Natural Event Summarization

Yexi Jiang
School of Computer Science
Florida International University
Miami, FL, 33199, USA
yjian004@cs.fiu.edu

Chang-Shing Perng
IBM T.J Watson Research Center
Hawthorne, NY, 10532, USA
perng@us.ibm.com

Tao Li
School of Computer Science
Florida International University
Miami, FL, 33199, USA
taoli@cs.fiu.edu

ABSTRACT

Event mining is a useful way to understand computer system behaviors. The focus of recent works on event mining has been shifted to event summarization from discovering frequent patterns. Event summarization seeks to provide a comprehensible explanation of the event sequence on certain aspects. Previous methods have several limitations such as ignoring temporal information, generating the same set of boundaries for all event patterns, and providing a summary which is difficult for human to understand.

In this paper, we propose a novel framework called *natural event summarization* that summarizes an event sequence using inter-arrival histograms to capture the temporal relationship among events. Our framework uses the minimum description length principle to guide the process in order to balance between accuracy and brevity. Also, we use multi-resolution analysis for pruning the problem space. We demonstrate how the principles can be applied to generate summaries with periodic patterns and correlation patterns in the framework. Experimental results on synthetic and real data show our method is capable of producing usable event summary, robust to noises, and scalable.

Categories and Subject Descriptors: H.2.8 [Database Management]: Database Applications-Data Mining

General Terms: Algorithms, Experimentation, Performance

Keywords: Event Summarization, Minimum Description Length, Wavelet Transformation

1. INTRODUCTION

Many system applications produce and process temporal events (i.e., event sequences with associated time-stamps), for example, system logs, HTTP requests, database queries, network traffic data, etc. These events capture system states and activities over time, and mining historical event data is a useful way to understand and optimize system behaviors. By examining event patterns, system administrators can set up event and incident management rules to eliminate or mitigate IT risks. It has become a standard way to manage large scale distributed systems in IT companies like IBM [13] and HP [3].

Most existing event mining research efforts focus on episodes mining or frequency pattern discovering [2, 16, 19]. These methods simply output a number of patterns that are independent to each other, so they fail to provide a brief and comprehensible event summary revealing the big picture that the dataset embodies. Obtaining insights from the huge number of patterns remains to be a daunting task for practitioners.

Instead of discovering frequent patterns, recent works on event mining have been focused on event summarization, namely how to concisely summarize temporal events [14, 15, 25]. Current state-of-art event summarization techniques are based on sequence segmentation where an event sequence is first split into disjoint segments and patterns are then generated to describe events in each segment. Kiernan and Terzi [14, 15] model the set segmentation problem as an optimization problem that balances between the shortness of the local models for each segment (i.e., the summary) and the accuracy of data description. Their method reveals the local patterns of the sequence but fails to provide the inter-segment relationships. Based on their work, [25] provides a more sophisticated method that not only describes the patterns in each segment but also learns a Hidden Markov Model (HMM) to characterize the global relationships among the segments.

1.1 A Motivating Example

Example 1. *Figure 1 shows an event sequence containing 4 event types, where event type A denotes "an event created by an antivirus process", event type B denotes "the firewall asks for the access privilege", event type C denotes "a port was listed as an exception", event type D denotes "the firewall operation mode has been changed". The approach presented in [25] segments the sequence into 4 segments based on the frequency changes of the events. For each segment, the approach further clusters the event types according to the frequency of each event type. Finally, an HMM is used to describe the transactions between intervals.*

The result of [25] can produce a comprehensive summary for the input sequence. However, these frequency-based methods have several limitations: (1) They only focus on the frequency changes of event types across adjacent segments and ignore the temporal information among event types within a segment. As a result, the generated summary

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.
Copyright 2011 ACM 978-1-4503-0717-8/11/10 ...$10.00.

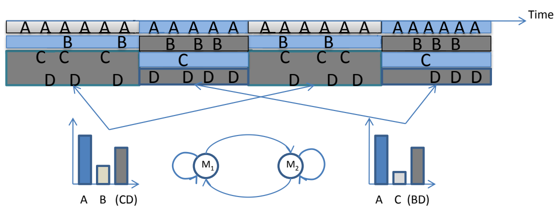


Figure 1: A motivating example. Event type A denotes "an event generated by antivirus monitor", event type B denotes "firewall asks for the access privilege", event type C denotes "a port was listed as an exception", event type D denotes "firewall operation mode has been changed".

fails to capture the temporal dynamics of event patterns. (2) They generate the same number of event patterns with the same boundaries for all event types. The unified pattern boundary requirement is impractical since different event types can have different underlying generating mechanisms. Consider in a distributed system, events may come from thousands of hosts that are not related to each other, forcing global segmentation means a simple event episode can be split to many segments because other events do not occur uniformly during the period; this yields a large number of small segments that may unnecessarily inflate the event summary. Having different boundaries for different event types could improve the event pattern identification and lead to better summaries. (3) Their generated summary (e.g., event distributions or HMMs) is difficult for system administrators to understand and take actions.

1.2 Our approach

In this paper, to address the above limitations, we take a novel approach called ***natural event summarization*** ***(NES** for short)* which first uses inter-arrival histograms to capture temporal relationships among same-type and different-type events, then finds a set of disjoint histograms to summarize the input event sequence based on minimum description length principle (MDL) [12], and finally represents the generated summary using event relationship networks. Using inter-arrival histograms allows for different boundaries for different event types. Moreover, inter-arrival histograms provide a conducive way to describe two main types of event patterns: the *periodic patterns* and the *correlation patterns*. These patterns capture the temporal dynamics of event sequences and can be used to generate the summaries. Many action rules can be derived almost directly from the generated summary. Figure 2 shows the output summary generated by our natural event summarization for Example 1. In the example event sequence, events of type C, D always appear after B for a short delay in both the second and fourth segments. Similarly, events of type C, D always appear after B for a long delay in both the first and third segments, Therefore, two correlation patterns: $B \to C, B \to D$ are identified. The change of periods implies that there may exist ill-configuration of the firewall. For events of type A, they appear regularly throughout the whole sequence, so all the instances of event type A belong to only one segment. Since event type A is the antivirus monitoring process event, its stable period shows that the antivirus process works normally. Note that in this paper we only consider pairwise cor-

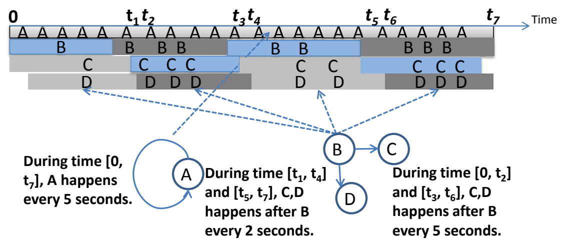

0
t1 t2
t3 t4
t5 t6
t7
Time
During time [0, t7], A happens every 5 seconds.
During time [t1, t4] and [t5, t7], C,D happens after B every 2 seconds.
During time [0, t2] and [t3, t6], C,D happens after B every 5 seconds.
A
B
C
D

Figure 2: Output summary of our proposed approach.

relations for events, since they are critical in understanding system behaviors. Discovering of correlation among three or more event types (two or more events trigger one event) is quite time-consuming and is one of our future works.

The framework of natural event summarization is shown in Figure 3. Our framework is based on inter-arrival histograms which capture the distribution of time intervals between events. These histograms provide a concise way to describe periodic patterns (of the same event type) and correlation patterns (of different event types). In this paper, we only focus on summarization on the aspect of temporal patterns of the events, since such information tells most of the story of the running status of the system.

The event summarization problem is formulated as finding a set of disjoint inter-arrival histograms, each representing a certain periodic pattern or correlation pattern, to approximate the original input sequence using MDL. MDL is an elegant theory to naturally balance the accurateness and the conciseness of the summarization information.

Although MDL has been used in previous event summarization methods for encoding the event segments, it is used here for encoding the inter-arrival histograms and for identifying the set of disjoint histograms for summarization. The problem of finding a set of disjoint histograms can be solved optimally in polynomial time by seeking a shortest path from the histogram graph that represents the temporal relations among histograms. To improve the efficiency of event summarization, we also explore an efficient alternative by using the multi-resolution property of wavelet transformation to reduce the size of the histogram graph. The final summary of our natural event summarization can be described as an easy-to-understand event relationship network (ERN) where many actionable rules are readily available.

In summary, our contributions are: (1) We propose a systematic and generic framework for natural event summarization using *inter-arrival histograms*. Through natural event summarization, an event sequence is decomposed into many disjoint subsets and well-fitted models (such as periodic patterns and correlation patterns) are used to describe each subset. Inter-arrival histograms demonstrate clear event patterns and capture temporal dynamics of events. To the best of our knowledge, this is the first work of using inter-arrival histograms for event summarization. (2) We formulate the event summarization problem as *finding the optimal combination of event patterns using MDL principle*. The usage of MDL principle automatically balances the complexity and the accuracy of summarization and makes our framework parameter free. (3) We present the *encoding scheme* for histograms and cast the summarization problem as seeking a shortest path from the histogram graph. (4) We also propose a *boundary pruning algorithm* that greatly accelerates

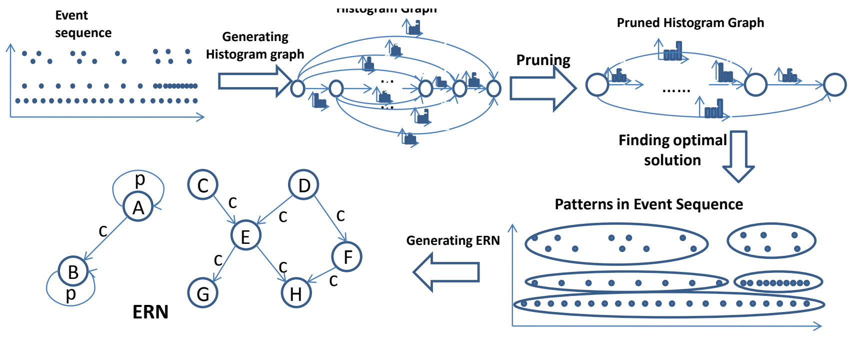


Figure 3: The framework of natural event summarization

the event summarization process by taking advantage of the multi-resolution property of wavelet transformation.

The rest of this paper is organized as follows. We present the interval histogram model for event summarization and its encoding scheme in Section 2 and 3 respectively. In Section 4, we introduce the algorithm of finding the best summarization solution. Then in Section 5, we present the segments boundary pruning algorithm that greatly improves the efficiency of our framework. In Section 6, we present our experiment evaluation. Finally, we discuss the related work in Section 7 and conclude in Section 8.

2. INTERVAL HISTOGRAMS

An event sequence D comprises of a series of event instances in the form of (e, t) ordered by their timestamps: $D = (< t_1, e_1 >, \cdots, < t_n, e_n >)$, where t_i is a timestamp and e_i is the type of the event. Each instance belongs to one of the m types $\mathcal{E} = \{e_1, \cdots, e_m\}$. We first describe the inter-arrival histograms used in our framework to represent the inter-arrival distribution of time interval between events.

Given two event types x and y, let S be the subsequence of D that only contains events of types x and y. Suppose S is split into k disjoint segments $S = (S_1, S_2, ...S_i, ..., S_k)$. We use an interval histogram (or inter-arrival histogram) $h_{xy}(S_i)$ to capture the distribution of time interval between events of type x and type y in S_i. Specifically, the bin $h_{xy}(S_i)[b]$ is the total number of intervals whose length is b. Let $next(t, y)$ denote the timestamp of the *first* type y event that occurs after t in S_i.

Definition 1. ***Inter-arrival histogram:***

$$h_{xy}(S_i)[b] = |\{i|e_i = x, next(t_i, y) - t_i = b\}|.$$

where t_i denotes the timestamp of e_i.

If $x \neq y$, then the inter-arrival histograms capture the time intervals for the events of different types; for the case of $x = y$, they capture the time intervals of events of the same type. Given an interval histogram, we can use a standard histogram to approximate it. The standard histogram is formally defined with definition 2.

Definition 2. ***Standard Histogram*** *is a special kind of interval histogram with one or two non-empty bin and all these non-empty bins have the same value $\frac{\#intervals}{n_{non}}$, where $\#intervals$ indicates the number of intervals and n_{non} indicates the number of non-empty bins. We use $\bar{h}_{xy}(S_i)$ to denote the corresponding standard histogram of $h_{xy}(S_i)$.*

Note that the two types of event relationships: *periodic patterns* and *correlation patters* can be easily described using standard histograms. The *periodic patterns* and *correlation patters* is formally defined in definition 3.

Definition 3. ***Periodic pattern and Correlation pattern:*** *A pattern is a 5-tuple (t_s, t_e, x, y, P), where (1) t_s and t_e denotes the start position and end position of the event sequence described by the pattern respectively. (2) x and y denote the types of events involved in the pattern, (3) P contains the periodic parameters. The pattern can contain 1 or 2 period parameters, which indicate the inter-arrival value between event x and y. Moreover, if $x = y$, this pattern is a* ***periodic pattern****, otherwise, it is a* ***correlation pattern****.*

Example 2 provides a detailed illustration about how to utilize standard histogram and periodic/correlation patterns to summarize a given event sequence.

Example 2. *Given an event sequence D, the timestamps for e_a, e_b and the related subsequences are listed in Table 1 (S for event type a, b is the same as D). There is only one segment in S and there exists a periodic pattern for e_a and a correlation pattern between e_a and e_b. The segment are described by inter-arrival histograms $h_{aa}(S)$ and $h_{ab}(S)$ in Figure 4 and are approximated by two standard histograms in Figure 5.*

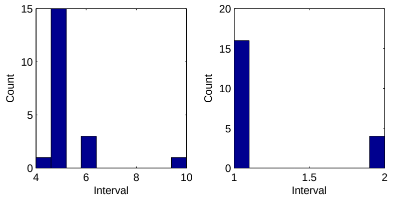


Figure 4: (a) inter-arrival histogram $h_{aa}(S)$; (b) inter-arrival histogram $h_{ab}(S)$.

Table 1: Occurrence of two events

Event type	Occurrence timestamp
a	5,11,16,21,26,30,35,41,46,51,61,66,71,76,81,86,92,97,102,107
b	6,12,17,22,28,31,36,43,47,52,63,67,72,77,82,87,93,99,103,108
S for $h_{aa}(S)$	The same as a
S for $h_{ab}(S)$	5,6,11,12,16,17,...,102,103,107,108

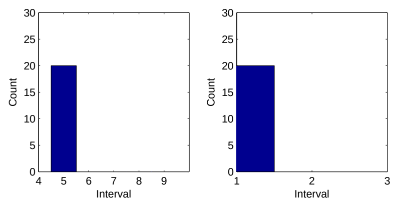


Figure 5: (a) standard histogram that best approximate $h_{aa}(S)$; (b) standard histogram that best approximate $h_{ab}(S)$.

It is obvious that the histograms demonstrate clear patterns. However, the total number of different histograms can be large for datasets with many events. The 2 histograms shown in Figure 4 and their corresponding standard histograms in Figure 5 are just one of many possible histogram combinations that depict the event relationships/patterns hidden in Example 2. For example, we can use two histograms to respectively depict the first and the second half of e_a. Hence one challenge is how to find the most suitable combination of histograms to describe the event sequence. In our framework, we use MDL principle to find out the best set of disjoint segments and the corresponding standard histograms to summarize the event sequence.

3. SUMMARIZATION USING MDL

In this section, we propose an information theoretic method to describe the histogram, i.e., encoding the histogram in *bits*. [22] proposed an encoding scheme for histograms with fixed number of bins and fixed number of bin elements (the total value of all bins). However, for the histograms used in event summarization, neither the number of bins and the number of bin elements is fixed beforehand. To the best of our knowledge, there is no encoding scheme that can be directly used in our scenario. Based on such situation, we first describe the scheme for encoding standard histograms (e.g., periodic patterns and correlation patterns). Then given an inter-arrival histogram, we show how it can be approximated using standard histograms. Finally, we formulate our event summarization problem. It should also be pointed out that, although MDL has been used in event summarization for encoding the event segments [14, 25], our work is the first one that uses MDL to encode the inter-arrival histograms and to identify the set of disjoint histograms for summarization.

3.1 Encoding Standard Histograms

Given an event subsequence S of event types x and y with disjoint segments $S = (S_1, S_2, ...S_i, ..., S_k)$, to encode a standard histogram $\bar{h}_{xy}(S_i)$, the following four components need to be encoded. These components are necessary and enough to describe the histogram.

Event type depicted by the histogram. Each histogram should be associated with one or two event types depending on the type of the relationship it depicts. Given the set of event types $\mathcal{E}$, it is enough to use $L(m) = \log|\mathcal{E}|$ bits to represent each event type.

Boundaries of S_i. The relationship depicted by an interval histogram has two boundaries indicating where the corresponding segment is. Each boundary requires $L(b) = \log|S|$ bits to encode its information.

Information of non-empty bins in the histogram. This piece of information can be encoded using

$$L(bin) = \log\delta + \log i_{max} + \log|S| + \sum_{i=1}^{n_{non}} \log i_{max} + \sum_{i=1}^{n_{non}} \log|S_i|.$$

The coding length consists of five terms. The first term uses $\log\delta$ bits to encode the largest interval i_{max} in S_i, where δ denotes the allowed largest interval. In our framework it is set as the number of seconds in one day. The second term uses $\log i_{max}$ bits to encode the number of non-empty bins n_{non}. The third term uses $\log|S|$ bits to encode the length of S_i. The fourth and fifth terms encode all the indices (1 to i_max) and the number of elements contained in the n_{non} bins respectively.

Putting them all together, the bits needed for encoding a standard histogram $\bar{h}_{xy}(S_i)$ is

$$L(\bar{h}_{xy}(S_i)) = L(m) + L(b) + L(bin). \qquad (1)$$

3.2 Encoding Interval Histograms

Given an inter-arrival histogram $h_{xy}(S_i)$, we want to measure how well it can be represented by event patterns, or equivalently, how well it can be approximated by standard histograms.

Histogram distance. Histogram distance describes how much information is needed to depict the *necessary bin element movements* defined in [6] to transform $\bar{h}(S_i)$ into $h(S_i)$ (To make notations uncluttered, we drop the subscripts xy and they should be clear from the context.). The code length of the distance can be calculated as:

$$L(h[S_i]|\bar{h}[S_i]) = \sum_{i \in Non} |be_i - bs_i| \log i_{max}, \qquad (2)$$

where Non is the union of the indices sets of non-empty bins in both histograms, be_i and bs_i denote the value (number of elements) of bin i in $h[S_i]$ and $\bar{h}[S_i]$ respectively. For each element at bin i, we can assign a new bin index to indicate where it should be moved. Equation 2 measures the bits of information needed by summing up the elements in unmatched bins.

In summary, the amount of information required to describe an inter-arrival histogram $h(S_i)$ using an event pattern $\bar{h}(S_i)$ equals to the summation of the code length for $\bar{h}(S_i)$ and the distance between $h(S_i)$ and $\bar{h}(S_i)$. Since there may be multiple standard histograms, we define the code length for an interval histogram $h(S_i)$ as follows:

$$L(h(S_i)) = argmin_{\bar{h}(S_i) \in \bar{H}(S_i)} L(\bar{h}(S_i) + L(h(S_i)|\bar{h}(S_i)), \qquad (3)$$

where $\bar{H}(S_i)$ is the set of all possible standard histograms on S_i.

3.3 Problem Statement

Given an event sequence D, for each subsequence S containing event types x and y, the minimum coding length $L(S)$ for S is defined as

$$L(S) = \arg\min_{\{S_1, S_2, ..., S_k\}} \sum_i L(h(S_i)). \qquad (4)$$

Since the boundaries for different subsequences are independent, then the minimum description length for the input event sequence D is

$$L(D) = \sum_{S \in D} L(S). \qquad (5)$$

Hence the event summarization problem is to *find the best set of segments $\{S_1, S_2, ..., S_k\}$ as well as the best approximated standard histograms to achieve the minimum description length.*

4. THE NES ALGORITHM

In this section, we first introduce a heuristic algorithm that can find the best set of segments of S in polynomial time. Then we present the method of generating ERN using the event patterns.

4.1 Finding the best segmentation

The problem of finding the best set of segments can be easily reduced to the problem of finding a shortest path from the generated histogram graph G. The histogram graph G is generated as follows:

1. Given S, let n_{xy} be the number of inter-arrivals between event x and y (x, y can be the same type) in S, generate n_{xy} vertices and label them with the positions of each x (1 to n_{xy}).
2. Add $edge(a, b)$ from vertex $v[a]$ to vertex $v[b]$ for each vertex pair $v[a], v[b]$, where $1 \leq a < b \leq n_{xy}$. Assign the weight of each $edge(a, b)$ as $L(h(S_i))$, where S_i starts at position a and ends at b. Note that to compute $L(h(S_i))$, we need to find the best standard histogram $\bar{h}(S_i)$ for $h(S_i)$ in the sense that $L(h(S_i))$ is minimized (as shown in Eq.(3). This can be done using a greedy strategy. Given $h(S_i)$, we can sort the bins in decreasing order based on their values. Then iteratively perform the following two steps: (1) generating $\bar{h}(S_i)$ using the top i bins, and (2) computing $L(h(S_i))$. (3) increase i by 1. The iteration continues till $L(h(S_i))$ begins to increases. The $\bar{h}(S_i)$ corresponding to the minimum description length $L(h(S_i))$ is often referred as the best standard histogram for $h(S_i)$.

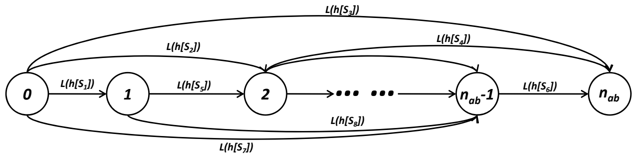


Figure 6: An example histogram graph

After generating the histogram graph, we can use the classical shortest path algorithm (e.g., *Dijkstra* algorithm) to output the vertices on the shortest path from $v[1]$ to $v[n_{xy}]$. Figure 6 shows an example histogram graph and Algorithm 1 illustrates the *summarization* process of our natural event summarization framework (*NES* for short). In line 4, the algorithm generates a set of event subsequences from D, there are m subsequences for the same event type (i.e., $x = y$) and $m^2 - m$ subsequences for different event types (i.e., $x \neq y$). Line 6 generates the directed acyclic histogram graph for each subsequence S and uses *Dijkstra* algorithm to find the shortest path $P = (v[i_1], v[i_2], ..., v[i_p])$. The best segmentation solution contains the segments $< v[i_1], v[i_2] >, < v[i_2], v[i_3] >, ..., < v[i_p - 1], v[i_p] >$. Line 8 and 9 represent each segment using the best fitted event patterns (i.e., the best standard histograms), and put them into the set $\mathcal{R}$.

Algorithm 1 The NES Algorithm

```
 1. input: event sequence $D$.
 2. output: all relationships $\mathcal{R}$.
 3. Identify $m$ from $S$, relationship set $\mathcal{R} \leftarrow \emptyset$;
 4. Separate $D$ into a set of $S$;
 5. for all $S$ do
 6.     Generate directed graph $G$;
 7.     Use Dijkstra($G$) to find shortest path $P$;
 8.     Generate relationships $R$ from $P$;
 9.     $\mathcal{R} \leftarrow \mathcal{R} \bigcup R$;
10. end for
11. return $\mathcal{R}$;
```

For each histogram graph G, *Dijkstra* algorithm requires $O(|E| + |V| \log |V|) = O(|S|^2)$ time. Therefore, the total running time is $O((m + m(m-1))|S|^2) = O(|D|^2)$.

4.2 Generating summarization graph

Although the set of event patterns can be described in text, we show that they can be used to construct an easy-to-understand event relationship network (ERN) [26] which produces a concise yet expressive representation of the event summary. ERN is a graph model that represents the relationship between event types. The procedure for building ERN is straightforward: it first generates the vertices for each event type involved in any event patterns, and then adds edges for event patterns and stores necessary information (e.g., segmentation, periods and time intervals) onto the edges.

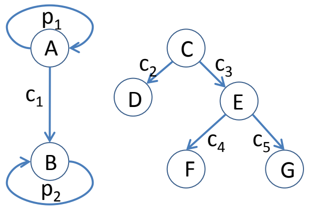


Figure 7: An example ERN graph.

Figure 7 shows an example ERN graph. It contains two periodic patterns: $A \xrightarrow{p_1} A$, $B \xrightarrow{p_2} B$ and five correlation patterns: $A \xrightarrow{c_1} B$, $C \xrightarrow{c_2} D$, $C \xrightarrow{c_3} E$, $E \xrightarrow{c_4} F$, $E \xrightarrow{c_5} G$. For simplicity, the ranges of segments are ignored.

5. EFFICIENCY IMPROVEMENT

The NES algorithm described in Section 4 finds the best segmentation by checking all positions in S, which is computational intensive. In fact, given an event subsequence S, boundaries should only locate at positions where inter-arrival times change rapidly, because segmentation at smooth places would waste encoding bitsj. Based on this observation, we propose an effective pruning algorithm which is able to prune a large part of boundaries that are unlikely to be the boundaries of segments. This algorithm greatly accelerates the summarization process by reducing the size of the histogram graph. By utilizing the multi-resolution analysis (MRA) of wavelet transformation, the pruning can be done in linear time in the worst case.

5.1 Preprocessing

For ease of pruning, some preprocessing steps are needed. Given an event subsequence S, we preprocess it as follows:

1. *Obtaining inter-arrival sequences*: given $S = (< e, t_1 >, \cdots, < e, t_{|S|} >)$, transform it into inter-arrival sequence $V = (t_2 - t_1, t_3 - t_2, \cdots, t_{|S|} - t_{|S|-1})$;
2. *Padding*: Append 0's to the tail of sequence until the length of interval sequence equals to a power of 2;
3. *Transforming*: Use *Haar* [4] wavelet as the the wavelet function and apply fast wavelet transformation (FWT) on V to obtain the transformation result W.

The resulting W contanis the wavelet coefficients of the inter-arrival sequence, which is the input of our pruning algorithm.

5.2 Pruning unlikely Boundaries

Due to the multi-resolution analysis (MRA) property of wavelet transformation, the deviation of inter-arrivals in V can be viewed from W at different resolutions. Since wavelet transformation captures both the resolution and the location information, for each $W[i]$, we can quickly locate the corresponding segment in V. For example, if $|V| = 1024$, the elements $W[1]$ and $W[2]$ contain the averaging information and the diff information of the highest resolution (in this example, resolution 1024) of original series respectively. The elements $W[3]$ and $W[4]$ contain the information of V at resolution 512. Specifically, $W[3]$ corresponds to the first half of V and $W[4]$ corresponds to the second half of V.

By taking advantage of such a property, our pruning algorithm identifies the inter-arrival deviation in a top-down manner. The basic idea of algorithm is as follows: it checks the element $W[i]$, starting from $i = 2$, which contains the deviation of inter-arrival for the whole sequence V. There are three possible cases:

Case I: If $W[i]$ is small, it means the corresponding subsequence in V for $W[i]$ is smooth enough and segmentation is not needed. In this case, the algorithm records the boundaries of the segment, and then stops.

Case II: If $W[i]$ is large, it means the deviation of inter-arrivals in the corresponding subsequence in V is too large. In this case, the subsequence needs to be split into two halves and the algorithm needs to perform recursively.

Case III: If $W[i]$ records the deviation of inter-arrivals at the lowest resolution, the algorithm just records the start and end boundaries and returns.

Algorithm 2 provides the pseudocode for *BoundaryPruning*. The parameter *level* (initialized as $\log |W|$) denotes the current level of resolution that the algorithm checks, and i denotes the current position in transformed result W (i.e. $W[i]$) to be checked. The algorithm calculates the corresponding threshold as $\frac{W[1]}{2^{\log |W| - level}}$, which reflects the average deviation of inter-arrival at resolution 2^{level} and changes dynamically according to the *level*.

Example 3 uses a small dataset to illustrate how the pruning algorithm works.

Algorithm 2 Algorithm of *BoundaryPruning*

```
1.  input: $W, level, i$.
2.  output: Boundary set $B$ after pruning.
3.  Set $B \leftarrow \emptyset$;
4.  $threshold \leftarrow \frac{W[1]}{2^{\log |W| - level}}$;
5.  $spectrum \leftarrow W[i]$;
6.  if $spectrum < threshold$ or reaches to the lowest resolution then
7.      Add corresponding boundaries to $B$;
8.  else
9.      $i_1 = 2i - 1, i_2 = 2i$;
10.     $B_1 \leftarrow$ BoundaryPruning($W, level - 1, i_1$);
11.     $B_2 \leftarrow$ BoundaryPruning($W, level - 1, i_2$);
12.     $B \leftarrow B \cup B_1 \cup B_2$
13. end if
14. return $B$;
```

EXAMPLE 3. *An input inter-arrival sequence V is shown in Figure 8. The algorithm starts from the highest resolution. It finds that $W[2] = 3$ is too large, so the whole inter-arrival sequence cannot be segmented with only one segment. At a lower resolution, the algorithm finds that $W[4]$ is small enough, so the corresponding subsequence $< 2, 1, 2, 1, 2, 1, 3, 1 >$ can be considered as one segment. For the element $W[3] = -2$ representing the first half of V, it is still too large and the corresponding subsequence $< 1, 1, 1, 1, 1, 3, 1, 1 >$ cannot be considered as only one segment. Hence the algorithm drills down to a lower resolution to do the same check task. Finally, the algorithm divides V into 5 segments, and 6 boundaries are recorded and the remaining 11 boundaries are pruned. Figure 9 shows the effect of BoundaryPruning in reducing the size of the histogram graph.*

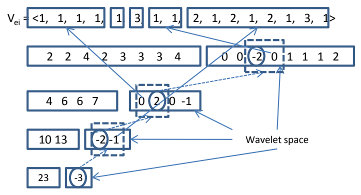


Figure 8: Segments information in wavelet spectrum sequence.

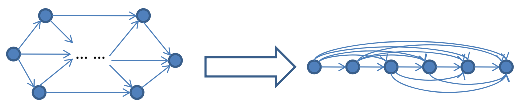


Figure 9: Before pruning, the histogram graph contains 17 vertices and 136 edges, after pruning, the histogram graph contains only 6 vertices and 14 edges.

The pruning algorithm only scans W once. In general cases, the algorithm does not need to check all the elements in W. If the inter-arrival sequence is smooth enough, the algorithm would stop checking at a high resolution. Only in the worst case, the algorithm has to check every elements in W. Since $|W| = |V| = |S| - 1$, the algorithm runs in $o(|S|)$ for the average case and $O(|S|)$ in the worst case.

6. EXPERIMENTAL EVALUATION

In order to investigate the effectiveness and efficiency of our proposed approach, we design two sets of experiments to evaluate our algorithm on both synthetic and real datasets. There are two reasons that we evaluate our algorithm on synthetic data:

1. There is no ground-truth summarization result for real world data, thus there is no way to evaluate the summarization result if we directly run our algorithm on real world data.

2. The real world event sequence is generated automatically by the operating system, there is no way to precisely control the size of the dataset. We cannot just pick a subset of the real world sequence with a fixed size, since it may destroy a lot of hidden patterns.

With synthetic data, we can freely set the parameters of dataset such as the number of patterns of each type, the location of each patterns, the level of noise, and the size of dataset. Since we know the ground-truth of the synthetic data, we are able to investigate the capabilities as well as limitations of our algorithm by comparing its output with the ground-truth.

Our tasks in synthetic data experiments part are trying to answer the following answers: (1) Can our event summarization framework indeed discover the hidden relationships from the event sequence and effectively summarize them? (2) Is our framework efficient in handling event summarization task? (3) Is the approximation algorithm precise enough that covers most of the relationships comparing with the optimal counterpart?

For the real world data experiments part, there are two tasks: (1) Does our algorithm find out any useful patterns from the real world data? (2) Is the summarization result more meaningful and friendly than the counterparts?

6.1 Experimental Setup

The whole framework along with the experiments code is implemented in Java 1.6. All the experiments are conducted on a machine with Intel Core2 CPU @2.83GHz, 2G main memory and running on Windows XP SP2. We use the open source library JWave[1] for wavelet transformation and Jung[2] for ERN visualization.

We preprocess the datasets by transforming all event instances into a predefined format $inst =< event\_type, date$, $time$, $source$, $category, event\_ID >$. Since different event types generated by different programs may share the same event_ID in some systems like Windows series operating system, it is not enough to distinguish the events just by their ID. We map each distinct tuple $< event\_type, source$, $category, event\_ID >$ as unique event type e.

6.2 Experiments on Synthetic Data

We generate several groups of synthetic datasets to evaluate our approach, each group consists a set of datasets generated by changing a particular parameter and fixing the remains. The meaning of each parameter is listed in Table 2. For each dataset, we intentionally plant some relationships/patterns and add the noise indicated by noise level [3] to simulate the real scenario.

[1]http://code.google.com/p/jwave/

[2]http://jung.sourceforge.net/

Table 2: Parameters and their meanings

Parameter	Description
n	Number of events in sequence.
m	Number of event types.
n_p	Number of periodic patterns.
n_c	Number of correlation patterns.
r_{noise}	Degree of noise.
l	Number of events in one pattern.

We use *NES-Prune* to represent summarization with *BoundaryPruning*, *NES* to represent summarization without *BoundaryPruning*.

Accuracy of the approaches. The first goal of experiments on synthetic data is to check whether our approach can really discover the correct hidden patterns from the data. We generate 5 datasets by fixing $n = 15000, m = 150, n_p = 50, n_c = 50, l = 100$ and iterate the noise levels from 0.1 to 0.5 with an increment of 0.1. Fore each dataset, we generated 50 periodic and 50 correlation patterns respectively with distinct period parameters and distinct event types. Then we randomly order these patterns to build the synthetic dataset. Table 3 shows the result of how many planted patterns are found via *NES* and *NES-Prune* respectively. In this table, NES_p and NES_c denote the proportion of planted periodic and correlation patterns found by *NES*, $NESP_p$ and $NESP_c$ denote those found by *NES-Prune*.

Table 3: Accuracy for synthetic datasets, $n = 15000, m = 150, n_p = 50, n_c = 50, l = 100$

No.	NES_p	NES_c	$NESP_p$	$NESP_c$
1	48/50	48/50	48/50	48/50
2	48/50	48/50	46/50	48/50
3	46/50	48/50	46/50	48/50
4	41/50	45/50	40/50	45/50
5	10/50	11/50	9/50	11/50

From the result, we observe that both *NES* and *NES-Prune* can discover most of the relationships. We only count the relationships that with the exact same event type, the exact periodical parameters and more than 95% overlap with the ground-truth patterns we plant in, so the criterion is quite demanding.

Compression ratio. We evaluate the quality of event summarization by compression ratio CR with the formula used in [15]: $CR(A) = \frac{L(A)}{L(direct)}$, where A is an algorithm, $L(A)$ denotes the code length achieved by A and $L(direct)$ denotes the code length of directly encoding the event sequence.

Figure 10 shows the compression ratio of our two algorithms as a function of *noiselevel*. For this experiment, we use the same datasets of the accuracy experiments. As expected, CR increases as the *noiselevel* increases. It is reasonable because more extra bits are required to describe the noise.

The proportion of planted patterns in the datasets is another factor that affects CR. We define the proportion of

[3]We set the noise levels as the probability of inter-arrival deviation, e.g, if noise level is 0.2, then with the probability of 20% the inter-arrival time will be shifted by a random value.

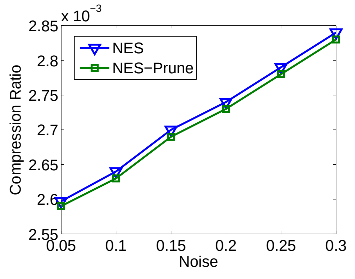


Figure 10: CR vs. noise, $n = 15000, m = 150, n_p = 50, n_c = 50, l = 100$

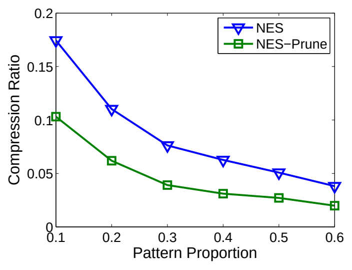


Figure 11: CR vs. Proportion of Pattern, $n = 5000, m = 50, l = 100, noise\ level = 0.1$

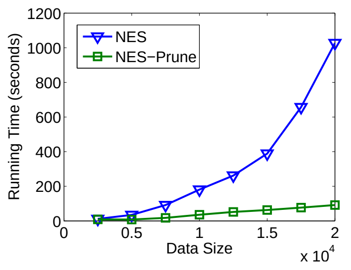


Figure 12: Scalability of *NES* and *NES-Prune*, $m = 60, l = 100, noise\ level = 0.1$

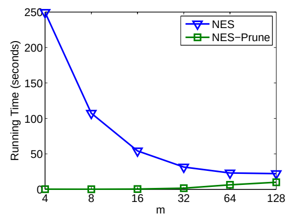
patterns as $\frac{|Patterns|}{|S|}$, where $|Patterns|$ denotes the number of events belonging to the injected patterns. We run the experiment on 6 datasets by fixing $n = 5000, m = 50, l = 100, noiselevel = 0.1$ and ranging the proportion of patters from 10% to 60% with an increment of 10%. Figure 11 shows that *CR* decreases as the pattern proportion increases. The result is straightforward because as the proportion of patterns increases, more events can be well fitted by event patterns, which results in shorter description length for the event sequence.

It should be pointed out that our *NES* summarizes an event sequence from different aspects simultaneously using event patterns with the goal of providing natural, interpretable and comprehensive summaries. It is quite possible that some events may belong to multiple correlation patterns and their coding length might be counted multiple times. So *CR* might not be the criterion to measure the quantity of summarization for our approach. This phenomenon also explains why *NES* leads to longer code lengths than *NES-Prune*.

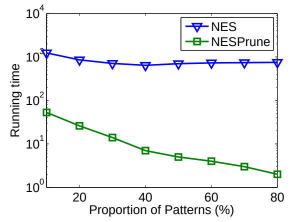
Performance and scalability. We generate 8 datasets by fixing $m = 100, n_c = 0, noiselevel = 0.1$ to evaluate the scalability of our approach. The length of event sequence for the datasets ranges from 2500 to 20000 with an increment of 2500, and n_p is set proportional to the length of sequence. Figure 12 shows the scalability results. As expected, *NES-Prune* runs much faster than *NES* and it shows better scalability, since *BoundaryPruning* prunes a large number of boundary candidates.

The number of event type m is an important factor that affects the running time, we use 6 datasets to do the evaluation by fixing $n = 4096, n_c = 0, noise\ level = 0.1$ and set m from 4 to 128 with an increment of a power of 2. Figure 13(a) shows that as m increases, the running time of *NES* decreases but the running time of *NES-Prune* increases. This is because the running time of *NES* procedure is $O(|D|^2) = O(m^2|S|^2)$. Since $S \gg m$ in general cases, for *NES*, the running time is dominated by $|S|$. By fixing n, the average length of S decreases as m increases, therefore the running time of *NES* decreases. For *NES-Prune*, since the input is pruned, the average length of S does not affect running time. Hence, the running time is dominated by m and would increase as m increases,

The proportion of patterns is another important factor that affects the running time. We generate 8 datasets by fixing $n = 5000, m = 50, l = 100, noiselevel = 0.1$ and ranging the proportion of patterns from 10% to 80%. Figure 13(b) shows that as the proportion of patterns increases, the running time of *NES-Prune* decreases, but the running time of *NES* is independent of the proportion of patterns. This is because as the proportion of patterns increases, more boundaries are removed using *BoundaryPruning*. And without *BoundaryPruning*, the size of input for *NES* is independent of the proportion of patterns.



(a) Running time vs. m, $n = 4096, noiselevel = 0.1$



(b) Running time vs. Proportion of Patterns, $n = 5000, m = 50, l = 100, noiselevel = 0.1$

Figure 13: Running Time Analysis

6.3 Experiments on Real Data

In order to explore whether our framework can really help system administrators, we test our approach with real log datasets recorded in Windows event viewer.

The real datasets consist of **application log**, **security log** and **system log**. The detailed information of the three datasets with their running times and compression ratios are listed in Table 4.

Our methods show a significant improvement over previous approaches. The algorithm in [15] needs more than a thousand seconds to find the optimal summarization solution while our methods just need 1/10 of the time. Due to inherent difficulty of event summarization, our methods are still not efficient enough. Further improvement in efficiency is needed and this is one of our future works.

Note that our algorithm utilize MDL to summarize the event sequence on the aspect of period pattern and correlation pattern, we successfully compressed the pattern information to just several percent of the original amount without loss any information.

Our algorithm finds and summarizes a lot of interesting event relationships. For example in system log, we find event <35, W32Time> occurs every 3600 seconds. This event type is a well known periodic event that synchronizes the computer time with the remote server. We also find that there

Table 4: Experiments with real datasets

	application	security	system
Period	09/10-02/11	11/10-12/10	09/10-02/11
Time range (secs)	12,005,616	2,073,055	12,000,559
Event instances	5634	21850	3935
Event types	100	17	50
Running Time (secs)			
NES	173	3102	108
NES-Prune	22	56	4
Compression Ratio $CR(A)$			
NES	0.0597	0.0312	0.0454
NES-Prune	0.0531	0.0216	0.0464

exist correlation relationships among the event types <6005, eventlog>, <6006, eventlog>, and <6009, eventlog>. The description on EventID.NET[1] verifies that such results are meaningful. Moreover, we find several important relationships from security log and application log. Figure 14 shows the whole ERN graph for the security log which includes 15 vertices and 36 edges. In this figure, the labels of vertices denote the event types and the labels on edges denote the relationships in form of $C[param1, param2]$, where C or P denotes the type of relationship and $param1, param2$ denote the time interval parameter of the relationship.

6.4 Application of Event Summarization

ERNs like the one shown in Figure 14 are immediately useful for managing computer systems. A system administrator can devise automatic event handling for each pattern type. For example, in the context of security monitoring, periodic event patterns with a short period like the one formed by event type 861 is usually caused by a malicious or ill-configured program that attempts to exploit security holes. Once such a pattern is identified, a security response process should be triggered. A correlation pattern usually represents an episode of an evolving problem going through various phases. One can devise proactive event-condition-action rules to prevent serious incidents from happening.

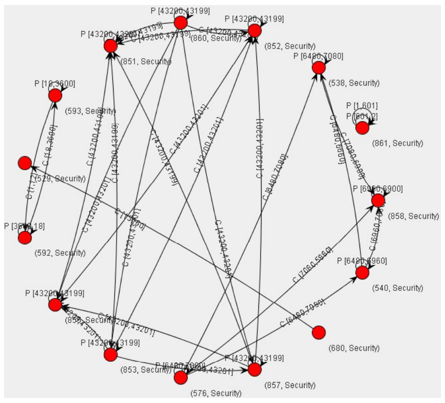


Figure 14: ERN for security log

Moreover, after we applying clique finding algorithm [5] to find all the cliques of this ERN by ignoring the directions, we further find larger patterns by combining the pairwise patterns. There are 4 sets of event groups found: Firewall related events (6 types), network access events (4 types), process management events (2 types) and antivirus monitoring events (1 type). The detailed information for discovered cliques are listed in Table 5.

[1]http://www.eventid.net

7. RELATED WORK

Event Summarization. Event Summarization is a relatively new research area that combines the area of data mining and computer systems. It can be deemed as an extension of frequent itemset mining [1, 7] and frequent episode mining [9, 16, 20, 21]. These frequent pattern mining techniques can reveal some interesting patterns by identifying the correlations of discrete events and are the building blocks of event summarization. Event summarization has attracted a lot of research attention recently [14, 15, 25, 24]. Peng et al. [24] proposed an approach to find the patterns in the event logs by measuring inter-arrivals of the events. Kiernan et al. [14, 15] proposed a summarization method by segmenting the event sequence according to the frequency changes of events. Based on the work of Kiernan, Wang et al. [25] further improved the summary by proposing a Hidden Markov Model to describe the transition of states among sequence segments. However, since they are either too general or too hard to be understood by data mining outsider, none of the results provided by these works are helpful enough for the system administrators.

Minimum Description Length. Minimum Description Length Principle (MDL) [12] is a general method for inductive inference. It is effective for generalization, data compression, and de-noising. In data mining community, it has been successfully used for temporal pattern mining [7], clustering [29], graph mining [10], trajectory outlier detection [17] and social network analysis [30]. Although MDL has been used in event summarization for encoding the event segments [14, 25], our method is the first one that uses MDL to encode the inter-arrival histograms and to identify the set of disjoint histograms for summarization.

Wavelet Transformation. Wavelet transformation is a useful tool for dividing up data, functions, or operators into different frequency components and then studies each component with a resolution matched to its scale [11, 18]. It has been widely used in many data mining tasks such as clustering [28], similarity search [27] and visualization [23]. The most relevant application to event summarization is wavelet-based time-series analysis [8, 27] where wavelet tools are used to measure the similarity of time-series in a global perspective. used in event summarization. In this paper, we use wavelet transformation as a subroutine to find the possible boundaries of the patterns hidden in an event sequence. We rely on its multi-resolution property to improve the efficiency of our approach.

8. CONCLUSION AND FUTURE WORK

In this paper, we propose a framework called *natural event summarization* and an algorithmic solution to the problem of summarizing event sequences that are generated by system logger. Our framework summarizes the event sequence from the perspective of depicting the event patterns by capturing the temporal relationships among same-type and different-type of events. The framework finds the best set of disjoint histograms to summarize the input event sequence based on MDL. Moreover, to improve the efficiency, we present a heuristic boundary pruning algorithm to prune

Table 5: Cliques of event relationships and their interpretations

Size	Event Types	Interpretation (After matching clique back to original ERN)
6	(851, security), (860, security) (852, security), (850, security) (853, security), (857, security)	This clique is related to the firewall actions. At first, event 860 (The windows firewall has switched the active policy profile) is triggered, then the other five types are triggered by 860.
4	(576, security) , (540, security) (538, security), (858, security)	This clique is related to the operation of network accessing. At first, event 76 (Special privileges assigned to new logon) is triggered, then event 540 (Logon successful) is triggered, and then event type 538 (User logoff) and 858 (Unknown) happens afterward.
2	(592, security), (593, security)	This clique is related to the creation and destroy of process. Event type 593 (A new process has been created) and type 593 (A process has exited) occur alternatively.
1	(861, security)	This clique is related to the actions of firewall. Event type 861 (The windows firewall has detected an application listening for incoming traffic) appears periodically. This is because the antivirus scanner installed on this machine continuously monitors the ports.

the unlikely boundaries and reduce the size of the histogram graph. Finally, we use ERN to present the final event summary in a more interpretable way.

There are several directions for the future work. Firstly, there is still a long way to go for the efficiency improvement. Secondly, one limitation of our current solution is the fixed time interval across time. In the future, we should also consider the pseudo temporal patterns for summarization. Moreover, many of the event logs in modern distributed systems are in form of streams, but our current event summarization is designed based on the static event log. One immediate goal is to design a new framework that is able to handle stream event sequences. Finally, we plan to build a whole event summarization system, which includes modules like event log structure extraction module, event summarization module, summarization postprocess module and summarization visualization module.

Acknowledgements

The work is partially supported by NSF CAREER grant IIS-0546280.

9. REFERENCES

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. *Fast Discovery of Association Rules*, pages 307–328. AAAI/MIT Press, 1996.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, 1995.

[3] M. Aharon, G. Barash, I. Cohen, and E. Mordechai. One graph is worth a thousand logs: Uncovering hidden structures in massive system event logs. In *PKDD*, 2009.

[4] Haar A.Zur. Theorie der orghogonalen funktionensysteme. *Mathematische*, 69:pp331–371, 1910.

[5] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16:575–577, 1973.

[6] S. Cha and S.N. Srihari. On measuring the distance between histograms. *Pattern Recognition*, 35:1355–1370, 2002.

[7] S. Chakrabarti, S. Sarawagi, and B. Dom. Mining surprising patterns using temporal description length. In *VLDB*, 1998.

[8] K. Chan and A. Fu. Efficient time series matching by wavelets. In *ICDE*, 1999.

[9] D. Chudova and P. Smyth. Pattern discovery in sequences under a markov assumption. In *KDD*, 2002.

[10] D.J. Cook and L.B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

[11] I. Daubechies. *Ten Lectures on Wavelets*. Captital City Press, 1992.

[12] P.D. Grunwald. *The Minimum Description Length Principle*. MIT Press, 2007.

[13] J.L. Hellerstein, S. Ma, S. M. Weiss, R. Vilalta, and C. Apte. Predictive algorithms in the management of computer systems. *IBM Systems Journal*, 4, 2002.

[14] J. Kiernan and E. Terzi. Constructing comprehensive summaries of large event sequences. In *KDD*, 2008.

[15] J. Kiernan and E. Terzi. Constructing comprehensive summaries of large event sequences. *ACM Transactions on Knowledge Discovery from Data*, 3:1–31, 2009.

[16] S. Laxman and K.P. Unnikrishnan P.S. Sastryand. Discovering frequent episodes and learning hidden markov models: A formal connection. *IEEE Transactions on Knowledge and Data Engineering*, 17:1505–1517, 2005.

[17] J.G. Lee, J. Han, and X. Li. Trajectory outlier detection: A partition-and-detect framework. In *ICDE*, 2008.

[18] T. Li, Q. Li, S. Zhu, and M. Ogihara. A survey on wavelet applications in data mining. *ACM SIGKDD Explorations*, 4:pp49–68, 2002.

[19] T. Li, F. Liang, S. Ma, and W. Peng. An integrated framework on mining logs files for computing system management. In *KDD*, 2005.

[20] S. Ma and J.L. Hellerstein. Mining partially periodic event patterns with unknown periods. In *ICDE*, 2001.

[21] H. Mannila and M. Salmenkivi. Finding simple intensive description from event sequence data. In *KDD*, 2001.

[22] S. Marsland, C. J. Twining, and C.J. Taylor. A minimum description length objective function for groupwise non-rigid image registration. In *Image and Vision Computing*, 2008.

[23] N.E. Miller, P.C. Wong, M. Brewster, and H. Foote. Topic islands - a wavelet-based text visualization system. In *IEEE Visualization*, 1998.

[24] W. Peng, C. Perng, T. Li, and H. Wang. Event summarization for system management. In *KDD*, 2008.

[25] W. Peng, H. Wang, M. Liu, and W. Wang. An algorithmic approach to event summarization. In *SIGMOD*, 2010.

[26] C. Perng, D. Thoenen, G. Grabarnik, S. Ma, and J. Hellerstein. Data-driven validation, completion and construction of event relationship networks. In *KDD*, 2003.

[27] I. Popivanov and R.J. Miller. Similarity search over time-series data using wavelets. In *ICDE*, 2002.

[28] G. Sheikholeslami, S. Chartterjee, and Aidong Zhang. Wavecluster: a wavelet-based clustering approach for spatial data in very large databases. *VLDB Journal*, 8, 2000.

[29] R.S. Wallace and T. Kanade. Finding natural clusters having minimum description length. In *Pattern Recognition*, 1990.

[30] K. Xu, C. Tang, C. Li, Y. Jiang, and R. Tang. An mdl approach to efficiently discover communities in bipartite network. In *DASFAA*, 2010.