J Math Chem (2010) 48:381–394
DOI 10.1007/s10910-010-9679-1

ORIGINAL PAPER

A fast algorithm and software for analysis of FT-ICR data

David R. Gibson · Hari Pulapaka

Received: 21 September 2009 / Accepted: 2 April 2010 / Published online: 18 April 2010

Abstract In this paper, we present and analyze a fast algorithm that determines the possible molecular formulae corresponding to the resolved peaks in the spectral data obtained from a mass analyzer like an FT-ICR-MS. In contrast to our approach, most known algorithms and software that attempt to solve this important problem are brute-force in nature and consequently, highly prone to combinatorial explosion when dealing with the volume of real data. We also present an object-oriented implementation of our algorithm in a general-purpose, user-friendly, interactive, and easily extensible software tool PG COMPOUND MATCH FINDER. A run-time performance analysis of our software shows that even when dealing with several billion theoretical possibilities matched against tens of thousands of resolved peaks, a complete analysis using today's standard desktop machines can take only a few minutes.

Keywords Spectrometry · Analysis · FT-ICR · Algorithm · Software

1 Introduction

Fourier Transform Ion Cyclotron Resonance Mass Spectrometry (FT-ICR-MS or FT-ICR or just FT-MS) is widely considered the most versatile and complex method of mass analysis and detection. First developed in the early 1970s at the University of British Columbia by researchers Alan Marshall and Melvin Comisarow [1], today it

D. R. Gibson
Department of Mathematics and Computer Science, Valdosta State University, Valdosta, GA 31698, USA
e-mail: dgibson@valdosta.edu

H. Pulapaka (✉)
Department of Mathematics and Computer Science, Stetson University, DeLand, FL 32723, USA
e-mail: hpulapak@stetson.edu

is one of the most sensitive methods of ion detection and enjoys a myriad of important applications in areas like Chemistry, Biology, Biochemistry, Biomedicine, Forensics, and Environmental Science. FT-ICR depends heavily upon high speed computing and also the mathematical algorithms that determine the spectroscopic parameters from the numerical computer data.

In this paper, we build on the work done in [2] by providing a complete theoretical and computational solution to the problem of analyzing data from a mass spectrometer. While our initial work on this project was motivated by the spectral analysis problem of comparing marine sediment to oil samples described in [3], the full scope of our algorithm and accompanying software tool (PG COMPOUND MATCH FINDER) may be understood by considering the general problem of trying to efficiently identify the molecular structure of compounds corresponding to the significant peaks obtained from an instrument such as a mass spectrometer. In the case of FT-ICR-MS, we may be dealing with tens of thousands of data points (resolved peaks). When faced with this computationally taxing problem, a researcher must find effective ways to *preprocess* the large (often numbering several billions) list of theoretically possible compounds that may be potential matches for the real FT-ICR data. In the past, several techniques have been proposed to help solve the problem of reducing the number of theoretical possibilities [4–6]. For example, recently, as an extension of the work done on the utilization of isotopic abundance patterns of mass spectra to reduce the number of formula candidates [7], the authors in [8] propose a set of rules that help in paring down the list of theoretical possibilities. Our approach in [2] was more efficient because by studying the geometry of the algebraic constraints, we were able to develop an algorithm that constructed only the valid possibilities. Furthermore, our methods are easily extensible to incorporate many families of quantitative rules in the search for theoretical compounds. In this paper, we extend the procedure developed in [2] to include highly efficient automation of the full problem of identifying the molecular formulae for the resolved peaks contained in an FT-ICR Mass Spectra.

We hope that PG COMPOUND MATCH FINDER will prove to be an effective tool for researchers who use Mass Spectrometry and with feedback form the research community, we hope to continually upgrade our software to be more efficient and useful.

2 Full problem statement

We begin by defining the variables necessary to describe the problem and our solution. Let $E = \{x_1, x_2, \ldots, x_N\}$ denote the set of N elements chosen by the user. For each i, $1 \leq i \leq N$, let $L(x_i)$ and $U(x_i)$ denote the user-specified lower and upper bounds, respectively, for the number of atoms of element x_i and let $n(x_i)$ denote some integer value in this range. A user may provide a range of charges (k) associated with each choice of N elements. Next, a user of the computer program can also specify that the ratios of the number of atoms of certain pairs of elements be fixed within a specified tolerance. For i and j where $1 \leq i \neq j \leq N$, let $r(x_i, x_j)$ denote the ratio of $n(x_j)$

to $n(x_i)$ with tolerance $t(x_i, x_j)$. That is to say, the user specifies that the following must be true:

$$r(x_i, x_j) - t(x_i, x_j) \leq \frac{n(x_j)}{n(x_i)} \leq r(x_i, x_j) + t(x_i, x_j)$$

Given N, $L(x_i)$, $U(x_i)$, $r(x_i, x_j)$, $t(x_i, x_j)$, let V denote the set of combinations (theoretical compounds) of N elements chosen from E which satisfy the following constraints:

1. For each i, $1 \leq i \leq N$,we must have

$$0 \leq L(x_i) \leq n(x_i) \leq U(x_i).$$

2. **(Ratio Rule)** For i and j where $1 \leq i \neq j \leq N$, we must have

$$r(x_i, x_j) - t(x_i, x_j) \leq \frac{n(x_j)}{n(x_i)} \leq r(x_i, x_j) + t(x_i, x_j).$$

3. $(C - H)$ **Rule** Denoting the elements carbon and hydrogen by C and H, respectively, if $n(C) \neq 0$, then we must have

$$n(H) \leq 2n(C) + 2.$$

Let $S(r)$ denote the set of values (peaks) obtained from an FT-ICR-MS and let t denote the tolerance at which the user is seeking *matches* with the real data. Now we are ready to state the full problem.

Problem Statement: For each value of the charge k and each real data value r in the set $S(r)$, determine all the combinations from the set V whose masses/charge are within tolerance t of the value r.

In other words, find the set of theoretical compounds, that adhere to any specified ratio rules, whose mass/charge falls within tolerance of any ICR values, given the user's choice of elements, bounds, ratios, tolerances, charges, and ICR values.

Remark Constraint 3 above may be easily generalized to include any algebraic inequality relating $n(H)$ and $n(C)$, or any other pair of elements. Also, any mathematically describable chemical constraints such as say those mandated by molecular graphs [9,10] may be included as constraints. Based on user feedback, future versions of our software may include such additional rules as user-activated options.

3 Analysis of mass spectrometry data

The speedy analysis of raw spectral data is one of the principal bottlenecks in the effective use of increasingly sophisticated spectroscopic techniques. This is because the high resolution capabilities of techniques like FT-ICR using very powerful mass spectrometers results in large data sets with tens of thousands of values. One then has to determine the molecular composition corresponding to each value by comparing it to each of the theoretical possibilities. Certainly, one way to analyze mass spectra is through completely brute force techniques. With thousands of data values and billions (even trillions) of theoretical compounds, it is easy to see that such techniques have very limited scope because of enormous runtime even for very small problems (see Sect. 6.2). Unfortunately, from what we can tell, there are few (if any) published techniques that make creative use of mathematics and computer science to make this task efficient. In fact, many published software tools appears to be brute-force in nature. In this paper, we provide a fast algorithm and develop a user-friendly software tool to accomplish this complex task. We use mathematical and computational optimization techniques in our development of an efficient method to preprocess the spectral data and then determine matches to desired quantitative constraints and tolerances.

We can think of this task of complete analysis of ICR data as two problems: (1) How do we reduce the theoretical possibilities (based on the user's input) to a set of only valid possibilities that satisfy all ratio rules? We refer to this as the *Preprocess* step. (2) How do we compare a set of valid compounds to the ICR data? We refer to this as the *Analyze* step.

3.1 *Preprocess* step

Here, we consider two different algorithms to preprocess the user inputs to reduce the set of theoretical compounds. First, note that it is possible to *optimize* our choice of the upper bound for each element chosen by the user. Of course, the user is allowed to input any values he/she chooses for lower and upper bounds (as long as $L(x_i) \leq U(x_i)$), but in fact, since the maximum value in the ICR data set forces a limit on the upper bound for any element (by considering a theoretical compound comprised entirely of that element), we describe the algorithm that allows us to consider only the necessary upper bound for each element x_i chosen in the analysis. Let

m_i = mass of an atom of the element x_i.
n_i = number of atoms of the element x_i.
k = charge of the compound
R = maximum ICR value contained in the data set $S(r)$.
t = tolerance specified by user.

Then, for a compound composed solely of element x_i, for each k, we must have

$$\left|\frac{n_i m_i}{k} - R\right| \leq t.$$

This is equivalent to

$$\frac{k(R-t)}{m_i} \leq n_i \leq \frac{k(R+t)}{m_i}.$$

We note that n_i is maximized when k is the maximum value provided by the user. Suppose the maximum value of k is denoted as k_m. Then, before computing the valid theoretical possibilities using the algorithm developed in [2], we re-define the upper bound for each element x_i to be equal to

$$\max\left\{L(x_i), \min\left(U(x_i), \left\lfloor \frac{k_m(R+t)}{m_i} \right\rfloor\right)\right\}$$

where $L(x_i)$ and $U(x_i)$ are the user specified lower and upper bounds for the element x_i.

In conjunction with this algorithm, we also consider the algorithm considered in [2] which specifies a technique for automatically determining complete set of valid compounds. The algorithm to tighten the upper bounds, followed by the algorithm in [2] is referred to as *PG*.

As a contrast to *PG*, we consider a brute force approach to preprocess the input data. For this approach, we compare each possible theoretical compound against each of the rules that are specified to arrive at a set of valid compounds. We refer to this algorithm as *BF*. We compare the two *Preprocess* algorithms, *PG* and *BF* in Sect. 4, in terms of complexity and in Sect. 6 in terms of experimental run-times.

3.2 *Analyze* step

We also consider two different algorithms for comparing the set of valid compounds with the ICR values to determine if there are matches. The first approach, for each valid compound, uses a series of binary searches [11] on the ICR data. Traditionally, a binary search is a very fast way to find a particular search value in a sorted structure of N values and has worst-case complexity $O(logN)$. However, it can be modified to return the position of the first value that is greater than or equal to the search value. The general idea with a binary search is that the initial list of sorted values is divided in half, resulting in a *low* and *high* side. Next, the side containing the search value is determined by a comparison with the largest value on the *low* side. Finally, this side where the search value was found is subdivided into a *low* and *high* side and the process is repeated. To adapt a binary search for the *Analyze* step, consider these definitions:

m = mass of a valid theoretical compound determined by the algorithm.
k = charge of the compound.
t = tolerance specified by user.
r = real data value from the FT-ICR analysis.

The implementation of binary search that we use, returns the index of the first value in the ICR list that is greater than or equal to our search value. In our case, for a particular valid compound and for each charge, we use binary search twice, to search for $m/k - t$ and $m/k + t$ in the (sorted) list of ICR values. Any ICR values that fall between the two positions returned from the two binary searches are *matches*. We refer to this algorithm as *BS*.

A completely brute force approach for the *Analyze* step is, for each valid compound, for each charge, for each ICR value, check to see if $|m/k - r| < t$, the definition for a *match*. However we can make an improvement on this in the case where a range of charges have been specified. The modified brute force approach we describe below essentially eliminates the step of looping over the charges by automatically determining the correct charge (if it exists) for a valid compound and an ICR value.

Now, for each $m, r,$ and t, we are seeking values of k (if any) such that

$$r - t \leq \frac{m}{k} \leq r + t$$

It is easy to see that the inequality above is satisfied if and only if $k \in \left[\frac{m}{r+t}, \frac{m}{r-t}\right]$.

Now suppose the range for k specified by the user is $[L(k), U(k)]$. Then for each $m, r,$ and t, we are seeking the intersection (if any) of the two intervals. If this intersection is non-empty, it must be equal to the interval

$$\left[\max\left\{\left\lceil\frac{m}{r+t}\right\rceil, L(k)\right\}, \min\left\{\left\lfloor\frac{m}{r-t}\right\rfloor, U(k)\right\}\right].$$

It must be noted that the intersection is empty (i.e. no such k exists for given $m, r,$ and t) in one of two cases: Either $\left\lceil\frac{m}{r+t}\right\rceil > U(k)$ or $\left\lfloor\frac{m}{r-t}\right\rfloor < L(k)$.

We refer to this modified brute force algorithm as *BFk*. We compare the two *Analyze* algorithms, *BS* and *BFk* in Sect. 4, in terms of theoretical complexity and in Sect. 6 in terms of experimental run-times.

3.3 Recommended algorithm

The complete algorithm that we propose for solving the problem is *PG* for *Preprocess* and *BS* for *Analyze*. We refer to this algorithm as *PG-BS*. However, in Sects. 4 and 6, we consider other combinations of the algorithms for the sake of comparison with our recommended algorithm. For instance, we contrast *PG-BS* with the most basic approach, *BF-BFk*. The software implements all four combinations of algorithms.

4 Worst-case complexity analysis

In this section, we discuss the worst-case complexity of our approach, *PG-BS* compared to the brute force approach, *BF-BFk*. We assume that the *primitive* operations (such as division, comparison etc.) are independent of the inputs. This is a valid assumption in general and certainly in the context of our specific application [12]. We will use the "big-oh" notation [12,13] to express the complexity.

4.1 Brute force (BF-BFk)

It was shown in [2] that the worst-case complexity of a brute-force approach for preprocessing is at least $O(N^2K^N)$ where $K = \max\{U(x_i) - L(x_i) + 1 \mid 1 \leq i \leq N\}$.

Thus, K is the largest range provided by the user and N is the number of elements chosen by the user.

Next, for the *Analyze* step, we have to compare each of the $|V|$ elements with each of the $|S(r)|$ real data values. Thus, the worst-case complexity of this analysis is $O(|V||S(r)|)$.

Consequently, the worst-case complexity of a brute force approach is $O(N^2K^N) + O(|V||S(r)|)$.

4.2 Recommended algorithm (PG-BS)

Again, from [2], we know that the worst-case complexity of the PG preprocess algorithm is $O(N^3K^2)$. However, we have improved on even this performance by redefining the user's upper bounds (as described in Sect. 3.1) to consider only those values that are possible based on the largest ICR value, element mass, and charge in question. Note that a user is still in full control of the bounds for each element, but we are now dealing with the smallest necessary K (say, $\hat{K}$), namely

$$\max\left\{\left[\max\left\{L(x_i), \min\left(U(x_i), \left\lfloor\frac{k_m(R+t)}{m_i}\right\rfloor\right)\right\} - L(x_i) + 1\right] \mid 1 \le i \le N\right\}.$$

To this, we must add the worst-case cost of the analyze algorithm, BS. Using the well-known worst-case complexity of binary search, it follows that for each theoretical compound, the worst-case complexity of trying to find a match with a real data value is at most $O(\log(|S(r)|)$. Since this must be done for each valid compound, we must undertake at most $O(|V|\log(|S(r)|)$ operations.

Thus, the worst-case complexity of our algorithm is

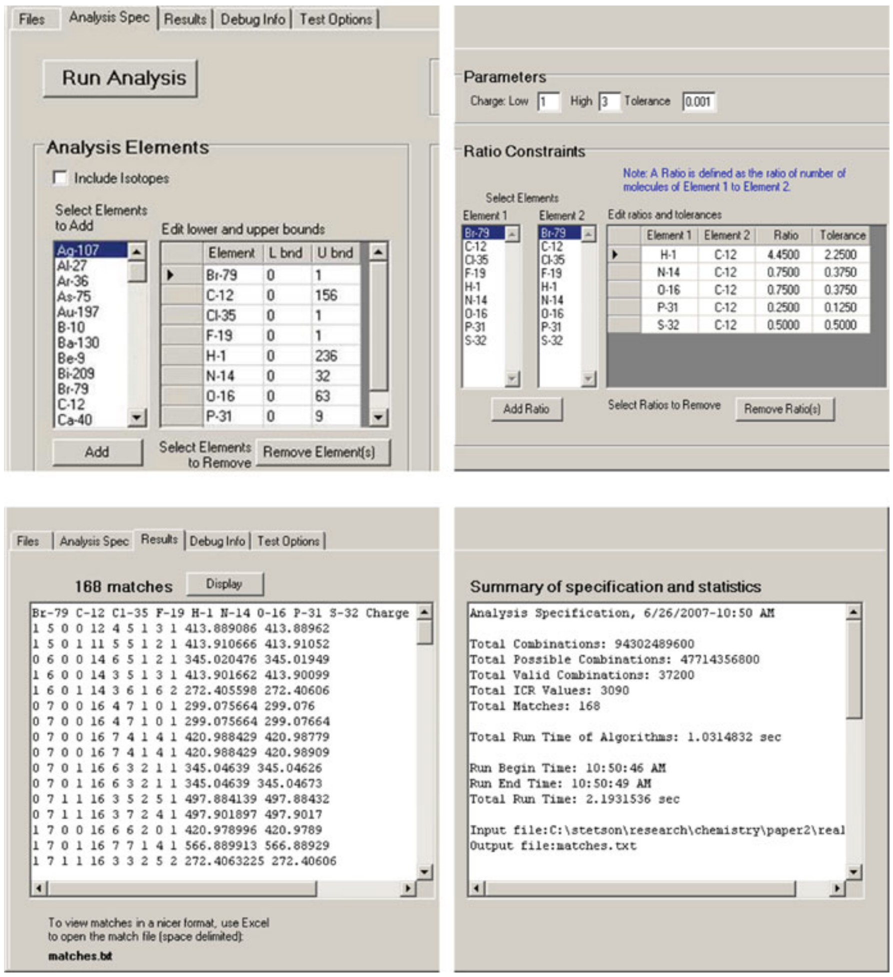
$$O(N^3\hat{K}^2) + O(|V|\log(|S(r)|)).$$

It is easy to see that our algorithm requires significantly fewer operations compared to any brute-force approach. In Sect. 6., we present run-time data analysis that unequivocally demonstrates the speed of our algorithm and implementation.

5 Software development

5.1 Software

The software, PG COMPOUND MATCH FINDER is a GUI based MS Windows program developed in C#.NET, under the .NET framework 1.1 and runs on any computer running Windows XP or better. Anticipating requests for additions and changes prompted a very flexible design which was implemented using object orientation and design patterns. During preliminary testing, it was found that this flexible design took 15–20% longer to complete an ICR analysis than a streamlined version that did away with some of the design patterns and the strict object orientation. The faster version of the program was adopted, continues to be developed and is available from the authors.



Edit lower and upper bounds

Element	L bnd	U bnd
Br-79	0	1
C-12	0	156
Cl-35	0	1
F-19	0	1
H-1	0	236
N-14	0	32
O-16	0	63
P-31	0	9

Parameters

Charge: Low 1 High 3 Tolerance 0.001

Ratio Constraints

Note: A Ratio is defined as the ratio of number of molecules of Element 1 to Element 2.

Edit ratios and tolerances

Element 1	Element 2	Ratio	Tolerance
H-1	C-12	4.4500	2.2500
N-14	C-12	0.7500	0.3750
O-16	C-12	0.7500	0.3750
P-31	C-12	0.2500	0.1250
S-32	C-12	0.5000	0.5000

168 matches

```
Br-79 C-12 Cl-35 F-19 H-1 N-14 O-16 P-31 S-32 Charge
1 5 0 0 12 4 5 1 3 1 413.889086 413.88962
1 5 0 1 11 5 5 1 2 1 413.910666 413.91052
0 6 0 0 14 6 5 1 2 1 345.020476 345.01949
1 6 0 0 14 3 5 1 3 1 413.901662 413.90099
1 6 0 1 14 3 6 1 6 2 272.405598 272.40606
0 7 0 0 16 4 7 1 0 1 299.075664 299.076
0 7 0 0 16 4 7 1 0 1 299.075664 299.07664
0 7 0 0 16 7 4 1 4 1 420.988429 420.98779
0 7 0 0 16 7 4 1 4 1 420.988429 420.98909
0 7 0 1 16 6 3 2 1 1 345.04639 345.04626
0 7 0 1 16 6 3 2 1 1 345.04639 345.04673
0 7 1 1 16 3 5 2 5 1 497.884139 497.88432
0 7 1 1 16 3 7 2 4 1 497.901897 497.9017
1 7 0 0 16 6 6 2 0 1 420.978996 420.9789
1 7 0 1 16 7 7 1 4 1 566.889913 566.88929
1 7 1 1 16 3 3 2 5 2 272.4063225 272.40606
```

To view matches in a nicer format, use Excel to open the match file (space delimited):

matches.txt

Summary of specification and statistics

```
Analysis Specification, 6/26/2007-10:50 AM

Total Combinations: 94302489600
Total Possible Combinations: 47714356800
Total Valid Combinations: 37200
Total ICR Values: 3090
Total Matches: 168

Total Run Time of Algorithms: 1.0314832 sec

Run Begin Time: 10:50:46 AM
Run End Time: 10:50:49 AM
Total Run Time: 2.1931536 sec

Input file:C:\stetson\research\chemistry\paper2\real
Output file:matches.txt
```

Fig. 1 PG compound match finder panels: **a**, **b** on row 1, **c**, **d** on row 2

Figure 1 shows the user-friendly interface of PG COMPOUND MATCH FINDER. A tabbed dialog is used organize the various inputs and outputs. The *Files* tab (not shown) is used to specify the ICR data file and an output file where matches are written. The *Analysis Spec* tab is shown in Fig. 1a and b. In Fig. 1a, the user can choose to add or remove elements (including isotopes) as well as adjust their bounds. Figure 1b shows that the user can supply global parameters such as the charge range and overall tolerance for matches. Also shown in Fig. 1b is how the user can select elements to form a ratio and edit the ratios and tolerances. When the problem specification is complete, the user can press the Run Analysis button to perform the ICR analysis. When it is complete, the *Results* tab is displayed (Fig. 1c, d) which shows the matches and various run statistics. The software also has a feature (not shown) that allows the user to load a complete specification from a text file.

5.2 Future enhancements

Currently, we are investigating an algorithm to estimate the length of time that a run will take using the PG-BS algorithm. The idea is to preprocess (using PG) the data and then present the user with an estimate of the time required to complete the analysis and the ability to abandon further analysis. This could be a very useful feature as it is easy to specify a set of elements and rules that generate a very large number of valid compounds and consequently takes a long time to complete the analysis. One other usability feature that is under design is providing a mechanism for the user to stop a long running analysis *gracefully*. Of course we would also like to implement any rules that individuals might suggest that could help the preprocess step reduce the size of the set of valid compounds in certain circumstances.

6 Computational analysis of algorithm and software

In this section we test the software's run-time performance using various combinations of the algorithms for *Preprocess* and *Analyze* for different input values. A complete analysis of all the factors that affect the algorithms and their implementations is beyond the scope of this paper. However, we do present compelling evidence that the PG-BS algorithm is extremely fast and several orders of magnitude faster than any brute force algorithms.

6.1 Testing environment

All testing was done on a brand new, fairly standard HP Compaq dc7700 Core 2 Duo with 2 GB RAM and two, 1.86 GHz processors, running Windows XP. The computer was not connected to any networks, no virus protection software was running, nor any process other than the standard Windows processes. In order to make cleaner inferences on the results, all timing data excludes all disk IO. Also, in order to have a fair comparison of the *Preprocess* algorithms, we did not want the algorithm for reducing the upper bounds (Sect. 3.1) to come into play. In order to achieve this, we simply chose a single, very large ICR value to include in our ICR data file so that no bounds would ever be reduced.

6.2 Analysis of preprocess algorithms

In the first experiment we consider the PG algorithm with 10 randomly chosen elements, six ratios rules (all with a ratio of 1 and a tolerance of 0.5), charge from 1 to 3, and a single ICR value. A single value was chosen because the *Preprocess* step does not depend on the number of ICR values (except as noted above, but controlled for in this experiment). For any given run, the upper bounds for each element were all the same. We varied the upper bound on each element from 5 to 16 which produced 9.8 million to 1.1 trillion possible compounds, respectively. The setup for considering

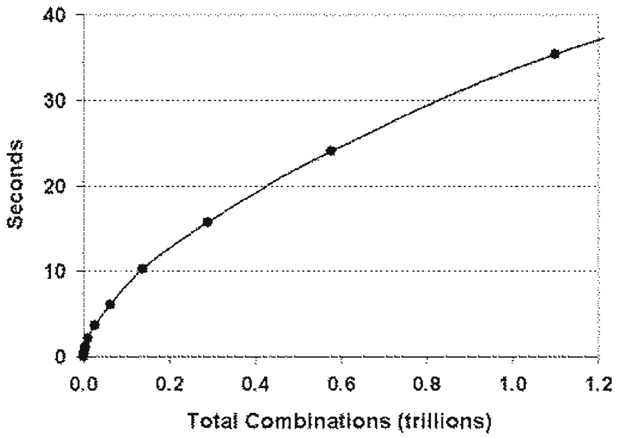


Fig. 2 PG preprocess runtimes, 10 elements, 6 ratios

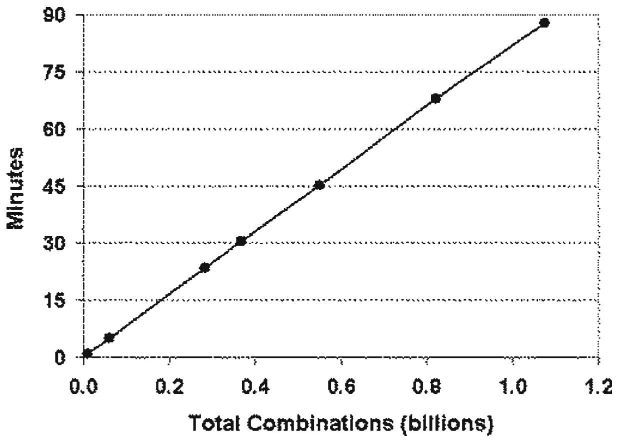


Fig. 3 BF preprocess runtimes, 10 elements, 6 ratios

the BF algorithm was essentially the same. A few extra runs on the low end were done in order to fill in the graphs.

Figure 2 shows the results for the PG algorithm and Fig. 3 for the BF algorithm. Note that the horizontal and vertical axes for the PG algorithm are in *trillions* and *seconds*, respectively, while for the BF algorithm they are in *billions* and *minutes*, respectively. We see that 1.1 trillion possible combinations were preprocessed by PG in only 35 s compared with BF which took almost 90 min to preprocess only 1.1 billion possible combinations. Clearly, there is an extreme difference in the results with the PG algorithm running several orders of magnitude faster than the BF algorithm. We also note that the growth in run-time is not as fast for PG compared to BF.

In the next experiment, we choose 10 elements and 8 ratios as recommended in [8]. In this case, there were two dominant elements, Carbon and Hydrogen where we consider upper bounds of up to 162 and 208, respectively. Also, all the ratios involve

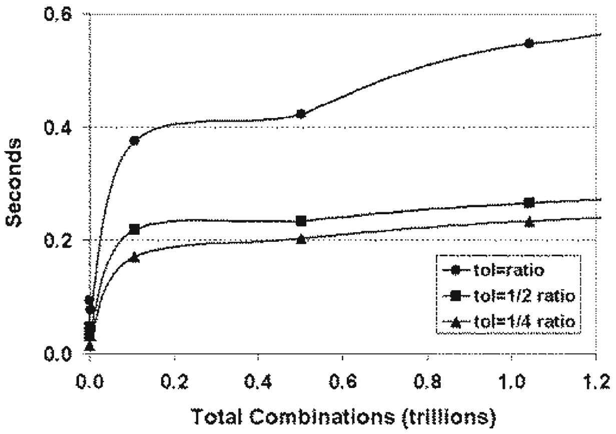


Fig. 4 PG preprocess runtimes, 10 elements, 8 ratios

Carbon. Figure 4 shows the results of the PG algorithm. The times here are much faster than the values in Fig. 2. For instance the case with 1.1 trillion possibilities was reduced from about 35 s (Fig. 2) to about 0.5 s (Fig. 4). We conjecture that, as in Fig. 2, when all the upper bounds are nearly equal, this serves as a sort of *worst case* in terms of preprocessing effort.

As shown in Fig. 4, we also examine the effect of the size of the tolerance on the ratios. We vary this for the Hydrogen-Carbon ratio from the case where the tolerance is roughly equal to the ratio down to where the tolerance is about one-fourth the ratio. More dramatic than the differences in run-times are the differences in the number of valid compounds that were found. As expected, as the tolerance gets close in value to the ratio, the valid region opens up more allowing for more valid compounds. This situation seems to be accentuated when there are several dominant elements as in this experiment. For instance, with the tolerance approximately equal to the ratio in the case with 1.1 trillion possible combinations, more than 200 billion were found to be valid. When the Hydrogen-Carbon tolerance was reduced to one-half the ratio only about 1 million valid compounds were found.

6.3 Analysis of analyze algorithms

The Analyze algorithms depend only on the number of elements and the number of ICR values. They do not depend on the number or makeup of the ratio rules, nor the upper bounds of the elements. Thus, no mention of these are included in the following experiments. Figure 5 shows the results of comparing BS and BFk, with 1000 ICR values, charge from 1 to 3, where the number of valid combinations is varied. Clearly the BS algorithm is vastly superior to the BFk algorithm. For instance, when the number of valid combinations is about 32 million, *BS* takes about 3 min compared to *BFk* which take about 115 min. Also, as the complexity suggests, growth in either case appears to be approximately linear (for a fixed number of ICR values).

Though not analyzed here, it should be noted that the number of elements does play a role in the run times for Analyze algorithms. This so because the compound mass must be calculated by summing the product of the mass of each element with the number of atoms of that element, for each element, before comparison with ICR values.

Figure 6 shows the run time performance of the BS algorithm for 1, 1000, and 100,000 ICR values. As shown, approximately 32 million valid compounds compared against 100,000 ICR values takes about 3.8 min.

6.4 Analysis of complete algorithm

To compare both algorithms (PG-BS and BF-BFk) for a complete problem, we chose the setup in Sect. 6.1 where we had 10 elements and 6 ratios which resulted in a fairly

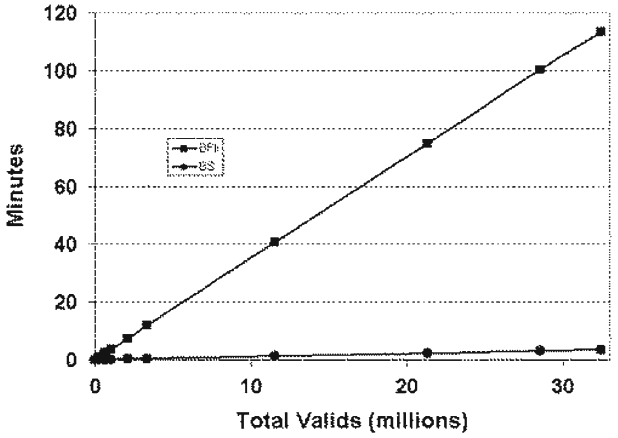


Fig. 5 BS, BFk analyze runtimes, 10 elements, 1000 ICR values

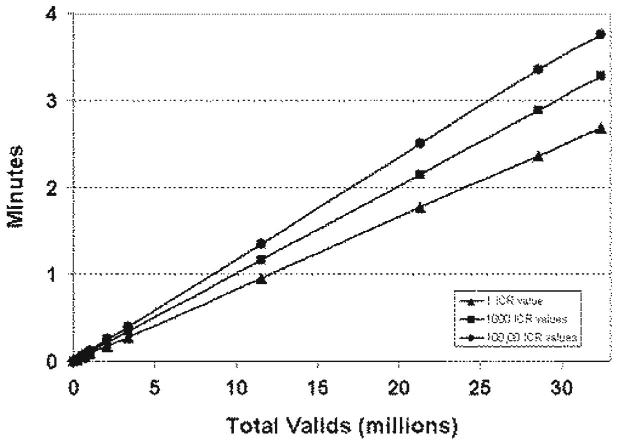


Fig. 6 BS analyze runtimes, 10 elements

Table 1 Run time (seconds), 10 elements, 6 ratios, 1000 ICR values

	Preprocess	Analyze	Total	
PG-BS	1	49	50	<1 min
BF-BFk	5250	1787	7037	≈ 2 h

small number of possible compounds (1.1 billion) and valid compounds (8.2 million). The results are shown in Table 1 where we see that the PG-BS algorithm is clearly superior, running about 140 times faster than the BF-BFk algorithm.

So far, we haven't encountered cases where the PG algorithm blows up. We have tested PG where there were almost 1,000 trillion possible compounds and it completed Preprocess in just a few seconds. However, it is conceivable that there are certain combinations of input factors that would fracture the set of valid compounds so much that software would run out of memory trying to intersect (as described in [2]) all the rule matrices. The real bottleneck, however is with BS. Its run-time will always be a linear function of the number of valid compounds (for a fixed number of ICR values). However, as shown in Fig. 6, if the number of valid compounds is kept below 30 million, the run time should be no more than a few minutes.

The BF-BFk algorithm can take a very long time if either the total number of possible combinations is large or (to a lesser degree) the number of valid combinations is large. In one case we considered, based on the conditions in [8], There were about 1 trillion possible combinations that were reduced to about 700,000 valid compounds. Although the Analyze time (using BFk) is quite modest, about 2.5 min, the Preprocess time (using BF) would be enormous, estimated to be about 59 days!

7 Conclusions

This work has clearly shown the robustness and efficiency of our algorithm and its implementation. This has been a truly inter-disciplinary project drawing from mathematics, computer science, statistics, and chemistry. We hope that by presenting our algorithm(s) and software to the research community, we are able to contribute to the important problem of efficient mass analysis of spectral data. Furthermore, it is our hope that feedback from the research and industrial communities will spawn further developments in this arena. We are excited about the prospect of continually enhancing features of our algorithms and software. Ultimately, our goal is to meet the needs of researchers using mass spectrometry in seeking better and faster technological tools to help them conduct their basic research in areas like proteomics, petroleomics, and pharmacokinetics.

References

1. M.B. Comisarow, A.G. Marshall, Fourier transform ion cyclotron resonance spectrometry. Chem. Phys. Lett. **25**, 282 (1974)
2. H. Pulapaka, D.R. Gibson, An efficient algorithm for chemical fingerprinting. J. Math. Chem. **44**(1), 75–87 (2008)

3. J. Bryant, T.J. Manning, D.R. Gibson, A.G. Marshall, R. Rogers, *Comparing chemical fingerprints: is oil the product of bacterial production?* 58th Southeast Regional Meeting of the Americal Chemical Society, Augusta, GA, United States, November 1–4 (2006), SRM06-704. Publisher: American Chemical Society, Washington, DC CODEN: 69INUY Conference; Meeting Abstract written in English. AN 2006:1191233 CAPLUS
4. B. Seebass, E. Pretsch, Automated compatibilty tests of the molecular formulas or structures of organic compounds with their mass spectra. J. Chem. Inf. Comput. Sci. **39**(4), 713–717 (1999)
5. J. Lederberg, Rapid calculation of molecular formulas from mass values. J. Chem. Educ. **49**(9), 613 (1972)
6. A.L. Rockwood, P. Haimi, Efficient calculation of accurate mass of isotopic peaks. J. Am. Soc. Mass. Spectr. **17**(3), 415–419 (2006)
7. T. Kind, O. Fiehn, Metabolomic database annotations via query of elemental compositions: mass accuracy is insufficient even at less than 1 ppm. BMC Bioinformatics **7**, 234 (2006)
8. T. Kind, O. Fiehn, Seven golden rules for heuristic filtering of molecular formulas obtained by accurate mass spectrometry. BMC Bioinformatics **8**, 105 (2007)
9. J.K. Senior, Partitions and their representative graphs. Am. J. Math. **73**(3), 663–689 (1951)
10. T. Morikawa, B.T. Newbold, Analogous odd-even parities in mathematics and chemistry. Chemistry (Bulg. J. Chem. Educ.) **12**(6), 445–450 (2003)
11. A.W. Weiss, *Data Structures and Problem Solving Using Java* (Addison Wesley, Reading, MA, 2006)
12. T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduction to Algorithms* (McGraw-Hill Publishing/MIT Press, NY/Cambridge, 1990)
13. C. Tovey, Tutorial on computational complexity. Interfaces (INFORMS) **32**(3), 30–61 (2002)