Discovering actionable patterns in event data

by J. L. Hellerstein
S. Ma
C.-S. Perng

Applications such as those for systems management and intrusion detection employ an automated real-time operation system in which sensor data are collected and processed in real time. Although such a system effectively reduces the need for operation staff, it requires constructing and maintaining correlation rules. Currently, rule construction requires experts to identify problem patterns, a process that is time-consuming and error-prone. In this paper, we propose reducing this burden by mining historical data that are readily available. Specifically, we first present efficient algorithms to mine three types of important patterns from historical event data: event bursts, periodic patterns, and mutually dependent patterns. We then discuss a framework for efficiently mining events that have multiple attributes. Last, we present Event Correlation Constructor—a tool that validates and extends correlation knowledge.

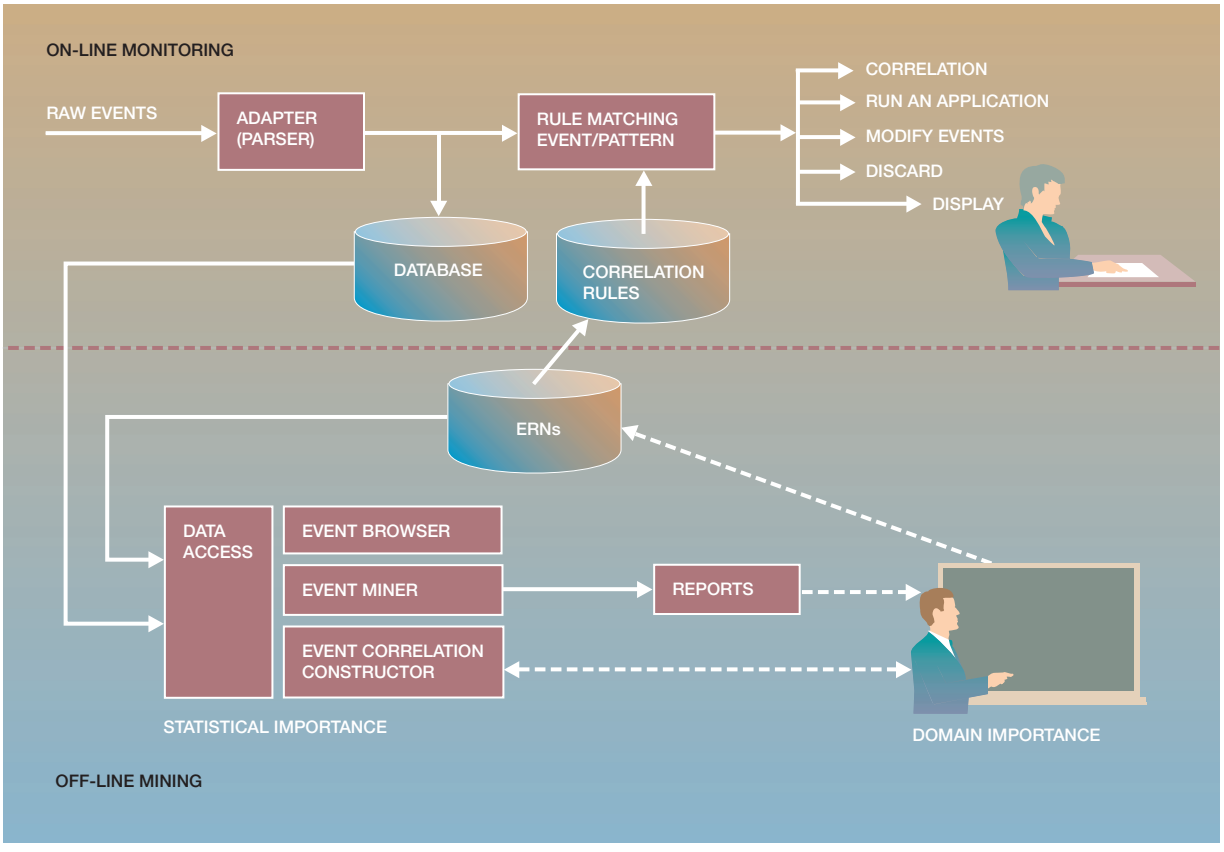
With advances in computer technology and sensor technology, various applications employ automated real-time, rule-based systems in which application-critical data are collected from various sensors and are processed and correlated based on predefined rules for identifying problems, diagnosing their root causes, and taking corrective action. Such applications include those used for computer availability and performance management, intrusion detection, and other surveillance tasks. In this paper, we focus on the off-line data analysis for constructing and maintaining correlation rules, and we apply our techniques to systems management tasks.

Figure 1 illustrates the event management task for a complex computer system and our vision of how it can be improved. The area above the dashed line depicts a typical on-line monitoring system such as Tivoli's TEC[1] and CA's Unicenter.[2] A raw event is generated when the state of a device changes (e.g., the router's interface goes down) or an exceptional event occurs (e.g., CPU utilization goes above 90 percent). Raw events generated by various devices and sensors flow into the event management system for automated operation (e.g., Reference 3), in which raw events are parsed and stored. Then a correlation engine uses correlation rules to interpret these events. Some events are filtered, others are coalesced. Some of them trigger notification mechanisms, such as alarms and e-mails, and some trigger automatic recovery actions. Correlation rules are structured as if-then statements. An example of a correlation rule is the following:

> If a "network card failure" event is followed by an "interface failure" event occurring on a router, within a five-minute window, then send an e-mail to the network support supervisor.

The condition, or *if* part, describes a situation in which actions are to be taken. The action, or *then*

Figure 1 On-line monitoring and off-line event mining



part, details what is to be done if the conditions of the if part are satisfied. In our example, the action to be taken is sending an e-mail for notification.

The if part of a rule is usually difficult to construct, because we must know the situation to be addressed before an action can be identified. Two broad approaches are used to specify the condition part of rules. The first approach is based on models, such as exploiting topology information to make inferences about connectivity (see, for example, Reference 4). Even so, there remains a broad range of problems that cannot be modeled easily. The second approach is a knowledge-based approach, in which human experts are consulted for constructing correlation rules. For example, Thoenen et al.[5] developed an event management and design methodology that has been widely used by IGS (IBM Global Services) consultants over the past three years. The core of this methodology is a graphic representation called Event Relationship Network (ERN) that describes how events are correlated. For example, an ERN graph can be used to represent the if part of our previous example, in which "network card failure" and "interface failure" are drawn as two nodes, and a link between these two events indicates that these two events are correlated. More details are found in the section "ERN constructor," later in this paper. Clearly, ERNs provide a means for an expert to design correlation rules graphically at a higher level, independent of a specific correlation engine. In current practice ERNs are constructed manually and may not always be complete or correct.

We describe in this paper an alternative approach—data-driven rule construction—in which we apply data mining in order to identify patterns used in constructing correlation rules. The lower part of Figure 1 illustrates the three components involved: Event Browser, Event Miner, and Event Correlation Con-

structor. Event Browser[6] provides an interactive environment for a user to visualize and explore large volumes of event data. Event Miner[7] employs data mining techniques to automatically search for patterns. Event Correlation Constructor takes as inputs existing ERNs, if any, as well as the results of Event Miner, and produces annotated ERNs that will be further presented to domain experts to review and construct correlation rules. Further, these tools are integrated to provide (A) seamless operation and early insights through visualization, (B) pattern discovery through event mining, and (C) assistance in knowledge representation of ERNs. Preliminary results have been reported previously. References 6 and 7 address the system aspect of Event Browser and Event Miner. Reference 8 discusses our early results of pattern discovery. References 9 and 10 describe several case studies. This paper provides a more systematic treatment of the problems we address and the algorithms employed. We also discuss how our data mining approach can be used together with traditional knowledge-based approaches.

Specifically, we discuss pattern discovery from large data sets of historical events. We define several types of patterns that are pertinent to systems management and propose algorithms for efficiently discovering those patterns. We note here that there are some key differences between our work and previous work, especially regarding frequent pattern discovery and frequent association rules[11,12] in data mining. In applying data mining to systems management tasks, we face some fairly unique challenges. For example, severe operation problems are of great interest in this domain. However, they do not occur frequently enough in well-maintained production environments. Hence the popular *frequent itemset* mining requires different types of patterns that also address important but less frequently occurring phenomena. Furthermore, event data usually have multiple attributes, usually from six to thirty. There is no apparent way to transform event data to transactional data. Hence this motivates us to propose a type of pattern that explores all possible selections of attributes for grouping and itemizing events.

We also discuss knowledge validation and completion from historical event data. This is important because considerable domain knowledge may be required to correlate event data. For example, ERNs provide a graphic representation of event correlation. To utilize existing correlation knowledge while exploring historical data, we develop a mixed system that is capable of validating existing knowledge as well as constructing new knowledge.

The remainder of the paper is organized as follows. In the next section we describe traditional data mining and related work as they pertain to mining events. Then we provide a motivating example and discuss the unique aspects of event mining. In the section that follows, we focus on the three main data patterns and the algorithms used for their discovery. In the next section we present a unified framework to handle multiple attributes. In the section "ERN Constructor" we develop our method for validating correlation knowledge. Our conclusions are contained in the last section.

Overview of data mining

In this section we provide a brief overview of data mining as it pertains to the analysis of event data. We begin by describing the market-basket analysis, the context in which data mining was first proposed. Then, we discuss efficiency considerations, a topic of particular importance given the large size of event histories.

Market-basket analysis. Market-basket analysis[13] originates from analyzing data from supermarkets, in which each supermarket customer has a basket of purchased goods. The main goal is to find association rules, according to which purchasing a set of items indicates that another set of items is likely to be also purchased. For example, as one early study found, when diapers are purchased, beer is frequently purchased as well. Association rules indicate a one-way dependency. For example, purchasers of beer are, in general, not particularly inclined to buy diapers.

We introduce some notations. Let I be the set of items that can be purchased. Thus, each market basket contains a subset of these items. We use $T = \{T_i\}_{i=1}^{N}$, where the i th transaction $T_i \subseteq I$, to denote a set of N market baskets.

A key data mining problem is to find sets of items, typically referred to as itemsets or patterns, with occurrences above a predefined threshold called minimum support (*minsup*). A second and closely related problem is prediction, in which we are looking for patterns that have a high probability of predicting that a given item will be in the same basket. The metric used is "confidence," and it is expressed as a conditional probability.

Generic level-wise search algorithm. A naive approach of frequent itemset mining is to exhaustively examine every possible pattern. This is computationally infeasible because the search space is huge, $O(n^k)$ to be precise, where n is the number of distinct items and k is the maximal length of an itemset. It is not uncommon that n can be 1000 or more. Thus, this brute-force search is computationally intractable even for modest k values.

Fortunately, the search for frequent patterns can be made more efficient. Doing so rests on the following observation: The support for a pattern E can be no greater than the support for its subset. Put differently, if one subset of E is not frequent, then E cannot be frequent. This means that if we find a pattern with low support, there is no need to consider any pattern that contains that pattern. This property, referred to as downward closure property[14] or antimonotonicity,[15] guarantees that we will not miss any frequent pattern while eliminating search space based on the current level. A level-wise search algorithm called *apriori*[11,13] was developed to efficiently discover frequent patterns from large market-basket data. Clearly, such a level-wise algorithm can be generalized to discover any type of pattern satisfying the downward closure property. Algorithm 1 shows a generic level-wise algorithm.

Algorithm 1: Generic level-wise algorithm

Input: Transaction data T
Output: All patterns

1. Find all qualified patterns of size 1: Q_1; $i = 2$.
2. C_i = candidate patterns at level i based on qualified patterns at level $i - 1$.
3. Scan data to obtain the count of each candidate pattern in C_i.
4. Find qualified patterns at level i. $Q_i = \{c | c \in C_i$ and c is qualified$\}$.
5. $i = i + 1$; go to (2) until there is no more qualified pattern.

A typical level-wise algorithm has four steps. The initialization step finds qualified patterns with the smallest size. This is often done by exhaustive search. Then, the candidate patterns (C_i) are constructed by using qualified patterns in the previous level. This can be done through joining patterns in C_{i-1} followed by a pruning operation.[13] Next, data are scanned to count instances of candidate patterns. Some care is needed here to efficiently count candidate patterns. See Reference 13 for details. Last, the qualified patterns (Q_i) are found by checking the qualification condition. For frequent patterns, we need only to check whether the count of a candidate is above *minsup*.

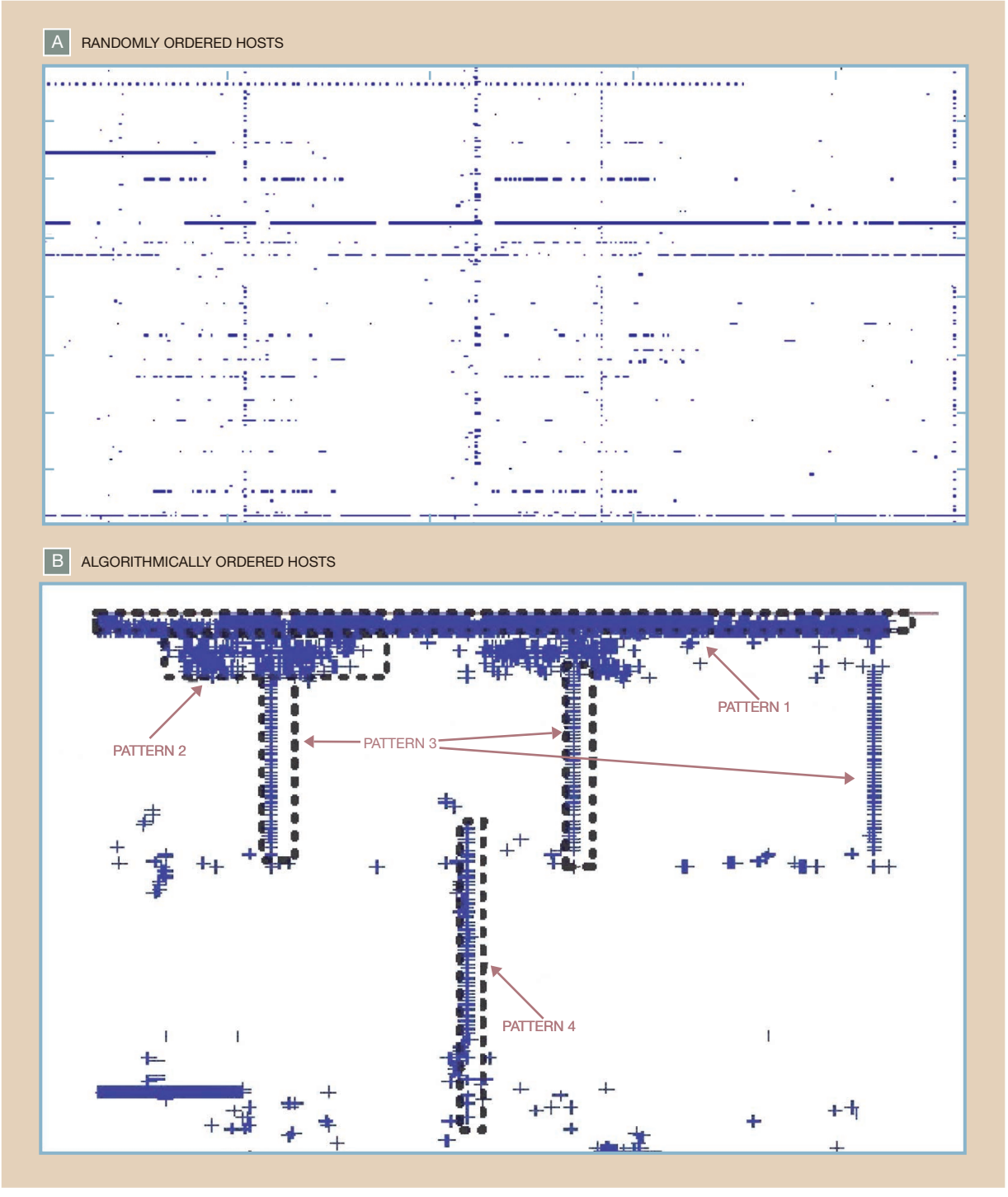
We note that the downward closure property holds for some patterns and not for others. In particular, downward closure does not hold for the confidence of association rules and neither does the χ^2 test. Also, we note that although we focus on level-wise search algorithms in this paper, much work has been developed to improve the algorithm by exploring different search strategies (see References 16–18 and references therein). All this work is built on the downward closure property of frequent itemsets.

Related work. Our approach makes use of data mining. Data mining is a mixture of statistical, machine learning, and data management techniques that provides a way to mine categorical data so as to find *interesting* combinations (e.g., the condition cold start trap is often preceded by the condition CPU threshold violated). Considerable work has been done in mining transactional data (e.g., supermarket purchases), much of which is based on References 11 and 13. For event data, time plays an important role. Follow-on research has pursued two directions that address this requirement. The first, sequential data mining (see for example Reference 19), takes into account the sequences of events rather than just their occurrence. The second, temporal data mining (e.g., Reference 12), considers the time between event occurrences. Data mining has been applied to numerous domains. Reference 20 discusses predictive rules for capital markets. Reference 21 describes approaches to finding patterns in Web accesses. Reference 22 discusses prediction of defects in disk drives. Reference 12 addresses sequential mining in the context of telecommunications events. The last one, although closely related to our interests, uses event data to discover temporal associations, not to identify characteristic patterns and their interpretation. Thus, although much foundational work has been done in data mining and some consideration has been given to mining event data, no one has studied the specific patterns that arise in enterprise event management.

Event mining

Figure 2A is a scatter plot of event data collected from a corporate intranet over a three-day period. The events consist of SNMP (Simple Network Management Protocol) traps such as threshold violated,

Figure 2 Patterns in SNMP trap data



Reprinted with permission from J. L. Hellerstein and S. Ma, "Mining Event Data for Actionable Patterns," Figures 2 and 3, *Proceedings of the CMG 2000 International Conference*, December 2000, Orlando, FL, The Computer Measurement Group, Inc. (2000).

connection-closed, port-up, and port-down. The x-axis represents time whereas the y-axis designates the host where the event originated. There are 149 hosts, numbered 1 through 149. A dot at (x, y) means an event occurred on host y at time x. Note that although this plot contains a considerable amount of information, few patterns are easily identifiable.

Figure 2B shows the same data, and the hosts are algorithmically ordered in a way to reveal patterns (the algorithms are described in References 23 and 24). Many of these patterns are used for constructing correlation rules. For example, pattern 1 consists of threshold violated and threshold reset events that occur every 30 seconds. Such a pattern may be indicative of hosts nearing their capacity limits. Pattern 2 has a cloud-like appearance that consists of port-up and port-down events generated as a result of mobile users connecting to, and disconnecting from, hubs. Such patterns are probably of little interest to the operations staff and hence should be filtered out since they represent normal behavior. Pattern 3, consisting of events occurring every day at 2:00 P.M., represents SNMP request and authentication failure events. This is most likely due to an improperly configured monitor. Pattern 4 is a series of link-up and link-down events, the result of a software problem on a group of hubs.

In a well-managed installation, errors are rare. Thus months of data are needed to identify actionable abnormalities. The volume of data can be substantial. For example, several installations we examined routinely collect five million events per week. Given the large volume of data and the different time scales at which patterns may be present, it is difficult to systematically identify patterns by relying only on visual inspection by a human. Clearly, automatic pattern discovery is needed.

For event data, the previously described concept of a market basket does not apply. However, each event has a timestamp and so looking for patterns means looking at events that co-occur within a time range. These ranges may be time windows (either fixed or variable size) or they may be contiguous segments of data that are characterized in some other way. In the data mining literature, this problem is referred to as temporal mining or temporal association.[12]

It is usually the case that we are looking for patterns related to problem-related situations, which occur infrequently. The frequently occurring patterns in systems management are usually related to normal operation. This requires that patterns be defined differently. In the next section we define three types of patterns pertinent to system management tasks: event burst, periodic pattern, and mutually dependent pattern.

When dealing with event mining, we often need to consider multiple attributes for characterizing membership in itemsets (or patterns). The event type attribute describes the nature of the event. The event origin attribute specifies the source of the event, a combination of the host where the event originated and the process and/or application that generated the event. In addition to type and origin, there is a plethora of other attributes that depend on these two, such as the port associated with a port down event and the threshold value and metric in a threshold violated event. The section "Multiattribute frequent pattern mining," later, develops a framework and an efficient mining algorithm for systematically exploring patterns with various membership definitions.

When there exists rich domain knowledge, and especially correlation knowledge associated with systems management event data, the question arises, how do we incorporate this knowledge in mining for historical events? The section "ERN constructor" presents a method and an implementation that validate existing correlation knowledge and construct new knowledge from historical data.

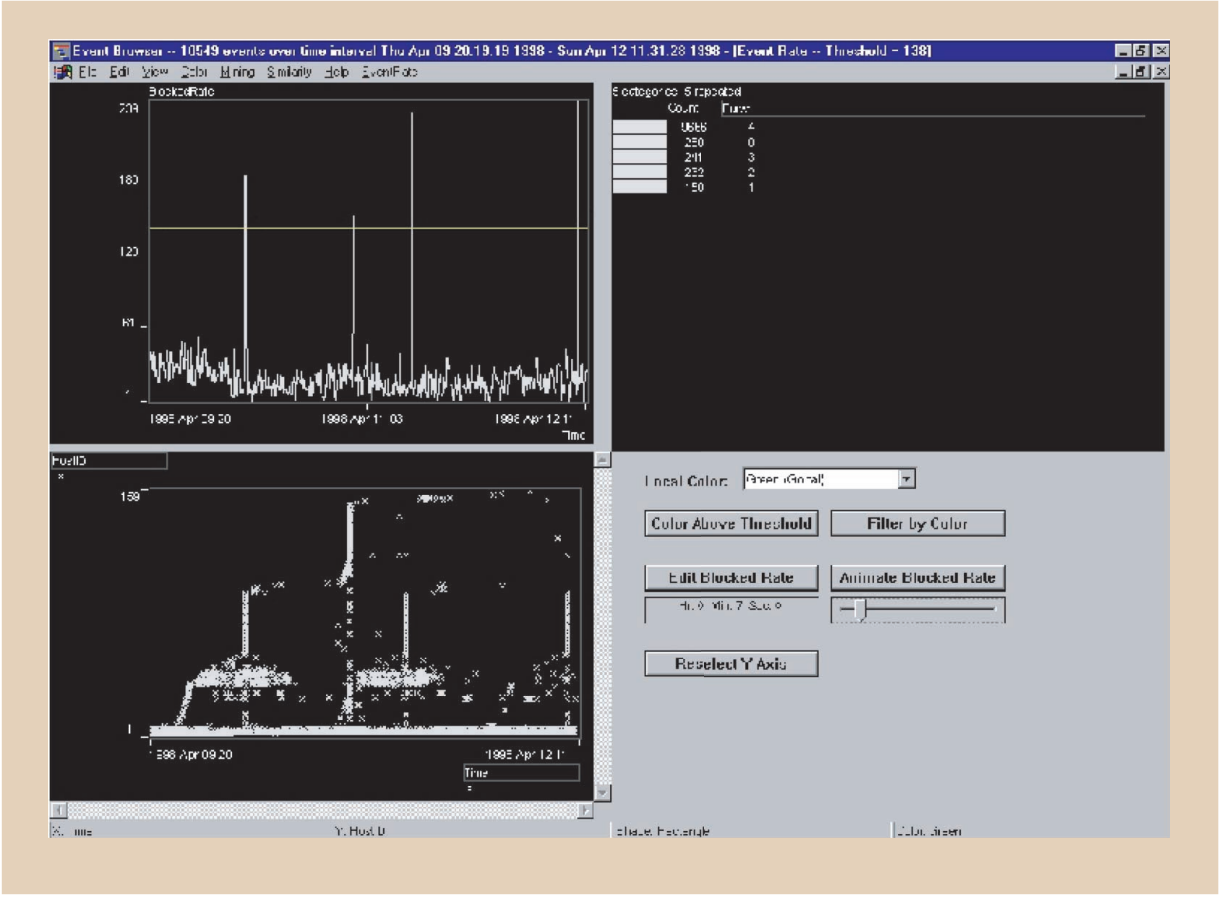
Patterns and pattern discovery algorithms

This section describes commonly occurring patterns in systems management tasks as illustrated in Figure 2. Specifically, we discuss event bursts (patterns 3 and 4 in Figure 2B), periodic patterns (patterns 1 and 2), and mutually dependent patterns (pattern 3).

Event burst analysis. Event bursts (or event storms) may arise under several circumstances. When a critical element fails in a network that lacks sufficient redundancy (e.g., there is only one name server in the network and it fails), communications are impaired thereby causing numerous cannot reach destination events to be generated in a short time period. Or, when a cascading problem occurs, such as the one caused by a virus or by switching loads after a failure, it may result in additional failures due to heavier load.

Figure 3 provides a means for visual identification of event bursts in our corporate intranet data. The

Figure 3 Event burst



plot in the lower left contains the raw data in the same form as in Figure 2B. Given the coarse time scale of the plot relative to the granularity of event arrivals, there are many cases in which more than one event occupies the same pixel. As a result, it is difficult to discern event rates visually. We could drill down to various sections of the plot to better determine event rates, but this is labor-intensive.

Instead, the upper left plot summarizes the rates of events for a specific window size (as indicated in the lower left). The table in the upper right of Figure 3 summarizes those situations in which large event rates are present. This provides a convenient way to select subsets of the data to study in detail.

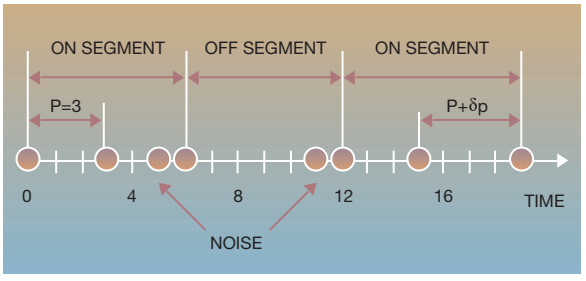
Mining event bursts consists of two steps.

1. Finding periods in which event rates are higher than a specified threshold
2. Mining for patterns common to the periods identified in Step 1

For Step 1, we proceed by first intervalizing the data. Then, event rates within each interval are computed. Those intervals in which rates exceed a specified threshold are then identified. In Figure 2B, these intervals are indicated by the vertical lines that lie above the threshold (which is indicated by the horizontal line).

Step 2 uses the intervals identified in Step 1 as the market baskets of events. For example, mining the three intervals with the largest event rates in Figure 3 finds the pattern SNMP request, Authentication

Figure 4 Partial periodicity



Reprinted with permission from S. Ma and J. L. Hellerstein, "Mining Partially Periodic Event Patterns with Unknown Periods," Figure 2, *Proceedings of the 2001 International Conference on Data Engineering* (ICDE'01), Heidelberg, Germany, April 2001, IEEE, New York (© 2001 IEEE).

Failure. Note that the mining employed here is essentially that performed by Algorithm 1. However, our market baskets are just those intervals that have high event rates.

Partially periodic event patterns. Periodic patterns consist of repeated occurrences of the same event or event set. Our experience has been that such patterns are common in event data, often accounting for one half to two thirds of the events present.

Periodic behaviors are common in networks. Two factors contribute to this phenomenon. The first is the monitoring model—when a managed element emits a high-severity event, the management server often initiates periodic monitoring of key resources (e.g., router CPU utilization). The second is the routine scheduling of maintenance tasks, such as rebooting print servers every morning or backing up data every week.

Our experience with analyzing events in computer networks suggests that periodic patterns often lead to actionable insights. Indeed, a periodic pattern indicates a persistent and predictable behavior, valuable in identifying and characterizing the periodicity. In addition, the period itself often provides a signature of the underlying phenomenon, thereby facilitating diagnosis. In either case, patterns with a very low support are often of great interest. For example, we found a one-day periodic pattern due to a periodic port scan. Although this pattern only happens three times in a three-day log, it is a strong indication of a potential intrusion.

Unfortunately, mining such periodic patterns is complicated by several factors.

1. Periodic behaviors are not necessarily persistent. For example, in complex networks, periodic monitoring is initiated when an exception occurs (e.g., CPU utilization exceeds a threshold) and stopped once exceptional situation is no longer present. During the monitoring interval or *on* segment, the monitoring request and its response occur periodically. The *off* segment consists of a random gap in the periodicity until another exceptional situation initiates periodic monitoring. This makes it difficult to apply well-established techniques such as the fast Fourier transforms.
2. There may be phase shifts and variations in the period due to network delays, lack of clock synchronization, and rounding errors.
3. Period lengths are not known in advance. This means that either an exhaustive search is required or there must be a way to infer the periods. Further, periods may span a wide range, from milliseconds to days.
4. The number of occurrences of a periodic pattern typically depends on the period. For example, a pattern with a period of one day has, at most, seven occurrences in a week, whereas one with a one-minute period may have as many as 1440 occurrences in a day. Thus, mining patterns with longer periods requires adjusting support levels. In particular, mining patterns with low support greatly increases computational requirements in existing approaches to discovering temporal associations.

In order to capture all the factors above, we employ the concept of partially periodic temporal association. We refer to it as a p-pattern. P-patterns generalize the concept of partial periodicity[25] defined by combining it with temporal associations (akin to episodes in Reference 12) and including the concept of time tolerance to account for imperfections in the periodicities.

Figure 4 illustrates the structure of a partially periodic pattern. Such patterns consist of an on segment and an off segment. During the on segment, events are periodic with a period of 3. No periodic event is present during the off segment. Spurious events (or noise) may arise during both on segments and off segments.

Pattern 1 in Figure 2B is an example of a partial periodicity. These partial periodicities contain two types

of events: threshold violated and threshold reset. Here the periodicities occur approximately every 30 seconds, although some are closer to 28 seconds and others are near 33 seconds.

Algorithm. Mining for p-patterns consists of two steps: (1) finding period lengths for each event type; and (2) finding temporal associations. Although a variation of level-wise search can be employed to address the second subtask, the first subtask has not been addressed (to the best of our knowledge). Our approach to finding the periods of p-patterns is to compute event interarrival times and then test if interarrival counts exceed what would have been expected by chance. Note that a simple threshold test is not sufficient here, since small interarrival times are much more common than longer ones and hence the threshold must be adjusted by the size of the interarrival time. We address this by using a Poisson distribution as our null hypothesis for the count of events at specified interarrival times. A χ^2 test is used to assess statistical significance. Next, we mine for the patterns at each statistically significant interarrival time. This is done by level-wise search on each interval that comprises each period. The details can be found in Reference 26.

Results. Table 1 displays the results of mining p-patterns in the corporate intranet data. The left-most column indicates the number of events in the pattern (i.e., size of the itemset). Note that patterns range in size from 1 event to 13 events. Column two specifies the number of candidate patterns searched for each pattern size. Patterns are distinguished by their host, event type, and period. Column three lists the number of distinct p-patterns. The last two columns specify the range of periods and the number of occurrences of patterns listed in each row. Note that the table includes each pattern only once (i.e., subpatterns are not listed).

Mutually dependent patterns. As we discussed before, in a systems management application, normal behavior dominates; abnormal behavior, such as failures and intrusions, is rare. Thus, it is important to discover *infrequent* patterns that relate to problem situations. Consider the example in Reference 8 in which the following events are generated on a router: network interface card failure, unreachable destination, and cold start trap. The last event indicates that the router has failed and restarted. If these events commonly occur together, then the first two may provide advance warning of an occurrence of the third. In Figure 2B, pattern 4 is an m-pattern.

Table 1 Result of mining p-patterns

Itemset Size	Candidate Size	Distinct p-patterns	Min:Max Periods	Min:Max Count
1	100	28	0:1-day	6:680
2	307	22	0:300	3:689
3	938	5	0:30	3:8
4	1917	1	4	3
5	3010	5	4	3
6	3525	3	4	3
7	3104	0		
8	2057	2	4:1-day	3
9	1017	0		
10	366	1	1 day	5
11	91	2	1 day	5
12	14	1	20	20
13	1	1	10	21

Reprinted with permission from S. Ma and J. L. Hellerstein, "Mining Partially Periodic Event Patterns with Unknown Periods," Table 3, *Proceedings of the 2001 International Conference on Data Engineering* (ICDE'01), Heidelberg, Germany, April 2001, IEEE, New York (© 2001 IEEE).

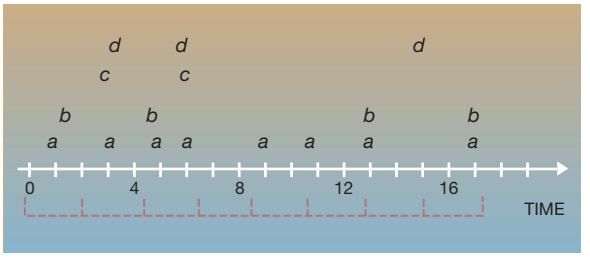
It consists of a combination of link down and link up events.

One naive approach to mining for infrequent patterns is to discover all frequent patterns (e.g., using the apriori algorithm) with very low minimum support (*minsup*), and then apply postprocessing to discover significant patterns. This approach, however, is impractical for finding infrequent patterns, because the minimum support in the apriori algorithm must be set very low, which in turn results in long processing times as well as many irrelevant patterns that must be examined. Irrelevant patterns (i.e., patterns occurring simply by chance) are particularly problematic in real applications with a wide disparity in the frequency of items. As an example, some events have a very high probability of occurrence, such as "connection closed" events issued by network hubs that support the Dynamic Host Configuration Protocol (DHCP). As a result, there are a large number of patterns that have "connection closed" in them even though this event has little correlation with other events.

With this background, we define an m-pattern to be an itemset for which any subset is significantly mutually dependent on all other subsets. That is, if any subset of an m-pattern is present, the remaining items occur with high probability. This is formalized[26] as follows.

Definition 1: A nonempty itemset E is an m-pattern with minimum mutual dependence threshold *minp*, iff

Figure 5 m-patterns vs frequent patterns



$$P_D(E_1|E_2) \geq minp \tag{1}$$

holds for any nonempty subsets E_1 and E_2 of E, where $0 \leq minp \leq 1$.

Note that this definition does not refer to a support level. Thus, unlike frequent associations, m-patterns will be discovered regardless of the frequency of their occurrences. However, we note that it is quite reasonable to consider frequent m-patterns that use both *minp* and *minsup*.

Figure 5 compares m-patterns and frequent patterns. Here, a, b, c, and d are events, and the intervalization consists of two time units (as indicated by the dashed lines). Suppose that: (1) the support threshold for a frequent pattern is 40 percent (i.e., a pattern must appear in 40 percent of the intervals to be considered frequent) and (2) the m-pattern co-occurrence threshold is 60 percent. (The latter is much higher because of the semantics of m-pattern.) Observe that the pattern $\{a, b\}$ is frequent in that this pattern occurs in 50 percent of the intervals. However, there are four cases in which a occurs but b does not. Thus, $\{a, b\}$ is not an m-pattern. Now consider, $\{d, c\}$. This pattern is much less frequent than $\{a, b\}$ in that $\{d, c\}$ occurs in only two of the eight intervals, which is below the support threshold. However, whenever c or d occurs the other does as well. Thus, $\{d, c\}$ is an m-pattern.

M-patterns can be seen as basic semantic units one level above events. Logically, the events in the pattern can be treated as a group. So, from an analysis perspective, it is desirable to coalesce an m-pattern into a single event. This not only reduces the number of events, it also provides the opportunity for higher level semantics (e.g., the m-pattern caused by a router failure).

Algorithm. This section develops an efficient algorithm for discovering all m-patterns. Efficiency is obtained by addressing two issues: (1) how to test (qualify) that an itemset is an m-pattern; (2) how to exploit level-wise search.

For (1), if we use the definition of an m-pattern to test for its presence, then the number of tests we must make is exponential in the size of the pattern. Clearly, this scales poorly. Fortunately, there is a way to simplify matters.

Theorem 1: (Equivalent definition of m-patterns.) An itemset E is an m-pattern as in Definition 2, iff

$$P_D(E - \{a\}|\{a\}) \geq minp \tag{2}$$

for every item a in E.

Following is the sketch of a proof. The necessary condition is followed by m-pattern definition directly. The sufficient condition can be proved by the fact that

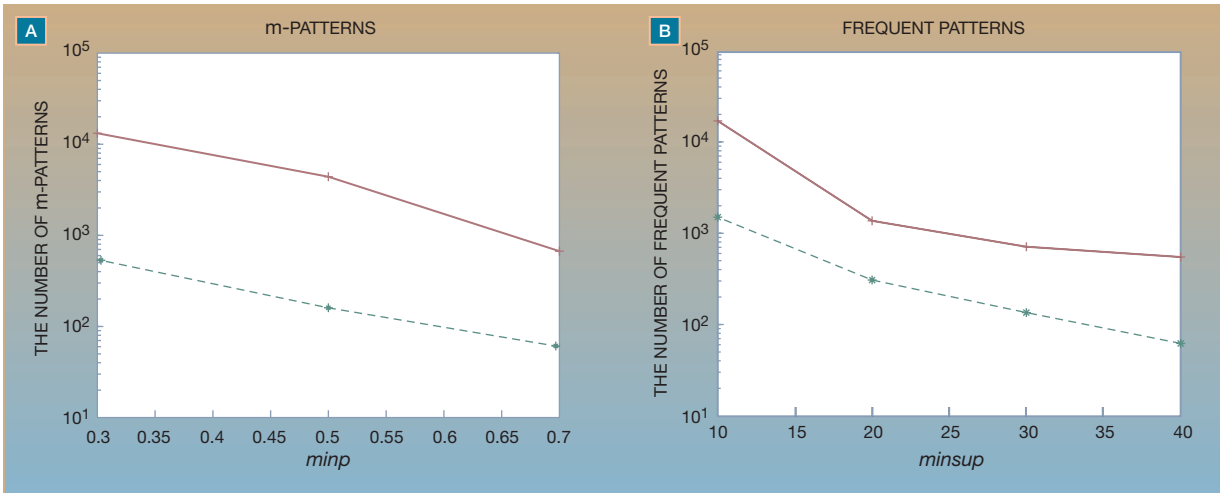
$$P(E_1|E_2) = \frac{P(E_1 \cup E_2)}{P(E_2)} \leq \frac{P(E_1 \cup E_2)}{P(a)} \leq \frac{P(E)}{P(a)}$$
$$= P(E - a|a) \leq minp$$

This equivalent definition provides a means to check for the presence of an m-pattern in only linear time of the length of E.

Second, observe that m-patterns have the downward closure property. This is implied by Definition 1. Since m-patterns are downward closed, a level-wise algorithm (Algorithm 1) can be used. Conceptually, we just need to use the m-pattern definition to qualify a pattern (Theorem 1) in the fourth step of Algorithm 1. Detailed elaboration of the algorithm and the experimental evaluation can be found in Reference 27.

Results. We apply our algorithm for m-pattern discovery to a set of operational data and compare the results to those from mining frequent itemsets. We fix *minsup* to be 3 so as to eliminate a pattern with only one or two instances, and we vary *minp*. Our results are reported in Figure 6A, which plots the total number of m-patterns (the solid line) and the number of border m-patterns (the dashed line) against *minp*. Here, a border pattern refers to a pattern that is not a subset of any other pattern. The

Figure 6 Patterns and border patterns found



Reprinted with permission from S. Ma and J. L. Hellerstein, "Mining Mutually Dependent Patterns," Figures 5 and 6, *Proceedings of the 2001 International Conference on Data Mining* (ICDM'01), San Jose, CA, November 2001, IEEE, New York (©2001 IEEE).

x-axis is *minp* and the *y*-axis is the number of m-patterns discovered on a log scale. Clearly, *minp* provides a very effective way to select the strongest patterns in that the number of m-patterns discovered drops dramatically as *minp* increases. Many of these patterns have very low support levels. For example, we found 59 border m-patterns with length from 2 to 5 in the data set when $minp = 0.7$. Half of these patterns have support levels below 10.

Figure 6B reports the result of mining for frequent patterns. Here the *x*-axis is *minsup*, and the *y*-axis is the number of patterns found on a log scale. Note that the number of frequent patterns is huge—in excess of 1000—even when *minsup* is 20. Examining the frequent patterns closely, we find that most are related to items that occur frequently, not necessarily items that are causally related. This is not surprising as the marginal distribution of items in our data is highly skewed. Indeed, a small set of items accounts for over 50 percent of total events and, consequently, these items tend to appear in many frequent patterns.

Multiattribute frequent pattern mining

In this section, we introduce a special class of patterns, multiple attribute frequent patterns or maf-patterns. The basic concept of maf-patterns follows the apriori algorithm reviewed in the section "Generic level-wise search algorithm." However, as previously mentioned, although events usually have multiple attributes, there is no obvious way of defining transactions as well as labeling events as items. In short, maf-patterns are designed in such a way that all possible ways of grouping events to transactions and labeling events to items are explored.

Table 2 shows a set of events collected from a production environment. We made the following observations.

1. Host 23 generated a large number of InterfaceDown events on 8/21/00. Such situations may indicate a problem with that host.
2. When Host 45 generates an InterfaceDown event, Host 16 generates a CiscoLinkUp (failure recovery) event within the same five-minute interval. Thus, a Host 45 InterfaceDown event may provide a way to anticipate the failure of Host 16.
3. The event types NetworkManagerUp and RouterLinkUp tend to be generated from the same host and within the same minute. This means that when a Cisco router recovers a link, it will discover that its midlevel manager is accessible.
4. Host 24 and Host 32 (not shown in Table 2) tend to generate events with same severity in the same day. This suggests a close linkage between these hosts. If this linkage is unexpected, it should be

Table 2 Systems management events

Rec	Date	Time	Interval	EventType	Host	Severity
1	08/21/00	2:12AM	2:10AM	TcpCnnctClose	3	harmless
2	08/21/00	2:13AM	2:10AM	InterfaceDown	45	severe
3	08/21/00	2:14AM	2:10AM	InterfaceDown	23	severe
4	08/21/00	2:14AM	2:10AM	InterfaceDown	5	severe
5	08/21/00	2:15AM	2:10AM	InterfaceDown	24	severe
6	08/21/00	2:16AM	2:15AM	CiscoLinkUp	16	harmless
7	08/21/00	3:16AM	3:15AM	NetworkManagerUp	16	harmless
8	08/21/00	3:16AM	3:15AM	RouterLinkUp	16	harmless
9	08/21/00	3:33AM	3:30AM	InterfaceDown	45	severe
10	08/21/00	3:34AM	3:30AM	CiscoLinkUp	16	harmless

Reprinted with permission from C.-S. Perng, H. Wang, S. Ma, and J. L. Hellerstein, "FARM: A Framework for Exploring Mining Spaces with Multiple Attributes," Figure 1, *Proceedings of the 2001 International Conference on Data Mining* (ICDM'01), San Jose, CA, November 2001, IEEE, New York (© 2001 IEEE).

investigated in order to avoid problems wherein one host causes problems with the other host.

These patterns have been proven to be very useful in systems management. However, two challenges emerge, as one tries to build a system to discover these patterns. First, we need a language that has sufficient power to express the mining goals that produce the above patterns but is not so verbose that it becomes computationally intractable. We need such a language to define a problem space so that we can *explore* that space and find all significant patterns. We can see that it is not very difficult to write programs to discover or verify each individual statement. However, it is less clear how to discover all these patterns through a systematic process. Second, can the mining algorithm deal efficiently with many attributes? As the field of frequent itemset mining is mostly based on the apriori property,[13,11] what are the similar properties in this new problem that we can utilize to speed up the mining process?

The maf-patterns go beyond fixed attribute mining to mining directly from multiattribute data. We are given data D with attributes $A = \{A_1, \cdots, A_k\}$. Thus, each record in D is a k tuple. For a given pattern, a subset of these attributes is used to define how transactions are grouped and another (disjoint) subset of attributes determines the items. The former are called the *grouping attributes*, and the latter are the *itemizing attributes*.

We begin with an example based on Table 2. Here, $k = 6$. For pattern 3, the grouping attributes are *Host* and *Time*; the itemizing attribute is *EventType*. The pattern has length two, which means that a pattern instance has two records. The items specified by these records are determined by the value of the *EventType* attribute. That is, one record must have *EventType* = *NetworkManagerUp* and the other has *EventType* = *RouterLinkUp*. Further, these records must have the same value for their *Host* and *Time* attributes. Records 7 and 8 form an instance of pattern 3 with *Host* = 16 and *Time* = 3:16 A.M. Note that items may be formed from multiple attributes. For example, pattern 2 has the itemizing attributes *Host* and *EventType*.

We use the term *mining camp* to provide the context in which patterns are discovered. The context includes pattern length (as in existing approaches), grouping attributes, and itemizing attributes. For example, pattern 3 has the mining camp (2, {*Host*, *Time*}, {*EventType*}). Or, more formally, a mining camp is a triple (n, G, S) where n is the number of records in a pattern, G is a set of grouping attributes, and S is the set of itemizing attributes.

Next, we formalize the notion of a pattern. There are several parts to this. First, note that two records occur in the same grouping if their G attributes have the same value. Let $r \in D$. We use the notation $\pi_G(r)$ to indicate the values of r that correspond to the attributes of G. Given a set of attributes G, two records r_1 and r_2 are G-equivalent if and only if $\pi_G(r_1) = \pi_G(r_2)$.

In Table 2, records 7 and 8 are G-equivalent, where $G = \{Host, Time\}$.

In maf-patterns, items are determined by the combinations of values of the attributes of S. Consider pattern 2 for which we require one record with *EventType* = *InterfaceDown*, *Host* = 45 and a second for which *EventType* = *CiscoLinkUp*, *Host* = 16. Thus, (*InterfaceDown*, 45) is one component (or item) of

the pattern and (*CiscoLinkUp*, 16) is the other component. More formally, consider a mining camp (n, G, S) where $S = \{S_1, \cdots, S_m\}$. A pattern component (also called a pattern item) is a sequence of attribute values $sv = \langle s_1, \cdots, s_m \rangle$ where $s_i \in S_i$ for $1 \leq i \leq m$. $p = \{sv_1, \cdots, sv_n\}$ is a pattern of length n for this mining camp if each sv_i is a pattern component for S.

An instance of a pattern is a set of records that are in the same grouping and whose itemizing attributes match those in the pattern. Let $p = \{sv_1, \cdots, sv_n\}$ be a pattern in mining camp (n, G, S) and let D be a set of records. An instance of pattern p is a set of n records $R = \{r_1, \cdots, r_n\}$ such that $r_i \in D$ and $\pi_S(r_i) = sv_i$ for $1 \leq i \leq n$, and r_i and r_j are G-equivalent for all $r_i, r_j \in R$.

Now we can define the concept of support in the maf-pattern mining framework: given a mining camp (n, G, S) and a set of records D that can be partitioned into G-equivalent classes $GEC_1, \cdots, GEC_w$, the support of a pattern p is defined as $|GEC_1|_p + \cdots, + |GEC_w|_p$ where $|GEC_i|_p$ is the number of disjoint instances of p in GEC_i for $1 \leq i \leq w$.

We now have in place all of the definitions necessary to discuss mining in the framework. First, note that if G and S are fixed, then we have the traditional fixed attribute data mining problem. Here, downward closure of the pattern length is used to look for those patterns in $(n + 1, G, S)$ for which there is sufficient support in (n, G, S).

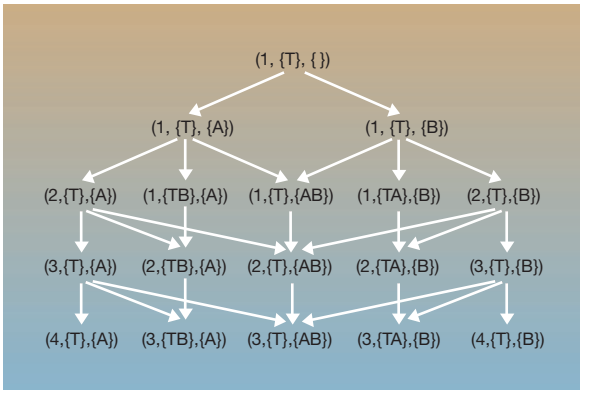
In maf-pattern mining, G and S need not be fixed. In essence, a separate search is done for each combination of G, S. This scales poorly. In particular, the number of permitted combinations of G and S is $3^k - 2^k$, where k is the number of attributes (which follows from observing that A_i may be in G, S, or neither and eliminating the 2^k cases for which $S = \emptyset$).

We propose the following ways for connecting mining camps. Given a mining camp $c = (n, G, S)$ and an attribute $A_i \notin G \cup S$ then

1. $(n + 1, G, S)$ is the type-1 successor of c.
2. $(n, G \cup \{A_i\}, S)$ is a type-2 successor of c.
3. $(n, G, S \cup \{A_i\})$ is a type-3 successor of c.

Figure 7 depicts the predecessor/successor relationships. The root precedes all other mining camps. (In this case, it is not a real camp since $S = \emptyset$.) The level

Figure 7 A mining space



Reprinted with permission from C.-S. Perng, H. Wang, S. Ma, and J. L. Hellerstein, "FARM: A Framework for Exploring Mining Spaces with Multiple Attributes," Figure 3, *Proceedings of the 2001 International Conference on Data Mining* (ICDM'01), San Jose, CA, November 2001, IEEE, New York (© 2001 IEEE).

of mining camp (n, G, S) is defined as $n + |G| + |S|$. Since n is at least 1 and S is nonempty, a minable mining camp has level no less than 2. We structure the mining camps so that the successor relationships only exist between mining camps at different levels. This imposes a partial order. Figure 7 is an example of a mining space. More formally, a mining space $MS(c)$ is a partially ordered set (poset) of mining camps containing c and all of its successors.

To make the notation more readable, we use $MS(n, G, S)$ to denote $MS((n, G, S))$.

Algorithm. This section shows that several types of downward closure can be present in the maf-pattern mining framework. Exploiting these properties provides considerable benefit in terms of efficiency. We begin by presenting the main theorem of the framework.

Theorem 2: Given a mining camp $c = (n, G, S)$ such that the support of a pattern $p = \{sv_1, \cdots, sv_n\}$ is less than θ.

1. For any type-1 successor of c, any pattern that is a superset of p has support less than θ.
2. The support of p in any of type-2 successor of c is less than θ.
3. The support of pattern $p' = \{sv'_1, \cdots, sv'_n\}$ of any type-3 successor of c is less than θ if $sv_i \subset sv'_i$ for all $1 \leq i \leq n$.

Downward closure properties are the foundation of our algorithm as they are in traditional (fixed attribute) mining for frequent itemsets. The more downward properties the mining algorithm uses, the greater the efficiencies that can be realized in mining.

Applications. The framework of maf-pattern mining provides twofold benefits. First, it significantly improves the performance of frequent itemset mining on data with multiple attributes. The problem of mining data with multiple attributes is hard because the number of possible mining camps grows exponentially with the number of attributes. The downward closure properties covered in this section can greatly reduce the computation time. Second and more important, maf-pattern is a notion that unifies frequent itemset mining and aggregation queries. For those mining camps with $n = 1$, the results are aggregations. Here are some examples:

1. The mining camp (1, ∅, {*Host*}) produces a list of hosts that generate a large number of events.
2. The mining camp (1, {*Interval*}, {*EventType*}) finds all event types that tend to be bursty.
3. The mining camp (1, {*Interval*}) outputs the busiest moments in a day.

When $n > 1$, the results are frequent itemset mining. Here we show an example that is particularly interesting in systems management. The mining camp (n, {*Interval*}, {*Host*}) finds the groups of hosts that tend to emit events at the same time. Furthermore, (n, {*Interval*, *EventType*}, {*Host*}) finds the groups of hosts that tend to encounter the same problems at the same time, and implies there is some sort of correlation among hosts in the same group.

The expressive power of maf-patterns can even go beyond the above examples by defining more derived attributes. For example, Interval is a derived attribute in Table 2. Intervals of other lengths can also be used. Network topologies can also be used to add attributes like Network Segment, Subnet, and Functionality. See Reference 28 for a more detailed discussion.

Event Relationship Network constructor

The Event Relationship Network is becoming a popular graphical representation of event correlation. In this section, we first introduce the ERN and the architecture of Event Correlation Constructor (ECC) and describe how domain experts can work with it. Next we present a statistical interpretation of the graphical model. Last, we discuss the application of Event Correlation Constructor in a case study.

Our approach to describing correlation logic uses the conceptual framework of ERN. An ERN is a directed acyclic graph. Nodes are events and are labeled with the role of the event within the case. Arcs from one event to the next indicate that the latter is associated with the former.

A simple event relationship network for a Cisco router is shown in Figure 8. The Cisco monitoring agent emits minor and major alarms for *power supply* device when the device is encountering an abnormal situation. As the status backs to normal, the monitoring agent emits *AlarmOff* events.

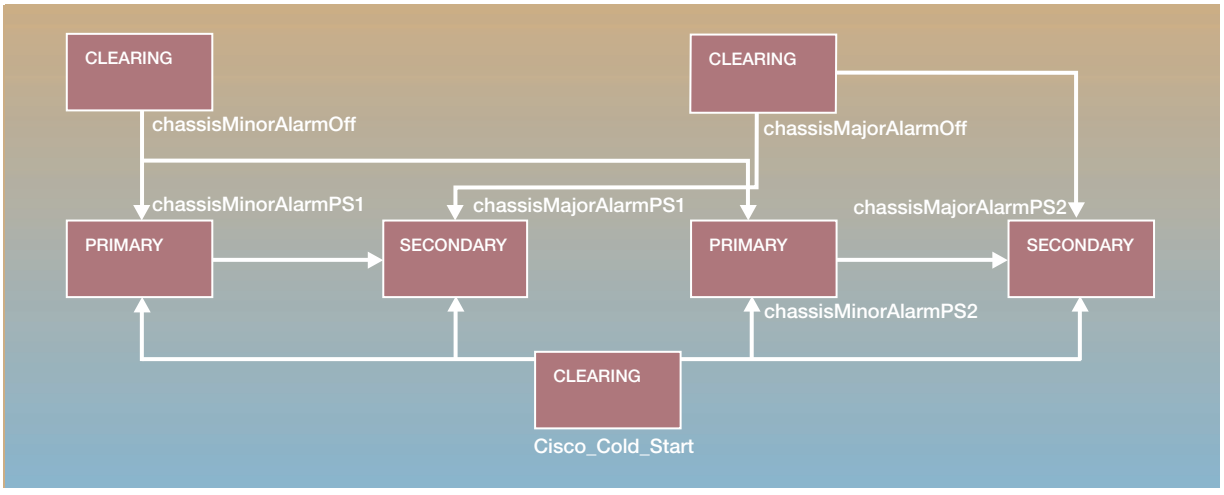
We now introduce a key concept: event roles. An event plays a primary role (is a primary event) if it provides an immediate, often unambiguous, indication as to the corrective action to take. For example, if a warning trap is the first event in the correlation case, then it is a primary event. Proactive management uses the receipt of a primary event to trigger a first level of response. Referring to Figure 8 the role of the *chassisMinorAlarmPS1* and *chassisMinorAlarmPS2* event is primary within the context of this example correlation case.

An event plays a secondary role (is a secondary event) if it is always extraneous in terms of selecting the corrective action in an exceptional situation. Although secondary events do not affect the choice of correct action, they may invoke actions of their own.

If events were always either primary or secondary, then correlation would be much less complex. However, in a large number of cases, the role of an event depends on context within the correlation case. Events that may be either primary or secondary are called primary/secondary events. Within our example correlation case two events act in the role of primary/secondary, the *chassisMinorAlarmOff* and *chassisMajorAlarmOff* events.

ERNs play a central role in building event correlation rules. Event correlation rules in most correlation engines—for example, Tivoli event console—are in the form of *Event-Condition-Action*. The structure of an ERN and the event roles of events can be translated to the event and condition part of correlation rules. With an action template provided by users, usually a means to notify system administrators, each ERN can be translated to a set of correlation rules.

Figure 8 ERN for Cisco chassis



ECC usage scenarios. The ECC is capable of working in three modes.

1. ERN validation: In situations where there are existing ERNs, either constructed previously or from other similar production environments, the ECC can validate the existing ERNs against historical event log from the environment.
2. ERN completion: For a given ERN, it identifies incorrect links. This completion process is done in an iterative manner. In each iteration, all event types correlated to any event types in current ERNs are attached with corresponding links. The process proceeds until no more event type can be added.
3. ERN construction: In situations where no existing ERN can be used as the starter set, the ECC is responsible for generating ERNs for subject matter experts to review. ERN construction can be treated as a special case of ERN completion where no ERN is available.

Figure 9 shows how domain experts and ECC can work together in an iterative fashion to produce high quality ERNs. There are two general cases. When the ERN Starter Set exists, then the ERN validation and completion modules are used; otherwise, the ERN construction module is used. The resulting ERNs are then reviewed by domain experts and the process repeated until the ERNs obtained are satisfactory. Then, the ERNs are translated into correlation rules for real-time monitoring (Figure 1).

A statistical view of ERNs. Now we need to establish a notion of validity of ERNs. We propose here a quantitative method that combines event counting and statistical testing to justify whether an ERN is *valid* based on the event history.

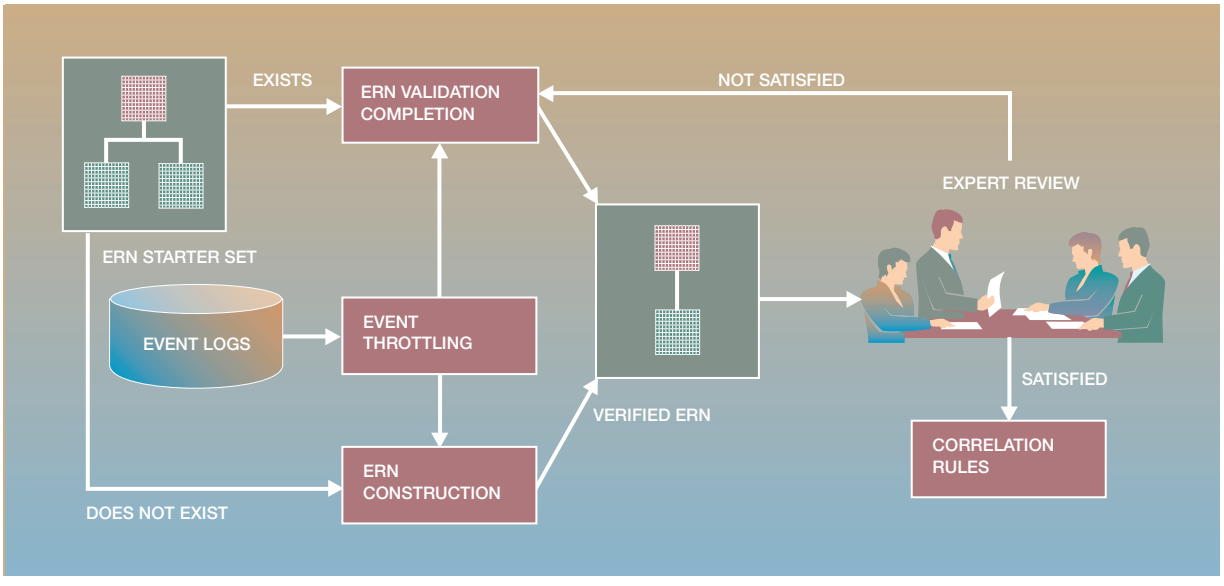
Given a predefined window length w, for each link (A, B) we compute the following statistics $Conf_{AB} = \langle N_A, P_{B|A}, \kappa_{AB} \rangle$.

1. N_A: The total number of occurrences of event type A. N_A indicates whether the event type A as well as the link are worth including in an ERN. In a sense, N_A represents the possible cost of applying incomplete ERN. The cost of processing these redundant trouble tickets caused by missing link (A, B) is proportional to N_A. So for large N_A, it is suggested to include the link if other statistics also indicate high correlation. For small N_A, the cost is a judgment call for the domain experts.
2. $P_{B|A}$: The conditional probability that an occurrence of event type A is followed by an occurrence of event type B within time no later than w. This is defined as:

$$\frac{\text{number of windows containing both } A \text{ and } B}{\text{the number of windows containing } A}$$

3. κ_{AB}: The dependence test score of (A, B) coupling, which indicates the deviation of A and B's distribution from random distribution. High κ

Figure 9 Process of ERN validation, completion, and construction



score indicates it is likely that the correlation of the two events is not due to randomness.

The dependence test score is defined through the following statistics. The probability of observing an event A in a window is $P_A = N_A/T$ where T is the time covered in the log. The expected probability of finding both event A and event B in a window with event A occurring before event B is $E(P_{AB}) = P_A \times P_B/2$. The actual probability of finding both event A and event B in a window with event A occurring before event B is $P_{B|A} = N_{AB}/2T$ where N_{AB} is the number of (A, B) event pairs. The variance of co-occurrences of event A and event B is defined as $VAR_{AB} = (P_{AB}(1 - P_{AB}))/T$. The dependence test score is defined as $\kappa_{AB} = (P_{B|A} - E(P_{AB}))^2/VAR_{AB}$.

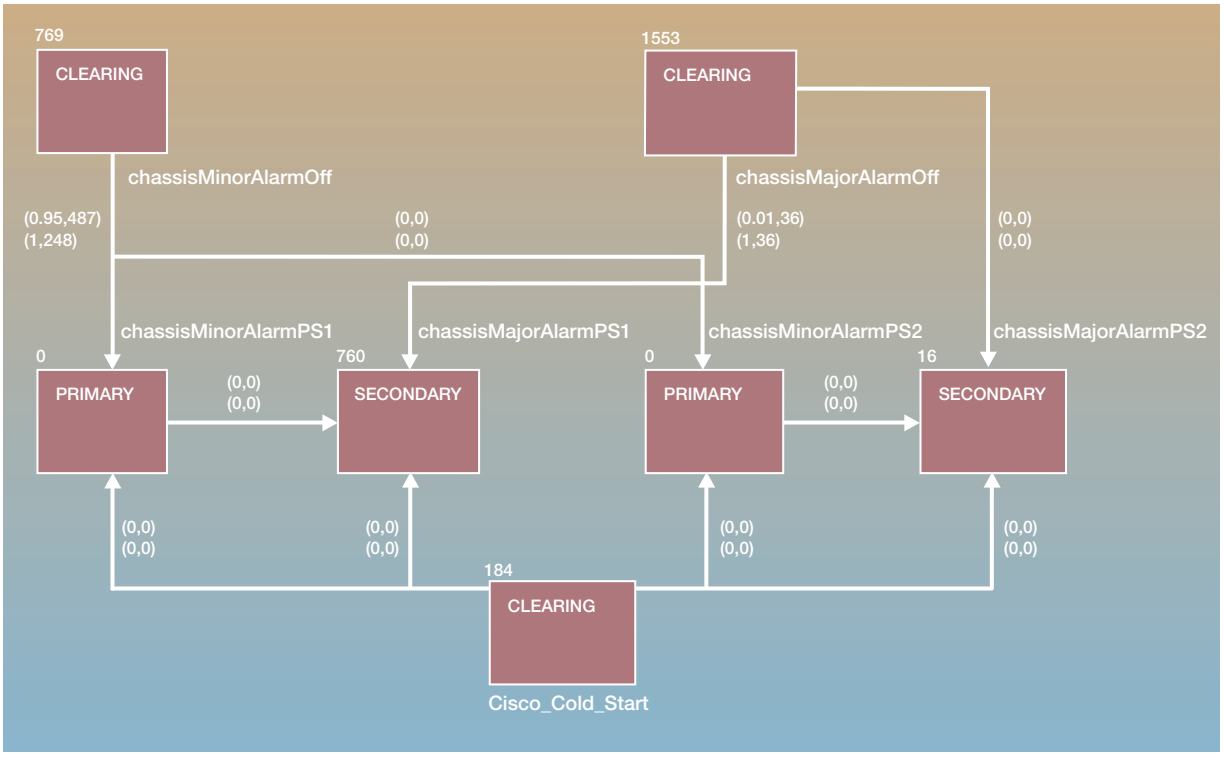
Thresholds of link confidence are also in the form of a triple $\langle N_t, P_t, \kappa_t \rangle$ such that a link (A, B) is valid if $N_A \geq N_t$, $P_{AB} \geq P_t$ and $\kappa_{AB} \geq \kappa_t$. Note that it is possible that both links (A, B) and (B, A) are valid. In such cases, we suggest the direction of link (A, B) should be from A to B if $P_{B|A} \geq P_{A|B}$, otherwise, the direction should be from B to A.

Application. The ERN in Figure 10, which was created by an IBM consultant team for a large financial services company, is annotated with validation statistics. The count, N_A, appears as a label adjacent to each node. For each link (A, B), $(P_{B|A}, \kappa_{AB})$ and $(P_{A|B}, \kappa_{BA})$, in this order, appear as labels on the link.

When domain experts drew the ERN shown in Figure 8, they interpreted *MinorAlarm*s and *MajorAlarm*s as different stages of the same underlying problem. Correspondingly, *AlarmOff* events indicate the problem has been cleared. Also, *Cisco_Cold_Start*, as the event name implies, should clear all alarms.

The statistics shown in Figure 10 contain some surprising results. First, while the number of major alarms in the event log is considerable (760 for power supply 1 and 16 for power supply 2), minor alarms never occur prior to major alarms, or more accurately, minor alarms never occur. Second, the Cisco_Cold_Start event has no correlation with any of the alarm events. We consulted the original designers of the ERN and found that both cases are due to design errors. In the first case, the minor alarms and major alarms are actually assigned to indicate problems of very different nature instead of different severities. In the second case, the designers already doubted the correlation between the Cisco_Cold_Start event and alarm events, but there was no way to check out the correlation at the time. So

Figure 10 An ERN with validation statistics



they included the event in this ERN because the cost of making this type of error is smaller.

There are undoubtedly some constraints in using event data to validate, complete, and construct ERNs. The most noticeable one is the inability to construct and refute the correlations that are not shown in event logs. However, ECC does provide significant improvement in ERN qualities and reduction of the cost of ERN construction.

Conclusions

Event management is the foundation of systems management. Over the last 15 years, automated operations based on the use of correlation rules for interpreting event patterns has lead to increased operator productivity. Although productivity has improved, a substantial bottleneck remains, namely determining the correlation rules.

In this paper we describe how data mining can be used to identify patterns of events that indicate underlying problems. In particular, we identify several patterns of interest to event management: event bursts, periodicities, and mutual dependencies. We provide interpretations for each, and we show how pattern discovery can be structured to exploit a level-wise search, thereby improving scalability. Scalability is particularly important because, as experience shows, tens of millions of events may need to be analyzed in order to discover actionable patterns.

We also address the issue of related multiple attributes of events. We develop here a framework that provides a means to systematically and efficiently explore patterns with different membership definitions. Last, we present the Event Correlation Constructor, a tool that can validate existing correlation knowledge and further construct such knowledge.

Acknowledgment

Our thanks to Luanne Burns, Dave Taylor, David Rabenhorst, and Genady Grabarnik for developing the visualization and mining facility prototype used

for the studies in this paper. We also thank Herb Lee for stimulating discussion and for his comments on this work.

Cited references and notes

1. See http://www.tivoli.com and the Tivoli TEC manual.
2. See http://www.ca.com/products and the Computer Associates Unicenter manual.
3. K. R. Milliken, A. V. Cruise, R. L. Ennis, A. J. Finkel, J. L. Hellerstein, D. J. Loeb, D. A. Klein, M. J. Masullo, H. M. Van Woerkom, and N. B. Waite, "YES/MVS and the Automation of Operations for Large Computer Complexes," *IBM Systems Journal* **25**, No. 2, 159–180 (1986).
4. S. A. Yemini, S. Sliger, M. Eyal, Y. Yemini, and D. Ohsie, "High Speed and Robust Event Correlation," *IEEE Communications Magazine* **34**, No. 5, 82–90 (1996).
5. D. Thoenen, J. Riosa, and J. L. Hellerstein, "Event Relationship Networks: A Framework for Action Oriented Analysis for Event Management," *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, Seattle, WA, May 2001, IEEE, New York (2001), pp. 593–606.
6. S. Ma and J. L. Hellerstein, "Eventbrowser: A Flexible Tool for Scalable Analysis of Event Data," *Proceedings of the 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management* (DSOM'99), Zurich, Switzerland, October 1999, *Lecture Notes in Computer Science*, Vol. 1700, Springer, New York (1999), pp. 285–296.
7. S. Ma, C. Perng, and J. L. Hellerstein, "Eventminer: An Integrated Mining Tool for Scalable Analysis of Event Data," *Proceedings of the Visual Data Mining Workshop* (KDD'01), San Francisco, August 2001, ACM, New York (2001), pp. 1–9.
8. J. L. Hellerstein and S. Ma, "Mining Event Data for Actionable Patterns," *Proceedings of the CMG 2000 International Conference*, Orlando, FL, December 2000, The Computer Measurement Group (2000).
9. L. Burns, J. L. Hellerstein, S. Ma, C. Perng, and D. A. Rabenhorst, "A Systematic Approach to Discovering Correlation Rules for Event Management," *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, Seattle, WA, May 2001, IEEE, New York (2001), pp. 345–359.
10. D. Taylor, N. Halim, J. L. Hellerstein, and S. Ma, "Scalable Visualization of Event Data," *Proceedings of the 11th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management* (DSOM'00), Austin, TX, December 2000, *Lecture Notes in Computer Science*, Vol. 1960, Springer, New York (2000), pp. 47–58.
11. R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proceedings of the 20th International Conference on Very Large Data Bases* (VLDB'94), September 1994, Santiago de Chile, Chile, Morgan Kaufmann, San Francisco (1994), pp. 487–499.
12. H. Mannila, H. Toivonen, and A. Verkamo, "Discovery of Frequent Episodes in Event Sequences," *Data Mining and Knowledge Discovery* **1**, No. 3, 259–289 (1997).
13. R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules Between Sets of Items in Large Databases," *Proceedings of the ACM SIGMOD International Conference of Management of Data*, Washington, DC, May 1993, ACM, New York (1993), pp. 207–216.
14. S. Brin, R. Motiwani, and C. Silverstein, "Beyond Market Baskets: Generalizing Association Rules to Correlations," *Data Mining and Knowledge Discovery* **2**, No. 1, 39–68 (1998).
15. J. Pei and J. Han, "Can We Push More Constraints into Frequent Pattern Mining?" *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD'00), Boston, MA, August 2000, ACM, New York (2000), pp. 350–354.
16. R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad, "Depth First Generation of Long Patterns," *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, Boston, MA, August 2000, ACM, New York (2000), pp. 108–118.
17. R. J. Bayardo, "Efficiently Mining Long Patterns from Databases," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Seattle, WA, June 1998, ACM, New York (1998), pp. 85–93.
18. J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns Without Candidate Generation," *Proceedings of the ACM SIGMOD International Conference on Management of Data* (SIGMOD'00), Dallas, TX, May 2000, ACM, New York (2000), pp. 1–12.
19. R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proceedings of the 11th International Conference on Data Engineering*, Taipei, Taiwan, March 1995, IEEE, New York (1995), pp. 3–14.
20. C. Apte and S. J. Hong, "Predicting Equity Returns from Securities Data with Minimal Rule Generation," *Advances in Knowledge Discovery and Data Mining*, U. M. Fayyad et al., Editors, AAAI/MIT Press (1984), pp. 541–560.
21. R. Cooley, J. Srivastava, and B. Mobasher, "Web Mining: Information and Pattern Discovery on the World Wide Web," *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*, Newport Beach, CA, November 1997, IEEE, New York (1997).
22. C. Apte, S. Weiss, and G. Grout, "Predicting Defects in Disk Drive Manufacturing: A Case Study in High-Dimensional Classification," *IEEE Conference on Artificial Intelligence Applications*, IEEE, New York (1993), pp. 212–218.
23. S. Ma and J. L. Hellerstein, "Ordering Categorical Data for Best Visualization," *Proceedings of the IEEE Symposium on Information Visualization* (InfoVis'99), IEEE, New York (1999), pp. 15–19.
24. A. Beygelzimer, C. Perng, and S. Ma, "Fast Ordering of Large Categorical Datasets for Better Visualization," *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD'01), San Francisco, CA, August 2001, ACM, New York (2001).
25. J. Han, G. Dong, and Y. Yin, "Efficient Mining of Partially Periodic Patterns in Time Series Database," *Proceedings of the 1999 International Conference on Data Engineering* (ICDE'99), Sydney, Australia, March 1999, IEEE, New York (1999), pp. 106–115.
26. S. Ma and J. L. Hellerstein, "Mining Partially Periodic Event Patterns with Unknown Periods," *Proceedings of the 2001 International Conference on Data Engineering* (ICDE'01), Heidelberg, Germany, April 2001, IEEE, New York (2001), pp. 205–214.
27. S. Ma and J. L. Hellerstein, "Mining Mutually Dependent Patterns," *Proceedings of the 2001 International Conference on Data Mining* (ICDM'01), San Jose, CA, November 2001, IEEE, New York (2001), pp. 409–416.
28. C.-S. Perng, H. Wang, S. Ma, and J. L. Hellerstein, "FARM: A Framework for Exploring Mining Spaces with Multiple Attributes," *Proceedings of the 2001 International Conference on Data Mining*, San Jose, CA, November 2001, IEEE, New York (2001), pp. 449–456.

Accepted for publication April 25, 2002.

Joseph L. Hellerstein *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: hellers@us.ibm.com).* Dr. Hellerstein manages the Systems Management Research Department with projects such as event mining, event prediction, intelligent probing, automated diagnosis, and generic adaptive control. He received his Ph.D. degree in computer science from the University of California at Los Angeles in 1984. He has taught at Columbia University and has published approximately 70 papers.

Sheng Ma *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: shengma@us.ibm.com).* Dr. Ma received his B.S. degree in electrical engineering from Tsinghua University, China, in 1992. He received M.S. and Ph.D. (with honors) degrees in electrical engineering from Rensselaer Polytechnic Institute in 1995 and 1998, respectively. He joined the T. J. Watson Research Center as a research staff member in 1998, where he is now manager of the Machine Learning for Systems Department. His current research interests include network and computer system management, machine learning, data mining, and network traffic modeling and control.

Chang-Shing Perng *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: perng@us.ibm.com).* Dr. Perng is a member of the Systems Management Research Department, working on event mining, system profiling, and anomaly detection. He received his Ph.D. degree in computer science from the University of California at Los Angeles in 2000. His current research interests include data mining, time series data, and applying human knowledge in knowledge discovery and data mining.