the CHNOPS elements formulae with integer mass

$M \leq 2000$ can be counted using code for polynomial multiplication: there are 39,026,736,558 different formulae in total. This is larger than the 7,995,776,805 counted by brute force enumeration with the HiRes software by Kind and coworkers [15], since our method does not employ any heuristics to eliminate nonsensical chemical formulae.

4.3 Finding the molecular formula of Cangrelor

To assess the performance of this method in an application to find possible molecular formulae of "real" data, conceivably obtainable from a hypothetical HRMS experiment, we chose the drug molecule Cangrelor, $C_{17}H_{25}Cl_2F_3N_5O_2P_3S_2$, with a mass of $M = 775$ and an exact monoisotopic mass of $M = 774.94831$. This molecule was chosen due to its relatively high mass, the fact that it was composed of eight different elements, and because Kind and Fiehn used it to generate 4,465 possible molecular formulae within 1 ppm of the exact mass, using the MS software HR2, a modified HiRes [15].

Using the unmodified version of our program in Listing 1, it took 3.7 seconds to generate the formula tree and 175 seconds to expand the formula tree, resulting in a list of 37,001,983 formulae, corresponding to all molecular formulae containing some or all of the 8 elements and possessing a mass of 775.

4.4 Using chemical knowledge-based heuristics

It is very straightforward to implement chemical heuristics using our method, both by truncating element vectors as detailed in Section 2.5 and in the process of tree traversal to recover molecular formulae. By implementing just two chemical rules on the multiplicities generated during formation of the molecular formula tree, namely:

$$n_{\mathrm{H,max}} < 2n_{\mathrm{C,max}} + 2, \quad \text{where} \quad n_{\mathrm{C,max}} = \lfloor M/12 \rfloor, \quad \text{and} \tag{16}$$

$$n_{\mathrm{C,min}} = \lfloor M/(4 \cdot 12) \rfloor \tag{17}$$

Table 1. A partial list of molecular formulae generated by the algorithm, ranked in order of mass deviation, $|M^{\text{precise}} - M|$, from the formula mass of $M = 774.9483131704699$. The correct formula is starred, the 1st in the list, and has a non-zero mass deviation due to imprecision in the floating point computer arithmetic. The masses of all formulae generated are monoisotopic masses, and the exact isotopic masses are taken from [1].

Molecular formula	Mass deviation
$C_{17}H_{25}N_5O_{12}F_3P_3S_2Cl_2$*	1.1×10^{-13}
$C_{17}H_{16}N_{15}O_9S_3Cl_3$	3.1×10^{-8}
$C_{22}H_{19}N_4O_7F_7PS_5$	3.5×10^{-8}
$C_{29}HN_3O_3F_{16}S$	2.5×10^{-7}
$C_{31}H_{26}N_5P_2Cl_7$	2.9×10^{-7}
$C_{19}H_{13}N_9O_6F_8P_3SCl$	3.4×10^{-7}
$C_{19}H_4N_{19}O_3F_5S_2Cl_2$	3.7×10^{-7}
$C_{20}H_{31}O_{13}F_2PS_6Cl$	3.7×10^{-7}
$C_{23}H_9N_{12}O_{10}F_2P_4$	6.7×10^{-7}
$C_{21}HN_{13}F_{13}P_3$	6.7×10^{-7}
$C_{16}H_{38}N_6OF_2PS_3Cl_8$	7.9×10^{-7}
$C_{21}H_{32}NO_3F_6Cl_9$	8.4×10^{-7}
$C_{18}H_{23}NO_3F_{13}S_6Cl$	9.7×10^{-7}

i.e the percentage composition of carbon by mass should be more than 25%. By truncating the corresponding C and H vectors, the size of the formula tree was reduced substantially; the time was 1.1 seconds to generate the formula tree and 20.3 seconds to traverse all of its branches, yielding a slightly smaller set of 4,899,086 molecular formulae.

This list can be pruned further by modifying the traversal of the formula tree in order to extract the list of formulae. For example, in the case of $M = 100$, $n_{\text{C,max}} = 8$; so according to the rules above, $n_{\text{H,max}} = 18$. However, even with the rules above, a formulae such as $C_3H_{16}O_3$, which contains too many hydrogens to be chemically relevant, would still be allowed. This shows that if many heteroatoms are present, then there can be large discrepancies between n_{C} and $n_{\text{C,max}}$, which cannot be dealt with by truncating element vectors

However, to save the excess time required to extract these formula from

the formula tree, tree traversal can be stopped immediately after establishing that the H-to-C ratio is too high, discarding the chemically unreasonable formulae all at once rather than generating each one individually by tree traversal and then discarding. The H-to-C ratio and heteroatom checks documented by Kind and coworkers can be incorporated like so, in order to increase computational efficiency [15].

Other heuristics, perhaps applicable to specific classes of chemical compounds or concerning discrepancies between integer and exact masses, can also be applied this way, in order to circumvent traversing much of the formula tree. To showcase this, the rule that the H-to-C ratio should be less than 3 was incorporated into the program, and this further reduced run time to 11.5 seconds and yielded a list of 3,259,436 formulae.

4.5 The effect of high-precision molecular masses

The list that resulted from the previous steps can be further filtered down using high resolution mass spectral data. To do this, we used high-precision isotopic masses, available from the CRC handbook [1], to compute the "formula mass" of each of the molecular formulae generated, filtering out branches where the mass difference was greater than 1 ppm between the formula mass of Cangrelor (the "measured mass") and the formula mass of other formulae. The very high resolution of 1 ppm represents a best-case scenario; in reality a value of 5 ppm is more realistic, and would yield many more solutions.

The result was a list of 9731 formulae, one of which was the the correct formula of $C_{17}H_{25}Cl_2F_3N_5O_2P_3S_2$. The closest matching results are also shown in Table 1. This shows that, given a high mass and even a relatively small number of elements, there exist formulae with virtually identical mass; there exist molecular formulae with a mass deviation less than 10^{-7} from the formula mass of the correct formula.

4.6 Comparison with other methods

The difference between our algorithm and that of Böcker and Lipták [5] is that the latter authors proposed a "top-down" query approach, whereas we

propose a "bottom-up" explicit construction approach. By a clever use of integer remainders and divisors, encoded in an "extended residue table", Böcker and Lipták are able to explore the search space of all possible formulae, terminating each branch of the search as soon as it is feasible. Due to only requiring the construction of a relatively small table, their method is more memory-efficient than our method.

However, a key disadvantage with their method when applied to HR-MS is that the mass measured will inevitably possesses an associated uncertainty ϵ which is *higher* than the uncertainty of the constituent atomic masses, so in practice *all* masses in an interval $[M - \epsilon, M + \epsilon]$ should be decomposed, requiring the compomer problem to be solved many times. This "interval problem" was addressed by both Agarwal and coworkers and Dührkop and coworkers, again utilising dynamic programming-based methodologies [2, 12]. On the other hand, our algorithm is conceptually simpler and produces a compact molecular formula tree containing all formulae with integer mass M, from which a list of formulae matching the HR-MS data can be directly extracted (refer to Section 4.5), thus bypassing the "interval problem" entirely.

Another advantage of our algorithm is that restrictions on the maximum/minimum number of elements make a very large difference in the time and memory required. As we have shown in Section (4.3), implementing simple restrictions on the search space, which can be done by simply truncating the element vectors to a specified size without changing the rest of the program, can lead to a drastic speedup in the run time of the program. Should a domain expert reasonably suspect a narrower range of "acceptable" elemental compositions, a further speedup is expected.

Like our algorithm, Morikawa and coworkers describe a generating function based method to count [19] molecular formula but with the added restriction that these formula correspond to molecular structures which obey Senior's standard valency rules [25]. An earlier method of Morikawa used a Diophantine equation approach [18]. Neither of these methods are completely general, and the concept of a molecular formulae tree did not emerge.

Our algorithm can be regarded as a generalisation of that of Dromey

and Foyster [10]. Their algorithm involves dividing the alphabet [C, H, N, O, P, S] into subalphabets [C, H] and [N, O, P, S], generating all possible decompositions with $m \leq M$ for both subproblems, and then taking the cartesian product of the two arrays of partial solutions to arrive at the full solutions. Our algorithm instead builds up all solutions as the molecular formula tree, by recursively taking the cartesian product of a tree of partial solutions with the trivial partial solution; mass decompositions with a size 1 subalphabet.

4.7 The effect of using polyisotopic patterns

It was already noted by Schum and coworkers that run time considerations render the possibility of simultaneous enumeration of polyisotopic and monoisotopic molecular formulae impractical [23]. Consequently, polyisotopic ion filtering needs to be conducted prior to molecular formula generation. The linear scaling of run time with respect to N suggests that our method could potentially bypass this step and simultaneously enumerate all polyisotopic and monoisotopic molecular formulae, especially since a limit for the maximum "reasonable" number of certain isotopes (e.g ^{13}C) within a fragment can be set based on its mass, meaning truncation of the element vectors can be done. Likewise, for a given chemical compound, all of its isotopomers up to a given mass (e.g $M + 10$) can be quickly and efficiently enumerated by this algorithm, by simply incorporating the isotopic masses into the alphabet. We did not test this possibility here.

4.8 Treating high resolution masses and integers

In the case of high resolution MS with masses recorded up to n decimal places, the integer-based method described here could be applied by multiplying all the masses by a factor of 10^n, say, or some other factor; and then rounding to the nearest integer, as the use of integer arithmetic on a computer is faster than using real arithmetic. However, as we already showed theoretically and in practice, run-time of the algorithm scales roughly as M^2, so that enumeration of all formulae with unscaled integer masses, followed by subsequent filtering, is more suitable in practice. In this case,

rather than needing to fully expand the molecular formula tree and eliminate formulae one by one, it would be possible to remove portions of the molecular formula tree, which are guaranteed to give an answer too different from the required mass, according to some preset tolerance, for example 5 ppm [3]. Removing portions of the molecular formula tree as soon as practicable mitigates both the computational cost and memory requirements for cases where there would otherwise be large numbers of molecular formulae. Furthermore, this enumeration-followed-by-filtering approach eliminates the need to perform mass decompositions on all integers within a range, as is the case with Böcker's method [5]. Again, we have not tested this possibility here.

5 Conclusion

An algorithm that enumerates all possible molecular formula given a specific mass M was developed using the generating function method of counting integer partitions. The algorithm works by constructing a molecular formula tree, essentially a tree associated with the monomials of the generating function, which scales linearly with the number of elements in the molecular formula, and roughly quadratically with respect to the total mass M.

Although the algorithm costs a substantial amount of computational memory, it was in some cases faster than algorithms reported for this purpose, indicating that this may not be a limiting problem using computers currently available. Our method can also allow for rapid pre-computation of molecular formulae in a specified range of masses.

Due to its simplicity and ability to be applied to arbitrary alphabets, this algorithm can be readily integrated into any molecular formula assignment software package, in any application where the speed of molecular formula generation becomes a limiting factor. Indeed, as the molecular formula of a compound, in addition to sub-formulae of its fragments, play a crucial role in providing insight into molecular structure, we plan to integrate this method into subsequent work involving *de-novo* structural identification of compounds using mass spectrometry.

***Acknowledgment*:** SL acknowledges funding of a Ph.D. scholarship from the Forrest Research Foundation.

References

[1] Robert C. West (Ed.), *CRC Handbook of Chemistry and Physics*, CRC Press, Cleveland, 1978.

[2] D. Agarwal, *Topics in mass spectrometry based structure determination.* Theses, Université Nice Sophia Antipolis, 2015.

[3] M. P. Balogh, Debating resolution and mass accuracy, *LC GC Asia Pacific* **7** (2004) 16–16.

[4] M. Beck, I. M. Gessel, T. Komatsu, The polynomial part of a restricted partition function related to the frobenius problem, *El. J. Comb.* **8** (2001) 1-5.

[5] S. Böcker, Z. Lipták, A fast and simple algorithm for the money changing problem. *Algorithmica* **48** (2007) 413–432.

[6] S. Böcker, M. C. Letzel, Z. Lipták, A. Pervukhin, Sirius: decomposing isotope patterns for metabolite identification, *Bioinformatics* **25** (2009) 218–224.

[7] S. Böcker, Z. Lipták, M. Martin, A. Pervukhin, H. Sudek, Decomp—from interpreting mass spectrometry peaks to solving the money changing problem, *Bioinformatics* **24** (2008) 591–593.

[8] B. Bohman, G. R. Flematti, R. A. Barrow, Identification of hydroxymethylpyrazines using mass spectrometry, *J. Mass Spectrom.* **50** (2015) 987–993.

[9] T. H Cormen, *Introduction to Algorithms*, MIT Press, Cambridge, 2009.

[10] R. G. Dromey, G. T. Foyster, Calculation of elemental compositions from high resolution mass spectral data, *Anal. Chem* **52** (1980) 394–398.

[11] K. Dührkop, F. Hufsky, S. Böcker, Molecular formula identification using isotope pattern analysis and calculation of fragmentation trees, *Mass Spectrom. (Tokyo)* **3** (2014) 37–37.

[12] K. Dührkop, M. Ludwig, M. Meusel, S. Böcker, Faster mass decomposition, arXiv:1307.7805, 2013.

[13] A. Fürst, J. T. Clerc, E. Pretsch, A computer program for the computation of the molecular formula, *Chemom. Intell. Lab. Sys.* **5** (1989) 329–334.

[14] N. W. Green, E. M. Perdue, Fast graphically inspired algorithm for assignment of molecular formulae in ultrahigh resolution mass spectrometry, *Anal. Chem.* **87** (2015) 5086–5094.

[15] T. Kind, O. Fiehn, Seven golden rules for heuristic filtering of molecular formulas obtained by accurate mass spectrometry, *BMC Bioinform.* **8** (2007) 105–105.

[16] S. K. Lando, *Lectures on Generating Functions*, AMS, Providence, 2003.

[17] J. Meija, Mathematical tools in analytical mass spectrometry, *Anal. Bioanal. Chem.* **385** (2006) 486–499.

[18] T. Morikawa, Algebraic enumeration and generation of molecular formulas for a given molecular weight, *J. Chem. Edu.* **63** (1986) 1053–1055.

[19] T. Morikawa, Y. Tada, Algebraic enumeration of molecular formulae for given molecular weight, *Int. J. Math. Edu. Sci. Techn.* **24** (1993) 467–472.

[20] E. M. Perdue, N. W. Green, Isobaric molecular formulae of c, h, and o: A view from the negative quadrants of Van Krevelen space, *Anal. Chem.* **87** (2015) 5079–5085.

[21] M. Rojas-Cherto, P. T. Kasper, E. L. Willighagen, R. Vreeken, T. Hankemeijer, T. H. Reijmers, Elemental composition determination based on MS(n), *Bioinformatics* **27** (2011) 2376–2383.

[22] K. Scheubert, F. Hufsky, S. Böcker. Computational mass spectrometry for small molecules, *J. Cheminf.* **5** (2013) #12.

[23] S. K. Schum, L. E. Brown, L. R. Mazzoleni, Mfassignr: Molecular formula assignment software for ultrahigh resolution mass spectrometry analysis of environmental complex mixtures, *Env. Res.* **191** (2020) #110114.

[24] S. K. Schum, B. Zhang, K. Dzepina, P. Fialho, C. Mazzoleni, L. R. Mazzoleni, Molecular and physical characteristics of aerosol at a remote free troposphere site: Implications foratmospheric aging, *Atmospheric Chem. Phys.* **18** (2018) 14017–14036.

[25] J. K. Senior, Partitions and their representative graphs, *Am. J. Math.* **73** (1951) 663–689.

[26] O. Serang, The probabilistic convolution tree: Efficient exact Bayesian inference for faster lc-ms/ms protein inference, *PloS One* **9** (2014) #e91507.

[27] N. Stoll, E. Schmidt, K. Thurow, Isotope pattern evaluation for the reduction of elemental compositions assigned to high-resolution mass spectral data from electrospray ionization fourier transform ion cyclotron resonance mass spectrometry, *J. Am. Soc. Mass Spectrom.* **17** (2006) 1692–1699.

[28] D. Valkenborg, I. Mertens, F. Lemiére, E. Witters, T. Burzykowski, The isotopic distribution conundrum, *Mass Spectrom. Rev.* **31** (2012) 96–109.