Automated Construction of Fuzzy Event Sets and Its Application to Active Databases

Yücel Saygın and Özgür Ulusoy

Abstract— **Fuzzy sets and fuzzy logic research aims to bridge the gap between the crisp world of math and the real world. Fuzzy set theory was applied to many different areas, from control to databases. Sometimes the number of events in an event-driven system may become very high and unmanageable. Therefore, it is very useful to organize the events into fuzzy event sets also introducing the benefits of the fuzzy set theory. All the events that have occurred in a system can be stored in event histories which contain precious hidden information. In this paper, we propose a method for automated construction of fuzzy event sets out of event histories via data mining techniques. The useful information hidden in the event history is extracted into a matrix, called sequential proximity matrix. This matrix shows the proximities of events and it is used for fuzzy rule execution via similarity based event detection and construction of fuzzy event sets. Our application platform is active databases. We describe how fuzzy event sets can be exploited for similarity based event detection and fuzzy rule execution in active database systems.**

Keywords— **Fuzzy event sets, fuzzy triggers, fuzzy rule execution, similarity based event detection, active databases.**

I. Introduction

The fuzzy set theory introduced by Zadeh [1] has been utilized in many areas of research, from control to databases and expert systems [2], [3], [4]. Fuzzy rules in expert systems and control that make use of the concepts in fuzzy sets were proven to increase the flexibility and effectiveness of the systems. Special inferencing mechanisms have been developed for fuzzy control to be used to infer a fuzzy conclusion from a set of fuzzy rules [2].

Modularization of rules is an important research issue in Active Database Management Systems. In systems where thousands of events may occur and thousands of rules may be fired by these events, it is really very difficult to keep track of the rules. Therefore, partitioning the whole event space into smaller groups would be helpful in controlling the rules fired by those events. Partitioning the event space enables the users of the system to concentrate on a smaller group of related events, increasing the efficiency and effectiveness of the system.

Consider the demand side management of an electricity producing and selling company where the power consumption information of individual companies is stored in a history. Consider also that there are hundreds of companies, which is usually the case, and each company has more than two power consumption events, like low, medium, high power consumption. The number of events in the system can easily become unmanageable. If there were a way to divide the whole event space into subsets and deal with each subset individually, the job of the power dealers would be much easier.

Classification of rules by using semantic constraints in an active database system is considered before by Baralis et. al. in [5]. This process is called *stratification*. Event-based stratification techniques aim to modularize the rules by classifying the event space which is done by human interaction and can not be used when semantic knowledge of events is not available. The rule modularization approach of Baralis et. al. is based on semantic techniques and it is not automated. However, it is always desirable to automate this process whenever possible. In this paper, we deal with the problem of automated rule modularization by partitioning the events which trigger the rules. Our model for modularizing rules is based on the analysis of past occurrences of events which are stored in a time based event history. We propose an incremental method for event history analysis based on a sliding window approach. With the sliding window approach, the whole event history is traced, and only the events in the current window are considered for determining the proximity of events. The window advances one event at each iteration. This is where the name "sliding window" comes from. The information in an event history is extracted into a matrix, called a *sequential proximity matrix*, which is an $n \times n$ matrix, where n is the number of events in the system.

We use graph partitioning techniques for dividing the event space using sequential proximity matrix into subsets. These subsets are modeled as fuzzy sets, therefore we call them fuzzy event sets. Each event has a degree of membership to a fuzzy event set. Similarity based event detection was proposed as a useful method in an event based system [6]. Fuzzy event sets are considered to be the basis of similarity based event detection and fuzzy rule execution in active databases which is our application platform. Similarity based event detection method of [6] assumes that similarities of the events are available by domain knowledge of experts. In this paper, we propose a method for extracting the similarities from event histories. This is a new idea for enabling similarity based event detection. By utilizing fuzzy event sets, we introduce the benefits of fuzzy set theory to active database rule modularization and rule execution. We also describe a new rule execution model based on fuzzy event sets which gives more flexibility to rule execution.

Main contributions of our work can be summarized as:

This research is supported by the Research Council of Turkey (TÜBİTAK) under grant number EEEAG-246.

The authors are with the Department of Computer Engineering, Bilkent University, 06533 Turkey. Email: {saygin, oulusoy}@cs.bilkent.edu.tr

- automated construction of relationships among events via a sliding window algorithm for mining event histories,
- partitioning the event space into fuzzy event sets for rule modularization, using graph partitioning techniques, and
- enabling a new event detection and rule execution model utilizing the fuzzy event sets. The new model allows us to perform:
  - similarity based event detection, and
  - fuzzy rule execution and rule prioritization.

Rest of the paper is organized as follows: In Section II, we provide the necessary background and related work on active databases and data mining. We explain the concept of event histories and introduce an algorithm for mining event histories in Section III. In Section IV, we describe the construction of fuzzy event sets using a graph partitioning algorithm for dividing the event space into fuzzy event sets. In Section V, we describe similarity based event detection using sequential proximity matrix. We also describe scenario-based rule execution in active databases. We present how fuzzy event sets can be used for fuzzy rule execution via rule scenarios. Finally, in Section VI we summarize the results and discuss the future work.

II. Background and Related Work

Before getting into the problem of fuzzy event set construction and its application to active databases, we should describe active databases and explain how they differ from conventional databases. Active Databases as our application platform are discussed in Section II-A. As we use data mining techniques for extracting fuzzy event sets, we also provide the necessary background on data mining in Section II-B.

A. Active Databases

Active Database Management Systems (ADBMSs) can be considered as an extension of conventional database management systems, in such a way that the database system can respond to the state changes in database by automatically executing some actions. The production rule concept in Artificial Intelligence was modified for the active database context so that rules can respond to state changes caused by database operations [7]. Expert systems and ADBMSs are very much related in that they are both based on the concept of rules although their rule structures are different. Fuzzy concepts were integrated into expert systems and database systems [3], [8], [9], [10], [11], and it was previously shown that incorporation of fuzzy concepts into databases is desirable and does solve the problems of uncertainty and inherent fuzziness of acquired data [12], [13]. The use of fuzziness in active database context which extends the standard databases by rules was also shown to be useful [14], [15], [6]. In a typical ADBMS, system responses are declaratively expressed using Event-Condition-Action (ECA) rules [16]. An ECA rule is composed of an *event* that triggers the rule, a *condition* describing a given situation, and an *action* to be performed if the condition is satisfied. Primitive events can be combined to form composite events. Composition of primitive events can be done with various event constructors, like conjunction, disjunction, or closure. Coupling modes between event and condition, and between condition and action determine when the condition should be evaluated relative to the occurrence of the event, and when the action should be executed relative to the satisfaction of the condition, respectively. Rules can be executed sequentially or concurrently depending on the underlying application [17]. An abstract ECA rule for electricity producing and selling company is given below:

> **Event :** Power consumption of company A is *high*
> **Condition:** If the temperature and humidity is *high* and A is a major customer
> **Action:** Increase the production and the price of electricity

In this example we can see that the event itself is fuzzy. Wolski and Bouaziz [14] adopted the fuzzy rules and fuzzy inferencing techniques in fuzzy control theory to the rules in ADBMSs and formalized the fuzzy events and fuzzy rule execution. In [18], [6] we extended the formal description of the fuzzy events provided by Wolski and Bouaziz, to composite fuzzy events. We also studied fuzzy coupling modes and introduced the concepts of similarity based event detection and fuzzy rule execution via scenarios.

B. Data Mining

Mining of large volumes of data for useful information is currently a hot research topic. Data mining problems can be categorized as:

- *classifications* where we try to partition the data into disjoint groups [19],
- *associations*, where some correlations among data items are sought [20], and
- *sequences* where we try to find sequences among data items [21] .

Discovery of event patterns from event histories (sequences) is very similar to finding association rules among a set of items. Informally, we have a set of items and a database consisting of transactions where each transaction contains some items bought by a customer at a time. The problem is to find rules like "if somebody buys diapers then he/she buys a bottle of beer as well", using the database of customer transactions.

Discovery of event patterns from event sequences is presented by Bettini et al. in [22] and Mannila et al. in [23]. Mannila et al. propose efficient algorithms for finding event patterns (they call it *frequent episodes*) by analyzing event sequences. The application area used in their work is telecommunication alarm management. They use a sliding window approach, where the window size is specified by the user. Bettini et al. give algorithms based on Mannila et al.'s sliding window approach for discovering event patters. They tackle a more complex problem of finding event patterns where events have multiple granularities.

Bettini et al. use a sliding window approach for the discovery of event patterns from event histories [22]. In the sliding window approach of Bettini et al., each window can be viewed as a customer transaction where transaction identifiers are just the window numbers in the event history. The set of events recorded in the history can be mapped to a set of items. The problem of finding frequent episodes is the same as finding large item sets with a given support (i.e., frequency) in the problem of finding associations. The problem of finding large item sets is a subproblem of finding associations where an item set is large if its support (i.e., the number of transactions that contain that item set) is greater than or equal to minimum support value provided by the user. However, the problem of finding frequent episodes becomes more complex when the user is allowed to specify complex episode structures.

There are various algorithms for finding large item sets in a given database of customer transactions. Two well-known algorithms are *Apriori* and *AprioriTID* proposed in [24]. Apriori algorithm starts with large item sets of size 1, generates candidate item sets, and then tests whether they are large or not. In each of the subsequent iterations, candidate item sets are generated using the large item sets found in the previous step, recursively. AprioriTID is similar to Apriori with some optimizations. The algorithms proposed in [22] for finding frequent episodes also use the approach of Apriori; i.e., generate candidate episodes starting with episodes of size 1. In latter iterations, the frequent episodes found in the previous step are used for generating the candidates.

III. A Sliding Window Algorithm for Mining Event Histories

In this section we explain the concepts of events and event histories, and propose a method for obtaining the relationships among events.

A. Events and Event Histories

Events are state changes in a system. In a control system, events can be defined using the measurements coming form the sensors. In a database environment, events can be tuple insertions, deletions and transactions operations. In an ADBMS, the sequence of events fired over a long period of time is called the *event history* of the system. The events in the history are stored in chronological order of event signaling.

Values that are coming from the sensors are stored in a database together with their time of occurrence, and the sensor values are then converted to events to be stored in the event history. This process is shown in Figure 1. The examples regarding this process are given below.

Example III.1: Consider a power consumption history kept by some power selling company. For simplicity, assume that there exist only three customers a, b, and c, where a and b correspond to small scale industrial customers and c corresponds to a typical residential customer. Sample power consumption values for these three customers are provided in Figure 2.

The crisp sensor values are converted to fuzzy events by using the membership functions of the fuzzy events in the system. Analyzing the event history is more effective than analyzing the actual sensor values since events are at a more abstract level and give us more compact information which is easier to interpret.

Example III.2: Assume that the power consumption events for the power consumption of the customers given in Example III.1 are $a_h, a_l, b_h, b_l, c_h, c_l$ where low power consumption is indicated by subscript **l**, and high power consumption is indicated by subscript **h**. For example, event a_h denotes high power consumption event of company a, and c_l denotes low power consumption event for company c. The fuzzy events corresponding to crisp sensor values are obtained by choosing the event which gives the highest membership value. Sample membership functions for the fuzzy sets high and low are given in Figure 3. A sample event history obtained from the crisp values in Figure 2 is shown in Figure 4.

Assume that we have a set of events $\xi = \{e_1, e_2, ..., e_n\}$, and a set of time stamps $\tau = \{t_1, t_2, ..., t_m\}$.

Definition III.1: The event occurrence binds an event to a timestamp. Event occurrence is denoted by a pair (e, t) where $e \in \xi$ and $t \in \tau$ denoting the time of the occurrence of e.

Definition III.2: Two event occurrences, (e_{i_p}, t_{i_p}) and (e_{i_q}, t_{i_q}) in an event history are said to be concurrent event occurrences if $t_{i_p} = t_{i_q}$; i.e., they occur at the same time unit.

In Figure 4, (a_h, t_1) is an event occurrence, and $(a_h, t_1), (b_h, t_1)$, and (c_l, t_1) are concurrent event occurrences at time t_1. The occurrences of events in a system are recorded in an event history as defined below.

Definition III.3: Let $\xi = \{e_1, e_2, ..., e_n\}$ be a set of events. An event history, H, is the set of past event occurrences. It is formally defined as:
$H = \{(e_{i_1}, t_{i_1}), (e_{i_2}, t_{i_2}), ..., (e_{i_n}, t_{i_n})\}$, where t_{i_j}, $j = 1, \ldots, n$, is a timestamp and $e_{i_j} \in \xi$, $j = 1, \ldots, n$. n is called the size of the event history.

In Figure 4, $\xi = \{a_h, a_l, b_h, b_l, c_h, c_l\}$ and $\tau = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$. If in an event history, there is at least one event at each time unit, we call these types of histories *complete event histories*. An event history constructed by using the sensor values that are sent periodically to the database is an example of a complete event history since there is a measurement value at each point of time.

B. Mining the History

Assume that we have a set of events $E = \{e_1, e_2, ..., e_n\}$, and an event history H where the events are stored in the form: $< e, p, t >$ where $e \in E$, p is the event parameter (i.e., the value returned by the crisp event such as the power consumption), and t is the time-stamp of the event occurrence. Our problem is to find similarities among the events in E using event history H. There might be different approaches for obtaining the similarities among events. Our approach uses the proximity of event occurrences in

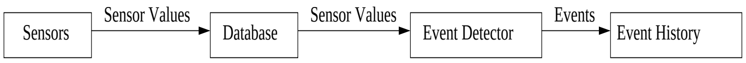

Fig. 1. The Process of Event Generation.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_10
Industrial (a)	30	25	20	5	10	10	15	20	25	30
Industrial (b)	25	20	16	8	12	10	10	25	30	30
Residential (c)	0.5	0.6	0.8	1.2	0.9	0.9	0.7	0.6	0.6	0.4

Fig. 2. Sample Values for Power Consumption of Residential and Industrial Customers in Kilo Watts.

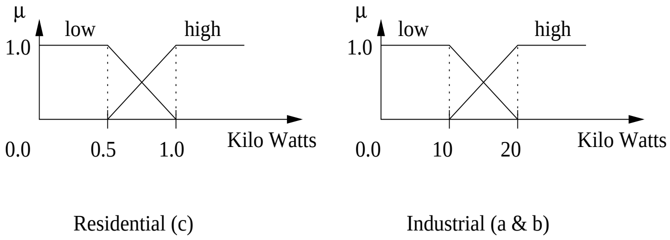

Fig. 3. Membership Functions for Fuzzy Sets *low* and *high* of Residential and Industrial Power Consumption.

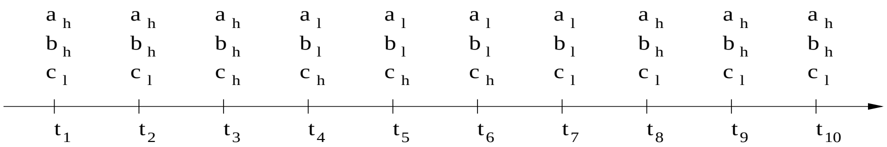

Fig. 4. A sample event history of power consumption events.

the event history. If two events are frequently being fired together in a short period of time, then it is likely that these events belong to the same context, therefore they are *similar* in that sense. We use a sliding window approach to analyze the event history and obtain the similarities among the events. In what follows, we explain our approaches in more detail.

B.1 Preliminary Definitions

In this section we define the terms used for event histories and the sliding window algorithm.

Definition III.4: A window, win, of size m is a collection of m consecutive time stamps and the events belonging to these time stamps. $win[i]$ denotes the events in the i^{th} timestamp of win, where $1 \leq i \leq m$.

Definition III.5: Pivot events are the events that belong to the first time stamp of the current window, i.e., pivot events $= win[1]$.

Example III.3: In Figure 5, w_1 is a window that covers timestamps, t_1 and t_2. Pivot events of w_1 are a_h, b_h, and c_l.

Definition III.6: Inverse distance between two event occurrences, (e_i, t_l) and (e_j, t_k) mimics the distance of occurrence of e_i and e_j. Inverse distance is a value in the range $(0, 1]$ and it is calculated by the formula $\frac{1}{|t_k - t_l| + 1}$.

Example III.4: In Figure 5, the inverse distance between (a_h, t_1) and (b_h, t_2) is $\frac{1}{|2-1|+1} = 0.5$. The inverse distance between two concurrent event occurrences (a_h, t_1) and (c_l, t_1) is $\frac{1}{|1-1|+1} = 1$.

The inverse distance of concurrent event occurrences is 1, which is the maximum possible value and the inverse distance value between (e_i, t_l) and (e_j, t_k) goes to 0 as $t_k - t_l$ goes to infinity.

Definition III.7: Sequential proximity matrix is an $N \times N$ matrix where N is the number of possible events that may occur in the system. Each element $m_{i,j}$ of the matrix is the cumulative inverse distance of the occurrence of event e_j after the occurrence of event e_i observed in the history.

We also take into account the order of event occurrences; i.e., the proximity of event occurrences in the order e_i, e_j (i.e., $e_i \rightarrow e_j$) may be different from that of the same events in the order e_j, e_i (i.e., $e_j \rightarrow e_i$). In that sense our notion of distance has a direction.

B.2 Sliding Window Algorithm

In order to construct the sequential proximity matrix, we use a sliding window approach similar to the one described by Mannila et. al., in [23]. The difference of our approach from that of Mannila et. al. is that, we do not find event patterns, but try to find pairwise correlations among events in terms of proximity of event occurrences. We store the results of pairwise correlations in a matrix and use this intermediate structure to construct fuzzy event sets.

Our sliding window algorithm for mining event histories is presented in Figure 6. The algorithm traces the whole

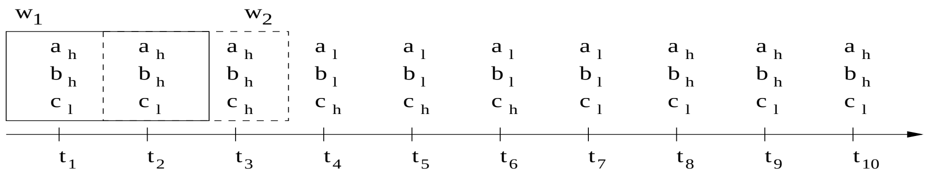

Fig. 5. Sliding windows, illustrated on the history of power consumption events.

INPUT: window size m, and the event history H

OUTPUT: sequential proximity matrix, M,
showing the relationships among the events in H

Begin
1. initialize matrix M to 0
2. initialize *window* to first m events
3. **repeat**
{
4. Initialize *pivot_events* to the events in the first time unit of *window*
5. **foreach** pair of events e_i, e_j in *pivot_events*
{
6. increment $M[e_i, e_j]$ by one
7. increment $M[e_j, e_i]$ by one
}
8. **foreach** event e_p in *pivot_events*
{
9. **foreach** event e_r in *window* and not in *pivot_events*
{
10. increment $M[e_p, e_r]$ by inverse distance of e_r to e_p
}
}
}
11. **until** the end of H is reached
End

Fig. 6. Sliding Window Algorithm.

event history in sequence and considers the events in a window of size m. In Figure 5, we have set the window size to 2. The first window, w_1, is shown with a rectangle drawn using straight lines. It covers the events at the first two time stamps, t_1 and t_2. The window slides one time unit at each iteration. The second window, w_2, is shown in Figure 5 with a rectangle drawn with dashed lines. The second window covers the events with time stamps t_2 and t_3. At each iteration, proximity of the pivot events to the rest of the elements in the window is updated one by one in the matrix. After this process, window slides one event to the right and pivot events are updated again. This process continues until the end of the history is reached, i.e., time stamp of the pivot events is the last time stamp in the event history. The output of the algorithm is the sequential

	a_h	a_l	b_h	b_l	c_h	c_l
a_h	0.0	0.0	1.5	0.0	0.0	1.5
a_l	0.0	0.0	0.0	0.0	0.0	0.0
b_h	1.5	0.0	0.0	0.0	0.0	1.5
b_l	0.0	0.0	0.0	0.0	0.0	0.0
c_h	0.0	0.0	0.0	0.0	0.0	0.0
c_l	1.5	0.0	1.5	0.0	0.0	0.0

Fig. 7. Sample Sequential Proximity Matrix after processing the first window.

	a_h	a_l	b_h	b_l	c_h	c_l
a_h	0.0	0.0	3.0	0.0	0.5	2.5
a_l	0.0	0.0	0.0	0.0	0.0	0.0
b_h	3.0	0.0	0.0	0.0	0.5	2.5
b_l	0.0	0.0	0.0	0.0	0.0	0.0
c_h	0.0	0.0	0.0	0.0	0.0	0.0
c_l	3.0	0.0	3.0	0.0	0.5	0.0

Fig. 8. Sample Sequential Proximity Matrix after processing the second window.

proximity matrix obtained by tracing all the event history.

The sequential proximity matrix keeps the proximity of events incrementally. Its elements are initially set to zero[1]. We would like to give an example to facilitate understanding of the sliding window algorithm. Assume that we have an event set $E = \{a_h, a_l, b_h, b_l, c_h, c_l\}$. The sequential proximity matrix, say M, would be a 6×6 square matrix. A sample event history for the given event set is provided in Figure 5 with the first and second event windows, w_1 and w_2, each of size 2 shown. Initially, all the entries in the sequential proximity matrix are 0. During the first iteration, the pivot events are a_h, b_h, and c_l. As the first step, proximity values of the pivot events are updated in the matrix. As a second step, proximity values of the pivot events to the rest of the events in the current window are incremented in matrix M. After the first pass, the matrix is updated as shown in Figure 7.

[1]In case of incremental runs of the algorithm, the contents of the previous matrix are used as initial values.

	a_h	a_l	b_h	b_l	c_h	c_l
a_h	0.0	0.5	8.0	0.5	2.0	6.5
a_l	0.5	0.0	0.5	5.5	4.0	2.0
b_h	8.0	0.5	0.0	0.5	2.0	6.5
b_l	0.5	5.5	0.5	0.0	4.0	2.0
c_h	1.0	5.0	1.0	5.0	0.0	0.5
c_l	7.5	1.0	7.5	1.0	0.5	0.0

Fig. 9. Sample Sequential Proximity Matrix after processing the whole event history.

In the second pass, the window is moved one position right, as shown in Figure 5, and the new pivot events are set to a_h, b_h and c_l. The same process is repeated for the new pivot events. After the second pass, entries of the matrix are as shown in Figure 8. The final proximity values of the events are given in Figure 9.

Giving more weight (increased proximity) to the events occurring very close to the pivot event in the window improves the accuracy of the sliding window algorithm.

A nice property of the sliding window algorithm is that, it is incremental. Assume that the time-stamp of the last event processed by the previous run of the algorithm is t_1. As the event history grows with time, the algorithm can be reapplied to the event history starting from t_1 and the matrix values are updated according to the new event patterns. For some applications, like stock exchange, very old portions of the event history may be obsolete and misleading. Therefore, after a reasonable time it may be necessary to discard the old proximity values and produce them from scratch starting from some point in the history.

The similarity matrix that is constructed by mining the event history is used for fuzzy rule execution in ADBMSs. Fuzzy rule execution is performed by similarity based event detection and construction of rule scenarios, which are explained in Section V.

B.3 Time Complexity of the Algorithm

Time complexity of the algorithm is determined by the size of the event history, size of the window and the maximum possible number of concurrent events.

Lemma III.1: Time complexity of the sliding window algorithm is $O(n \times c^2 \times m)$, where n is the event history size, m is the window size and c is the maximum number of concurrently occurring events in the event history.

Proof: This result is due to the fact that the whole event history is traced for once and at each slide of the window m comparisons are made. The window is initialized to first m timestamps, and then the window slides $n-1$ times. Therefore, the total number of iterations at the repeat-until loop is n. At each iteration of the loop, first the pivot events are initialized and for each pair of events in the pivot events, the matrix entries are updated. Updating an entry in the matrix takes constant time, and the number of pairs in the pivot events is $\binom{c}{2}$ where c is the maximum number of concurrent events at a time. After that, for each event in the pivot events, we update the proximity of that event with the rest of the events in the window. Since there are $m-1$ time points in the rest of the window (excluding the time point of pivot events), this operation is done at most $c \times c \times (m-1)$ times. Therefore, the maximum possible number of operations is, $n \times (\binom{c}{2} + c^2 \times (m-1))$ which equals to $n \times (\frac{c\times(c-1)}{2} + c^2(m-1))$. Since we have calculated the maximum possible number of operations with respect to the maximum number of concurrent events, the time complexity of the sliding window algorithm is $O(n \times c^2 \times m)$ ■

The only parameter that we can control in the time complexity formula is the window size. As the window size increases the precision of the result of the mining increases, however the time complexity also increases. There is a tradeoff between the time complexity and precision. The selection of window size should consider this tradeoff.

IV. Construction of Fuzzy Event Sets

The theory of fuzzy sets was introduced by Zadeh [1]. A fuzzy set is characterized by its membership function which gives us the degree of membership of each element in the universal set to the fuzzy set. Membership function of a fuzzy set F on the universal set U is generally denoted by μ_F and maps each element $x \in U$ to a real number in the range $[0, 1]$; i.e., $\mu_F(x) : U \rightarrow [0, 1]$.

Fuzzy event sets are fuzzy sets where the elements are events, and each event has a degree of membership to the fuzzy event set A fuzzy event set, E_F, over a universal event set ξ is a tuple of the form $< E, \mu >$ where $E \subseteq \xi$ and μ is the membership function that describes the degree of membership of the events in E to E_F. In the following subsection, we explain how the sequential proximity matrix is used for constructing fuzzy event sets and their membership function.

A. Partitioning The Event Space Using Sequential Proximity Matrix

The fuzzy event set construction is done by partitioning the event space using the sequential proximity matrix. The events in each partition have a degree of membership greater than 0, and the rest of the events have 0 as their degree of membership to the fuzzy event set. The exact membership degree functions are provided later on. The problem here is to partition the set of events into k subsets such that the resulting subsets are balanced, the total proximities of the events in all subsets are maximized, and the total similarities among the events belonging to different subsets are minimized. This problem is similar to the *graph partitioning problem* which was introduced by Kernighan and Lin in [25]: Given a graph G with costs on its edges, partition the nodes of G into subsets no larger than a *given maximum size*, so as to minimize the total cost of the edges cut. Following is the formal definition of the graph partitioning problem [25]:

Definition IV.1: Let G be a graph of n nodes, of size (weights) $w_i > 0, i = 1, ..., n$. Let p be a positive number, such that $0 < w_i \leq p$ for all i. Let $C = c_{ij}, i, j = 1, ..., n$ be a weighted connectivity matrix describing the edges of G.

Let k be a positive integer. A *k-way partition* of G is a set of nonempty, pairwise disjoint subsets of G, $v_1, ..., v_k$ such that $\cup_{i=1}^{k} v_i = G$. A partition is admissible if $|v_i| \leq p$ for all i, where v_i stands for the size of the set $|v_i|$ and equals to the sum of the sizes of all the elements of v_i.

It has been shown in [25] that it is not feasible to find an exact solution to this problem. Therefore Kernighan and Lin proposed heuristics for solving the graph partitioning problem.

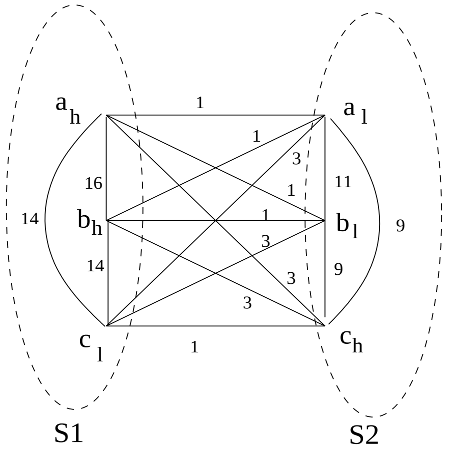

Fig. 10. A sample event graph.

We simplify the graph partitioning problem defined above so that the weights of the nodes are all equal to 1. We also set p to the constant value $\frac{n}{k}$ for a balanced partition. Balanced partitioning is important to avoid trivial partitions. If we do not assume a balanced partitioning then the partition that minimizes the sum of the weights of the edges among the partitions will produce a single partition that has no outgoing edges which has no use. The number of partitions, i.e., the number of fuzzy event sets is application dependent. The weighted connectivity matrix is the sequential proximity matrix described in Section III.

The algorithms provided by Kernighan and Lin assume undirected graphs. In our case the proximity of event e_i to event e_j may be different from the proximity of e_j to e_i. We may consider our sequential proximity matrix as a directed complete graph where the events correspond to nodes and weights of directed edges of the form $e_i \rightarrow e_j$ correspond to the matrix entry $M[i,j]$. In order to map our problem to the graph partitioning problem of Kernighan and Lin, we convert our directed graph to a single undirected graph by replacing the directed edges $e_i \rightarrow e_j$ and $e_j \rightarrow e_i$ with an undirected edge $< e_i, e_j >$. Weight of the edge $< e_i, e_j >$ is the sum of the weights of $e_i \rightarrow e_j$ and $e_j \rightarrow e_i$. Collapsing the directed edges connecting the same nodes to an undirected edge does not affect the outcome of the partition since either both of the edges or none of them are in the cutset.

METIS Software package can be used for graph partitioning [26]. The event graph that resulted from the sequential proximity matrix given in Figure 9 is provided in Figure 10. The nodes of the graph represent the events and the edges between the nodes represent the similarities between the corresponding events. The undirected edges of the graph are obtained by summing the weights of the edges with the same end-points. Two event sets resulting from the partitioning step are shown in Figure 10 by ellipses with dashed lines. The first set, S_1 consists of the events a_h, b_h, and c_l while the second set, S_2 consists of a_l, b_l, c_h. Similar to Example III.1, a and b here correspond to the power consumption of industry and c corresponds to a residential customer. This grouping may show us that in the snapshot of the power consumption event history, high power consumption of a residential customer corresponds to low consumption of industry. This situation may occur in summer where lots of ACs are operated in the houses and the electricity price goes up forcing the large scale companies to lower their production resulting in lower power consumption in the industry.

B. Computation of Membership Functions

One method of obtaining the membership functions for fuzzy event sets is to rely on the experts of the particular application which is usually the case in fuzzy control. Since we have the sequential proximity matrix ready at hand, we can utilize it to construct fuzzy event sets. As we have explained in Section IV-A, fuzzy event sets are constructed using a graph partitioning algorithm. Graph partitioning is used to cluster strongly related events. We can deduce the *average coherence* among the events in a fuzzy event set by taking the average proximity of all event pairs. The proximity between the event pair (e_i, e_j) is denoted by $proximity(e_i, e_j)$ and it is the value stored in the i^th row and j^th column in the sequential proximity matrix. The average coherence is further used for determining the membership function of a fuzzy event set.

Definition IV.2: Coherence of an event e_i in a fuzzy event set S is denoted by $C_S(e_i)$, and defined as the average proximity of event e_i to the rest of events in S. Coherence is calculated as follows:

$$C_S(e_i) = \frac{\sum_{e_j \in S \wedge i \neq j} proximity(e_i, e_j)}{|S|-1}$$

where $|S|$ is the cardinality of the fuzzy event set S and $proximity(e_i, e_j)$ is the proximity of e_i to e_j. We exclude the coherence of an event to itself in coherence calculation.

Coherence of an event e, in a fuzzy event set S specifies how closely e is related to S.

Definition IV.3: Average Coherence for a fuzzy event set S is denoted by AC_S and calculated as follows:

$$AC_S = \frac{\sum_{ei \in S} C_S(e_i)}{|S|}$$

Definition IV.4: Coherence Deviation of an event e_i, in a fuzzy event set S is denoted by $CD_S(e_i)$ and calculated as:

$$CD_S(e_i) = C_S(e_i) - AC_S.$$

Maximum absolute coherence deviation for a fuzzy event set S, denoted by $MaxCD_S$, is the maximum of absolute coherence deviations in that fuzzy event set, i.e., $MaxCD_S = max_{e_i \in S}\{|CD_S(e_i)|\}$.

Coherence Deviation is a measure of how far is the coherence of an event from the average coherence.

Definition IV.5: Mean Absolute Deviation of a fuzzy event set S is denoted by MAD_S and calculated as:

$$MAD_S = \frac{\sum_{ei \in S} |CD_S(e_i)|}{|S|}$$

Mean Absolute Deviation is an approximate measure of how close are the different pairwise similarity values in a fuzzy event set.

Lemma IV.1: The formula, $\frac{CD_S(e_i)\times MAD_S}{2\times MaxCD_S^2}+0.5$ produces a number in the range $[0,1]$ when $MaxCD_S \neq 0$.

Proof: For any event e_i, $CD_S(e_i)$ is in the range $(-\infty,\infty)$, while $MaxCD_S$ and MAD_S are both in the range $[0,\infty)$. By definition, $CD_S(e_i) \leq MaxCD_S$, and also $MAD_S \leq MaxCD_S$. Therefore, we have
$-1 \leq \frac{CD_S(e_i)}{MaxCD_S} \leq 1$ and $-1 \leq \frac{MAD_S}{MaxCD_S} \leq 1$.
Since $MaxCD_S \neq 0$, both of the fractions are defined.
$-1 \leq \frac{CD_S(e_i)}{MaxCD_S} \leq 1$ and $-1 \leq \frac{MAD_S}{MaxCD_S} \leq 1$ together imply that $-1 \leq \frac{CD_S(e_i)\times MAD_S}{MaxCD_S\times MaxCD_S} \leq 1$.
This leads to, $-0.5 \leq \frac{CD_S(e_i)\times MAD_S}{2\times(MaxCD_S)^2} \leq 0.5$.
When we add 0.5 to the inequality, we get
$0 \leq \frac{CD_S(e_i)\times MAD_S}{2\times(MaxCD_S)^2}+0.5 \leq 1$.
Thus, $\frac{CD_S(e_i)\times MAD_S}{2\times(MaxCD_S)^2}+0.5$ is in the range $[0,1]$ ∎

The membership function of a fuzzy event set S is denoted by μ_S and calculated as follows:

$$\mu_S(e_i) = \begin{cases} 0 & if\ e_i \notin S \\ \frac{CD_S(e_i)\times MAD_S}{2\times MaxCD_S^2}+0.5 & if\ e_i \in S \wedge MaxCD_S \neq 0 \\ 1 & if\ e_i \in S \wedge MaxCD_S = 0 \end{cases} \quad (1)$$

An example for the membership calculation for the graph in Figure 10 is given below:

Example IV.1: There are two fuzzy event sets, S_1 and S_2,in Figure 10. We calculate the membership function for S_1 as follows:

1. First we need to calculate the coherence for each event and find the average coherence.
   - The coherence of event a_h is $C_{S_1}(a_h) = \frac{similarity(a_h,b_h)+similarity(a_h,c_l)}{|S_1|-1}$ which equals to $C_{S_1}(a_h) = \frac{8.0+6.5}{2} = 7.25$.
   - $C_{S_1}(b_h) = \frac{similarity(b_h,a_h)+similarity(b_h,c_l)}{|S_1|-1}$ which is $C_{S_1}(b_h) = \frac{8.0+6.5}{2} = 7.25$.
   - $C_{S_1}(c_l) = \frac{similarity(c_l,a_h)+similarity(c_l,b_h)}{|S_1|-1}$ which is $C_{S_1}(c_l) = \frac{7.5+7.5}{2} = 7.5$.
2. Average coherence for S_1 is calculated as: $AC_{S_1} = \frac{C_{S_1}(a_h)+C_{S_1}(b_h)+C_{S_1}(c_l)}{|S_1|} = \frac{7.25+7.25+7.5}{3} = 7.33$
3. Coherence deviation for each event is calculated to find the maximum coherence deviation.
   - $CD_{S_1}(a_h) = C_{S_1}(a_h) - AC_{S_1} = 7.25 - 7.33 = -0.08$
   - $CD_{S_1}(b_h) = C_{S_1}(b_h) - AC_{S_1} = 7.25 - 7.33 = -0.08$
   - $CD_{S_1}(c_l) = C_{S_1}(c_l) - AC_{S_1} = 7.5 - 7.33 = 0.17$
   - Maximum absolute coherence deviation is, $MaxCD_{S_1} = 0.17$
4. Mean Absolute Deviation is calculated as: $MAD_{S_1} = \frac{|-0.08|+|-0.08|+|0.17|}{3} = 0.11$

After obtaining all the intermediate values needed, we can calculate the membership values of the individual events as:

- $\mu_{S_1}(a_h) = \frac{CD_{S_1}(a_h)\times MAD_{S_1}}{2\times MaxCD_{S_1}^2}+0.5$ which is $\frac{-0.08\times 0.11}{2\times(0.17)^2}+0.5 = 0.35$
- $\mu_{S_1}(b_h) = \frac{-0.08\times 0.11}{2\times(0.17)^2}+0.5$ which is also 0.35.
- Finally the membership value of c_l is $\mu_{S_1}(c_l) = \frac{0.17\times 0.11}{2\times(0.17)^2}+0.5 = 0.82$

V. Application of Sequential Proximity Matrix and Fuzzy Event Sets to Active Databases

Fuzzy rules have been involved in many areas from control to expert systems. The application of fuzzy concepts into Active Database Management Systems (ADBMSs) has attracted the attention of some researchers; however, we cannot say that a considerable amount of research has been conducted addressing this issue. This section is devoted to the description of fuzzy rule execution in ADBMSs. In the following subsections we describe how fuzzy event sets are used for fuzzy rule execution and how sequential proximity matrix is used for similarity based event detection in ADBMSs.

A. Fuzzy Rules in Active Database Management Systems

Although incorporation of fuzziness to active databases introduces much flexibility, not much attention has been paid so far to this issue. To the best of our knowledge, only a research group in VTT (Finland) has worked on fuzzy triggers [27], [15], [28], [14]. In [28], a *Condition-Action (CA)* fuzzy trigger was proposed which means that fuzziness was introduced to the CA part of an *Event-Condition-Action (ECA)* rule. In a later work [15], the concept of CA trigger was extended to a fuzzy ECA rule by introducing the notion of fuzzy events. A CA fuzzy trigger consists of a fuzzy predicate (i.e., a predicate that has linguistic terms) on the database as its condition, and a fuzzy action which is an overall conclusion obtained after evaluating fuzzy conditions. Wolski and Bouaziz compiled their previous work on fuzzy ECA rules and based their contributions on a sound theoretical background in [14]. A rule with a fuzzy condition and a crisp action is called a *C-fuzzy* trigger. The C-fuzzy trigger model is based on linguistic terms. The *max-min* inference method is applied to the rule set to determine the truth value of the fuzzy predicates. In fuzzy ECA rules, an event may fire a set of rules. Fuzzy events are defined as fuzzy sets and use linguistic terms like *high, low,* and *strong* [15]. Formally a primitive fuzzy event is represented as a tuple $< e_c, e_f >$ where e_c is a crisp event, and e_f is a fuzzy event predicate. When a crisp event is signaled (such as a database update), the current value v produced upon the operation causing the crisp event is fed into the membership function of e_f. The output of the membership function is called the *event match factor*, and the fuzzy event is signaled only if the event match factor is greater than zero [15]. Upon the occurrence of the fuzzy event, the corresponding rules are fired and their conditions (which are fuzzy predicates on the database) are checked. The action of a rule may be started to execute depending on the result of condition evaluation. In a nuclear power plant control system, we may define rules like,

Event: On a *significant* temperature change
Condition: Check whether the water level of the river is *low*
Action: Decrease the capacity to *medium* and issue a warning signal

where the linguistic terms like *significant*, *low*, and *medium* increase the understandability of the rule. It is very difficult to give exact numbers for the temperature change and water level. Instead, predefined linguistic terms, which are in fact fuzzy sets, can be used without difficulty by the people who define the rules.

In the above rule, the event "*significant* temperature change" is actually a fuzzy event. When a temperature change event is signaled, which is a crisp event, its current value, say 2000^oC, is fed into the membership function of the corresponding fuzzy event (i.e., "*significant* temperature change"). Assuming that the membership function of the fuzzy event is μ_f, then the value $\mu_f(2000)$ is called the event match factor.

Another possible application area is the stock exchange market. Price changes of certain stocks can mimic the price changes of some other related stocks, and dealers can take actions accordingly. A rule for a stock exchange market control system can be defined as

Event: On a *considerable* price reduction in stock A
Condition: $TRUE$
Action: Sell a *considerable* amount of stock A and stock B

In the above rule, the event, *considerable* price reduction in stock A, is a fuzzy event. When a price reduction event is signaled, its value, say 12% is fed into the membership function of the corresponding fuzzy event and the result is used in fuzzy rule execution. Such a rule is very useful since it automatically issues an action upon the occurrence of an event, therefore reducing the time required to take an action.

The rule execution model provided in [29] is modified and depicted in Figure 11. Events are detected by the event detector using the sensor values that are continuously stored in the database. Rules corresponding to the detected events are obtained from the rule base and their conditions are evaluated. Those rules whose conditions are satisfied are scheduled for execution by the scheduler. As an extension to this model, we add the component in the rectangle with dashed lines. In this modified model, detected events are stored in the event history and they are fed to the mining process periodically. The mining process together with the partitioning, produces the fuzzy event sets which are further used by the event detector. Utilization of the fuzzy event sets and sequential proximity matrix is explained in the following subsections.

B. Fuzzy Rule Execution and Fuzzy Event Sets

In the previous sections, we have shown how we can partition the whole event space ξ into fuzzy event sets. Scenarios in the active database context are the aggregation of fuzzy event sets and rules. The idea of scenarios comes from the need to group rules into sets corresponding to different situations such as a "high power consumption scenario" in the summer. Dividing the rule space into scenarios has many advantages. First of all, dealing with a group of related rules is much easier than dealing with all the rules. Also, the number of rules triggered can be tuned by pruning the triggered rules using the membership function of the scenario as will be explained later on. This combined effect of decreased number of events and rules in consideration will improve the efficiency and effectiveness of the system.

Assume that we have a set of fuzzy event sets $\{E_{F_1}, E_{F_2}, ..., E_{F_k}\}$ where $E_{F_i} =< E_i, \mu_i >$. Assume also that the set of rules in the system is R where each rule is a triple of the form $< e, c, a >$, e being the event, c being the condition, and a being the action. Let $events(R_i) = \{e | r :< e, c, a >\in R_i\}$ be the events of the rule set R_i. We partition the set of rules R, into subsets, $\{R_1, R_2, ..., R_k\}$ such that $events(R_j) \subseteq E_j$ for some fuzzy event set $E_{F_j} =< E_j, \mu_j >$ and $R_1 \cup R_2 \cup ... \cup R_k = R$ and for $1 \leq i < j \leq k$, $events(R_i) \cap events(R_j) = \emptyset$. The scenario corresponding to R_i, and E_{F_j}, $1 \leq i, j \leq k$, is a triple $< R_i, E_j, \mu_j >$.

Example V.1: Assume that we have the set of rules
$R = \{r_1 :< a_h, c_1, a_1 >, r_2 :< b_h, c_2, a_2 >,$
$r_3 :< a_l, c_3, a_3 >, r_4 :< b_l, c4, a_4 >\}$
and the two fuzzy event sets
$E_{F_1} =< \{a_h, b_h, c_l\}, \mu_1 >$ and $E_{F_2} =< \{a_l, b_l, c_h\}, \mu_2 >$.
We first partition R into
$R_1 = \{r_1 :< a_h, c1, a_1 >, r_2 :< b_h, c_2, a_2 >\}$ and
$R_2 = \{r_3 :< a_l, c_3, a_3 >, r_4 :< b_l, c4, a_4 >\}$.
The corresponding scenarios are
$< R_1, E_1, \mu_1 >$ and $< R_2, E_2, \mu_2 >$.

Event signaling is done by considering the membership degree of the event parameter in the fuzzy event. We use the fuzzy event structure described in [15] where a primitive event is a tuple, $e :< e_c, e_f >$, consisting of a crisp part e_c which is the crisp parameter coming from the system and a fuzzy part e_f which denotes the fuzzy event.

Definition V.1: The strength of an event $e :< e_c, e_f >$ for the rule $r :< e, c, a >$ in scenario $R_{S_i} :< R_i, E_i, \mu_i >$ is defined as :

$$\text{strength(e,r)} = \mu_i * \mu_{e_f}(value(e_c))$$

where $value(e_c)$ is the value of the crisp event detected, and μ_{e_f} is the membership function of the fuzzy event e_f. The value of an event can be e_c itself in case e_c is coming from a sensor. Or the value can be a parameter to the crisp event such as a database update event and the updated value.

Each rule has a firing threshold which is used to decide if a rule will be fired or not. In order to decide whether a rule r will be fired in response to the signaling of a fuzzy event e, the strength of event e for rule r is calculated and the result is compared to the threshold value for rule r. If the

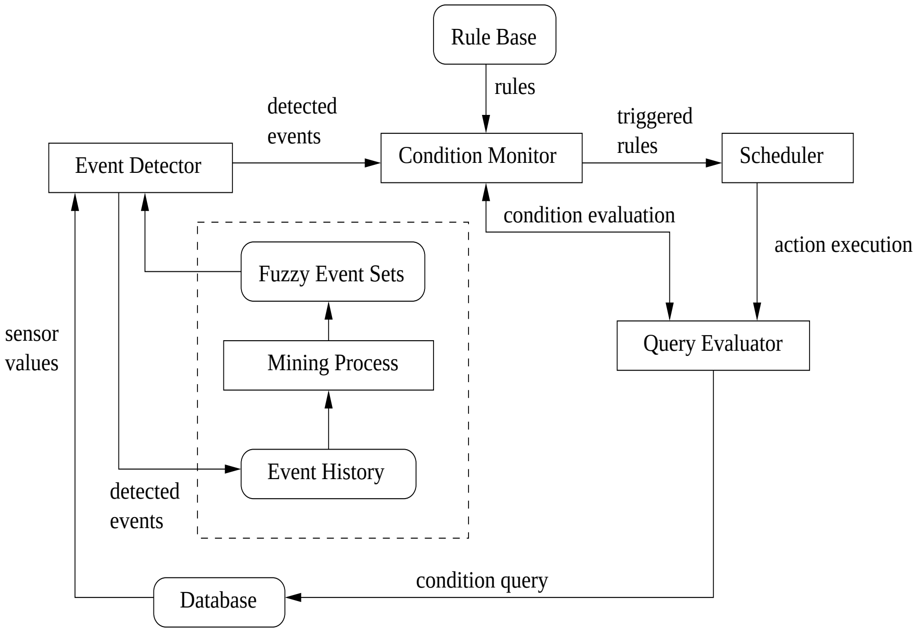

Fig. 11. Event Detection and Rule Execution Model.

result is greater than or equal to the threshold value, then the rule is fired. Threshold values of rules can be changed dynamically to tune to particular scenarios.

Prioritization of rules in an ADBMS is very important. Some rules may be more urgent than the others. The priority of rules can be set by the people who defined the rules. Another method is to use the events of the rules to set the priorities. Consider two rules whose events are "significant temperature change" and "significant pressure change". If the membership value of the significant temperature change is much higher than the membership value for the pressure change, the rule with event "significant temperature change" should be given higher priority of execution since late execution of that rule may be hazardous. Strength of the rules is a good indicator of rule priorities and they can be used to order the rules during rule execution. Prioritization based on the strength of the rules will give higher priority of execution to the rules with higher strength values.

C. Similarity Based Event Detection

Detection of similar events upon an event occurrence is something very useful when the cost of missing events is very high in supported applications, like a nuclear reactor control system. Assume that an event such as "update in temperature level" is detected. Events with a high similarity degree, like "update in pressure level" should also be signaled automatically. This way, the system can predict that an event which escaped the event detection process has occurred. In a power producing and selling company, occurrence of some events can imply the occurrence of some other events. Advantage of similarity based event detection in such a system is that, the dealers may assume that the events similar to the signaled events will occur and can take the necessary actions in advance.

In similarity based event detection, when an event is signaled, other events which are similar to it should also be fired. Similarity based event detection considers only primitive events. In order to facilitate similarity based event detection, the sequential proximity matrix explained in Section III is utilized. We also need similarity thresholds in order to avoid the system to continuously detect irrelevant events via similarity based event detection.

Definition V.2: Similarity threshold for an event e in a scenario S is the minimum similarity requirement for the events in S to be detected automatically by e via similarity based event detection.

Similarity threshold for an event e in a scenario S can be set to be the coherence of e in S which is explained in Section IV-B. Upon the detection of an event, e, all the events in the same scenarios as e whose similarity to e is grater than the similarity threshold will be detected automatically. Similarities of the events in a scenario are the proximity values in the sequential proximity matrix. An example would be helpful in explaining similarity based event detection.

Example V.2: Consider the events and the sequential proximity matrix in Figure 9 and the fuzzy event sets in Figure 10. Assume that the current scenario is S_1 and a_h is detected. Coherence of a_h in S_1 was calculated in Example IV.1 as 7.25. Among the rest of the events in S_1 which are b_h and c_l, only b_h will be detected by similarity based event detection since $proximity(a_h, b_h) = 8.0 > 7.25$, but c_l will not be detected since $proximity(a_h, c_l) = 6.5 \leq 7.25$.

Conventional event detection in active databases is a special case of similarity based event detection where the similarity relation among the events is an identity relation and similarity thresholds are equal to infinity.

VI. Conclusions

In this paper, we have proposed some methods for automated construction of fuzzy event sets which are sets of events where each event has a degree of membership to a set. Fuzzy event sets are constructed by analyzing event histories. We have shown how an event history can be mined to produce the sequential proximity matrix which stores similarities among events in terms of the proximity of event occurrence. We have proposed a sliding window algorithm for mining event histories and proposed an automated rule modularization method that does not rely on semantic knowledge. As an application platform, we have chosen active databases. The sequential proximity matrix produced by the sliding window algorithm is used for similarity based event detection in active databases. Fuzzy event sets are also constructed by the sequential proximity matrix. Construction of fuzzy event sets is mapped to the well known graph partitioning problem, and fuzzy event sets are constructed using graph partitioning algorithms.

Incorporation of fuzziness to rule execution in Active Database Management Systems is still an open and important research area. Fuzzy Event-Condition-Action (ECA) rules are obviously more flexible compared to standard ECA rules. With fuzzy events, we can involve a priority assignment mechanism based on membership functions of fuzzy events. Organization of rules into scenarios, which are mapped to fuzzy event sets, increases the efficiency and modularity of rule execution. In summary, the main contribution of our work is to propose the idea of fuzzy rule execution via scenarios and to describe how it could be realized using fuzzy event sets. As a future work, we are planning to define new similarity metrics other than the proximity of event occurrences and construct the similarity matrix according to these new metrics.

Acknowledgments

The authors wish to thank Dr. Adnan Yazıcı for his help in establishing the fuzzy concept background, and Dr. Erdem Başçı for his comments in the mathematical formulations of this work.

References

[1] L.A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

[2] W. Pedrycz, *Fuzzy Control and Fuzzy Systems*, Research Studies Press, 1993.

[3] B. Buckles and F. Petry, "A fuzzy model for relational databases," *International Journal of Fuzzy Sets and Systems*, vol. 7, pp. 213–226, 1982.

[4] K.S. Leung and W. Lam, "Fuzzy concepts in expert systems," *IEEE Computer*, September 1998.

[5] E. Baralis, S. Ceri, and S. Paraboschi, "Modularization techniques for active rules design," *ACM Transactions on Database Systems*, vol. 21, no. 1, 1996.

[6] Y. Saygın, O. Ulusoy, and A. Yazıcı, "Dealing with fuzziness in active mobile databases," *Information Sciences*, vol. 120, no. 1-4, 1999.

[7] E. N. Hanson and J. Widom, "*An Overview of Production Rules in Database Systems*," Tech. Rep., University of Florida, Department of Computer and Information Sciences, October 1992.

[8] R. George, R. Srikanth, F. E. Petry, and B. P. Buckles, "Uncertainty management issues in object-oriented database systems," *IEEE Trans. on Fuzzy Systems*, vol. 4, no. 2, pp. 179–192, May 1996.

[9] B. P. Buckles and F. E. Petry, "Information-theoretic characterization of fuzzy relational databases," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 13, no. 1, pp. 74–77, Feb. 1983.

[10] A. Yazici, B. P. Buckles, and F. E. Petry, "Handling Complex and Uncertain Information in the ExIFO and NF^2 Data Models," *IEEE Trans. on Fuzzy Systems*, vol. 7, no. 6, pp. 659–676, 1999.

[11] A. Yazici, A. Soysal, B. P. Buckles, and F. E. Petry, "Uncertainty in a nested relational database model," *Data and Knowledge Engineering*, vol. 30, no. 3, pp. 275–301, 1999.

[12] A. Yazici and R. George, *Fuzzy Database Modeling*, Springer-Verlag, Heidelberg, 1999.

[13] F. E. Petry, *Fuzzy Databases: Principles and Applications*, Kluwer Press, 1996.

[14] A. Wolski and T. Bouaziz, "Fuzzy triggers: Incorporating imprecise reasoning into active databases," in *Proceedings of the 14th Int'l Conference on Data Engineering(ICDE'98)*, Orlando, Florida, February 1998.

[15] T. Bouaziz and A. Wolski, "Applying fuzzy events to approximate reasoning in active databases," in *Proceedings of the 6th IEEE International Conference on Fuzzy Systems(FUZZ-IEEE'97)*, Barcelona, Catalonia, Spain, July 1997.

[16] U. Dayal, "Active Database Management Systems," in *Proceedings of the Third International Conference on Data and Knowledge Bases*, Jerusalem, June 1988, pp. 150–169.

[17] Y. Saygın, O. Ulusoy, and S. Chakravarthy, "Concurrent rule execution in active databases," *Information Systems*, vol. 23, no. 1, 1998.

[18] Y. Saygın and O. Ulusoy, "Involving fuzzy concepts in active mobile databases," in *Proceedings of the 9th International Conference and Workshop on Database and Expert Systems Applications (DEXA'98)*. 1998, Lecture Notes in Computer Science, Springer Verlag.

[19] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami, "An interval classifier for database mining applications," in *Proceedings of the 18th Int'l Conference on Very Large Databases*, Vancouver, August 1992, pp. 560–573.

[20] R. Agrawal R. Srikant, "Mining generalized association rules," in *Proceedings of the 21st Int'l Conference on Very Large Databases*, Zurich, Switzerland, September 1995.

[21] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proceedings of the 11th Int'l Conference on Data Engineering (ICDE'95)*, Taipei, Taiwan, March 1995.

[22] C. Bettini, X. S. Wang, S. Jajodia, and J. Lin, "Discovering frequent event patterns with multiple granularities in time sequences," *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, no. 2, 1998.

[23] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of frequent episodes in event sequences," in *Proceedings of the 1st International Conference on Knowledge Discovery and Datamining*, Montreal, Canada, August 1995.

[24] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th Int'l Conference on Very Large Databases*, Santiago, Chile, September 1994.

[25] B. W. Kerninghan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 49, no. 2, 1970.

[26] G. Karypis and V. Kumar, *METIS Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparce Matrices, Version 4.0*, University of Minnesota, Department of Computer Science/ Army HPC Research Center, Mineappolis, MN 55455.

[27] T. Bouaziz and A. Wolski, "*Incorporating Fuzzy Inference into Database Triggers*," Tech. Rep., VTT Information Technology, November 1996.

[28] T. Bouaziz, J. Karvonen, A. Pesonen, and A. Wolski, "Design and implementation of tempo fuzzy triggers," in *Lecture Notes in Computer Science*, Toulouse, Fran, September 1997, Springer, vol. 1308, pp. 91–100.

[29] Norman W. Paton and O. Diaz, "Active database systems," *ACM Computing Surveys*, vol. 31, no. 1, pp. 1–29, 1999.