Fast Elastic Peak Detection for Mass Spectrometry Data Mining

Xin Zhang, Dennis Shasha, Yang Song, *Student Member*, *IEEE*, and Jason T.L. Wang, *Member*, *IEEE*

Abstract—We study a data mining problem concerning the elastic peak detection in 2D liquid chromatography-mass spectrometry (LC-MS) data. These data can be modeled as time series, in which the X-axis represents time points and the Y-axis represents intensity values. A peak occurs in a set of 2D LC-MS data when the sum of the intensity values in a sliding time window exceeds a user-determined threshold. The elastic peak detection problem is to locate all peaks across multiple window sizes of interest in the data set. We propose a new data structure, called a Shifted Aggregation Tree or AggTree for short, and use the data structure to find the different peaks. Our method, called PeakID, solves the elastic peak detection problem in 2D LC-MS data yielding neither false positives nor false negatives. The method works by first constructing an AggTree in a bottom-up manner from the given data set, and then searching the AggTree for the peaks in a top-down manner. We describe a state-space algorithm for finding the topology and structure of an efficient AggTree to be used by PeakID. Our experimental results demonstrate the superiority of the proposed method over other methods on both synthetic and real-world data.

Index Terms—Knowledge discovery from LC-MS data, time series data mining, bioinformatics, computational proteomics, algorithms and data structures.

1 Introduction

1.1 Motivation

Recently, mass spectrometry data mining has drawn much attention in the computational proteomics community [4], [11], [29], [32]. Typical mining processes include peak detection [9], [17], [22], [25], spectrum alignment [31], data correlation [15], biomarker discovery [14], [20], among others. In this paper, we propose a new approach, called PeakID, for identifying peaks in liquid chromatography-mass spectrometry (LC-MS) data. LC-MS data have three dimensions, namely retention time, mass-to-charge ratio (m/z), and intensity [1], [11]. One important step in mass spectrometry data mining is to detect peaks in the three-dimensional (3D) LC-MS data [1], [4], [9], [11], [25], [28]. Due to the complex nature of the 3D data with different peak shapes, this is known to be a difficult problem [9], [11], [13], [25]. One approach is to convert the 3D data to lower dimensional data as explained below.

For each given m/z value, 3D LC-MS data can be expressed as a two-dimensional (2D) map in which the X-axis represents time points and the Y-axis represents intensities. A peak in the 2D map, M_t, is a collection of intensity values occurring within a certain time window where the sum of the intensity values is greater than or equal to a user-specified threshold. Suppose a peak occurs in a time window $[t_l, t_r]$ in which the largest intensity value occurs at t_{top} in M_t. For the given t_{top} value, one can check the corresponding 2D map, M_{mz}, whose X-axis has m/z values and Y-axis has intensities. Find a small range $[mz_l, mz_r]$ that surrounds each mz_{pos} whose intensity is a sufficiently large positive value in M_{mz}. Then, the intensities in the cube constructed based on $[t_l, t_r]$ and $[mz_l, mz_r]$ form a 3D peak. Thus, by detecting peaks in the 2D map M_t whose X-axis has time points and Y-axis has intensities, one is able to derive peaks in the 3D LC-MS data.[1] Finding peaks in the 2D LC-MS data M_t in an efficient way is the subject of this paper.

If the size of a sliding time window in which a 2D peak occurs is known a priori, then peak detection can easily be done in linear time by summing up the intensity values within each time window of the known size. However, in practice, the window size is unknown a priori. The size itself may be an interesting subject to be discovered. Also, in many cases, it is required to detect peaks across a variety of window sizes [1], [11].

1.2 Problem Formulation

The *elastic peak detection* problem is to detect peaks across multiple window sizes. Formally, given a set of nonnegative intensity values $x_1, x_2, \ldots, x_N$, a set $\mathcal{W}$ of window sizes $w_1, w_2, \ldots, w_m$, where $w_i < w_j$, $1 \le i < j \le m$, and a threshold associated with each window size, $f(w_j)$, $j = 1, 2, \ldots, m$, the elastic peak detection is the problem of finding all pairs (t, w) such that t is a time point, w is a window size in $\mathcal{W}$ and

- *X. Zhang and D. Shasha are with the Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, NY 10012. E-mail: {xinzhang, shasha}@cs.nyu.edu.*
- *Y. Song and J.T.L. Wang are with the Department of Computer Science, New Jersey Institute of Technology, University Heights, Newark, NJ 07102. E-mail: {ys54, wangj}@njit.edu.*

Manuscript received 8 Aug. 2009; revised 6 Jan. 2010; accepted 9 Aug. 2010; published online 18 Nov. 2010.
Recommended for acceptance by Y. Chen.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2009-08-0596.
Digital Object Identifier no. 10.1109/TKDE.2010.238.

1. As m/z represents the mass-to-charge ratio, the following scenario might happen [2]: a peptide seen as a singly-charged ion (e.g., for a m/z of approximately 1,800) can also be observed as a doubly charged ion (e.g., for a m/z of approximately 900) or a triply charged ion (e.g., for a m/z of approximately 600). This would lead to different mz_l, mz_r values and hence different 3D peaks, though they may present the same thing. Under this circumstance, our scheme can be used as a filter, and further analysis of the 3D LC-MS data would be needed.

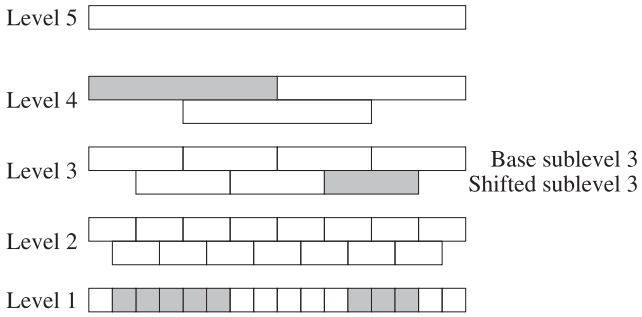


Fig. 1. An example of Shifted Binary Trees. Each cell at level 1 is a leaf node; each leaf node corresponds to a time point in the input 2D LC-MS data. Each cell at base sublevel i (shifted sublevel i, respectively) represents a node at base sublevel i (shifted sublevel i, respectively), and corresponds to or contains 2^{i-1} time points in the input data. In the figure, the first highlighted sequence at level 1 is contained in the highlighted node at base sublevel 4; the second highlighted sequence at level 1 is contained in the highlighted node at shifted sublevel 3.

$$\sum_{p=t}^{t+w-1} x_p \geq f(w).$$

A brute-force algorithm is to check each window size of interest one at a time. To detect peaks across m window sizes in a sequence of intensity values over N time points, the brute-force algorithm requires $O(mN)$ time.

In [23], [40], we showed that a simple data structure called a *Shifted Binary Tree*, abbreviated as a *BinaryTree*, could be the basis of a filter that can be used to detect all peaks in time independent of the number of window sizes when the probability of peaks is low. This tree is a hierarchical data structure, inspired by the Haar wavelet tree [5]. Each leaf node at level 1 of the Shifted Binary Tree corresponds to a time point in the input data; a node at level 2 aggregates two adjacent nodes at level 1. In general, a node v at level i aggregates two nodes v_1, v_2 at level $i-1$; v is the parent of v_1, v_2 and v_1, v_2 are the children of v. Thus, v contains or corresponds to 2^{i-1} time points. There are $\log_2^N + 1$ levels in the Shifted Binary Tree where N is the number of time points in the input 2D LC-MS data. Except level 1 and the top level, each level i has two sublevels, namely base sublevel i and shifted sublevel i. Each node at shifted sublevel i is shifted by 2^{i-2} time points with respect to base sublevel i. Fig. 1 shows an example of Shifted Binary Trees.

The overlap between the base sublevels and the shifted sublevels guarantees that every time window of size w, $0 < w \leq 2^{i-2} + 1$, is contained in either a node at base sublevel i or a node at shifted sublevel i, or both. In [23], [36], [40], we exploited this property, developing an algorithm that uses Shifted Binary Trees to search for peaks in time sequence data. The algorithm works well when there are few peaks, but performs poorly if there are many near peaks [23].

1.3 Contributions

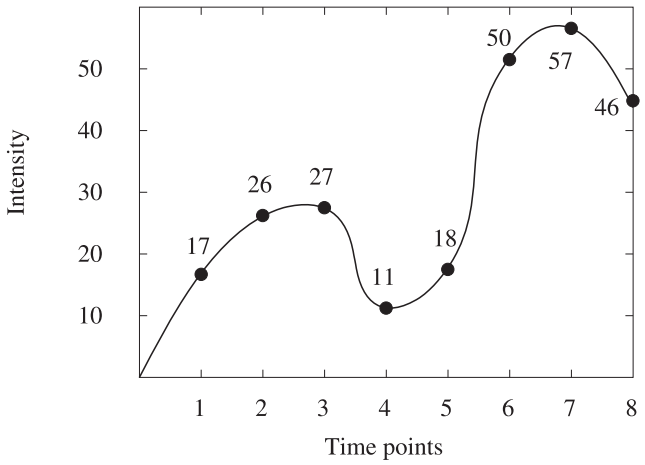
In [36], we introduced a new data structure called a *Shifted Aggregation Tree*, abbreviated as an *AggTree*, which improves the performance of a Shifted Binary Tree, and sketched the use of AggTrees in a general setting of time series mining. Here, we extend the work in [36] by presenting 1) the algorithmic details and theoretical foundation of AggTrees, 2) the PeakID method that adapts AggTrees to elastic peak detection in 2D LC-MS data, and 3) experimental results showing the superiority of PeakID over other methods.

The rest of the paper is organized as follows: Section 2 surveys related work and contrasts our approach with existing techniques. Section 3 describes the PeakID method in detail, introducing the concept of an Aggregation Pyramid that acts as a host data structure into which a Shifted Aggregation Tree can be embedded, and explaining how to find an efficient Shifted Aggregation Tree given input data. Section 4 evaluates the performance of PeakID and presents experimental results on both synthetic and real-world data. Section 5 concludes the paper.

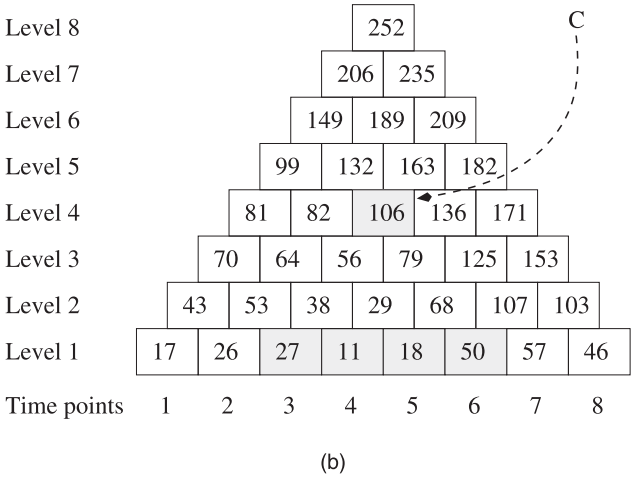
2 Related Work

There are two groups of work that are closely related to ours. The first group is concerned with 3D peak detection in mass spectrometry data. Most 3D peak detection algorithms are based on either statistical distributions or a variety of smoothing functions [13], [34]. To reduce the number of false positives (i.e., those that are nonpeaks but are detected as peaks), these algorithms often assume a minimum peak width. The algorithms focus on dealing with 2D data where the X-axis has m/z values and the Y-axis has intensity values [6], [9], [10]. An alternative approach, as described in Section 1, is to examine 2D data where the X-axis has time points and the Y-axis has intensity values. Stolt et al. [25], for example, developed a second-order peak detection algorithm capable of finding peaks of different widths in such 2D data. The authors set the minimum peak width to 3. One disadvantage of Stolt et al.'s algorithm is that it may produce false negatives (i.e., those that are real peaks but are predicted as nonpeaks). To reduce the number of false negatives, the brute-force algorithm checking different window sizes could be used. In contrast to Stolt et al.'s work, we develop a new data structure for identifying all peaks quickly without using the brute-force approach.

The second group of related work is concerned with burst modeling and detection in time series. Wang et al. [30] used a one-parameter model, b-model, to model the bursty behavior in self-similar time series and to synthesize realistic trace data. This type of time series occurs in a large number of real world applications, such as Ethernet, file systems, web, video and disk traffic. Kleinberg [19] studied the bursty and hierarchical structure in temporal text streams, with a focus on finding how high frequency words change over time. Vlachos et al. [27] mined the bursty behavior in the query logs of the MSN search engine. They used moving averages to detect time regions having high numbers of queries. Only two window sizes were considered, short term and long term. The detected bursts were further compacted and stored in a database to support burst-based queries. Other methods for finding surprising and periodic patterns in time series have also been developed [12], [18], [33]. The 2D LC-MS data we deal with here are time series in nature. However, in contrast to the above time series mining methods, our work mainly focuses on detecting bursts (peaks) across multiple window sizes.



(a)



(b)

Fig. 2. An example of an 8-level Aggregation Pyramid. (a) An example of hypothetical 2D LC-MS data. (b) The Aggregation Pyramid built over the 2D LC-MS data in (a).

3 The PeakID Method

3.1 The Aggregation Pyramid

An *Aggregation Pyramid* is an N-level isosceles triangular-shaped data structure built over the input 2D LC-MS data with N time points, satisfying the following properties:

- Level 1 has N cells where each cell stores the intensity value associated with each time point in the input 2D LC-MS data.
- Level 2 has $N-1$ cells where the first cell stores the sum of the first two intensity values (i.e., the intensity value at time point 1 and the intensity value at time point 2) in the 2D LC-MS data; the second cell stores the sum of the second two intensity values (i.e., the intensity value at time point 2 and the intensity value at time point 3), and so on.
- Level h has $N-h+1$ cells where the ith cell, c, stores the sum of the h consecutive intensity values starting at time point i and ending at time point $i+h-1$ in the 2D LC-MS data. The time window starting at time point i and ending at time point $i+h-1$ is called the *shadow window*, or simply the *shadow*, of cell c. When the context is clear, we also refer to the set of the h consecutive cells at level 1 starting with the ith cell as the shadow window, or the shadow, of cell c.
- The top level has one cell only, storing the sum of all intensity values in the input 2D LC-MS data.

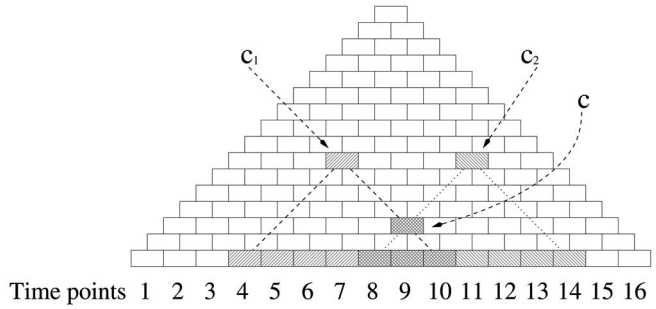


Fig. 3. Illustration of the overlap of two cells in an example Aggregation Pyramid.

Notice that each time window of size w is the shadow of some cell at level w. Conversely, each cell at level w has a shadow window of size w; the cell stores the sum of the intensity values within its shadow window. Fig. 2 shows an example of an 8-level Aggregation Pyramid. In Fig. 2b, for example, the highlighted cell, c, at level 4 stores a value of 106, which is the sum of the intensity values 27, 11, 18, and 50 at level 1 that are within the shadow window of the cell c.

By construction, an Aggregation Pyramid has the following properties:

- All the shadows of the cells along the 45 degree diagonal have the same starting time point. All the shadows of the cells along the 135 degree diagonal have the same ending time point.
- A cell at level h with the shadow ending at time point t is denoted as $cell(h,t)$, which stores the sum of the intensity values in $cell(1,t-h+1)$ to $cell(1,t)$. When the context is clear, we also use $cell(h,t)$ to represent the sum value stored in this cell.
- The shadow of any cell c in the subpyramid rooted at cell r is a subset of the shadow of cell r. We say c is *shaded* by r. By monotonicity, the value in cell c is guaranteed to be less than or equal to the value in cell r.
- The *overlap* of two cells c_1 and c_2 in that order at the same level is the cell c at the intersection of the 135 degree diagonal touching cell c_1 and the 45 degree diagonal touching cell c_2 (see Fig. 3). The shadow of the cell c is the intersection of the shadow of c_1 and the shadow of c_2. For example, in Fig. 3, the shadow of cell c_1 contains time points $4,5,\ldots,10$, the shadow of cell c_2 contains time points $8,9,\ldots,14$, and the shadow of the overlap c contains time points 8, 9 and 10.

The values stored in the cells of the Aggregation Pyramid can be calculated in a bottom-up manner using (1) below: for $1<h,t\leq N$,

$$cell(h,t)=cell(h-1,t-1)+cell(1,t). \tag{1}$$

If $cell(h,t)$ exceeds the threshold $f(h)$ for the time window size h, then there exists a peak starting at time point $t-h+1$ and ending at time point t.

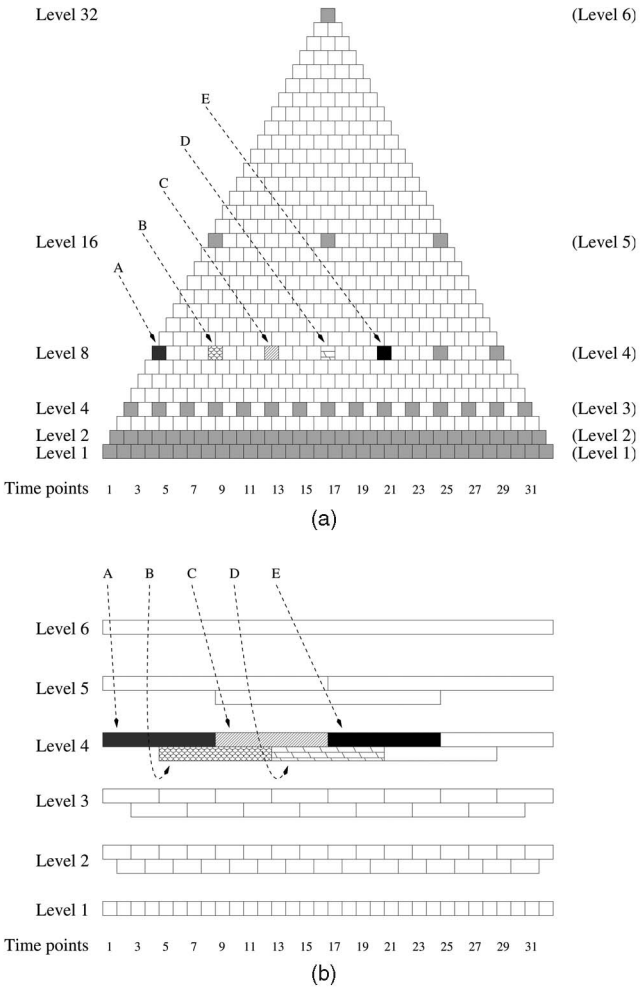


Fig. 4. Illustration of the correspondence between the Aggregation Pyramid in (a) and the Shifted Binary Tree in (b). The highlighted cells at level 1, 2, 4, 8, 16, 32, respectively, in the Aggregation Pyramid in (a) correspond to the nodes at level 1, 2, 3, 4, 5, 6, respectively, in the BinaryTree in (b).

3.2 Embedding a BinaryTree into an Aggregation Pyramid

Recall that in a Shifted Binary Tree, each node at level 1 corresponds to a time point in the input 2D LC-MS data, and each node at level i corresponds to or contains 2^{i-1} time points in the input data. Observe that each node in a Shifted Binary Tree with N time points corresponds to a cell in an Aggregation Pyramid with the same number of time points. Fig. 4 shows the correspondence between the nodes in a Shifted Binary Tree with 32 time points and the cells in an Aggregation Pyramid with 32 time points. Each cell in the Aggregation Pyramid that corresponds to a node in the Shifted Binary Tree is highlighted in Fig. 4a. Specifically, the cells labeled A, B, C, D, and E, respectively, in the Aggregation Pyramid in Fig. 4a correspond to the first five nodes, labeled A, B, C, D, and E, respectively, at level 4 of the Shifted Binary Tree in Fig. 4b. Notice that level i in the Shifted Binary Tree corresponds to level 2^{i-1} in the Aggregation Pyramid. This correspondence shows how to embed the Shifted Binary Tree in Fig. 4b into the Aggregation Pyramid in Fig. 4a.

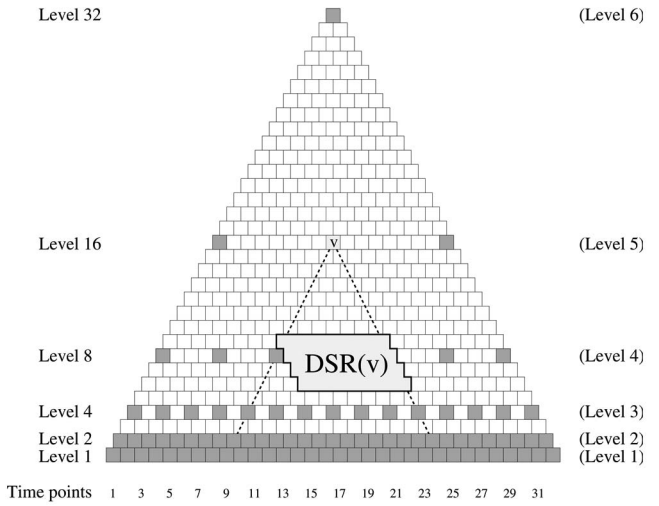


Fig. 5. Illustration of the detailed search region $DSR(v)$ in the Aggregation Pyramid in Fig. 4a for a node v at level 5 of the BinaryTree in Fig. 4b. The level numbers on the left represent the level numbers of the Aggregation Pyramid in Fig. 4a, and the level numbers in the parentheses on the right represent the level numbers of the BinaryTree in Fig. 4b. The cells of the Aggregation Pyramid that correspond to the nodes of the BinaryTree are highlighted.

An important property of the Shifted Binary Tree is that any given time window of size w, $w \leq 2^{i-2}+1$, is contained in at least one of the nodes at level i of the BinaryTree. By induction, any time window of size w, $w \leq 2^{i-3}+1$, is contained in at least one of the nodes at level $i-1$ of the BinaryTree. After the BinaryTree is constructed, we search the BinaryTree for peaks in a top-down manner. If the value stored in a node $v = cell(2^{i-1}, t)$ at level i of the BinaryTree exceeds the threshold $f(2^{i-3}+2)$ associated with the time window size $2^{i-3}+2$, then an alarm is raised, indicating the possible occurrence of a peak. A detailed search has to be performed to check the cells of the Aggregation Pyramid whose shadow sizes are in the range $[2^{i-3}+2,\ 2^{i-2}+1]$.

Note that we need to search and check only the cells in the Aggregation Pyramid whose shadows end after time point $t-2^{i-2}$, because the cells whose shadows end at or before time point $t-2^{i-2}$ are shaded by one of v's preceding nodes at level i of the BinaryTree. We refer to the region in which the search is performed as the *detailed search region* of v, denoted $DSR(v)$. The detailed search region consists of cells from level $2^{i-3}+2$ to level $2^{i-2}+1$ in the Aggregation Pyramid, where the shadows of these cells end at time points in $[t-2^{i-2}+1, t]$. The purpose of checking the cells in $DSR(v)$ is to find peaks occurring in time windows whose sizes are in the range $[2^{i-3}+2,\ 2^{i-2}+1]$ and that end at time points in $[t-2^{i-2}+1, t]$. Notice that, if v is the leftmost node at level i, since no node precedes v, $DSR(v)$ contains cells whose shadows end at time points in $[1, 2^{i-1}]$. It was proved [23] that searching the cells in $DSR(v)$ guarantees that all peaks are detected. Fig. 5 illustrates the detailed search region $DSR(v)$ for a node v in the BinaryTree in Fig. 4b.

TABLE 1
Comparison between BinaryTrees and AggTrees

BinaryTree	AggTree
$d_i = 2$	$d_i \geq 2$
$s_i = 2 \times s_{i-1}$	$s_i = k \times s_{i-1}$, $k \geq 1$
$o_i = a_{i-1}$	$o_i \geq a_{i-1}$

3.3 Generalizing a BinaryTree to an AggTree

A detailed search in $DSR(v)$ may turn out to be fruitless (i.e., no peak is found in $DSR(v)$). It has been observed that [23], [36], [40]

1. When peaks are rare but not very rare, the number of fruitless detailed searches grows, suggesting that we may want more levels than a Shifted Binary Tree provides.
2. Conversely, when peaks are exceedingly rare we may need fewer levels than a Shifted Binary Tree provides.

In other words, we want a data structure that adapts to the input data. For this reason, we generalize Shifted Binary Trees to Shifted Aggregation Trees. Like a BinaryTree, an AggTree is a hierarchical data structure defined on a subset of the cells of an Aggregation Pyramid. It has several levels, each of which contains several nodes. The nodes at level 1 are in one-to-one correspondence with the time points in the input 2D LC-MS data. The value stored in a node at level i is obtained by aggregating the values stored in some nodes below level i. The shadows of two neighboring nodes at the same level overlap.

A Shifted Aggregation Tree differs from a Shifted Binary Tree in two ways:

1. *The parent-child structure*. This defines the topological relationship between a node and its children, i.e., how many children the node has and their placements.
2. *The shifting pattern*. This defines how many time points apart there are between two neighboring nodes v_1 and v_2 at the same level i. Formally, letting the shadow of v_1 end at time point t_1 and the shadow of v_2 end at time point t_2, where $t_1 < t_2$, we call $(t_2 - t_1)$ the *shift* between v_1 and v_2, or the shift of level i.

In a BinaryTree, the parent-child structure for each node is always the same: one node at level i aggregates two nodes at level $i-1$. The shifting pattern is also fixed: the shadows of two neighboring nodes in the same level always half-overlap. In an AggTree, a node could have three children and be two time points away from its preceding neighbor, or could have 64 children and be 128 time points away from its preceding neighbor. We define the shadow size of level i, denoted a_i, to be the size of the shadow of a node at level i. Define the overlapping shadow size of level i, denoted o_i, to be the size of the intersection of the shadows of two neighboring nodes at level i. Define the degree of level i, denoted d_i, to be the degree of a node at level i, i.e., the number of children the node has. Let s_i denote the shift of level i. Table 1 gives a side-by-side comparison between AggTrees and BinaryTrees. Clearly, BinaryTrees are a special case of AggTrees.

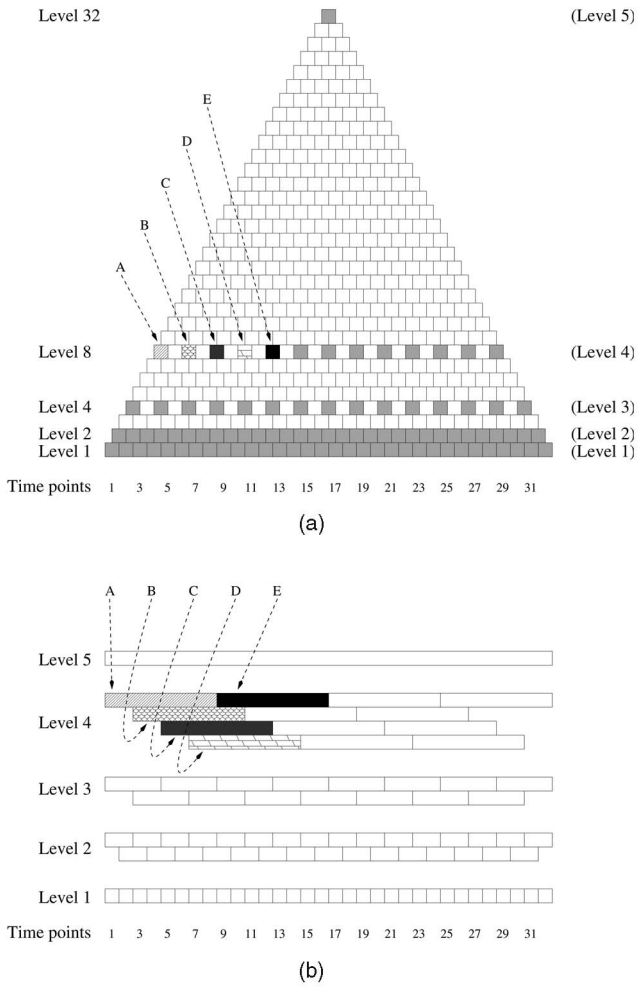


Fig. 6. Illustration of the correspondence between the Aggregation Pyramid in (a) and the Shifted Aggregation Tree in (b). In (a), the level numbers on the left represent the level numbers of the Aggregation Pyramid, and the level numbers in the parentheses on the right represent the level numbers of the AggTree in (b). The cells of the Aggregation Pyramid that correspond to the nodes of the AggTree are highlighted in (a).

Fig. 6 shows an example of a Shifted Aggregation Tree with 32 time points; the figure also shows how the AggTree is embedded into an Aggregation Pyramid with 32 time points. In the AggTree, the shift of level 2 is 1; the shift of level 3 is the same as the shift of level 4, which equals 2. Each node at level 2, 3, and 4, respectively, has two children, whereas the node at level 5 has four children. The overlapping shadow size of level 2 is 1, which equals the shadow size of level 1. The overlapping shadow size of level 3 is 2, which equals the shadow size of level 2. The overlapping shadow size of level 4 is 6, which is larger than the shadow size of level 3. The cells labeled A, B, C, D, and E, respectively, in the Aggregation Pyramid in Fig. 6a correspond to the first five nodes, labeled A, B, C, D, and E, respectively, at level 4 of the Shifted Aggregation Tree in Fig. 6b. This example shows the difference between an AggTree and a BinaryTree (cf. Table 1 and Fig. 4).

Recall that in the elastic peak detection problem whose goal is to find peaks across multiple window sizes, we are given the 2D LC-MS data, $\mathrm{LM}[N]$, with N time points, a set of nonnegative intensity values $x_1, x_2, \ldots, x_N$, a set $\mathcal{W}$ of

Procedure: BuildTree(LM$[N]$, s_i, a_i, d_i)
Input: The 2D LC-MS data LM$[N]$, shift s_i, shadow size a_i and degree d_i.
Output: The Shifted Aggregation Tree AggTree$[I][N]$.
/* Initialize the first level of the AggTree. */
1. **for** j = 1 **to** N **do**
2. AggTree[1][j] $\leftarrow$ LM[j];
/* Construct the AggTree in a bottom-up manner. */
3. **for** i = 2 **to** I **do**
4. **for** j = 1 **to** $\lfloor \frac{N-a_i}{s_i} \rfloor + 1$ **do**
5. calculate AggTree[i][j] using Equation (2);
6. **return** AggTree$[I][N]$;

Fig. 7. Algorithm for constructing the AggTree from the 2D LC-MS data $\mathrm{LM}[N]$.

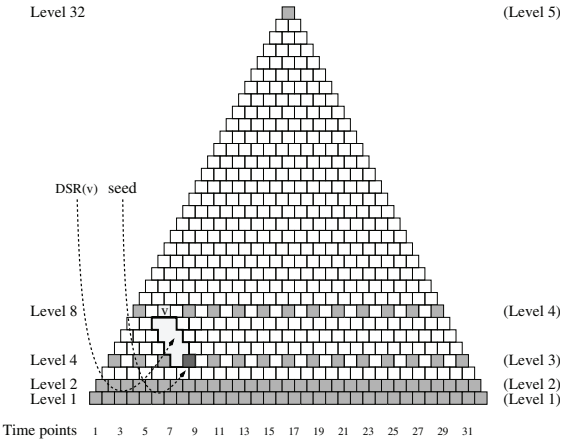


Fig. 8. Illustration of the detailed search region $DSR(v)$ for a node v at level 4 of the AggTree in Fig. 6b.

window sizes $w_1, w_2, \ldots, w_m$ where $w_i < w_j$, $1 \le i < j \le m$, and a threshold associated with each window size, $f(w_j)$, $j = 1, 2, \ldots, m$. Our approach to solving this problem, called PeakID, is to first construct an AggTree on the given 2D LC-MS data and then search the AggTree for peaks, where each peak is represented by a pair (t, w) such that t is a time point, w is a window size in $\mathcal{W}$ and $\sum_{p=t}^{t+w-1} x_p \ge f(w)$. The algorithm for constructing the AggTree, called BuildTree, takes as input the 2D LC-MS data $\mathrm{LM}[N]$ where $\mathrm{LM}[j]$, $1 \le j \le N$, contains the intensity value x_j associated with the jth time point. In addition, the input data of the algorithm include the shift s_i, shadow size a_i, and degree d_i of each level i in the AggTree to be constructed. We will use a state-space algorithm to find appropriate values for s_i, a_i, and d_i, as explained later in the paper. The algorithm builds the AggTree in a bottom-up manner. In the first level, each node corresponds to a time point in the input 2D LC-MS data, and stores the intensity value associated with that time point. The top level has one node only, whose shadow contains all the N time points.

In practice, we do not need to build the entire AggTree. It suffices for the BuildTree algorithm to construct nodes from level 1 to level I where $a_{I-1} - s_{I-1} + 1 < w_m \le a_I - s_I + 1$. Here, w_m is the largest window size of interest, a_I is the shadow size of level I, and s_I is the shift of level I in the AggTree. Let $\mathrm{AggTree}[i][j]$, $1 < i \le I$ and $1 \le j \le \lfloor \frac{N-a_i}{s_i} \rfloor + 1$, represent the value stored in the jth node at level i. We have

$$\mathrm{AggTree}[i][j] = \sum_{p=1}^{d_i} \mathrm{AggTree}[i-1] \left[\left\lfloor \frac{(j-1) \times s_i + (p-1) \times a_{i-1}}{s_{i-1}} \right\rfloor + 1 \right]. \quad (2)$$

Fig. 7 summarizes the algorithm for building the AggTree. Notice that for efficiency reasons, we do not actually build the Aggregation Pyramid into which the AggTree is embedded. As explained below, the partially constructed AggTree is sufficient for detecting all peaks exactly yielding neither false positives nor false negatives.

3.4 Detecting Peaks Using an AggTree

We search the AggTree constructed in the previous section to detect all peaks in a top-down manner. To detect peaks in the time windows of size w_m, we examine the nodes at level I where $a_{I-1} - s_{I-1} + 1 < w_m \le a_I - s_I + 1$. In general, to detect peaks in the time windows of size w_j, $1 \le j \le m$, we examine the nodes at level i where $a_{i-1} - s_{i-1} + 1 < w_j \le a_i - s_i + 1$. We check each node $v = cell(a_i, t)$ at level i to see if the value stored in v exceeds the threshold $f(w_j)$ associated with the window size w_j. If so, an alarm is raised, indicating the possible occurrence of a peak, and a check in the detailed search region $DSR(v)$ is performed. The shadow of the node v ends at time point t. $DSR(v)$ comprises cells in the Aggregation Pyramid into which the AggTree is embedded, where the sizes of the shadows of the cells are in the range $[a_{i-1} - s_{i-1} + 2, a_i - s_i + 1]$, and the shadows end at time points in $[t - s_i + 1, t]$. Notice that, if v is the leftmost node at level i, since no node precedes v, $DSR(v)$ contains cells whose shadows end at time points in $[1, a_i]$. Fig. 8 illustrates the $DSR(v)$ for a node v at level 4 of the AggTree in Fig. 6b. In Fig. 8, the sizes of the shadows of the cells in the $DSR(v)$ are in the range [4, 7], and the shadows end at time points in [9], [10].

In searching $DSR(v)$ where $v = cell(a_i, t)$, if $cell(w_j, t') \ge f(w_j)$, $t - s_i + 1 \le t' \le t$, then a peak is detected, which is represented and output as a pair $(t' - w_j + 1, w_j)$. That is, the peak occurs in the time window $[t' - w_j + 1, t']$ where $\sum_{p=t'-w_j+1}^{t'} x_p \ge f(w_j)$. Since we do not actually build the Aggregation Pyramid containing $DSR(v)$, the values of the cells in $DSR(v)$ are computed "on-the-fly," as explained below. Observing that the shadows of two neighboring cells overlap, to avoid duplicate computation, we start from one seed node in the AggTree, and then by adding or subtracting the difference between two neighboring cells, we can get the values of the cells in $DSR(v)$. Specifically, assuming the seed node is $cell(h, t_s)$, we have

$$cell(h, t_s + 1) = cell(h, t_s) - cell(1, t_s - h + 2) + cell(1, t_s + 1). \quad (3)$$

In general, for all $1 < t_s + k \le N$,

$$cell(h, t_s + k) = cell(h, t_s + k - 1) - cell(1, t_s + k - h + 1) + cell(1, t_s + k). \quad (4)$$

Procedure: SearchTree(AggTree[I][N], w_j, $f(w_j)$)
Input: The AggTree[I][N], window size w_j and its associated threshold $f(w_j)$.
Output: The set $\mathcal{P}$ of pairs (t, w) representing peaks.
1. $i \leftarrow I$, $j \leftarrow m$, $\mathcal{P} \leftarrow \emptyset$;
   /* Search AggTree[I][N] in a top-down manner for peaks. */
2. **while** $j \geq 1$ **do**
3. **if** $(a_{i-1} - s_{i-1} + 1 < w_j \leq a_i - s_i + 1)$ **then**
4. **for each** node v = $cell(a_i, t)$ at level i of AggTree[I][N] **do**
5. **if** (v's value $\geq f(w_j)$) **then**
   /* Search $DSR(v)$. */
6. **for each** $cell(w_j, t')$, $t - s_i + 1 \leq t' \leq t$, in $DSR(v)$ **do**
7. **if** $cell(w_j, t') \geq f(w_j)$ **then**
8. $\mathcal{P} \leftarrow \mathcal{P} \cup \{(t' - w_j + 1, w_j)\}$;
9. $j \leftarrow j - 1$;
10. **else** $i \leftarrow i - 1$;
11. **return** $\mathcal{P}$;

Fig. 9. Algorithm for searching AggTree[I][N] for peaks.

Furthermore,

$$cell(h+1, t_s) = cell(h, t_s) + cell(1, t_s - h). \tag{5}$$

In general, for all $1 < h + k \leq I$,

$$\begin{aligned} cell(h+k, t_s) = {} & cell(h+k-1, t_s) \\ & + cell(1, t_s - h - k + 1). \end{aligned} \tag{6}$$

Due to the properties of Shifted Aggregation Trees, it's guaranteed to find a seed node in or near a detailed search region. Here is how. Notice $s_{i-1} \leq s_i$ (cf. Table 1). The shadows of the cells in $DSR(v)$ end at time points in $[t - s_i + 1, t]$, i.e., the time span of $DSR(v)$ is s_i. Thus, there exists a node at level $i-1$ whose shadow ends at some time point in $[t - s_i + 1,\ t]$; call it the seed node (Fig. 8). Notice $a_{i-1} \leq a_i - s_i + 1$. If $s_{i-1} > 1$, then $a_{i-1} - s_{i-1} + 2 \leq a_{i-1} \leq a_i - s_i + 1$. The sizes of the shadows of the cells in $DSR(v)$ are in the range $[a_{i-1} - s_{i-1} + 2,\ a_i - s_i + 1]$. Therefore, the seed node lies in $DSR(v)$. If $s_{i-1} = 1$, then the seed node is immediately below $DSR(v)$.

Fig. 9 summarizes the algorithm, called SearchTree, for finding peaks in the AggTree[I][N] constructed by the BuildTree algorithm in Fig. 7. The SearchTree algorithm takes as input the AggTree[I][N], a set $\mathcal{W}$ of window sizes $w_1, w_2, \ldots, w_m$ where $w_i < w_j$, $1 \leq i < j \leq m$, and a threshold associated with each window size, $f(w_j)$, $j = 1, 2, \ldots, m$. The algorithm outputs all pairs (t, w) such that t is a time point in AggTree[I][N], w is a window size in $\mathcal{W}$ and $\sum_{p=t}^{t+w-1} x_p \geq f(w)$. Each pair (t, w) represents a peak occurring in the time window $[t, t + w - 1]$. Notice that in step 6 of SearchTree, if v is the leftmost node at level i, the shadows of the cells in $DSR(v)$ end at time points in $[1, t]$, which equals $[1, a_i]$. Under this circumstance, we need to check each $cell(w_j, t')$ where $1 \leq t' \leq t = a_i$ in $DSR(v)$.

Below we show that the proposed PeakID method, composed of the two algorithms BuildTree and SearchTree, finds all peaks exactly yielding neither false positives nor false negatives. Let T be the Shifted Aggregation Tree constructed by the BuildTree algorithm and let AP be the Aggregation Pyramid into which T is embedded.

Lemma 3.1. *Let W be a time window of size w where $a_{i-1} - s_{i-1} + 1 < w \leq a_i - s_i + 1$. Let $cell(w, t)$ be the cell at level w of AP whose shadow is W that ends at time point t. There exists a node v at level i of T such that $cell(w, t)$ is shaded by v. Furthermore, $cell(w, t)$ lies in $DSR(v)$.*

Proof. We use mathematical induction to show that the lemma holds for any positive integer k, where $t \leq a_i + (k-1) \times s_i \leq N$.

Base step. When $k = 1$, i.e., $t \leq a_i$, since the shadow of the first node, i.e., the leftmost node, v at level i of T ends at time point a_i, $cell(w, t)$ is shaded by v. $DSR(v)$ comprises cells in AP where the sizes of the shadows of the cells are in the range $[a_{i-1} - s_{i-1} + 2,\ a_i - s_i + 1]$. Since $a_{i-1} - s_{i-1} + 1 < w \leq a_i - s_i + 1$, $cell(w, t)$ lies in $DSR(v)$.

Hypothesis step. Assume the lemma holds when $k = p$. That is, when $t \leq a_i + (p-1) \times s_i$ and $a_{i-1} - s_{i-1} + 1 < w \leq a_i - s_i + 1$, there exists a node v at level i of T such that $cell(w, t)$ is shaded by v and $cell(w, t)$ lies in $DSR(v)$.

Induction step. We want to show that the lemma holds when $k = p + 1$. Based on the properties of T, the shadows of the cells in the detailed search regions of the first p nodes at level i of T end at time points in $[1, a_i + (p-1) \times s_i]$. The shadows of the cells in the detailed search region of the $(p+1)$th node at level i of T end at time points in $[a_i + (p-1) \times s_i + 1, a_i + p \times s_i]$. Now, consider the time window W of size w where $a_{i-1} - s_{i-1} + 1 < w \leq a_i - s_i + 1$ and the $cell(w, t)$ whose shadow is W. If $1 \leq t \leq a_i + (p-1) \times s_i$, by the induction hypothesis, there exists a node v at level i of T such that $cell(w, t)$ is shaded by v and $cell(w, t)$ lies in $DSR(v)$.

If $a_i + (p-1) \times s_i + 1 \leq t \leq a_i + p \times s_i$, which means W ends at some time point in $[a_i + (p-1) \times s_i + 1,\ a_i + p \times s_i]$, then W must start at some time point in $[a_i + (p-1) \times s_i - w + 2, a_i + p \times s_i - w + 1]$. Notice that, the shadow of the $(p+1)$th node at level i of T starts at time point $p \times s_i + 1$. Since $w \leq a_i - s_i + 1$, we have $a_i - s_i + 1 - w \geq 0$. Thus, $(a_i + (p-1) \times s_i - w + 2) - (p \times s_i + 1) = a_i - s_i + 1 - w \geq 0$, which means $(a_i + (p-1) \times s_i - w + 2) \geq (p \times s_i + 1)$. Therefore, all time points in W are in $[p \times s_i + 1, a_i + p \times s_i]$, which is the shadow of the $(p+1)$th node at level i of T. Hence, $cell(w, t)$ is shaded by the $(p+1)$th node at level i of T.

Furthermore, because $a_{i-1} - s_{i-1} + 1 < w \leq a_i - s_i + 1$, and $DSR(v)$ comprises cells in AP where the sizes of the shadows of the cells are in the range $[a_{i-1} - s_{i-1} + 2,\ a_i - s_i + 1]$, we know that $cell(w, t)$ lies in $DSR(v)$. This completes the proof. □

Theorem 1. *The SearchTree algorithm finds all peaks exactly, yielding neither false positives nor false negatives.*

Proof. First, due to the verification done in step 7 of the algorithm, it does not yield false positives.

Next, assume for contradiction that SearchTree yields a false negative. That is, there exists a peak $P = (t, w_j)$ for some j, $1 \leq j \leq m$, that cannot be detected by SearchTree. The peak P occurs in the time window

starting at time point t and having a size of w_j. This time window is the shadow of $cell(w_j, t + w_j - 1)$, and $cell(w_j, t + w_j - 1) \geq f(w_j)$. From Lemma 3.1, there exists a node v at level i of the AggTree T where $a_{i-1} - s_{i-1} + 1 < w_j \leq a_i - s_i + 1$, such that $cell(w_j, t + w_j - 1)$ is shaded by the node v. Thus, w_j satisfies the condition in step 3 of the SearchTree algorithm. Since SearchTree checks each node at level i in step 4, the algorithm must be able to find the node v. Since $cell(w_j, t + w_j - 1) \geq f(w_j)$ and $cell(w_j, t + w_j - 1)$ is shaded by the node v, by monotonicity, the value stored in v must be greater than or equal to $f(w_j)$. Thus, the condition in step 5 is satisfied and $DSR(v)$ is searched.

By Lemma 3.1, $cell(w_j, t + w_j - 1)$ lies in $DSR(v)$. Since the algorithm checks each cell at level w_j in $DSR(v)$ in step 6, and $cell(w_j, t + w_j - 1) \geq f(w_j)$, the algorithm is able to detect and output (t, w_j) in step 8, which contradicts the assumption. □

3.5 A State-Space Algorithm

PeakID employs a heuristic state-space algorithm to search for an efficient AggTree from a training data set. The algorithm treats each (partially constructed) AggTree as a state and considers the growth from one partially constructed AggTree to another as a transformation. The training process is done only once, and the shift s_i, shadow size a_i, and degree d_i of each level i in a final state will be used subsequently as the input of the BuildTree algorithm in Fig. 7 to construct AggTrees for different sets of 2D LC-MS data when detecting peaks in those data sets.

In a state-space algorithm, the problem to be solved is represented by a set of states and a set of transformation rules mapping states to states. The solutions to the problem are represented by final states that satisfy some conditions and have no outgoing transformations. The search algorithm starts with one initial state, and then repeatedly applies the transformation rules to the set of states currently being explored to generate new states. When at least one final state is reached, the algorithm stops. There are different strategies to choose the order to traverse the state space. Depth-first search, breadth-first search, best-first search, and A^* search are commonly used ones [21]. Below, we describe the main components of our best-first search algorithm.

- *Initial state*. Each partially constructed AggTree must contain the input 2D LC-MS data. Thus, the initial state is the partially constructed AggTree consisting of level one only, in which each leaf node corresponds to a time point in the input data (Fig. 10).
- *Transformation rule*. If by adding a level of nodes to the top of a partially constructed AggTree A, we can get another (partially constructed) AggTree B, we say AggTree A or state A can be transformed to AggTree B or state B. Recall that there are some constraints that nodes of the top level of AggTree B must satisfy. Each node at the top level of AggTree B must aggregate several children below the top level. Each node in AggTree A is shaded by a node at the top level of AggTree B. The shift of the newly added top level of AggTree B must be an integral multiple of the shift of the level below the top level (cf. Table 1). The transformation rule defines how to grow a more complicated AggTree from a simpler AggTree.
- *Final states*. Final states are those (partially constructed) AggTrees that can be used to detect peaks across all window sizes of interest. Let h be the shadow size of the top level of an AggTree T and let s be the shift of the top level of T. For any time window W of size w, $w \leq h - s + 1$, W is shaded by a node at the top level of T. Thus, T is a final state if $h - s + 1 \geq w_m$ where w_m is the largest window size of interest.
- *Traversing strategy*. In order to find an efficient AggTree, we use the best-first strategy to explore the state space. Each state (or AggTree) T is associated with a cost; the state with the minimum cost is picked as the next state to be explored. One can calculate this cost empirically by measuring the CPU time needed to build the AggTree T and search T for peaks based on the training data set or calculate the cost based on a theoretical model, as we will explain below.
- *Desired tree*. The final state (or AggTree) T with the minimum cost is picked as the desired tree, where the shift s_i, shadow size a_i, and degree d_i of each level i in T are used as the input for the BuildTree algorithm in Fig. 7.

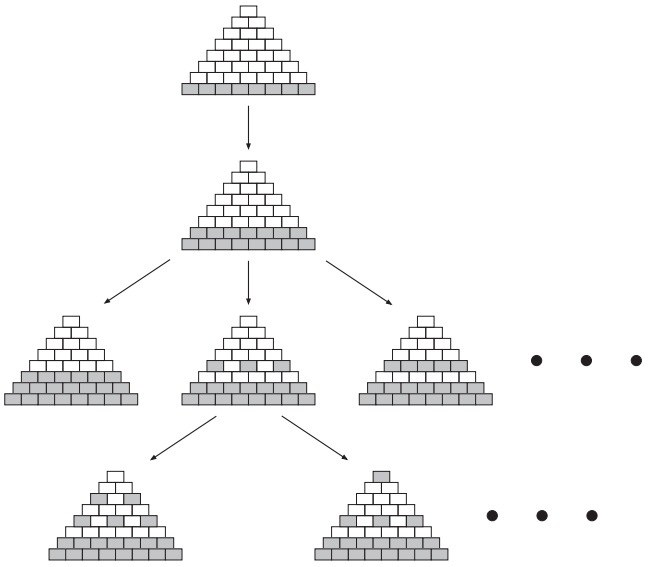


Fig. 10. Illustration of the state-space growth process.

In summary, the heuristic state-space algorithm starts with a partially constructed AggTree having level one only, and then keeps on growing the candidate set of AggTrees, until a set of final AggTrees is obtained. Fig. 10 illustrates how the state space grows.

Given an AggTree T, there are an exponential number of ways to grow T. We develop some constraints to reduce the complexity of our state-space algorithm. Let L be the maximum of the shadow sizes of the top levels of all the explored AggTrees (or states). Let S be the current state to be explored or visited. Instead of generating all possible next states from S, we generate only partially constructed AggTrees (or states) from S where the shadow sizes of the top levels of the AggTrees do not exceed $2L$. In addition, we introduce a parameter, $N = Max\_num\_states$, to govern

TABLE 2
Description of Synthetic Data

Data	Size	Max Intensity	Mean	StdDev
Set 0	2×10^4	6.256	3.182	0.968
Set 1	10^6	3.998	2.095	0.604
Set 2	10^6	23.723	11.211	4.384
Set 3	10^6	3.428	1.557	0.427
Set 4	10^6	6.133	2.922	0.826
Set 5	10^6	2.274	1.215	0.289
Set 6	10^6	5.627	2.817	0.772
Set 7	10^6	12.006	5.735	2.086
Set 8	10^6	14.033	6.153	2.698
Set 9	10^6	8.905	4.838	1.012
Set 10	10^6	18.333	9.688	3.972
Set 11	10^6	7.829	3.231	1.302
Set 12	10^6	2.408	1.008	0.354
Set 13	10^6	16.982	7.909	3.014
Set 14	10^6	6.018	2.757	0.706
Set 15	10^6	5.729	2.901	0.713
Set 16	10^6	10.692	4.724	1.528
Set 17	10^6	1.441	0.823	0.206
Set 18	10^6	3.697	1.775	0.678
Set 19	10^6	26.731	11.974	5.079
Set 20	10^6	3.156	1.669	0.518

the traversal of the state space. If we have visited N states in which the shadow sizes of the top levels are the same, we don't explore any more states whose top levels have this same shadow size. Likewise, if we have visited N final states, the algorithm stops and the final state we have visited with the minimum cost is returned as the output of the algorithm.

The cost associated with each state is used to decide which state is the next one to be explored. We develop a theoretical model to calculate the cost of an AggTree (or state) T. With this model, the cost of T equals the number of node construction operations needed to build T plus the number of cells to be checked in performing detailed searches in T when alarms are raised. The number of node construction operations equals the number of nodes in T. Let AP be the Aggregation Pyramid into which T is embedded. Let $P_a(w_j \mid a_i)$ be the probability that an alarm is raised, i.e., the probability that we check the cells at level w_j in AP given a node at level i of T with shadow size a_i (cf. step 5 of the SearchTree algorithm in Fig. 9). Let s_i be the shift of level i and n_i be the number of nodes at level i of T. The expected number of cells to be checked is

$$\sum_{i=1}^{I} \sum_{j=1}^{m} P_a(w_j \mid a_i) \times s_i \times n_i,$$

where the alarm probability $P_a(w_j \mid a_i)$ can be calculated from the training data set as explained in the next section.

4 Experiments and Results

We conducted a series of experiments to evaluate the performance of the PeakID method using both synthetic and real-world data. All the experiments were performed on a 2 GHz Pentium 4 PC having a memory of 2 G bytes. The operating system was Windows XP and the method was implemented in C++.

TABLE 3
Parameters and Default Values Used in Experiments

Parameter	Value
Max_num_states	200
Min_window_size	3
Max_window_size	500
Peak probability	10^{-5}

4.1 Experimental Results on Synthetic Data

We generated synthetic data using a mixture of Gaussians. A set of 20,000 intensity values, referred to as set 0, was used as training data. In addition, 20 sets, referred to as set 1 to set 20 with each set containing one million intensity values, were used as test data. In each data set, training or test, we randomly generated a number of peak profiles. Within each peak profile, we randomly generated intensity values that were normally distributed, where the smallest intensity values that were less than zero were discarded; the same proportion of the largest intensity values were also discarded. Table 2 gives details of the data sets, showing the size of each data set, i.e., the number of intensity values in each data set, the maximum intensity value, as well as the mean and standard deviation of the intensity values in each data set where each intensity value v corresponds to $v \times 10^5$. The minimum intensity value in each data set was zero.

We used set 0 to train our state-space algorithm to find the topology and structure of an efficient AggTree, i.e., to find the input parameter values s_i, a_i, and d_i for the BuildTree algorithm in Fig. 7.[2] We then ran the PeakID method, composed of the BuildTree algorithm in Fig. 7 and the SearchTree algorithm in Fig. 9, on each of the 20 test data sets, and the mean was plotted. Error bars, representing one standard error of the mean, were also plotted where the standard error was calculated by dividing the standard deviation by the square root of number of runs in the experiments. The error bars represent a description of how confident one is that the mean represents the true value. The smaller the error bars, the more reliable the plotted mean values are.

Table 3 lists parameters and their default values used in the experiments. The parameter Max_num_states is used by the state-space algorithm to reduce the time spent in traversing the state space. The window sizes of interest comprised consecutive integers in the range

$$[Min\_window\_size, Max\_window\_size].$$

The default value of Min_window_size was set to 3, as suggested in [13], [25], [34]. The peak probability is the probability that a peak occurs in a time window of some size w, i.e., it is the probability that the sum of the intensity values within the time window exceeds the threshold $f(w)$ associated with w. The peak probability is reversely proportional to the threshold—the smaller the peak probability, the larger the threshold is. We assumed that

2. This training is done once, and no more training is needed on the test data, since the training and test data were generated with the same form of randomness. When the data have a different form of randomness, e.g., gamma, we may need to train the state-space algorithm again.

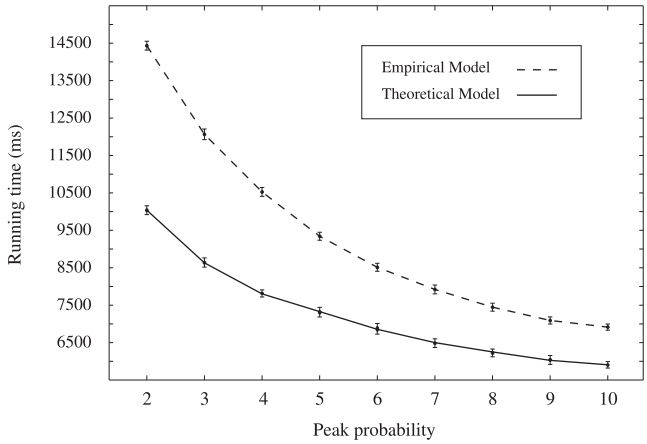


Fig. 11. Comparison of the theoretical cost model and the empirical cost model used in the state-space algorithm.

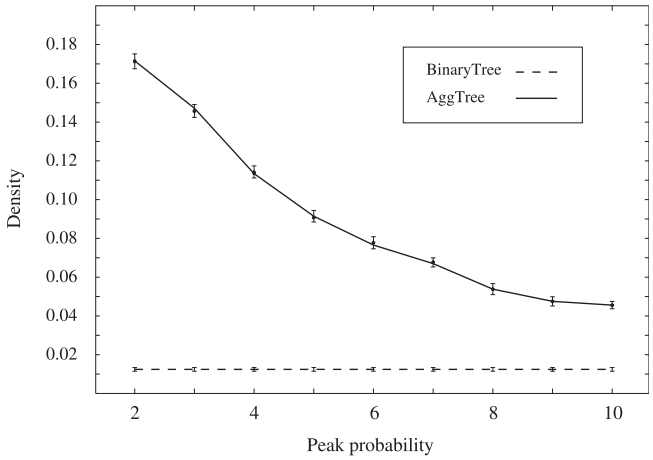


Fig. 12. The impact of peak probabilities on the density of a tree.

the peak probability was the same for each window size of interest.

The intensity values in the generated 2D LC-MS data are characterized by a mean μ and a standard deviation σ. The sum of the intensity values within a time window of size w has a mean $w\mu$ and standard deviation $\sqrt{w}\sigma$. Let p be the peak probability for the window size w. We can characterize this situation by saying that $Pr[S_o(w) \geq f(w)] \leq p$, where $S_o(w)$ is the observed sum of the intensity values within the time window of size w. Let $\Phi(x)$ be the normal cumulative distribution function for a normal random variable X

$$Pr[X \geq -\Phi^{-1}(p)] \leq p.$$

We have

$$Pr\left[\frac{S_o(w) - w\mu}{\sqrt{w}\sigma} \geq -\Phi^{-1}(p)\right] \leq p.$$

Therefore, the threshold $f(w)$ associated with w is $w\mu - \sqrt{w}\sigma\Phi^{-1}(p)$, where the values of σ, μ, and p can be found in Tables 2 and 3.

In Section 3.5, we introduced the alarm probability $P_a(w_j \mid a_i)$. This probability can be rewritten more generally as $P_a(w \mid W)$, $w \leq W$, which is the probability that the sum of the intensity values within a time window of size W exceeds the threshold $f(w)$ associated with the window size w. Referring to Fig. 9, W is the shadow size of the node v in step 4 and w is the window size w_j in step 5 in the figure. Thus, the alarm probability is $Pr[S_o(W) \geq f(w)]$. Therefore,

$$\begin{aligned} P_a(w \mid W) &= Pr[S_o(W) \geq f(w)] \\ &= Pr\left[\frac{S_o(W) - W\mu}{\sqrt{W}\sigma} \geq \frac{f(w) - W\mu}{\sqrt{W}\sigma}\right] \\ &= \Phi\left(-\frac{f(w) - W\mu}{\sqrt{W}\sigma}\right) \\ &= \Phi\left(\frac{(W-w)\mu}{\sqrt{W}\sigma} + \frac{\sqrt{w}\sigma\Phi^{-1}(p)}{\sqrt{W}\sigma}\right) \\ &= \Phi\left(\left(\sqrt{B} - \frac{1}{\sqrt{B}}\right)\sqrt{w}\frac{\mu}{\sigma} + \frac{\Phi^{-1}(p)}{\sqrt{B}}\right), \end{aligned} \tag{7}$$

where $B = \frac{W}{w}$ denotes the *bounding ratio* with respect to W, w.

So, $P_a(w \mid W)$ is determined by the distribution parameters μ and σ, the peak probability p, the bounding ratio B, and the cell level w in the Aggregation Pyramid into which the AggTree in Fig. 9 is embedded. It can be seen from (7) that the larger the ratio $\frac{\mu}{\sigma}$, the larger the alarm probability $P_a(w \mid W)$ is. As μ increases, there are more chances to raise an alarm. On the other hand, as σ increases, there are less chances to raise an alarm.

Fig. 11 compares the theoretical cost model with the empirical cost model used in the state-space algorithm described in Section 3.5. We first used the theoretical cost model to find an efficient AggTree T_1 from the training data set. The shift s_i, shadow size a_i, and degree d_i of each level i in T_1 were then used by the PeakID method to build an AggTree for each test data set and to search for peaks in that test data set. The CPU time used by PeakID on each test data set was recorded; the mean and error bars were plotted. We then used the empirical cost model to find an efficient AggTree T_2 from the same training data set. The shift, shadow size, and degree values of T_2 were then used by PeakID for peak detection in each test data set. The CPU time used by PeakID on each test data set was recorded; the mean and error bars were also plotted. Fig. 11 shows that the theoretical cost model is better than the empirical cost model; the AggTree T_1 produced from the theoretical cost model leads to a more efficient PeakID than that based on the AggTree T_2 produced from the empirical cost model. The X-axis in Fig. 11 shows different k values where each k corresponds to a peak probability 10^{-k}. As k becomes larger, the peak probability becomes smaller and consequently less time is required by PeakID to detect the fewer peaks.

In subsequent experiments, we used the theoretical cost model to generate the efficient AggTree employed by PeakID. Fig. 12 shows the density of an AggTree (BinaryTree, respectively) as a function of peak probabilities. The X-axis in the figure shows different k values where each k corresponds to a peak probability 10^{-k}. The Y-axis shows different densities where the density of a tree is defined as N_n/N_c; N_n is the number of nodes in the tree and N_c is the number of cells in the Aggregation Pyramid into which

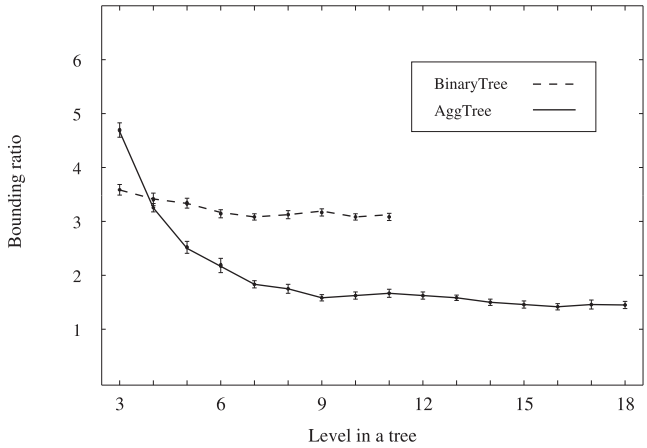


Fig. 13. The bounding ratio with respect to each level in a tree.

the tree is embedded. The figure shows that an AggTree becomes sparser or less dense when the peak probability becomes smaller. This happens because when peaks are rare, we only need few nodes in the AggTree to find those peaks. By contrast, the density of a BinaryTree is almost a constant, since the parent-child structure and the shifting pattern of the BinaryTree is fixed (cf. Table 1). Fig. 12 shows that an AggTree is able to adapt to the input data. Notice that AggTrees have a higher density than BinaryTrees. This implies that AggTrees generally have more levels and hence have smaller bounding ratios than BinaryTrees.

Fig. 13 shows the bounding ratio with respect to each level in an AggTree (BinaryTree, respectively). In an AggTree, the bounding ratio B is large at a low level where the shadow window size w is small; B is small at a high level where the shadow window size w is large. These changes of bounding ratios do not occur in BinaryTrees. The reason is that in a BinaryTree, for a node $v = cell(2^{i-1}, t)$, the shadow sizes of the cells in the detailed search region of v are in the range $[2^{i-3} + 2, 2^{i-2} + 1]$, cf. Section 3.2. We check the value stored in v; if the value exceeds the threshold for some window size in the range $[2^{i-3} + 2, 2^{i-2} + 1]$, an alarm is raised. Thus, the bounding ratio in a BinaryTree is always in the range $[2^{i-1}/(2^{i-3} + 2),\ 2^{i-1}/(2^{i-2} + 1)]$, which is approximately [2, 4].

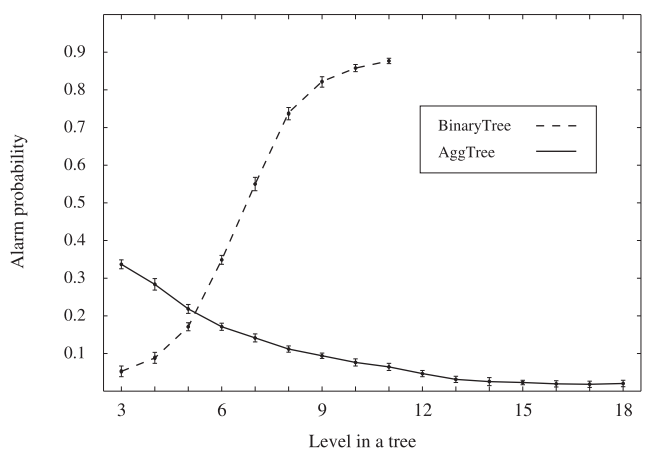


Fig. 14. The alarm probability with respect to each level in a tree.

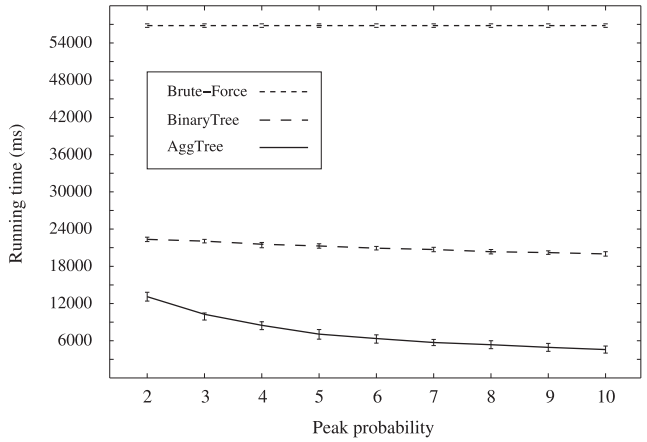


Fig. 15. Comparison of the three studied methods for varying peak probabilities.

Fig. 14 shows the alarm probability with respect to each level in an AggTree (BinaryTree, respectively). As explained above, in an AggTree, when the node level becomes larger, the bounding ratio becomes smaller. Referring to (7), as the bounding ratio decreases, so does the alarm probability, and consequently fewer detailed searches are performed. On the other hand, in a BinaryTree, the alarm probability is high and hence many detailed searches are needed when the node level is large.

The above analyses lead to the conclusion that searching for peaks using AggTrees would require less time and hence be more efficient than using BinaryTrees. Fig. 15 confirms this conclusion, showing the relative performance of these two data structures for varying peak probabilities. For comparison purposes, we also include the brute-force method in Fig. 15. Clearly, PeakID using AggTrees outperforms the other two methods. Notice that as the peak probability decreases, so does the alarm probability, cf. (7). Consequently, the running time of PeakID decreases.

Fig. 16 compares the relative performance of the three studied methods for varying window sizes of interest. The window sizes of interest comprised consecutive integers in the range $[3, Max\_window\_size]$. The X-axis shows different values of Max_window_size. Again, PeakID using AggTrees is the best. When Max_window_size becomes larger, the

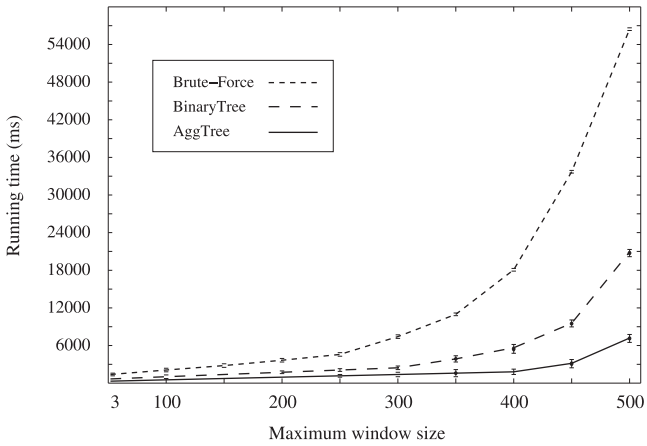


Fig. 16. Comparison of the three studied methods for varying window sizes of interest.

TABLE 4
Description of Real-World Data

Data	Size	Max Intensity	Mean	StdDev
Set 1	8341	8.936	3.024	1.322
Set 2	8852	8.612	2.737	1.175
Set 3	8199	9.138	2.804	1.283
Set 4	8068	8.533	2.671	1.199

superiority of PeakID becomes more obvious. This happens because with a large Max_window_size, there are more node levels in an AggTree where the bounding ratios can be adjusted, thereby speeding up the search for peaks in the AggTree.

In summary, an AggTree can adapt to the input data, adjusting its topology and structure through the training process to reduce alarm probabilities. By contrast, a BinaryTree does not employ the training process, and its parent-child structure and shifting pattern are always fixed (cf. Table 1). As a consequence, AggTrees are far more efficient than BinaryTrees when used in detecting peaks across different window sizes of interest.

4.2 Experimental Results on Real-World Data

We obtained four sets of 2D LC-MS data from cytochrome c, which is a protein commonly used to test biochemical characters [8]. The minimum intensity value in each data set was zero. Table 4 gives other details of the data sets, showing the size of each data set, i.e., the number of intensity values in each data set, the maximum intensity value, as well as the mean and standard deviation of the intensity values in each data set where each intensity value v corresponds to $v \times 10^5$. The experiments were performed in four phases. In phase i, $1 \leq i \leq 4$, data set i was used as the training data and three runs of experiments were performed where in run j, $1 \leq j \leq 4$, $j \neq i$, data set j was used as the test data. This led to 12 runs in total; the mean and error bars obtained from the 12 runs were plotted. The window sizes of interest in the experiments comprised consecutive integers in the range $[3, Max\_window\_size]$. The parameter value, Max_num_states, used by the state space algorithm in the training phase was fixed at 20.

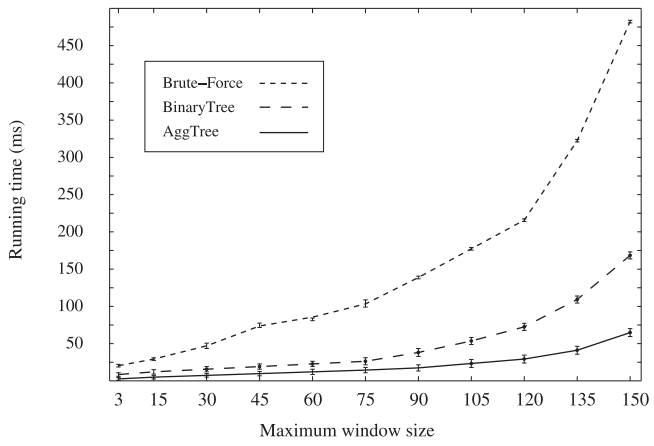


Fig. 17. Comparison of the three studied methods on real-world data.

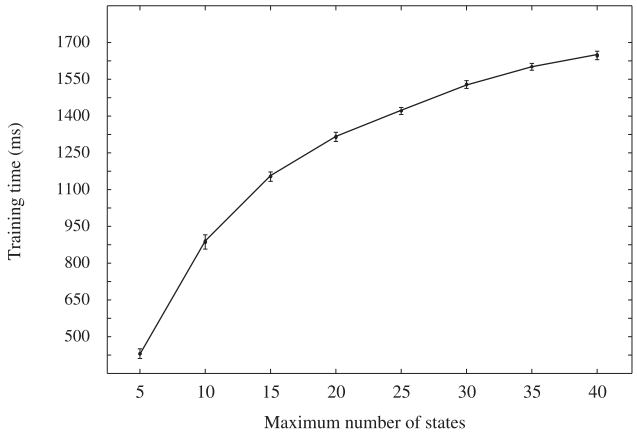


Fig. 18. The impact of Max_num_states on the training time of PeakID for real-world data.

Fig. 17 compares the relative performance of the three studied methods on the protein data for varying window sizes of interest. The trend observed here is consistent with that from the synthetic data (cf. Fig. 16). PeakID using AggTrees outperforms the method using BinaryTrees, which in turn is better than the brute-force method. We also tested the three methods by varying peak probabilities; the results were similar to those presented here.

Next, we examined how the heuristic state-space algorithm used in the training step affects peak detection. Fig. 18 shows the time spent in the training step as a function of the maximum number of states explored in the state-space algorithm. In phase i, $1 \leq i \leq 4$, data set i was used as the training data and the training time was recorded. The average training time was plotted in Fig. 18. It can be seen from the figure that the time used by the state-space algorithm, i.e., the training time of PeakID, is proportional to the maximum number of states explored in the algorithm.

Fig. 19 shows the running time of PeakID as a function of the maximum number of states explored in the state-space algorithm. It can be seen from Fig. 19 that as more states are explored, a more efficient AggTree can be built, and hence the less running time PeakID requires, though this speedup becomes less obvious when sufficient states are used (e.g.,

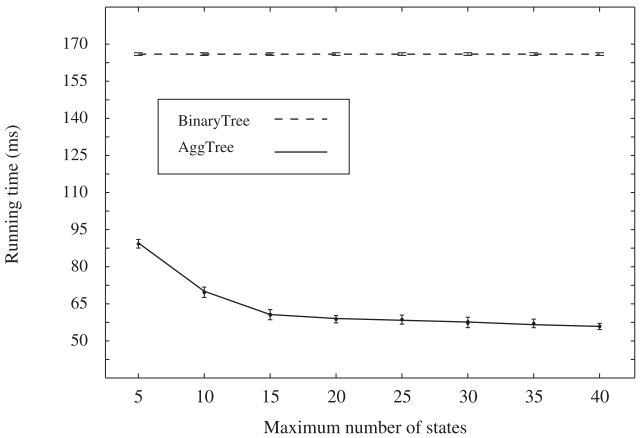


Fig. 19. The effect of Max_num_states on the running time of PeakID for real-world data.

TABLE 5
Experimental Results Obtained from Real-World Data

Win. Size	Set 1		Set 2		Set 3		Set 4	
	θ	n	θ	n	θ	n	θ	n
3	20.633	1	19.565	2	19.736	1	20.667	2
8	45.065	3	46.183	2	46.602	2	47.119	3
18	76.532	2	80.577	5	78.624	4	79.015	2
25	138.334	6	138.092	8	134.428	6	137.555	7
42	252.604	1	261.522	1	246.656	1	255.035	3
61	335.405	1	331.687	2	336.532	2	340.671	1
77	435.697	5	431.283	3	436.599	3	437.333	4
82	458.831	9	463.445	7	465.068	8	464.376	8
90	506.088	3	515.563	1	497.634	2	509.728	1
101	549.197	2	600.724	1	586.834	1	617.257	1
136	693.215	2	701.138	2	690.742	1	729.006	1

$Max\_num\_states \geq 20$). Since the method using BinaryTrees does not employ a state-space algorithm to generate a desired tree from the training data set, the running time for the method using BinaryTrees is almost a constant, independent of Max_num_states. Fig. 19 shows that by spending some time for training, PeakID runs much faster than the method using BinaryTrees. Notice that the training step affects the speed of PeakID only; it does not have any impact on the accuracy of PeakID. As shown in Theorem 1, the accuracy of PeakID is always 100 percent; no true peaks will be missed due to the training step.

Finally we compared PeakID with the techniques developed by Stolt et al. [25] on the protein data. To avoid false negatives, we combined Stolt et al.'s techniques with the brute-force algorithm for checking different window sizes. Table 5 summarizes the threshold, denoted by θ, and the corresponding number of peaks, denoted by n, with respect to each data set and window size found by Stolt et al.'s method. With the window sizes and threshold values in Table 5 where each threshold value v corresponds to $v \times 10^5$, PeakID obtained the same results while speeding up Stolt et al.'s method by a factor of 10. We have experimented with more (e.g., 30) data sets and other proteins (e.g., bovine serum albumin [26]). The results were similar to those presented here.

5 Discussion and Conclusions

In this paper, we have presented a new approach, called PeakID, for elastic peak detection in 2D LC-MS data. PeakID is comprised of two algorithms: BuildTree and SearchTree. It works by first constructing a Shifted Aggregation Tree or AggTree from input data in a bottom-up manner, and then searching the AggTree for different peaks in a top-down manner. This method is able to detect multiple peaks across a variety of window sizes yielding neither false positives nor false negatives. The PeakID program is written in C++, which can be downloaded from http://datalab.njit.edu/biosoft/PeakID.

The time complexities of the two algorithms, BuildTree and SearchTree, are closely related to the number of nodes in the AggTree used by the algorithms. Let N be the number of input time points, m be the number of window sizes of interest, and I be the number of levels of the AggTree. Let s_i represent the shift of level i, d_i represent the degree of level i, and n_i represent the number of cells at level i of the AggTree. BuildTree constructs $\sum_{i=1}^{I} \frac{n_i}{s_i}$ nodes in total. Since the value of each node is calculated by aggregating the values stored in its children, the time spent by BuildTree is therefore $\sum_{i=1}^{I} \frac{n_i}{s_i} \times d_i$. Thus, the time complexity of BuildTree is $O(IN)$, Notice that this is a very pessimistic upper bound, since n_i is much smaller than N as i becomes larger. The time complexity of SearchTree, in the worst case, is $O(mN)$, which happens when an alarm is raised and a check in the detailed search region $DSR(v)$ is performed for every node v in the AggTree. However, as shown in Fig. 14, the alarm probability with respect to each level of an AggTree is far smaller than 100 percent. This means that many of the fruitless detailed searches are skipped by PeakID. The method using BinaryTrees has the same time complexity as PeakID. However, in practice, that method is much slower than PeakID, as demonstrated in Figs. 16 and 17. This happens because AggTrees are adaptive to input data whereas BinaryTrees have a fixed topology regardless of the input data.

The space complexity of PeakID is also very much related to the number of nodes in its AggTree. Let N_i be the number of nodes at level i of the AggTree. The total number of nodes in the AggTree is $\sum_{i=1}^{I} N_i$. Since $N_i \leq N$, the space complexity is $O(IN)$. However, this is a very pessimistic upper bound. To be more precise, notice $N_i = \frac{n_i}{s_i}$. Hence, the space complexity can be formulated as $\sum_{i=1}^{I} \frac{n_i}{s_i}$, which is much smaller than IN. The space complexity of the method using BinaryTrees is $O(N)$ [40]. The space used by these methods is best illustrated in Fig. 12, where the density of a tree is shown as a function of different peak probabilities.

Our work concentrates on offline batch processing of 2D LC-MS data. It is possible to integrate our method into LC-MS instrumental software to perform online peak detection [25]. Online data stream processing has drawn much attention recently [7]. The proposed AggTree is adaptive to input data and hence can be easily extended for online peak detection. Here is how. Refer to the algorithm BuildTree in Fig. 7. Instead of taking the whole 2D LC-MS data $\mathrm{LM}[N]$ as input, we consider one time point at a time and build the AggTree dynamically. Let d_i be the degree of level i of the AggTree. As soon as we receive the first d_2 time points at level 1, we can build the first node at level 2. In general, as soon as we have the first $\frac{(j-1)\times s_i + a_i - a_{i-1}}{s_{i-1}} + 1$ nodes at level $i-1$, we can build the first j nodes at level i. Next, refer to the algorithm SearchTree in Fig. 9. Instead of taking the whole AggTree[I][N] as input, we can start the search as described in Fig. 9 as soon as the first node at level I is built. The reason is that when the first node at level I is built, all nodes below it have already been built and can be searched. Let the shift of level I be s_I. In general, after getting every s_I time points, one more node at level I is built; we can perform the search as described in Fig. 9 from that node immediately. The training process in which the topology of an efficient AggTree is obtained by the state-space algorithm can be done on historical data or on the first couple of time points of data streams. This training process is done only once; the parameter values obtained from the training phase can be used to build efficient AggTrees suitable for online peak detection of numerous

LC-MS data sets with the same or similar statistics. The method using BinaryTrees can be easily extended for online peak detection as well [40]. While that method does not require a training phase, PeakID with AggTrees can run much faster than that method in online peak detection, particularly when the peak probability is extremely low and the number of window sizes of interest is very large, cf. Figs. 15, 16, and 17.

It should be pointed out that in LC-MS data analysis, window sizes of interest are determined by biochemical experiments, instrumentation, experiences, and domain knowledge [1], [3], [14], [22], [25]. The threshold associated with each window size is determined by similar factors, or by mathematical formulas [10], [11]. In contrast to other peak detection methods [16], [24], [35], [37], which do not deal with peaks across multiple window sizes, may have inaccurate peak prediction or threshold estimation, or are variants of brute-force methods, the proposed PeakID method exploits a novel data structure (AggTree) to detect peaks across different window sizes with 100 percent accuracy.

In the empirical study presented here, we compared PeakID with two other methods (BinaryTrees and brute force) that also have 100 percent accuracy. Our experimental results indicated that by spending some time in a training phase to find the topology and structure of an efficient AggTree, PeakID can run much faster than the two other methods. The more window sizes of interest there are, the more speedup PeakID achieves (cf. Figs. 16 and 17). In practice, when the domain demands the detection of peaks across a large number of different window sizes, it is worthwhile to implement PeakID to meet that demand.

The proposed approach lays out a framework for solving the elastic peak detection problem on time series data. While we have focused on 2D LC-MS data in this paper, and have shown that PeakID can speed up the analysis of such data by a factor of 10 when compared with a state-of-the-art method [25], our techniques can be easily generalized to process other time series data in an efficient way.

ACKNOWLEDGMENTS

This work was supported in part by NIH Grant 2R01GM032877-25A1, and US NSF Grants DBI-0445666, MCB-0929339, and IOS-0922738. The authors thank the anonymous reviewers for their constructive suggestions, which helped improve the presentation and content of this paper. They also thank Dr. Tong Liu of the Center for Advanced Proteomics Research at UMDNJ-New Jersey Medical School for useful conversations and assistance on LC-MS data collection and analysis.

REFERENCES

[1] R.E. Ardrey, *Liquid Chromatography—Mass Spectrometry: An Introduction.* John Wiley & Sons, 2003.

[2] A.E. Ashcroft, "An Introduction to Mass Spectrometry," http://www.astbury.leeds.ac.uk/facil/MStut/mstutorial.htm, 2011.

[3] K.A. Baggerly, J.S. Morris, J. Wang, D. Gold, L.C. Xiao, and K.R. Coombes, "A Comprehensive Approach to the Analysis of Matrix-Assisted Laser Desorption/Ionization-Time of Flight Proteomics Spectra from Serum Samples," *Proteomics,* vol. 3, no. 9, pp. 1667-1672, 2003.

[4] *Analysis of Biological Data: A Soft Computing Approach,* S. Bandyopadhyay, U. Maulik, and J.T.L. Wang, eds. World Scientific, 2007.

[5] F.K.-P. Chan, A.W.-C. Fu, and C. Yu, "Haar Wavelets for Efficient Similarity Search of Time-Series: With and without Time Warping," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 3, pp. 686-705, May/June 2003.

[6] S. Chen, D. Hong, and Y. Shyr, "Wavelet-Based Procedures for Proteomic Mass Spectrometry Data Processing," *Computational Statistics & Data Analysis,* vol. 52, pp. 211-220, 2007.

[7] Y. Chen, G. Dong, J. Han, B.W. Wah, and J. Wang, "Multi-Dimensional Regression Analysis of Time-Series Data Streams," *Proc. 28th Int'l Conf. Very Large Data Bases,* pp. 323-334, 2002.

[8] C. Christin, A.K. Smilde, H.C.J. Hoefsloot, F. Suits, R. Bischoff, and P.L. Horvatovich, "Optimized Time Alignment Algorithm for LC-MS Data: Correlation Optimized Warping Using Component Detection Algorithm-Selected Mass Chromatograms," *Analytical Chemistry,* vol. 80, no. 18, pp. 7012-7021, 2008.

[9] M.C. Codrea, C.R. Jimenez, S. Piersma, J. Heringa, and E. Marchiori, "Robust Peak Detection and Alignment of nanoLC-FT Mass Spectrometry Data," *Proc. Fifth European Conf. Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics,* pp. 35-46, 2007.

[10] K.R. Coombes, S. Tsavachidis, J.S. Morris, K.A. Baggerly, M.C. Hung, and H.M. Kuerer, "Improved Peak Detection and Quantification of Mass Spectrometry Data Acquired from Surface-Enhanced Laser Desorption and Ionization by Denoising Spectra with the Undecimated Discrete Wavelet Transform," *Proteomics,* vol. 5, no. 16, pp. 4107-4117, 2005.

[11] I. Eidhammer, K. Flikka, L. Martens, and S.-O. Mikalsen, *Computational Methods for Mass Spectrometry Proteomics,* first ed. Wiley-Interscience, 2008.

[12] M.G. Elfeky, W.G. Aref, and A.K. Elmagarmid, "Periodicity Detection in Time Series Databases," *IEEE Trans. Knowledge and Data Eng.,* vol. 17, no. 7, pp. 875-887, July 2005.

[13] A. Felinger, *Data Analysis and Signal Processing in Chromatography.* Elsevier Science, 1998.

[14] T. Fushiki, H. Fujisawa, and S. Eguchi, "Identification of Biomarkers from Mass Spectrometry Data Using a Common Peak Approach," *BMC Bioinformatics,* vol. 7, article 358, 2006.

[15] T. Hankemeier, J. Rozenbrand, M. Abhadur, J.J. Vreuls, and U.A.T. Brinkman, "Data Correlation in On-Line Solid-Phase Extraction-Gas Chromatography-Atomic Emission/Mass Spectrometric Detection of Unknown Microcontaminants," *Chromatographia,* vol. 48, pp. 273-283, 1998.

[16] M. Karnstedt, D. Klan, C. Politz, K.U. Sattler, and C. Franke, "Adaptive Burst Detection in a Stream Engine," *Proc. ACM Symp. Applied Computing,* pp. 1511-1515, 2009.

[17] M. Katajamaa and M. Oresic, "Processing Methods for Differential Analysis of LC/MS Profile Data," *BMC Bioinformatics,* vol. 6, no. 1, article 179, 2005.

[18] E. Keogh, S. Lonardi, and W. Chiu, "Finding Surprising Patterns in a Time Series Database in Linear Time and Space," *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* 2002.

[19] J. Kleinberg, "Bursty and Hierarchical Structure in Streams," *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 91-101, 2002.

[20] J. Listgarten, R.M. Neal, S.T. Roweis, P. Wong, and A. Emili, "Difference Detection in LC-MS Data for Protein Biomarker Discovery," *Bioinformatics,* vol. 23, no. 2, pp. 198-204, 2007.

[21] Z. Michalewicz and D.B. Fogel, *How to Solve It: Modern Heuristics.* Springer, 2002.

[22] B.Y. Renard, M. Kirchner, H. Steen, J.A.J. Steen, and F.A. Hamprecht, "NITPICK: Peak Identification for Mass Spectrometry Data," *BMC Bioinformatics,* vol. 9, article 355, 2008.

[23] D. Shasha and Y. Zhu, *High Performance Discovery in Time Series: Techniques and Case Studies.* Springer, 2004.

[24] L. Singh and M. Sayal, "Privacy Preserving Burst Detection of Distributed Time Series Data Using Linear Transforms," *Proc. IEEE Symp. Computational Intelligence and Data Mining,* pp. 646-653, 2007.

[25] R. Stolt, R.J.O. Torgrip, J. Lindberg, L. Csenki, J. Kolmert, I. Schuppe-Koistinen, and S.P. Jacobsson, "Second-Order Peak Detection for Multicomponent High-Resolution LC/MS Data," *Analytical Chemistry,* vol. 78, pp. 975-983, 2006.

[26] M.H.M. van de Meent and G.J. de Jong, "Improvement of the Liquid-Chromatographic Analysis of Protein Tryptic Digests by the Use of Long-Capillary Monolithic Columns with UV and MS Detection," *Analytical and Bioanalytical Chemistry,* vol. 388, pp. 195-200, 2007.
[27] M. Vlachos, C. Meek, Z. Vagena, and D. Gunopulos, "Identifying Similarities, Periodicities and Bursts for Online Search Queries," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 131-142, 2004.
[28] *Pattern Discovery in Biomolecular Data: Tools, Techniques and Applications,* J.T.L. Wang, B.A. Shapiro, and D. Shasha, eds. Oxford Univ. Press, 1999.
[29] *Data Mining in Bioinformatics,* J.T.L. Wang, M.J. Zaki, H.T.T. Toivonen, and D. Shasha, eds. Springer, 2005.
[30] M. Wang, T. Madhyastha, N.H. Chan, S. Papadimitriou, and C. Faloutsos, "Data Mining Meets Performance Evaluation: Fast Algorithms for Modeling Bursty Traffic," *Proc. 18th Int'l Conf. Data Eng.,* pp. 507-516, 2002.
[31] J. Wong, G. Cagney, and H.M. Cartwright, "SpecAlign - Processing and Alignment of Mass Spectra Data Sets," *Bioinformatics,* vol. 21, pp. 2088-2090, 2005.
[32] C. Yang, Z. He, and W. Yu, "Comparison of Public Peak Detection Algorithms for MALDI Mass Spectrometry Data Analysis," *BMC Bioinformatics,* vol. 10, article 4, 2009.
[33] J. Yang, W. Wang, and P.S. Yu, "Mining Asynchronous Periodic Patterns in Time Series Data," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 3, pp. 613-628, May/June 2003.
[34] W. Yu, B. Wu, N. Lin, K. Stone, K. Williams, and H. Zhao, "Detecting and Aligning Peaks in Mass Spectrometry Data with Applications to MALDI," *Computational Biology and Chemistry,* vol. 30, pp. 27-38, 2006.
[35] Z. Yuan, K. Du, Y. Jia, and J. Miao, "Stream Event Detection: A Unified Framework for Mining Outlier, Change and Burst Simultaneously over Data Stream," *Proc. Seventh IEEE Int'l Conf. Data Mining-Workshops,* pp. 575-580, 2007.
[36] X. Zhang and D. Shasha, "Better Burst Detection," *Proc. 22 Int'l Conf. Data Eng.,* p. 146, 2006.
[37] A. Zhou, S. Qin, and W. Qian, "Adaptively Detecting Aggregation Bursts in Data Streams," *Proc. 10th Int'l Conf. Database Systems for Advanced Applications,* pp. 435-446, 2005.
[38] Y. Zhu and D. Shasha, "StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time," *Proc. 28th Int'l Conf. Very Large Data Bases,* pp. 358-369, 2002.
[39] Y. Zhu and D. Shasha, "Warping Indexes with Envelope Transforms for Query by Humming," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 181-192, 2003.
[40] Y. Zhu and D. Shasha, "Efficient Elastic Burst Detection in Data Streams," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 336-345, 2003.

Xin Zhang received the PhD degree in computer science from the Courant Institute of New York University. His research interests include time series data analysis and data mining.

Dennis Shasha received the BS degree from Yale University and the PhD degree from Harvard University in 1984. He is a professor of computer science at the Courant Institute of New York University, where he works with biologists on pattern discovery for microarrays, combinatorial design, network inference, and protein docking; with physicists, musicians, and financial people on algorithms for time series; and on database applications in untrusted environments. Other areas of interest include database tuning as well as tree and graph matching. Because he likes to type, he has written six books of puzzles about a mathematical detective, a biography about great computer scientists, and technical books about database tuning, biological pattern recognition, time series, and statistics. He has coauthored more than 60 journal papers, 70 conference papers, and 15 patents. He has written the puzzle column for various publications including *Scientific American*.

Yang Song received the BE degree in computer science from the University of Science & Technology of China. He is currently working toward the PhD degree in the Computer Science Department at the New Jersey Institute of Technology, and is a student intern with the Center for Advanced Proteomics Research, UMDNJ-New Jersey Medical School Cancer Center in Newark, New Jersey. His research interests include data mining and bioinformatics. He is a student member of the IEEE.

Jason T.L. Wang received the BS degree in mathematics from National Taiwan University, Taipei, Taiwan, and the PhD degree in computer science from the Courant Institute of Mathematical Sciences at New York University in 1991. He is a professor of computer science and bioinformatics at the New Jersey Institute of Technology and the director of the university's Data and Knowledge Engineering Laboratory. His research interests include data mining, databases, and computational biology and bioinformatics. He has published five books and more than 100 papers in these areas. He is a member of the IEEE.