iTree: Exploring Time-Varying Data using Indexable Tree

Yi Gu* Chaoli Wang†

Michigan Technological University

ABSTRACT

Significant advances have been made in time-varying data analysis and visualization, mainly in improving our ability to identify temporal trends and classify the underlying data. However, the ability to perform cost-effective data querying and indexing is often not incorporated, which posts a serious limitation as the size of time-varying data continue to grow. In this paper, we present a new approach that unifies data compacting, indexing and classification into a single framework. We achieve this by transforming the time-activity curve representation of a time-varying data set into a hierarchical symbolic representation. We further build an indexable version of the data hierarchy, from which we create the iTree for visual representation of the time-varying data. A hyperbolic layout algorithm is employed to draw the iTree with a large number of nodes and provide focus+context visualization for interaction. We achieve effective querying, searching and tracking of time-varying data sets by enabling multiple coordinated views consisting of the iTree, the symbolic view and the spatial view.

1 INTRODUCTION

A notable recent research effort in time-varying data visualization is to treat voxels' values over time as *time-activity curves* (TACs) for studying their temporal behaviors [3]. The TAC representation has been successfully applied to multiscale data clustering [21], temporal sequencing [22] and trend identification [10]. Several challenges, however, still remain. The first challenge is the ever-growing size and complexity of time-varying data we need to deal with. A straightforward way of adopting TACs raises the concern of speed and performance due to the large number of voxels involved and the potentially long time series present in the time-varying data. The second challenge is that the ability to index and query TACs in a coarse-to-fine manner is still missing. While many solutions allow multiscale data exploration, data indexing and querying are not naturally built in. Finally, the third challenge is the lack of techniques to interact with the space-time data. Due to the nature of four-dimensional spatiotemporal data, there is an increasing need to transform the data into another space of lower dimension for more desirable navigation and exploration. Therefore, a solution that can achieve efficient data compacting, effective data indexing and intuitive data exploration is highly desirable.

In this paper, we propose a novel solution that unifies data transformation and visual representation into an integrated framework to address the above challenges and support effective visual data analysis. Many high-level representations of time series, such as Fourier transforms, wavelets and piecewise polynomial models, have been presented for data mining. We utilize the *symbolic aggregate approximation* (SAX) technique [12] to represent time-series data in the TAC form. Compared to other symbolic representations, SAX not only allows dimensionality reduction, but also guarantees distance bound. To support fast indexing and coarse-to-fine data analysis in a progressive manner, we further organize SAX words into a multiresolution, bit-aware representation called *indexable* SAX (iSAX) [18]. Leveraging extensible hashing, iSAX offers a quantized and reduced SAX representation with variable granularity, enabling fast approximate search and exact search. To tailor these techniques for our needs, we modify the original SAX and iSAX algorithms by incorporating transfer function based breakpoint identification and a more balanced scheme for symbol splitting.

To support intuitive navigation and exploration of the large iSAX hierarchy, we design the *iTree*, a 2D hyperbolic browser that visually arranges the hierarchical symbolic representation for focus+context (F+C) visualization. Drawing the iTree in 2D instead of 3D avoids occlusion and simplifies user interaction. Leveraging the iTree and the SAX view, we further provide functions for the user to perform data querying and clustering and make connection to the original space-time data. These views work hand-in-hand to enable visual data interrogation, pattern identification and knowledge extraction, which are not possible with only a single view. We demonstrate the efficiency and effectiveness of our approach in assisting data analysis and visualization tasks with experimental results running on several time-varying data sets.

2 RELATED WORK

Data Compacting. Compressing time-varying data to supports cost-effective data processing, rendering and viewing has been a central focus of many research efforts. Efficient data structures for accelerating time-varying data retrieval and rendering have also been presented. Examples include the multidimensional (4D) tree [20], time-space partitioning tree (TSP) [17] and space-partitioning time (SPT) tree [2]. We utilize SAX to compact a voxel's spatial neighbors over time into a succinct symbolic representation, which is less time consuming than typical transform-based compression solutions. Leveraging iSAX, we organize the compact data in the SAX form into a hierarchical structure to facilitate data indexing and querying.

Data Indexing. Bitmap indexing [16] is an approach for accelerating multidimensional, multivariate range queries for read-mostly data. In its simplest form, a bitmap index consists of a vector of bits (i.e., bitmap) on an indexed attribute, where the size of each bitmap equals the cardinality of the indexed relation. Bitmap indexing is well suited for value range queries of multivariate data with flexible support of boolean operations [19], while SAX/iSAX is mainly designed for indexing and finding patterns or trends in time-varying data. Dynamic time warping (DTW) is an algorithm for measuring similarity between two sequences which may vary in time or speed. DTW and its variant SUBDTW have been utilized in finding trends in time-varying multivariate data [11, 10]. DTW has been shown to be better than the Euclidean distance for most data mining tasks in most domains, yet the difference diminishes when the data sets get larger [23]. SAX and iSAX use the Euclidean distance while they also allow indexing under DTW as well.

Visual Representation. To assist scientific visualization tasks, it is often useful to transform the data or the visualization process into another visual means commonly used in information visualization for exploring complex parameter and data relationships. Early examples include the image graph [15] and spreadsheet-like interface [7]. More recent examples include the storyboard [14], attribute

*e-mail: gyi@mtu.edu

†e-mail: chaoliw@mtu.edu

cloud [6] and TransGraph [4]. Our iTree addresses two difficulties in time-varying data analysis and visualization: supporting efficient data compacting and querying, and controlling the exploration of data relationships in an adaptive manner.

3 SAX AND iSAX

The simplest way to construct SAX/iSAX representations from a time-varying data set is to treat each voxel over time as a TAC. However, it may not be cost-effective for a large time-varying data set with a large number of voxels and/or a large number of time steps. Since we will visualize the iSAX hierarchy, it is important to effectively reduce the number of SAX words constructed. One solution is to organize spatially neighboring voxels into a group (such as $2\times2\times2$) and rearrange their corresponding TACs time step by time step (i.e., going through voxel by voxel for the first time step, then the second etc.), treating it as the TAC for the entire group. Furthermore, breaking the long time steps into time intervals would allow us to cluster data with finer temporal granularity. Therefore, to strike a good balance, we suggest to use group-wise voxels in conjunction with time intervals for generating input TACs. SAX has been applied to detect frequent or outlier patterns for time-varying data [5]. We modify the original algorithms [12, 18] to accommodate the characteristics of time-varying volumetric data in order to better differentiate SAX words, improve computation efficiency, and reduce the number of nodes shown in the iTree.

3.1 SAX

A SAX word is a symbolic aggregate approximation of a TAC, which reveals the trend of the TAC and allows indexing in iSAX. A SAX word can be represented by symbols (e.g., a, b, c and d etc.) or bits (e.g., 00, 01, 10 and 11 etc.). SAX achieves saving in storage by dimension reduction of time steps and bit compression.

Before transforming a TAC to a SAX, the user first specifies the value dimension α which indicates how many levels the entire value range will be partitioned into. It is essential to appropriately choose the "breakpoints" that determine the partitioning. Lin et al. [12] normalized each TAC to have zero mean and unit variance. Assuming the value distribution of a normalized TAC fulfills the Gaussian distribution, they determined a set of breakpoints by evenly partitioning the area covered by the distribution. Normalizing a TAC may better differentiate the value difference but the information of the original value range is lost. Moreover, solely considering the values may lead us to distinguish the value ranges not even showing up in the transfer function (i.e., the corresponding opacity is zero), which is not desirable either.

To resolve these issues, we present our transfer function based strategy to choose the breakpoints. Specifically, we first convert the TACs to the piecewise aggregate approximation (PAA) representations, then construct a global histogram H' based on all PAA representations. After that, we combine H' with the transfer function to generate a new histogram H. From H, we calculate a set of breakpoints and finally apply a further transformation to obtain discrete symbolic representations.

PAA Conversion. Given a TAC T with length of n, we convert T into a PAA by compressing n values into w dimensions. The ith element in the PAA representation C is calculated as

$$C[i] = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} T[j], \tag{1}$$

In practice, n does not need to be an exact multiple of w and we modify the summation to adopt fractional values when w does not evenly divide n.

Breakpoint Identification. From the PAA representations, we construct a fine-grained global histogram H'. Figure 1 illustrates such a histogram H' and the corresponding opacity transfer function. We treat the opacity as the weight for each bin and generate a new histogram H. Given H and the value dimension α, we find a set

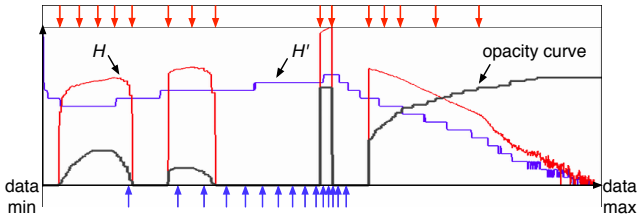


Figure 1: H' is the histogram after logarithm and normalization of the original histogram of the earthquake data set. H is the new histogram which results from multiplying H' by the opacity value. Blue and red arrows indicate the breakpoints determined by the original histogram and H, respectively.

of breakpoints to mark those ranges with zero opacity and evenly partition the rest of H. We first identify the continuous value ranges with zero opacity. For each of these ranges, we assign two breakpoints to mark the beginning and ending data values. If the beginning (ending) value is the minimum (maximum) value of the data set, we remove this breakpoint. Assuming m and n are the numbers of value ranges with zero and non-zero opacity respectively, we have three cases. If $n+m=\alpha$, the algorithm stops. If $n+m<\alpha$, we successively pick the value range covering the largest area in H and split it into two ranges with equal area. The value for splitting marks the new breakpoint. If $n+m>\alpha$, we successively choose the two neighboring non-zero opacity ranges with the smallest zero opacity value range in between to merge. The beginning and ending values of the new range after merging are the breakpoints left.

In Figure 1, the final breakpoints determined by H' and H are marked with blue and red arrows, respectively. As we can see, our transfer function based solution not only avoids focusing on the value ranges with zero opacity, but also better balances the frequencies of all SAX symbols in the final representation. Figure 4 (a) and (b) show a comparison and it is clear that using our transfer function based solution leads to a more meaningful layer-by-layer data classification as shown in (b).

SAX Word Generation. Once the breakpoints are obtained, we construct an alphabet Φ and transform C into an array of symbols $\hat{C}$ to form the SAX word. We assign the symbol $\hat{C}[i]$ by comparing $C[i]$ with the value ranges bounded by breakpoints β_{j-1} and β_j

$$\hat{C}[i] = \Phi_j, \text{ iff } \beta_{j-1} \le C[i] < \beta_j. \tag{2}$$

Given two symbols $\hat{C}[i]$ and $\hat{C}[j]$ with the same value dimension α, the distance between them is defined as

$$d(\hat{C}[i],\hat{C}[j]) = \begin{cases} 0, & \text{if } |\hat{C}[i]-\hat{C}[j]| \le 1 \\ \beta_{\max(\hat{C}[i],\hat{C}[j])-1} - \beta_{\min(\hat{C}[i],\hat{C}[j])}, & \text{otherwise} \end{cases} \tag{3}$$

where $\hat{C}[i]$ and $\hat{C}[j]$ are the indices of the corresponding symbols that are between 0 and $\alpha-1$.
The distance between two SAX words $\hat{C}_1$ and $\hat{C}_2$ is defined as

$$d(\hat{C}_1,\hat{C}_2) = \sqrt{\frac{1}{w}\sum_{i=1}^{w}\left(d(\hat{C}_1[i],\hat{C}_2[i])\right)^2}. \tag{4}$$

Note that the distance measure defined based on two SAX words is the lower bound of the Euclidean distance defined based on the PAA representation of the original time series. This allows us to transform TACs into vectors of discrete symbols and work on these symbols directly to speed up the performance.

3.2 iSAX

After converting all TACs to SAX words, we build the iSAX hierarchy through clustering. In the hierarchy, a node represents a set of TACs with the same or similar SAX words. When the number of SAX words at a node exceeds a certain threshold δ_n, the node

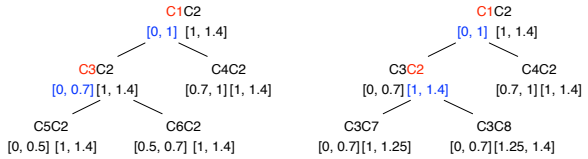


Figure 2: Comparing SAX symbol splitting using our algorithm (a) and the original algorithm [12] (b). Red indicates which symbol to split and blue indicates the largest value range.

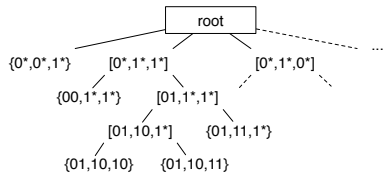


Figure 3: An illustration of an iSAX index. Internal nodes and terminal nodes are denoted with [ ] and { }, respectively.

is split into two children with bit promotion of a symbol. This is to keep the number of SAX words corresponding to each terminal node small enough for efficient search.

The key of this process is to find the proper symbol to split. Shieh and Keogh [18] always chose the symbol with the left-most smallest bit cardinality to split, which guarantees that the difference between the largest and smallest bit cardinalities in each SAX word is less or equal to 1. We choose a symbol covering the largest value range to split in order to maximize the difference between SAX words. Figure 2 illustrates a comparison using our algorithm and the original algorithm. After two iterations, our algorithm has three SAX words C_5C_2, C_6C_2 and C_4C_2 (i.e., leaf nodes of the hierarchy). The difference between C_5C_2 and C_4C_2 is $\sqrt{((0.7-0.5)+0)/2} = 0.32$. The other two differences are both zero because the corresponding SAX symbols are either the same or have neighboring value ranges. The three SAX words generated by the original algorithm are C_3C_7, C_3C_8 and C_4C_2. The differences among them are all zero. Thus, we can see that our algorithm maximizes the distances of the SAX words and minimizes the largest value range. Figure 4 (a) and (b) show a comparison and we can see that our symbol splitting scheme leads to a more balanced data classification result. It also produces a less number of nodes in the iTree which we will create from the iSAX hierarchy.

Bit Promotion and Reduction. Let us assume all the symbols s in a SAX word range from 0 to $\alpha - 1$ where α is the value dimension and $|\alpha| = \lceil \log \alpha \rceil$ is the bit cardinality. We write a SAX word with its symbol value and the corresponding bit cardinality in a particular form. For example, $0_2.2_2.6_3.3_2$ is a SAX word consisting of four symbols, 0, 2, 6 and 3, and the bit cardinalities are 2, 2, 3 and 2, respectively. We can also write the symbol values in bits, i.e., 00, 10, 110 and 11 for 0_2, 2_2, 6_3 and 3_2, respectively.

When comparing two SAX words, we actually compare their corresponding symbols in order. If the bit cardinalities of the two symbols are different, we can promote the bits with the smaller cardinality to the same as the larger one. Given two symbols s and t and the bit cardinality of s is less than that of t, we treat s as $s*$. There are three cases to consider. First, if s is the prefix of t, $*$ are the same as the corresponding bits in t for all unknown bits. Second, if s is lexicographically smaller than the corresponding bits in t, $*$ are all 1s for the rest unknown bits. Third, if s is lexicographically larger than the corresponding bits in t, $*$ are all 0s for the rest unknown bits. The guiding principle for these cases is to promote the bits in s so that s and t are closest in the SAX space. After bit promotion, we apply Equation 4 to compute the distance between these two SAX words.

Besides bit promotion, we can also reduce the bits of the symbol with the larger cardinality to the same as the smaller one. If the reduced SAX word is the same as the SAX word of the lower one, we consider these two SAX words as a match. In practice, *bit promotion* is used for SAX word comparison in search, while *bit reduction* is used for building the iSAX hierarchy.

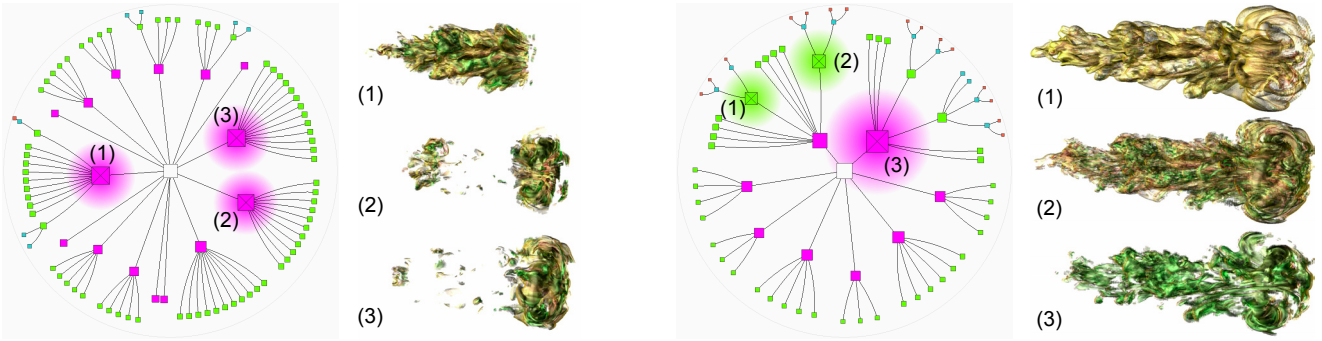
iSAX Construction. To construct the iSAX hierarchy, iSAX 2.0 [1] uses the raw TACs as input and loads the TACs into main memory till almost full. In main memory it converts each TAC into a SAX word, then places the SAX words into terminal nodes. At last it flushes them into disk and clears main memory. After each iteration, it continues to load the rest of TACs. Thus the benefits of iSAX 2.0 are that it reduces disk access, eliminates a second loading for each TAC, and provides the ability of handling large data sets. In our algorithm, we partition this process into two parts. First, we convert all the TACs to SAX words, then insert all SAX words one by one into the hierarchy. Our method not only offers the benefits of iSAX 2.0, but also increases the performance with GPU calculation and prevents system crashes by storing intermediate data into files.

As sketched in Figure 3, starting from the root whose children have the same property that the cardinality of each symbol is 1, we compare the input SAX word with these children. If the input SAX word matches one child c that is an internal node, we continue to check c's children in the next level of the hierarchy. Otherwise, we place the SAX word as a member of c. When the member of SAX words placed in a node c is larger than δ_n, we split c into two child nodes. In practice, we choose the symbol that covers the largest value range. For example, given a SAX word $2_2.3_2.3_2.1_2$, let us assume 2_2 covers a range of 0.5 while both 3_2 and 1_2 cover a range of 0.1. We split the SAX word into two: $4_3.3_2.3_2.1_2$ and $5_3.3_2.3_2.1_2$. Let us further assume 4_3 covers a range of 0.3 and 5_3 covers a range of 0.2. If we need to further split $4_3.3_2.3_2.2_2$, the two new SAX words will be $8_4.3_2.3_2.2_2$ and $9_4.3_2.3_2.2_2$. Finally, for each terminal node, we store all its corresponding voxel IDs into a file, using the SAX word itself as the file name to facilitate the subsequent search.

Acceleration Strategy. The above iSAX hierarchy construction algorithm has two limitations. First, each SAX word is inserted into the hierarchy in a sequential order, making it not amenable for parallel processing. Second, the entire iSAX hierarchy along with all voxel IDs needs to be stored in memory, making it not well scalable for handling large-scale data sets. We therefore propose an out-of-core algorithm for iSAX hierarchy construction to address these limitations. Our main idea is to first partition all voxels or groups into at most 2^w buckets, where w is the word length of their SAX words. We save each non-empty bucket into a file, corresponding to all the non-empty nodes in the first level from the root of the hierarchy. After that, we choose the file with the largest voxel/group count. If the voxel/group count is larger than δ_n, then we split it into two files following our symbol splitting scheme. We continue this process until there is no file which has its voxel/group count larger than δ_n. This algorithm performs file splitting in an out-of-core fashion, thus is more memory efficient. In practice, for each SAX word, we only need to load one symbol involved for splitting. In addition, this improved solution is flexible with the change of δ_n as only the splitting step need to be recomputed, which is not available with the previous solution.

3.3 Approximate Search and Exact Search

With the iSAX hierarchy built, we are able to perform approximate search and exact search. Both searches take the PAA representation of a voxel or voxel group and a threshold δ as the input. We find similar voxels or voxel groups with their distance to the input within δ. The approximate search converts the input PAA to a SAX word and compares each of the file names corresponding to the terminal



(a) (b)

Figure 4: (a) and (b) show the comparison of iTrees of the argon bubble data set generated using the original algorithm [12, 18] and our algorithm with new schemes for breakpoint identification and symbol splitting. Three main clusters are highlighted in the volume at time step 160.

nodes in the hierarchy with this SAX word. If the distance between the file name and the SAX word is larger than δ, we discard that file because the distance between two SAX words is the lower bound of their PAA's Euclidean distance. That is, if the SAX distance is larger than δ, then the Euclidean PAA distance must be larger than δ. Otherwise, we open the corresponding file and return all voxels or voxel groups in the file. The search result is the union of all voxels for all the files opened. The exact search needs an additional step. Instead of simply returning all voxels in the file which has a SAX distance within δ, we actually compute their PAA-based distances to the input PAA and only return those voxels that have a smaller distance less than or equal to δ.

4 iTree

The iSAX hierarchy is an *internal* representation of the time-varying data. We construct its corresponding *external* version of iTree for visual presentation and navigation. Unlike the VizTree [13] which draws the hierarchy as a regular tree, we utilize a hyperbolic graph drawing algorithm to draw the iTree and support interactive F+C visualization. Such a solution nicely organizes a large number of nodes in the iTree within a limited display area and allows the users to query the data in an adaptive manner. In conjunction with the intuitive SAX view, we enable coordinated multiple views to facilitate versatile exploration of time-varying data.

4.1 From iSAX Hierarchy to iTree

The original iSAX hierarchy we create is not readily suitable for visualization due to the following reasons. First, the number of non-empty children of the root is fairly large which easily leads to occlusion and clutter when drawing the hierarchy. Second, except for the first level from the root which has a very large fanout, all the internal nodes have exactly two children, which is a fairly small fanout. As a result, the iSAX hierarchy may have a very large number of levels which also makes the drawing of the hierarchy less effective for viewing and exploring. Third, sibling nodes are not arranged in the iSAX hierarchy according to their similarity. Therefore, we propose the following steps: *level promoting*, *sibling grouping* and *sibling reordering* to transform the iSAX hierarchy to the iTree suitable for drawing, navigation and query.

Level Promoting. We first promote each node in the iSAX hierarchy such that its new level in the hierarchy equals the maximal bit cardinality used for any symbol of its SAX word. For example, in Figure 3, nodes at the first level under the root remains the same. For the second level, we promote terminal nodes $\{01,10,10\}$, $\{01,10,11\}$ and $\{01,11,1^*\}$ to become the immediate children of node $[0^*,1^*,1^*]$. Accordingly, internal nodes $[01,1^*,1^*]$ and $[01,10,1^*]$ are removed. For this example, since the maximal bit cardinality used for any symbol in a SAX word is two, the height of the iSAX hierarchy after level promoting is three (i.e., three levels corresponding to 0 bit, 1 bit and 2 bits, respectively).

Sibling Grouping. After level promoting, for each non-leaf node, we use a hybrid k-means clustering algorithm [8] to group its children if the number of children n is larger than a certain threshold δ_n. The number of clusters is chosen as $\lfloor\sqrt{n}\rfloor$. We calculate the representative value for each SAX symbol with cardinality $|\alpha|$ as

$$v_i = \begin{cases} (v_{\min}+\beta_i)/2, & \text{if } |i|=0 \\ (v_{\max}+\beta_{i-1})/2, & \text{if } |i|=\alpha \\ (\beta_i+\beta_{i-1})/2, & \text{otherwise} \end{cases} \quad (5)$$

where $v_{\min}$ and $v_{\max}$ are the minimum and maximum values of the data set. β_i is the ith breakpoint. For a SAX symbol $C_{i.a}$ with value i and cardinality a where $a < |\alpha|$, we calculate its representative value $v_{(C_{i.a})}$ as follows

$$v_{(C_{i.a})} = (v_{i\ll(|\alpha|-a)} + v_{(i+1)\ll(|\alpha|-a)-1})/2, \quad (6)$$

where $\ll$ is the left shift operation, $|\alpha|-a$ is the difference between the maximum bit cardinality $|\alpha|$ and the current cardinality a. $i \ll (|\alpha|-a)$ and $(i+1) \ll (|\alpha|-a)-1$ indicate the smallest and largest value ranges that $C_{i.a}$ covers. For example, value 0 with cardinality 1 actually covers 0 (000) to 3 (011) with cardinality 3. Replace i as 0, $|\alpha|$ as 3, a as 1, we get $0 \ll (3-1) = 0$ and $(0+1) \ll (3-1)-1 = 3$ which are the smallest and largest value ranges that $C_{0.1}$ covers.

We define the distance between two SAX symbols as the distance between their representative values, and the distance between two SAX words as the average distance of their corresponding SAX symbols. For each cluster created, we identify a node that is closest to the centroid of the cluster as its representative. This process is performed in a top-down manner and we replace nodes in each non-root level with their representatives in the actual iTree drawing.

After level promoting and sibling grouping, the resulting iTree has the following two nice properties for drawing and querying. First, the height of the iTree is determined by the maximal bit cardinality for representing any symbol in the SAX words. Second, the iTree is balanced in the sense that any node does not have a large fanout. By transforming the iSAX hierarchy to the iTree, we maintain the connection between the internal and external representations (such as node correspondence) so that any further query of the iTree will still follow the underlying iSAX hierarchy correctly.

Sibling Reordering. Finally, we reorder sibling nodes under the same parent in the iTree based on the similarity of their SAX words. When neighboring sibling nodes are selected together, they will form a meaningful group because their corresponding voxels in the volume have a high degree of similarity in terms of spatial closeness and temporal trend. To get the optimal reordering, we need to compute the sum of distances for neighboring sibling nodes and identify the permutation with the smallest sum for every possible permutation of sibling nodes.

data set	volume dimension	group size	time interval	word length	quant. level	I/O time	PAA GPU time	BP CPU time	SAX GPU time
argon bubble	$640\times256\times256\times165$	$2\times2\times2$	33	10	16	28.27	0.10	9.82	2.09
combustion	$800\times686\times215\times53$	$2\times2\times5$	18	12	16	23.11	0.35	7.98	1.73
earthquake	$256\times256\times96\times599$	$4\times4\times3$	50	12	16	8.85	7.25	0.01	0.14
		$2\times2\times2$	60	10	16	11.00	7.05	8.07	0.67
		$1\times1\times1$	25	8	16	15.29	7.04	4.05	1.54
hurricane	$500\times500\times100\times48$	$2\times2\times2$	12	8	16	4.1	2.52	1.99	0.48
supernova (entropy)	$432\times432\times432\times60$	$2\times2\times2$	12	10	16	25.98	9.50	6.84	1.50
supernova (vel. mag.)	$864\times864\times864\times105$	$2\times2\times2$	7	10	32	1038.03	407.83	17.78	7.16

data set	δ_n	I/O time	bucketing GPU time	file splitting GPU time	# nodes for iSAX	# nodes for iTree	promoting CPU time	grouping CPU time
argon bubble	7500	242.55	1.19	0.20	1499	101	0.09	0.07
combustion	50000	596.66	0.98	6.05	4869	357	0.33	0.34
earthquake	5000	1.17	0.14	1.89	441	38	0.03	0.04
	1000	15.49	0.26	0.86	513	46	0.01	0.03
	50000	58.90	1.00	52.23	465	36	0.03	0.04
hurricane	100000	9.42	0.37	0.33	416	76	0.01	0.03
supernova (entropy)	50000	327.54	1.00	4.26	2048	387	0.11	0.08
supernova (vel. mag.)	500000	1381.17	3.92	21.54	1927	600	0.14	0.39

Table 1: Parameter values and timing results for generating SAX words, constructing iSAX hierarchy and iTree. All timing results reported are in seconds. The time for SAX word creation includes data I/O, PAA conversion, breakpoint (BP) identification and SAX word generation. The time for iSAX hierarchy construction includes data I/O, bucketing and file splitting. The time for iTree construction includes data I/O (negligible), level promoting, sibling grouping and sibling reordering (negligible).

data set	query location (x,y,z,t)	δ	BF time	exact time	approx. time	BF time	exact time	approx. time
			current interval			all time steps		
argon bubble	(99, 59, 59, 138)	0.103	0.44	0.23	0.14	12.10	6.81	0.13
combustion	(286, 152, 37, 10)	0.000002	1.14	0.34	0.18	45.77	8.22	0.18
earthquake, $4\times4\times3$ group size	(26, 52, 29, 66)	0.103	0.13	0.01	0.00	8.40	0.30	0.00
earthquake, $2\times2\times2$ group size	(51, 104, 45, 70)	0.094	1.92	0.04	0.01	76.54	6.20	0.50
earthquake, $1\times1\times1$ group size	(113, 204, 86, 68)	0.018	0.27	0.14	0.13	8.06	1.46	0.14
hurricane	(91, 31, 20, 48)	0.00071	0.41	0.29	0.07	13.53	11.01	0.06
supernova (entropy)	(108, 120, 97, 1312)	0.001	1.42	0.41	0.25	44.96	6.68	0.25
supernova (vel. mag.)	(191, 282, 153, 22)	0.015	3.20	2.42	2.08	49.27	22.83	2.23

Table 2: Timing performance comparison (in seconds) among brute-force (BF) search, exact search and approximate search running on a desktop PC. The timing reported included I/O time for reading PAA representation and voxel/group index files.

Since the complexity of optimal reordering follows a factorial growth, we opt for an approximate random swap solution for efficient handling a large number of sibling nodes. Random swap starts with an initial sibling ordering and randomly chooses two nodes to swap their positions. If the new ordering has a smaller sum of distances, then we replace this ordering with the new one; otherwise we keep the old ordering. Then, we randomly select another pair of nodes to swap. We continue this process for at most $2m^2$ iterations or until we have m consecutive random swaps without decreasing the sum of distances, where m is the number of sibling nodes.

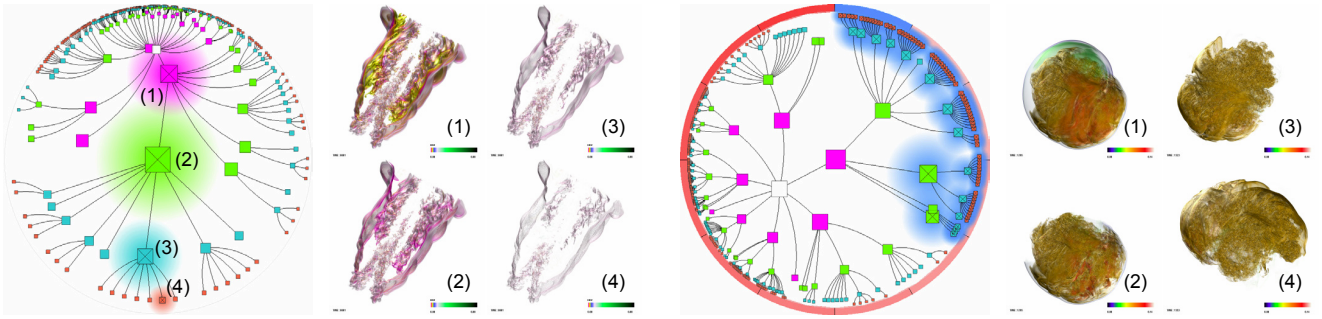
4.2 iTree Drawing and Focus+Context Visualization

We leverage the hyperbolic layout algorithm introduced by Lamping and Rao [9] to visualize the iTree. The hyperbolic layout organizes a given tree hierarchy within a 2D circle. Since a tree hierarchy tends to expand exponentially with depth, using a circular display provides exponentially more space along its radius to nicely accommodate the increasing number of nodes in successive levels of the hierarchy. To achieve this, the algorithm lays out the tree on the hyperbolic plane and then maps the structure to the Euclidean plane during the display. Compared to the conventional tree layout, the hyperbolic layout can display up to ten times more nodes within the same display region while providing more effective navigation, such as F+C visualization, around the hierarchy. Change of focus is achieved by changing the mapping from the hyperbolic plane to the Euclidean plane, which can be efficiently performed as node positions in the hyperbolic plane remain unchanged. It also allows for smooth blending of focus and context and continuous repositioning of the focus. There are two best known models to map the tree laid out on the hyperbolic plane to the Euclidean plane: the Klein model and the Poincaré model. The former preserves *straightness* of lines while the later preserves *angles* but maps lines in the hyperbolic space into arcs in the Euclidean space. We choose to draw arcs for a more pleasing visualization. Details about implementing the hyperbolic layout and F+C visualization can be found in [9].

4.3 Query in Multiple Coordinated Views

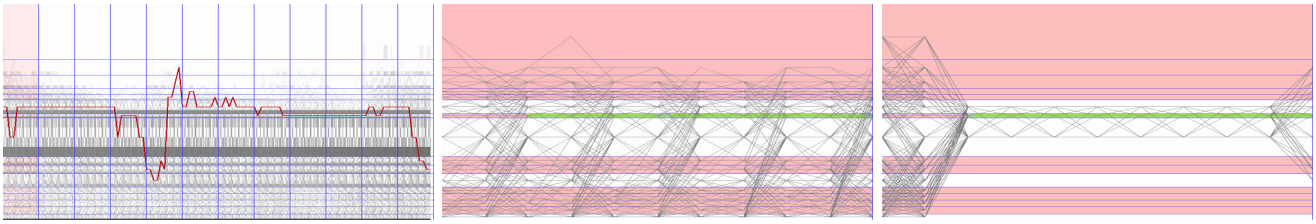
We dynamically link together the three views of the time-varying data, i.e., the volume view, the iTree view and the SAX view. The user interacts with the data in one view and the result is automatically reflected in the other view. To support effective exploration of time-varying data sets, we provide the following queries.

iTree Query. We allow the user to filter out nodes based on their levels in the hierarchy, or the number of voxels they contain. This helps reduce the clutter and facilitates the observation and exploration. Nodes on different levels in the iTree are distinguished using different colors. We map the size of a node to the number of descendants its contains, thus attracting the user's attention to those interesting nodes that are worth exploring. The user can select such a node and press keyboard shortcuts to explore its siblings or ancestors and descendants conveniently. Multiple nodes can be selected for a joint view. The corresponding selected voxels or data regions are highlighted in the spatial volume view. If the TACs are built based on time intervals instead of all time steps, animating over



(a) (b)

Figure 5: (a) level-of-detail exploration of the iTree of the combustion data set. We mark four nodes (1) to (4) at four different levels of detail in the iTree. The four images to the right show the corresponding clusters highlighted in the volume at the first time step. (b) exploring the supernova (entropy) data set using the iTree and volume views. We show the iTree with the corresponding nodes highlighted according to SAX filtering. (1) shows the full volume rendering at time step 1295. (2) to (4) show the corresponding volumetric region and the tracking results at three selected time steps: 1295, 1323 and 1353.



(a) overall density pattern of SAX words (b) zoom into the first time interval (c) SAX filtering result

Figure 6: Exploring the supernova (entropy) data set using the SAX view. (a) shows the overall density pattern of SAX words with a particular SAX word highlighted. (b) shows the zoom-in into the first time interval where user selections are highlighted in green. (c) SAX filtering shows the SAX words that are within a distance of 1.0 to the selected green regions.

time will reveal how the selected data clusters vary over space and time. In addition, when the user select a node, we draw a time ring to indicate the time intervals it covers.

SAX Query. Complementing the abstract iTree view, the SAX view allows the user to operate in an intuitive manner. We enable F+C visualization so that the user can closely exam the details in a particular value range or time interval. To combat the dense drawing of a large number of SAX curves, we draw quads with the binning of SAX value and time interval combination to show an overview of SAX curve distribution. The density (i.e., color saturation) show the number of curves falling into a bin. The user can brush bins to select SAX curves of interest and make connection to the volume view and the iTree view. We also show SAX curve clustering results so that the user can easily identify the different temporal trends exhibited in the time-varying data.

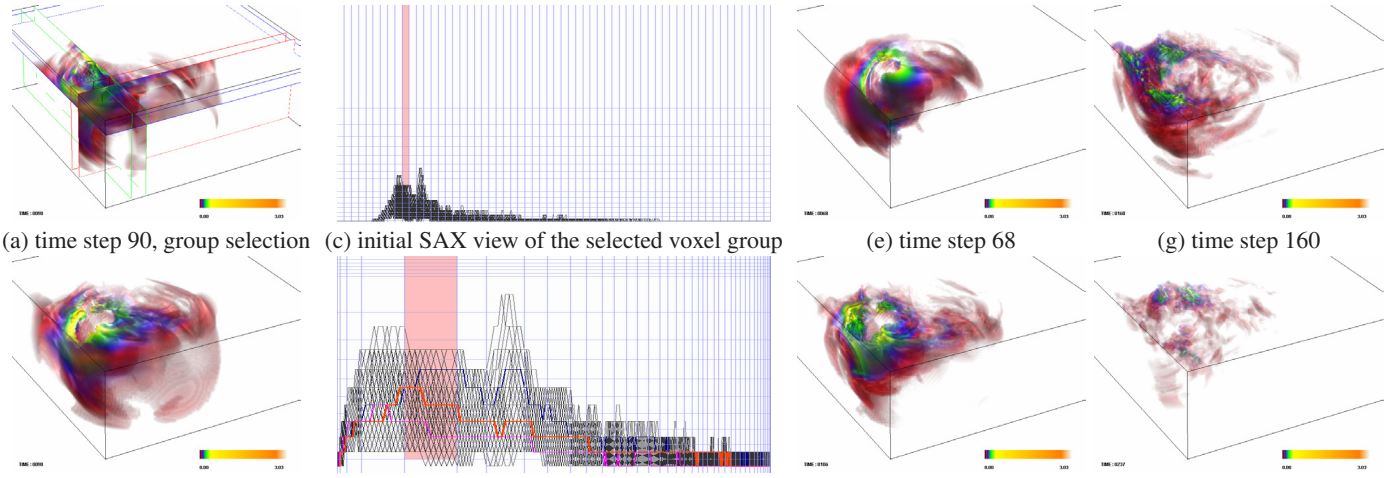
Volume Query. We allow the user to select a voxel or a group of voxels of interest from the volume at a certain time step (by bounding the ranges in the x, y and z directions) and search for similar voxels in the time-varying data. Two kinds of search using the iSAX hierarchy are supported: approximate search and exact search (Section 3.3). The search results can be highlighted in the iTree and in the volume. Again, the user can animate over time to reveal how the similar data distribute over space and time. Leveraging the iSAX hierarchy, indexing and searching the time-varying data is much faster than the brute-force solution without indexing.

5 Results and Discussion

Data Sets, Parameter Setting and Timing Performance. We experimented our approach with several time-varying data sets, as listed in Table 1. These data sets range from small (128^3) to large (864^3) in spatial extent and tens of time steps to hundreds of time steps in temporal extent. For SAX word generation, we allowed either a voxel-wise or group-wise setting. Splitting the entire time sequence into intervals was useful, leading to time-dependent clustering results. Choosing different word lengths and quantization levels affected the speed performance. A longer word length has a finer representation of temporal trend and a larger quantization levels has a finer representation of value range. This leads to a better discriminating power in the subsequent data clustering, indexing and searching while increasing the cost to process, store and search the reduced SAX/iSAX representations. Our experience shows that choosing 8 to 12 for word length and 16 or 32 for quantization level are appropriate for quality and speed tradeoff. For SAX hierarchy construction, the key parameter is the threshold for voxel/group count as it determines the time cost for file splitting. Normally, we chose tens of thousands as the threshold. A larger threshold leads to a smaller number of nodes produced for the iSAX hierarchy. For iTree construction, we were able to reduce the number of nodes to at least an order of magnitude smaller for the iTree. This process not only groups clusters together but also prepares for a more effective drawing, viewing and interacting with the iTree.

The timing was collected on a PC with an Intel Core i7-960 3.2GHz CPU, 24GB main memory, and an nVidia GeForce GTX 580 GPU with 1.5 GB graphics memory. For SAX word creation, breakpoint identification was performed in the CPU while PAA conversion and SAX word generation were performed in the GPU using CUDA. For SAX hierarchy construction, both bucketing and splitting were conducted in the GPU. Nevertheless, excluding I/O time, SAX hierarchy construction can be completed in less than 10 minutes for all data sets. For iTree construction, all tasks were performed in the CPU. Data I/O and sibling reordering tasks were negligible in terms of timing. The rest of two tasks can be completed within a second. All these three stages were done during preprocessing. At runtime, the drawing of and interaction with the iTree and the volume are interactive.

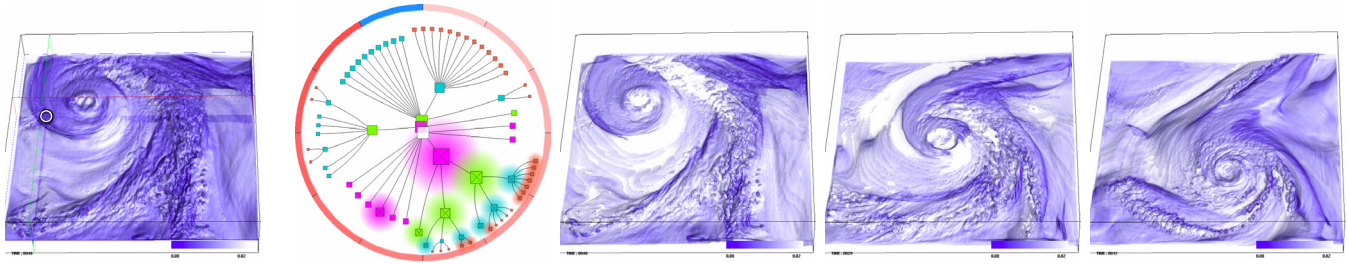
Data Classification and Tracking. The iSAX hierarchy essen-



(a) time step 90, group selection (c) initial SAX view of the selected voxel group (e) time step 68 (g) time step 160

(b) time step 90 (d) SAX cluster selection with F+C visualization (f) time step 106 (h) time step 237

Figure 7: Querying the earthquake data set. (a) a group of blocks at time step 90 are selected by bounding two slices along each of the x, y and z axes, respectively. (c) the initial SAX view of the group of blocks. (d) the F+C visualization with three clusters highlighted and the orange cluster selected. (b) and (e) to (h) are the query results where the SAX words are within a distance of 4.0 to the selected cluster at every SAX symbol.



(a) time step 48, block selection (b) iTree highlighting (c) time step 48 (d) time setp 29 (e) time step 12

Figure 8: Searching the iTree of the hurricane data set. (a) a block at $(x,y,z,t) = (91,31,20,48)$ is selected using three slices along the x, y and z axes, respectively. (b) and (c) show the exact search results with $\delta = 0.000711$. (d) and (e) are the results at other two selective time steps for the exact search with the same threshold.

tially classifies a time-varying data into multiple levels of detail based on the similarity of their trends. By exploring the iTree, we select nodes of interest and visualize the classification results in the volume view. Figure 4 (b) shows an example with the argon bubble data set. After exploring the child nodes of the root, we identify three nodes that correspond to three distinct regions in the volume. As we can see, these three regions are the outer, middle and inner layers of the bubble, respectively. We used one-dimensional transfer function where the scalar data value was mapped to color. The three regions classified have overlap in their data value ranges. Therefore, our iSAX-based classification actually goes beyond straightforward value-based segmentation. This result shows the nice capability of iTree in terms of classifying spatiotemporal data into meaningful groups for interpretation.

Figure 5 (a) shows the level-of-detail exploration of the iTree. In the actual interaction, as we move from the root to the leaf node, the selected node at higher levels of detail is displayed with a larger screen space in a F+C manner. The corresponding volumetric region highlighting shows the coarse-to-fine exploration of the combustion data set, starting from the main flame structure and narrowing down to a feature region at the boundary.

Since our iSAX hierarchy is created from a time-varying data organized into multiple time intervals, a node in the iTree corresponds to spatiotemporal regions that exhibit similar patterns or trends. Therefore, the iTree also enables us to track spatiotemporal regions over time. Figure 6 shows such an example with the supernova data set. In the SAX view shown in (a), we can see two dark value ranges with higher density values which indicates that these two ranges may contain interesting large-scale features. The vertical value ranges with non-zero opacity transfer function content are highlighted in pink in the first time interval. In (b), we zoom into the first time interval and brush some regions of interest. The corresponding SAX words that have a distance within 1.0 to the selected regions are displayed in (c). In Figure 5 (b), the corresponding iTree nodes are highlighted with halos. The time ring is also displayed. In the time ring, the first time step starts from the 12 o'clock direction and subsequent time steps follow the clockwise direction. Time intervals are marked in black and the current time interval is highlighted in blue. The corresponding volumetric regions satisfying the query are shown in (2) to (4) at three selected time steps. Compared to (1), we can see that the selected region actually corresponds to the main internal large-scale features of the supernova data set. The highlight results are consistent across different time steps to support meaningful tracking.

Data Indexing and Search. We conducted our data indexing and search by comparing brute-force search, exact search and approximate search. Brute-force search does not use any indexing scheme but simply goes over the PAA representation of data for identifying similar voxels or groups of voxel. We compared the performance for searching the current time interval and searching all time steps. The searching performance results with different data sets are shown in Table 2. The threshold δ selected for each data set is proportional to its value range. We can see that, in most cases, the time for brute-force search is much larger than that of ex-

act search under the same threshold, and the time for exact search is much larger than that of approximate search. Comparing searching the current time interval with searching all time steps, we find that usually the time cost does not increase much for approximate search, but does increase substantially for both brute-force and exact searches. This is because approximate search only involves using the names of index files for distance computation, while the other two searches need to examine the content in these index files.

In Figure 7, we show the querying of the earthquake data set using the SAX view. As shown in (a), a group of blocks are selected at time step 90 and their SAX words are displayed in the SAX view in (c). As shown in (d), using automatic F+C visualization based on the relative importance of each time interval (along the horizontal direction) and each value range (along the vertical direction), we are able to see more clearly the pattern of the corresponding SAX words. Next, we classify the SAX words using a k-mean clustering algorithm. Three clusters are displayed while we select one cluster of interest for further querying. The results of all SAX words that are within a distance of 4.0 to the selected cluster at every SAX symbol are highlighted in (b) and (e) to (h).

Figure 8 shows an example of the exact search with the hurricane data set. A block at a certain spatial position at time step 48 is selected in (a). We perform the exact search with the threshold $\delta = 0.000711$. The result captures the main hurricane structure that is similar to the selected voxel. Our experience shows that compared to approximate search, exact search gives a finer result since after finding all indexed files that match the query criteria, it also requires all voxels in the matched files to be compared with the PAA representation of the query voxel.

6 Conclusions, Limitations and Future Work

We have presented the iTree, a data organization, visual representation and user interaction framework for time-varying volume data analysis and visualization. Built on an indexable hierarchical structure, the iTree nicely combines data compacting, indexable data searching and querying with visually adaptive data exploration, a unique feature that makes our framework amenable for tackling large-scale time-varying data sets. We demonstrate the ability of the iTree in supporting fast data search and coarse-to-fine data classification and tracking. Our work has the following limitations. First, our breakpoints are identified based on the opacity transfer function. Thus when the opacity function changes, the iTree need be reconstructed due to the changes of breakpoints. Second, since our TACs are defined on data blocks, it is difficult to eliminate the block discontinuity in data classification. Third, the choice of parameters is important. For example, we have to choose a proper block size because too large blocks lead to serious blocky artifact in volume rendering while too small blocks lead to shorter word lengths and the iTree with fewer nodes. Nevertheless, as the size and complexity of scientific data sets continue to grow, we anticipate future visual analytics systems for scientific data equipped with such supports comparable to the iTree.

Acknowledgements

This work was supported in part by the U.S. National Science Foundation through grants IIS-1017935 and CNS-1229297. We thank Dr. Laura E. Brown for pointing out SAX/iSAX techniques which motivated us to conduct this research. We also thank the anonymous reviewers for their insightful comments.

References

[1] A. Camerra, T. Palpanas, J. Shieh, and E. J. Keogh. iSAX 2.0: Indexing and mining one billion time series. In *Proceedings of IEEE International Conference on Data Mining*, pages 58–67, 2010.

[2] Z. Du, Y.-J. Chiang, and H.-W. Shen. Out-of-core volume rendering for time-varying fields using a space-partitioning time (SPT) tree. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 73–80, 2009.

[3] Z. Fang, T. Möller, G. Hamarneh, and A. Celler. Visualization and exploration of time-varying medical image data sets. In *Proceedings of Graphics Interface*, pages 281–288, 2007.

[4] Y. Gu and C. Wang. TransGraph: Hierarchical exploration of transition relationships in time-varying volumetric data. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2015–2024, 2011.

[5] M. Imoto and T. Itoh. A 3D visualization technique for large scale time-varying data. In *Proceedings of International Conference on Information Visualisation*, pages 17–22, 2010.

[6] H. Jänicke, M. Böttinger, and G. Scheuermann. Brushing of attribute clouds for the visualization of multivariate data. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1459–1466, 2008.

[7] T. J. Jankun-Kelly and K.-L. Ma. Visualization exploration and encapsulation via a spreadsheet-like interface. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):275–287, 2001.

[8] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. A local search approximation algorithm for k-means clustering. In *Proceedings of ACM Symposium on Computational Geometry*, pages 10–18, 2002.

[9] J. Lamping and R. Rao. The hyperbolic browser: A focus + context technique for visualizing large hierarchies. *Journal of Visual Languages and Computing*, 7(1):33–55, 1996.

[10] T.-Y. Lee and H.-W. Shen. Visualization and exploration of temporal trend relationships in multivariate time-varying data. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1359–1366, 2009.

[11] T.-Y. Lee and H.-W. Shen. Visualizing time-varying features with tac-based distance fields. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 1–8, 2009.

[12] J. Lin, E. J. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of ACM SIGKDD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 2–11, 2003.

[13] J. Lin, E. J. Keogh, S. Lonardi, J. P. Lankford, and D. M. Nystrom. Viztree: A tool for visually mining and monitoring massive time series databases. In *Proceedings of International Conference on Very Large Data Bases*, pages 1269–1272, 2004.

[14] A. Lu and H.-W. Shen. Interactive storyboard for overall time-varying data visualization. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 143–150, 2008.

[15] K.-L. Ma. Image graphs - a novel approach to visual data exploration. In *Proceedings of IEEE Visualization Conference*, pages 81–88, 1999.

[16] P. O'Nell and D. Quass. Improved query performance with variant indexes. In *Proceedings of ACM SIGMOD Conference*, pages 38–49, 1997.

[17] H.-W. Shen, L.-J. Chang, and K.-L. Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree. In *Proceedings of IEEE Visualization Conference*, pages 371–377, 1999.

[18] J. Shieh and E. J. Keogh. iSAX: Indexing and mining terabyte sized time series. In *Proceedings of ACM SIGKDD Conference*, pages 623–631, 2008.

[19] K. Stockinger, J. Shalf, K. Wu, and E. W. Bethel. Query-driven visualization of large data sets. In *Proceedings of IEEE Visualization Conference*, pages 167–174, 2005.

[20] J. Wilhelms and A. V. Gelder. Multi-dimensional trees for controlled volume rendering and compression. In *Proceedings of IEEE Symposium on Volume Visualization*, pages 27–34, 1994.

[21] J. Woodring and H.-W. Shen. Multiscale time activity data exploration via temporal clustering visualization spreadsheet. *IEEE Transactions on Visualization and Computer Graphics*, 15(1):123–137, 2009.

[22] J. Woodring and H.-W. Shen. Semi-automatic time-series transfer functions via temporal clustering and sequencing. *Computer Graphics Forum*, 28(3):791–798, 2009.

[23] X. Xi, E. J. Keogh, C. R. Shelton, L. Wei, and C. A. Ratanamahatana. Fast time series classification using numerosity reduction. In *Proceedings of International Conference on Machine Learning*, pages 1033–1040, 2006.