analytical chemistry

pubs.acs.org/ac Article

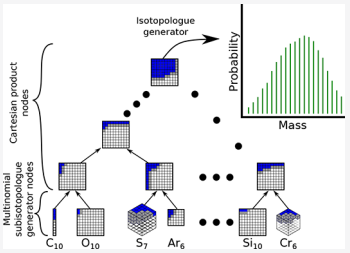
Fast Exact Computation of the *k* Most Abundant Isotope Peaks with Layer-Ordered Heaps

Patrick Kreitzberg, Jake Pennington, Kyle Lucke, and Oliver Serang*

Cite This: *Anal. Chem.* 2020, 92, 10613−10619 Read Online

ACCESS | Metrics & More | Article Recommendations

ABSTRACT: Computation of the isotopic distribution of compounds is crucial to applications of mass spectrometry, particularly as machine precision continues to improve. In the past decade, several tools have been created for doing so. In this paper we present a novel algorithm for calculating either the most abundant *k* isotopologue peaks of a compound or the minimal set of isotopologue peaks which have a combined total abundance of at least *p*. The algorithm uses Serang's optimal method of selection on Cartesian products. The method is significantly faster than the state-of-the-art on large compounds (e.g., Titin protein) and on compounds whose elements have many isotopes (e.g., palladium alloys).



Calculating the theoretical isotopic distribution of compounds is a valuable tool in mass spectrometry (MS); however, it poses a difficult combinatorics problem because there are exponentially many isotopologues to consider. Computation of the theoretical isotopic distribution is useful for targeted screening,[9,13] identifying unknown metabolites,[4] and in general MS workflows.[14]

There have been multiple methods developed in the past decade focused on more efficiently calculating the most abundant peaks from the isotope distribution.[3,7,8,10,15] In 2019, Wang et al. compared four of the top algorithms: ISOSPEC, ENVIPAT, ECIPEX, and their own ISOVECTOR. They found ISOSPEC to consistently be the fastest of the four.[15] While it is possible to simply enumerate the exponentially many peaks or possible to approximate the distribution of isotopologues,[10] methods like ISOSPEC compute the exact abundances and masses of the most abundant isotopologues without enumerating all possible isotopologues.[7] ISOSPEC does this by employing a central-limit theorem-based approximation to define outcomes that roughly account for a given total abundance *p*.

ISOSPEC works by first calculating the most abundant subisotopologues (all instances of the same element in a compound, for example H_2 and O are two subisotopologues of water) and then combining the subisotopologues to form whole isotopologues. The isotopologues are then put into a FIFO queue and when popped, if they exceed a threshold, it will be appended to the output. Each isotopologue's neighbors are inserted into the queue. An isotopologue is a neighbor of another if they differ by changing one isotope. Each threshold for the FIFO queue creates a new layer of isotopologues, where the cumulative output of all layers is $\in O(\cdot)$ of the optimal output; however, in practice ISOSPEC may produce significantly more isotopologues than are necessary, requiring a final selection step.

Once two subisotopologues have been calculated, selecting the top *k* isotopologues of the compound by merging the two subisotopologues is the same as selecting the top *k* terms in a Cartesian product of two lists, $X + Y$, where addition is used to add log abundances (equivalent to multiplying their frequencies). There are multiple methods for selecting the top *k* terms in the Cartesian product on $X + Y$ in optimal time.[5,12] Serang's optimal method utilizes layer-ordered heaps (LOH) and is the fastest in practice. LOHs create a continuum between unsorted and sorted data by partitioning a list into layers. Values in layer L_i must be less than or equal to all values in subsequent layers $L_j, j > i$, but values within layers are unordered. The sizes of the layers grow in an asmyptotically exponential manner such that $\frac{|L_{i+1}|}{|L_i|} \approx \alpha,\ i \gg 1$. Where comparison-based sorting is $\in \Omega(n \log(n))$, lists can be LOHified in $O(n)$ time. Unlike soft heaps,[2] LOHs are contiguous in memory, leading to greater cache performance.

The α parameter is very important in LOHs as it controls the level of order in the LOH, this is similar to the ϵ parameter of the soft-heap which controls the amount of corruption. If $\alpha = 1$ then each layer has size 1 and so the LOH is completely sorted and if $\alpha > n$ then the LOH will be two layers, the minimum element then $n - 1$ unsorted elements. A small α is desirable because it enforces more ordering on the LOH, the trade-off is the time to LOHify the list is increased. In practice, $\alpha \in [1.01, 1.1]$ seems to

Received: April 18, 2020
Accepted: July 14, 2020
Published: July 14, 2020

https://dx.doi.org/10.1021/acs.analchem.0c01670
Anal. Chem. 2020, 92, 10613−10619

have a good trade-off between enforcing enough ordering and a better-than-sorting LOHify cost.

In this paper, we present a method of efficiently calculating the top k isotopologues of a compound using LOHs. The method solves the problem exactly using a purely combinatorial approach. It does not approximate, bin or round any numbers. This is achieved by building a balanced binary tree where internal nodes perform online $X + Y$ selection and where leaves generate the most abundant subisotopologues by performing selection on multinomials. The reported peaks will not be sorted but will instead be layer-ordered. We also present a small C++17 implementation, NEUTRONSTAR, which is provided with a free license.

METHODS

Here, we present a method that computes the top k isotopologue peaks of a given chemical formula.

Calculating all subisotopologues of a compound is equivalent to expanding products of polynomials with each taken to a power: $(b_1 \cdot X^{\beta_1} + b_2 \cdot X^{\beta_2} + \ldots)^{q_1} \ldots$. The b_i are the isotopic abundances that correspond to each isotopic mass β_i. Each element has one subisotopologue polynomial taken to some power q_j, which reflects the number of occurrences of that element in the compound. For example, an element with four carbons has polynomial $(0.9893 \cdot X^{12.0} + 0.0107\ X^{13.003})^4$. These subisotopologue polynomials are combined via Cartesian product by multiplying the two polynomials. For example, H_3C_4 is $(0.9893 \cdot X^{12.0} + 0.0107 \cdot X^{13.003})^4 (0.999885 \cdot X^{1.008} + 0.000115 \cdot X^{2.014})^3$.

When expanding a polynomial such as $(c_1 \cdot X + c_2 \cdot Y)^2$ there will be terms that can be merged together: $(c_1 \cdot X + c_2 \cdot Y)^2 = (c_1^2 \cdot X^2 + c_1 c_2 \cdot X \cdot Y + c_2 c_1 \cdot Y \cdot X + c_2^2 \cdot Y^2) = (c_1^2 \cdot X^2 + 2 c_1 c_2 \cdot X \cdot Y + c_2^2 \cdot Y^2)$. For larger polynomials (both in the power and number of terms) there will be many terms that may combine. A significant speed-up can be found if only one of these terms is calculated then multiplied by the appropriate multinomial coefficient (2 in the previous example).

We do not compute the full polynomial expansion, because that would present exponentially many terms; instead, we perform selection of the largest coefficients of the multinomials (from the element subisotopologue polynomials taken to a power) and of Cartesian products (from the polynomial multiplications).

Selection on a Multinomial. Multinomial selection begins with the mode of that subisotopologue, which is the term in the subisotopologue expansion that has largest coefficient. Subsequent outcomes are generated in descending order of abundance using a binary max-heap, where keys are the probabilities of each outcome for the isotopologue. When a subisotopologue is popped from the heap, it proposes new subisotopologues to enter the heap based on their index tuple (a tuple that describes how many of each isotope is in the subisotopologue). For example, C_{100} begins at the mode (99,1), which corresponds to 99 copies of ^{12}C and 1 copy of isotope ^{13}C.

Let $(x_1, x_2, \ldots, x_m)$ be a mode of our distribution where p_i represents the abundance of the isotope at index i in the tuple. Because it is the mode, and therefore the most abundant isotopologue, $P(x_1,x_2,\ldots,x_m) \geq P(\ldots,x_i + 1,x_j - 1,\ldots)$ for any indices x_i and x_j. Examining the probability mass function of a multinomial we find that

$$\frac{p_i \cdot (x_j)}{p_j \cdot (x_i + 1)} \leq 1$$

for any indices x_i and x_j.

Furthermore, because

$$\frac{P(\ldots, x_i + (b+1), \ldots, x_j - (b+1), \ldots)}{P(\ldots, x_i + b, \ldots, x_j - b, \ldots)} = \frac{p_i^{x_i+(b+1)} \cdot p_j^{x_j-(b+1)} \cdot (x_i + b)! \cdot (x_j - b)!}{(x_i + (b+1))! \cdot (x_j - (b+1))! \cdot p_i^{x_i+b} \cdot p_j^{x_j-b}} = \frac{p_i \cdot (x_j - b)}{p_j \cdot (x_i + (b+1))}$$

and

$$\frac{p_i \cdot (x_j - b)}{p_j \cdot (x_i + (b+1))} \leq \frac{p_i \cdot (x_j - (b-1))}{p_j \cdot (x_i + b)} \leq \ldots \leq \frac{p_i \cdot (x_j - 1)}{p_j \cdot (x_i + 2)} \leq \frac{p_i \cdot (x_j)}{p_j \cdot (x_i + 1)} \leq 1$$

we can see that $P(\ldots,x_i + (b+1),x_j - (b+1),\ldots) \leq P(\ldots,x_i + b,x_j - b,\ldots)$ for any $b \leq \min(x_i,x_j)$. This means every time the ith entry in the index tuple is increased and the jth entry is decreased, thus moving further from the mode in $\mathbb{L}_1$ (or Manhattan) distance, the probability never increases.

The relationship between the $\mathbb{L}_1$ distance from the mode and the probability still holds when other index tuple entries have been perturbed away from the mode. $P(\ldots,x_i + 1,x_j - 1,\ldots) \geq P(\ldots,x_i + 2,x_j - 1,x_k - 1\ldots)$ because

$$P(\ldots, x_i + 2, x_j - 1, x_k - 1 \ldots) = P(\ldots, x_i + 1, x_j - 1, \ldots) \cdot \frac{p_i \cdot (x_k)}{p_k \cdot (x_i + 2)}$$

and

$$\frac{p_i \cdot (x_k)}{p_k \cdot (x_i + 2)} \leq \frac{p_i \cdot (x_k)}{p_k \cdot (x_i + 1)} \leq 1$$

Finally, we also have

$$P(\ldots, x_i + 1, x_j - 1, \ldots) \geq P(\ldots, x_i + 1, x_j - 1, x_k + 1, x_l - 1 \ldots)$$

because

$$P(\ldots, x_i + 1, x_j - 1, x_k + 1, x_l - 1 \ldots) = P(\ldots, x_i + 1, x_j - 1, \ldots) \cdot \frac{p_k \cdot (x_l)}{p_l \cdot (x_k + 1)}$$

and

$$\frac{p_k \cdot (x_l)}{p_l \cdot (x_k + 1)} \leq 1$$

Since the probability never decreases as we move closer to the mode, then wherever we are, if we head in a direction of ascending probability, we are heading toward the mode. Once we are at a location which can not increase in probability, we have reached the mode. Because the distribution is discrete and we move by the smallest possible amount (incrementing and decrementing a pair of indices by one), we will never overshoot the mode. The starting position for this greedy process is found by using the modes of each the binomial marginals and

https://dx.doi.org/10.1021/acs.analchem.0c01670
Anal. Chem. 2020, 92, 10613−10619

correcting if the sum is not = n, although any starting position will lead to a mode because there are no local maxima.

In order to populate the heap with the best possible next subisotopologue, any subisotopologue in the heap must have all subisotopologues between itself and the mode already in the heap (or have been popped from the heap); this ensures that $\mathbb{L}_1$ distance of proposed index tuples is always increasing and thus index tuples are visited in descending order of probability. This is accomplished by pushing all neighbors of the subisotopologue that has been popped from the heap. These neighbors are found as with the search for the mode: from some starting point, one index is increased and one index is decreased, thereby holding the sum constant; however, unlike the search for the mode, here we must guarantee that the $\mathbb{L}_1$ distance from the mode always increases, and so proposed neighbors that would move closer to the mode on any axis are discarded.

It is necessary to prevent the same neighbor from being inserted into the heap multiple times. For example, index tuple (30, 4, 3) has neighbors (29, 5, 3), (29, 4, 4), (31, 3, 3), (31, 4,2), (30, 5, 2), (and 30, 3, 4). Of these, (29, 5, 3) and (30, 5, 2) both have neighbor (29,6,2) . One way to prevent these duplicates from being reinserted into the heap is to store a set of the heap's contents; however, that requires additional memory and time. Although it is asymptotically comparable to the cost of pushing to and popping from the heap, it significantly harms performance in practice. For this reason, we use a proposal scheme that can reach all subisotopologues in increasing $\mathbb{L}_1$ order from the mode but without duplicates.

A proposal can be characterized by the two axes, i,j that are perturbed (without loss of generality, let index i increase and index j decrease). If two chains of neighbors, (i_1, j_1), (i_2, j_2),... and (i'_1, j'_1), (i'_2, j'_2),..., collide then multiset(i_1, i_2,...) = multiset(i'_1, i'_2,...), multiset(j_1, j_2,...) = multiset(j'_1, j'_2,...). Multisets are unique when their contents are sorted, and thus chains whose i and j are both in lexicographic order (by the index of the largest entry that has been perturbed as i and j respectively) will visit each index tuple only once. This proposal scheme means any index tuple may be proposed by only one unique neighboring index tuple. The first few proposals for the subisotopologue K_3 may be seen in (Figure 1).

Now that the top isotopologues can be generated with the most abundant (i.e., most probable) first, values can be requested in an online manner.

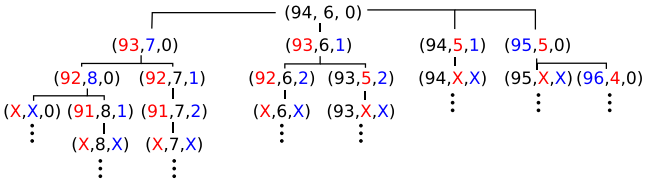


Figure 1. First few multinomial proposals for K_{100}. The figure shows index tuples in the multinomial and the neighbors they propose, from top to bottom, starting with the mode (94,6,0) (94 copies of ^{39}K, 6 copies of ^{41}K, and no copies of ^{40}K). Each index tuple proposes its neighbors in lexicographical order where, if the ith index has been incremented, it cannot propose any neighbors by incrementing an index less than i (this is the same pattern is used for decrementing an index). In the figure, the largest index to be incremented is in blue and the largest to be decremented is in red. In order to move away from the mode, any index which has been incremented may not be decremented to create a proposed tuple, and vice versa. Note that for clarity not all proposed indices are included.

Selection on Two Partial Isotopologues. Generating the largest coefficients in the polynomial product is accomplished by using Serang's selection algorithm on $X + Y$.

The $X + Y$ selection algorithm focuses on layer products (LPs) of the lohified input lists X and Y. The lists are lohified $\in O(n)$, avoiding the $\Omega(n \log(n))$ bound which would be required if they were sorted. An LP is the Cartesian product of some layer $i \in X$ and layer $j \in Y$; however, the presence of an LP does not ensure all values in the Cartesian product will be generated. All LPs are represented as a tuple which has a value, the layer indices in X and Y, and a boolean. There are two types of LPs: a min-corner which has the minimum value in the Cartesian product and the boolean FALSE (indicating it is a min-corner), and a max-corner, which has the maximum value in the Cartesian product and the boolean TRUE (indicating it is a max-corner).

A priority queue is utilized in order to select the minimum number of LPs such that the generated Cartesian products of the LPs will contain the top k values. The priority queue is initialized with the min-corner LP of the first LP in X and Y. Once a min-corner LP is popped from the queue, its max-corner equivalent is inserted. The size of the Cartesian product of an LP is accumulated when the max-corner is popped, the popping continues until this total has reached k.

Once k is reached, among the Cartesian products of the popped LPs (either min-corner or max-corner), there will be at least k many values less than the value of the last max-corner LP popped. To find the final top k values, all values in LPs which have been popped are generated and a one-dimensional k-select is performed. If both a min-corner and max-corner of the same indices in X,Y are popped, only one of them has its Cartesian product generated.

As this applies to selection on isotopologues, the inputs X and Y will be layers of either partially built isotopologues of the desired compound or whole subisotopologues. For example, if calculating $Au_2Ca_{10}Ga_{10}Pd_{76}$ there will be a selection where one list is Au_2 and the other is Ca_{10} and a different selection where one list is Au_2Ca_{10} and the other is $Ga_{10}Pd_{76}$.

We do not wish to generate all possible combinations of Au_2 and Ca_{10} in order to do the selection on $Au_2Ca_{10} + Ga_{10}Pd_{76}$. This is solved by performing "online" generation of the inputs into the selection $Au_2Ca_{10} + Ga_{10}Pd_{76}$. In order for an $X + Y$ selection to remain online, once the selection is done, the heap must not be modified. On the next selection the heap starts popping from where it stopped in the last selection, then some care must be taken in order to not admit overlapping values between selections. Since the layers requested by the parents grow exponentially in size, the work done by each selection node will be dominated by the last selection. Note that in Serang's manuscript, lemma 7 states that s', the total area of all min-corner LPs visited as postprocessing for this round of selection, will have $s' \in O(n + k)$; in fact, this can be improved to $s' \in O(k)$ because the contribution by an LP with either of $u,v = 1$ is limited by the previous layer along that axis.

Selecting Most Abundant Isotopologues from a Compound. The method described above is able to efficiently get the top k combinations of two subisotopologues; however, for compounds of more than two elements, this method alone is not sufficient. In order to combine all subisotopologues, a balanced binary tree of two different kinds of nodes is formed. The leaves of the tree are all multinomial subisotopologue generators while all internal nodes are $X + Y$ selection nodes (Figure 2). Axes X and Y of each internal node are extended when necessary by requesting another layer from the relevant

https://dx.doi.org/10.1021/acs.analchem.0c01670
Anal. Chem. 2020, 92, 10613−10619

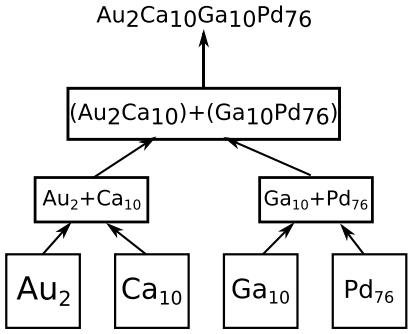


Figure 2. Illustration of the balanced binary tree for palladium alloy PGC, $Au_2Ca_{10}Ga_{10}Pd_{76}$. The leaves are subisotopologue generators of Au_2, Ca_{10}, Ga_{10}, and Pd_{76}. All nodes above the leaves combine their child compounds using the modified pairwise selection from Serang's method. The tree's root generates isotopologues of $Au_2Ca_{10}Ga_{10}Pd_{76}$, and every other node generates isotopologues for some smaller constituent compound.

LOH generator (either X or Y). This makes an invariant that the inputs to and output from each node are LOHs. In this manner, only LOHs are made at internal nodes and sorting is not performed in those nodes. Likewise, the LOHs guarantees the exponential growth necessary for the online k-selection on $X + Y$ mentioned in the paragraph above.

The final top k peaks are taken from the root by generating layers until their total number of elements is $\geq k$. A one-dimensional selection is used to narrow that result to exactly k in linear time.[1] If, instead of the top k peaks, the user requests the minimal set of isotopologue peaks which have a combined total abundance of at least p, the layers stop being produced once their combined abundance reaches p. Then, the final layer is sorted and only those which are needed to pass p are returned in the result.

Time Analysis. If the leaves are removed from the tree, then the tree has the same time complexity as the FastSoftTree algorithm presented by Kreitzberg et al.[6] The difference in the algorithms is that this algorithm uses LOHs where FastSoftTree uses soft-heaps.[2] LOHs and soft-heaps have the same theoretical runtime for selection, but in practice LOHs are significantly faster due to the data being contiguous in memory.

The subisotopologue generators form tensors that have the same dimensionality as the number of isotopes of the element. Thus, the subgenerators themselves have the same time complexity as the SortTensor method used in Kreitzberg et al., and therefore are $\in O(n_e \cdot m_e + k_e \cdot m_e^2 + k_e \log(k_e \cdot n_e))$, where n_e is the number of element e in the compound, m_e is the number of isotopes of e, and k_e is the number of subisotopologues generated. For small k the algorithm is leaf heavy, and for large enough k most of the work done will be in the interior nodes and so the tree becomes dominated by the $X + Y$ selections.

RESULTS

Here, we compare NEUTRONSTAR versus ISOSPEC. The C++ interface for ISOSPEC was compiled with flags set so that only the masses and log-probabilities are generated, specifically an instance of TotalProbFixedEnvelope was created with flags (true, false, true, true, false). Both ISOSPEC and NEUTRONSTAR were compiled with g++ -O3 -march = native -mtune = native -std = c++17. The executables ran on a computer with dual AMD Epycs 7351 and 256GB of RAM. All runtimes used α = 1.01 which is good for both the k and the p method; however, if only the k method is used, α = 1.05 is typically faster in practice.

Time. ISOSPEC generates a superset of the needed isotopologues and then performs one-dimensional selection to retrieve the most abundant. The isotopologues ISOSPEC generates are chosen as a function of p, the cumulative abundance threshold. NEUTRONSTAR ran with both parameters as input, p, and the corresponding k recorded from running ISOSPEC with p. The k parameter is a faster method for NEUTRONSTAR because the p parameter requires keeping track of the accumulated abundance of all previous isotopologues produced.

Both ISOSPEC and NEUTRONSTAR generate more peaks than are necessary and then do some form of selection to remove the extra peaks; however, ISOSPEC tends to generate many more extra peaks than NEUTRONSTAR. For example, on the averagine[11] molecule $C_{24692}H_{38792}N_{6788}O_{7386}S_{208}$, with p = 0.9 the trimmed number of peaks was 51 633, but ISOSPEC generated 73 415, an additional 42.19% while NEUTRONSTAR generated 51 776, an additional 0.2769%. For p = 0.999 ISOSPEC, generated an extra 38.46% of peaks while NEUTRONSTAR generated an extra 4.695%. The large difference in extraneous peaks being generated plays a large role in the runtime disparities.

Space. The memmory usage of both programs was gained under the same setup as the runtimes, except we do not bother producing output for NEUTRONSTAR using the two separate parameters since they will produce the same number of layers (and thus overall values) in either case. VALGRIND, specifically the callgrind tool, was used to track the memory usage throughout each programs execution.

Generated Spectra. Figure 3 depicts the most abundant 100 000 peaks of palladium alloy PGC, $Au_2Ca_{10}Ga_{10}Pd_{76}$; at a high resolution, these peaks are subtly staggered from one another. This would be seen by a high mass accuracy spectrometer.

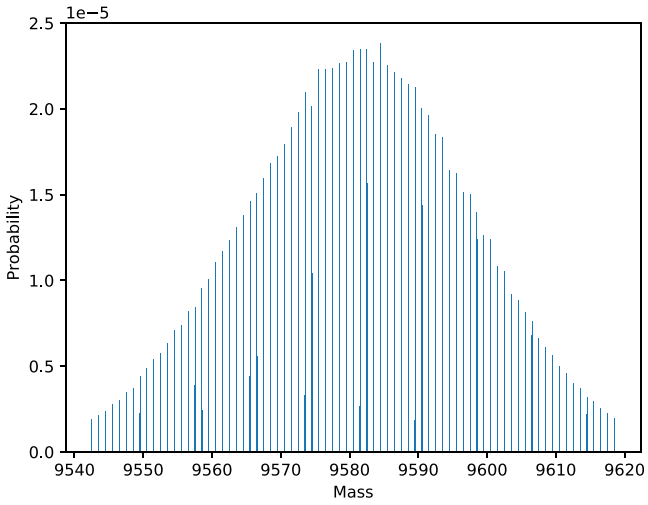


Figure 3. Theoretical spectra of the top 100 000 peaks of palladium alloy PGC, $Au_2Ca_{10}Ga_{10}Pd_{76}$. The top isotopologue peaks were generated by NEUTRONSTAR using α = 1.01. NEUTRONSTAR took 0.0210772 s to generate the peaks and they cover a cumulative probability of 0.408561.

Influence of α on Runtime. Table 3 shows the influence of α on runtime.

https://dx.doi.org/10.1021/acs.analchem.0c01670
Anal. Chem. 2020, 92, 10613−10619

Table 1. Runtimes of ISOSPEC and NEUTRONSTAR for Six Compounds, All with $\alpha = 1.01$[a]

compound	p	k	ISOSPEC	NEUTRONSTAR(k)	NEUTRONSTAR(p)
Averagine	0.1	698 668	0.191634	0.0337640	0.0422050
	0.3	3 958 459	0.391008	0.143400	0.168099
	0.5	11 442 227	2.65370	0.374090	0.434118
	0.7	30 264 581	2.58011	0.943990	1.10575
	0.9	110 437 547	18.7317	3.38632	3.89124
	0.99	541 404 815	31.4017	14.1528	17.8143
	0.999	1 524 764 796	80.6248	40.6456	55.2786
Ostalloy	0.1	6 719 141	9.21131	0.154020	0.184007
	0.3	65 366 950	11.8971	1.33645	1.56795
	0.5	279 712 408	8.52057	5.54898	6.45966
	0.7	1 084 729 667	153.463	21.7491	25.1355
	0.9	6 502 472 315	Segfaulted	171.363	195.902
palladium	0.1	9134	0.255972	0.00396640	0.00289220
alloy Pgc	0.3	52 855	0.253937	0.0114332	0.0115626
	0.5	162 857	1.470858	0.0234636	0.0230856
	0.7	473 917	1.478908	0.0433196	0.0451654
	0.9	2 074 266	4.250200	0.139940	0.140854
	0.99	13 466 926	8.929702	0.638875	0.675590
	0.999	47 409 787	16.13902	1.69085	2.06479
Xe_{50}	0.1	2510	0.799876	0.00353960	0.00324580
	0.3	12 909	0.800633	0.0117036	0.0138288
	0.5	35 243	0.801047	0.0298446	0.0292678
	0.7	91 046	8.40066	0.0707594	0.0670961
	0.9	332 449	8.38711	0.234639	0.237602
	0.99	1 564 230	35.9428	1.15119	1.17131
	0.999	4 208 537	104.811	3.15043	3.55441
$Sn_{20}Xe_{20}Nd_{20}Dy_{20}$	1e-12	1	0.000365	0.000179800	0.000204799
	1×10^{-11}	5	Segfaulted	0.000208200	0.000280400
	1×10^{-10}	50		0.000302600	0.000345199
	1×10^{-9}	554		0.000841600	0.00102380
	1×10^{-8}	6156		0.00344500	0.00338180
	1×10^{-7}	72 222		0.0114870	0.0124566
	1×10^{-6}	961 382		0.0470376	0.0486468
	1×10^{-5}	13 415 245		0.338715	0.374403
	1×10^{-4}	221 970 398		4.68355	5.47419
Titin	0.1	461 921 393	138.025	14.5096	15.9033
	0.2	1 345 272 073	141.805	40.9665	45.0096
	0.3	2 776 599 465	190.003	97.0903	116.240
	0.4	5 027 827 340	207.433	165.237	193.948

[a]There are two runtimes for NEUTRONSTAR, one using the p parameter and one using the k parameter. The first compound is averagine with $n = 5000$, $C_{24692}H_{38792}N_{6788}O_{7386}S_{208}$. The second is Ostalloy, $Bi_{50}Cd_{12}Pb_{25}Sn_{12}$, also known as Lipowit'z metal. The third is palladium alloy PGC, $Au_2Ca_{10}Ga_{10}Pd_{76}$, a dental amalgam. The fourth is just 50 copies of xenon. The fifth is the molecule $Sn_{20}Xe_{20}Nd_{20}Dy_{20}$ and the sixth is Titin protein, $C_{169719}H_{270466}N_{45688}O_{52238}S_{911}$, the largest protein in the human body. Both algorithms ran out of memory on Titin with $p = 0.5$.

DISCUSSION

As seen in Table 1, for all tested compounds, NEUTRONSTAR is faster than ISOSPEC. For organic compounds of moderate size (and therefore similar compounds whose elements have a small amount of isotopes), NEUTRONSTAR and ISOSPEC are of similar speed for smaller p; as p grows, NEUTRONSTAR starts to be consistently faster. On large organic molecules, such as Titin protein, NEUTRONSTAR gives a significant speed advantage over ISOSPEC of between 1.25543× and 9.51267×. For compounds whose elements have significantly more isotopes (e.g., the two compounds, Xe_{50}, and $Sn_{20}Xe_{20}Nd_{20}Dy_{20}$) NEUTRONSTAR shows a significant advantage over ISOSPEC. The most drastic result is the difference in runtimes for the compound $Sn_{20}Xe_{20}Nd_{20}Dy_{20}$, where ISOSPEC segfaults (on a machine with 256GB) and NEUTRONSTAR takes just 0.00028 s with $p = 1 \times 10^{-11}$. This is most likely due to ISOSPEC generating far too many peaks before the final trimming. If this is the case, then it would have had to produce more than 5 000 000 000 peaks (since Titin produced that many on the same machine) in order to select down to the proper five. This molecule highlights the stark differences between the two algorithms when it comes to handling molecules whose elements have many isotopes something that will become increasingly important for large compounds because, given enough copies of one element and sufficient resolution of machine, even trace isotopes may appear.

It may be possible to gain a further advantage by tuning α according the compound and number of peaks requested (Table 3) or even using heterogeneous α throughout the tree. Furthermore, where ISOSPEC requires the use of the Gaußian approximation, NEUTRONSTAR is a purely combinatorial approach. Efficient selection on $X_1 + X_2 + \ldots + X_m$, may be useful

https://dx.doi.org/10.1021/acs.analchem.0c01670
Anal. Chem. 2020, 92, 10613−10619

Table 2. Memory Usage of ISOSPEC and NEUTRONSTAR (with $\alpha = 1.05$) on Five Compounds[a]

compound	p	k	ISOSPEC	NEUTRONSTAR
Averagine	0.1	698 668	479.8 KiB	39.976 KiB
	0.3	3 958 459	479.8 KiB	83.7279 KiB
	0.5	11 442 227	3.17 MiB	206.233 KiB
	0.7	30 264 581	3.17 MiB	507.072 KiB
	0.9	110 437 548	12.4 MiB	3.48 MiB
	0.99	541 404 826	49.2 MiB	8.56 MiB
	0.999	1 364 841 014	82.0 MiB	21.4 MiB
Ostalloy	0.1	6 719 141	12.4 MiB	124.364 KiB
	0.3	65 366 950	12.4 MiB	1.06 MiB
	0.5	279 712 408	12.4 MiB	4.46 MiB
	0.7	1 084 729 667	82.0 MiB	5.35 MiB
	0.9	6 502 472 315	Segfautled	102.38 MiB
Palladium	0.1	9134	9.7 MiB	941.2 KiB
alloy PGC	0.3	52 855	9.7 MiB	4.2 MiB
	0.5	162 857	63.5 MiB	11.2 MiB
	0.7	473 917	63.5 MiB	29.5 MiB
	0.9	2 074 266	419.7 MiB	105.6 MiB
	0.99	13 466 926	831.8 MiB	552.0 MiB
	0.999	47 409 787	3.1 GiB	1.7 GiB
Xe_{50}	0.1	2510	30.7 MiB	1.4 MiB
	0.3	12 909	30.7 MiB	5.4 MiB
	0.5	35 243	30.7 MiB	21.6 MiB
	0.7	91 046	166.0 MiB	43.3 MiB
	0.9	332 449	166.0 MiB	86.9 MiB
	0.99	1 564 230	480.0 MiB	352.3 MiB
$Sn_{20}Xe_{20}Nd_{20}Dy_{20}$	1×10^{-12}	1	95.93 KiB	15.94 KiB
	1×10^{-11}	5	Segfaulted	15.94 KiB
	1×10^{-10}	50		15.94 KiB
	1×10^{-9}	554		16.07 KiB
	1×10^{-8}	6156		16.40 KiB
	1×10^{-7}	72 222		18.82 KiB
	1×10^{-6}	961 382		47.11 KiB
	1×10^{-5}	13 415 245		436.6 KiB
	1×10^{-4}	221 970 398		6.965 MiB

[a]Memory usage for first five compounds from Table 1. VALGRIND, the program used to acquire the memory usage, was, for reasons unknown, unable to run on Titin protein.

Table 3. Relationship between α and the runtime for $\alpha \in [1,2]$ on compound $Au_2Ca_{10}Ga_{10}Pd_{76}$ with $k = 100\,000$[a]

compound	α	Time(s)
$Au_2Ca_{10}Ga_{10}Pd_{76}$	1.0	0.0546169
	1.10	0.0218086
	1.20	0.0259033
	1.30	0.0348981
	1.40	0.037129
	1.50	0.0457096
	1.60	0.0528224
	1.70	0.10543
	1.80	0.103198
	1.90	0.0810971
	2.00	0.0977637

[a]The time reported is the average over 10 iterations and all times reported are only from NEUTRONSTAR. If $\alpha = 1$ then sizes of the layers in the LOHs do not increase and so a layer-ordering with $\alpha = 1$ is the same as sorting. The runtime is at its worst when the $\alpha = 1.8$ and best when $\alpha = 1.1$.

for other optimization problems, such as special cases of integer linear programs and set-cover problems (which can be applied to protein inference).

The memory efficiency of NEUTRONSTAR compared to ISOSPEC, as seen in Table 2, is likely due to the different proposal schemes. ISOSPEC will propose all neighbors of the popped isotopologues if they are not in the set that will form a tensor with dimension equal to the number of subisotopologues. In NEUTRONSTAR this space is drastically reduced due to using only pairwise selection.

Since both algorithms solve the problem exactly, we see a significant agreement in both mass, typically 15 significant figures, and log-abundance, typically to 10 significant figures. The difference between the log-abundance is likely from ISOSPEC using Stirling approximation or their calculation of log-abundance from scratch whereas NEUTRONSTAR calculates the log-abundance at each node in the tree.

Currently, NEUTRONSTAR is not configured to report the isotopic makeup of the resulting isotopologues. This could be achieved by keeping track of the index tuple as an isotopologue is created while climbing up the binary tree; however, this would result in a performance reduction and remove one of the more novel aspects of this algorithm.

While NEUTRONSTAR can accept p as a parameter, using k is desirable because the number of peaks generated according to p is difficult to estimate (e.g., for the compound $C_{27}H_{35}N_6O_8P_1$

https://dx.doi.org/10.1021/acs.analchem.0c01670
Anal. Chem. 2020, 92, 10613−10619

where $p = 0.999\,999\,999\,999\,9$ gives only 669 of the possible 317 520 peaks).

In the future, it may be possible to use LOHs inside the subisotopologue generators similar to the $X + Y$ selection nodes. This could be a considerable speed-up because it avoids the $\Omega(n\log(n))$ bounds created by sorting the subisotopologues.

CONCLUSION

NEUTRONSTAR is a fast and accurate algorithm for calculating the top k abundance isotopologue peaks. NEUTRONSTAR relies on the layer-ordered heap data structure to perform optimal select on two partial isotopologues. Multiple pairwise selections combined in a binary tree allow the NEUTRONSTAR to perform efficient selection to produce full isotopologues.

Compared to the current state-of-the-art algorithms, it is significantly faster, growing better as the compounds grow in size. For compounds whose element have many isotopes, NEUTRONSTAR is by far the best option: it finds the collection of peaks on the compound $Sn_{20}Xe_{20}Nd_{20}Dy_{20}$ in less than 0.0003 s using less than 16 KiB of memory, while ISOSPEC exceeded the 256 GB of RAM limit.

AUTHOR INFORMATION

Corresponding Author

Oliver Serang − *Department of Computer Science, University of Montana, Missoula, Montana, United States;* orcid.org/0000-0003-1245-7051; Email: Oliver.Serang@umontana.edu

Authors

Patrick Kreitzberg − *Department of Mathematics, University of Montana, Missoula, Montana, United States*

Jake Pennington − *Department of Mathematics, University of Montana, Missoula, Montana, United States*

Kyle Lucke − *Department of Computer Science, University of Montana, Missoula, Montana, United States*

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.analchem.0c01670

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

This work was supported by Grant No. 1845465 from the National Science Foundation. The NEUTRONSTAR algorithm, implemented in C++17, can be found freely at https://bitbucket.org/orserang/neutronstar/.

REFERENCES

(1) Blum, M.; Floyd, R. W.; Pratt, V. R.; Rivest, R. L.; Tarjan, R. E. *Journal of Computer and System Sciences* **1973**, *7* (4), 448−461.

(2) Chazelle, B. *J. Assoc. Comput. Mach.* **2000**, *47* (6), 1012−1027.

(3) Ipsen, A. *Anal. Chem.* **2014**, *86* (11), 5316−5322.

(4) Ji, H.; Xu, Y.; Lu, H.; Zhang, Z. *Anal. Chem.* **2019**, *91* (9), 5629−5637.

(5) Kaplan, H.; Kozma, L.; Zamir, O.; Zwick, U. *Symposium on Simplicity in Algorithms* **2019**, 5-1−5-21.

(6) Kreitzberg, P.; Lucke, K.; Serang, O. Selection on $X_1 + X_2 + \cdots + X_m$ with layer-ordered heaps. 2019, not yet submitted.

(7) Łacki, M. K.; Startek, M.Ł; Valkenborg, D.; Gambin, A. *Anal. Chem.* **2017**, *89* (6), 3272−3277.

(8) Loos, M.; Gerber, C.; Corona, F.; Hollender, J.; Singer, H. *Anal. Chem.* **2015**, *87* (11), 5738−5744.

(9) Ruff, M.; Mueller, M.; Loos, M.; Singer, H. P. *Water Res.* **2015**, *87*, 145−154.

(10) Sadygov, R. G. *J. Proteome Res.* **2018**, *17* (1), 751−758.

(11) Senko, M. W.; Beu, S. C.; McLaffertycor, F. W. *J. Am. Soc. Mass Spectrom.* **1995**, *6*, 229.

(12) Serang, O. Optimal selection on X + Y simplified with layer-ordered heaps. 2020.

(13) Singer, H. P.; Wössner, A. E.; McArdell, C. S.; Fenner, K. *Environ. Sci. Technol.* **2016**, *50* (13), 6698−6707.

(14) Sturm, M.; Bertsch, A.; Gropl, C.; Hildebrandt, A.; Hussong, R.; Lange, E.; Pfeifer, N.; Schulz-Trieglaff, O.; Zerck, A.; Reinert, K.; Kohlbacher, O.; et al. *BMC Bioinf.* **2008**, *9* (1), 163.

(15) Wang, Z.; Chen, X.; Ren, J.; Hu, G. *Int. J. Mass Spectrom.* **2019**, *443*, 70−76.