MINING TEMPORAL SEQUENCES TO DISCOVER INTERESTING PATTERNS

Edwin O. Heierman, III, G. Michael Youngblood, Diane J. Cook
Department of Computer Science
The University of Texas at Arlington
Arlington, Texas 76019-0015
{heierman, youngbld, cook@cse.uta.edu

ABSTRACT
When mining temporal sequences, knowledge discovery techniques can be applied that discover interesting patterns of interactions. Existing approaches use frequency, and sometimes length, as measurements for interestingness. Because these are temporal sequences, additional characteristics, such as periodicity, may also be interesting. We propose that information theoretic principles can be used to evaluate interesting characteristics of time-ordered input sequences. In this paper, we present a novel data mining technique based on the Minimum Description Length principle that discovers interesting features in a time-ordered sequence. We discuss features of our real-time mining approach, show applications of the knowledge mined by the approach, and present a technique to bootstrap a decision maker from the mined patterns.

Categories and Subject Descriptors
Mining data streams, novel data mining algorithms, preprocessing and post processing for data mining, spatial and temporal data mining.

General Terms
Algorithms.

Keywords
Discovering interesting episodes, knowledge discovery, mining sequential data streams.

1. INTRODUCTION
With the proliferation of computers comes the proliferation of data created by these computers. Every interaction with a computer system or sensor can be recorded and preserved. It is this abundance of data that has resulted in the emergence of the field known as Data Mining. The number of computer systems and the data these systems collect has far surpassed our ability to review and understand the collected data, so we turn to the computer itself to help us automatically analyze the data for important information.

Data mining techniques have evolved that perform sequential pattern mining by processing time-ordered input streams and discovering the most frequently occurring patterns. Because these input sequences contain temporal information, additional interesting characteristics, such as periodicity, may also exist in the data. The world, at times, tends to operate in a periodic manner. The earth orbits around the sun every 365 days, and completes a revolution about its axis every twenty-four hours. In addition, mankind has subdivided these periods into years, months, weeks, days, hours, and seconds. By doing so, we can predict when the sun will rise again, when winter will approach, and when the earth once again will orbit around the sun. By taking advantage of the periodic behavior of our environment to organize time, we also tend to exhibit behavior that is periodic. We rise at about the same time everyday to eat. We go to work five days a week, and take the weekends off. Our favorite TV programs are shown at the same time on the same day of each week. These periodic interactions provide predictability and add stability to our lives. Thus, data collected by a computer system responding to interactions with a human or an environmental occurrence may also contain patterns that are governed by periodic behavior.

In our work, we use information theoretic principles to evaluate characteristics in an input sequence other than frequency, such as length and periodicity. By using information theory as a foundation, we anticipate that other characteristics can also be evaluated for interestingness. In this paper, we describe our work, discuss the knowledge that is discovered, and present a technique that uses the mined knowledge to bootstrap a decision maker that operates in a real-time environment.

2. SEQUENTIAL PATTERN MINING
Our work is related to techniques for sequential data mining, which is the task of mining frequently occurring patterns related in time or to other sequences [3]. An example of a sequential pattern is:

> *An individual who bought a car three months ago is likely to change the oil in the car within the next two weeks.*

A common characteristic of techniques that mine sequences is the discovery of patterns that are frequent [1][6]. The more frequently a pattern occurs, the more likely it is that the pattern is important. In addition, some approaches also take into account the length of a pattern [1]. Important knowledge is provided if a system can report that the symbols *{a, b}* occur frequently together, as

KDD'04, August 22–25, 2004, Seattle, WA, USA.

opposed to only reporting that *{a}* occurs frequently and *{b}* occurs frequently. However, little attention has been given to evaluating other characteristics of the patterns, such as periodicity. This is surprising, given the emphasis individuals place on organizing their time. Thus, we are interested in a data-mining technique that can discover features about a pattern like the following:

> *At 6:00pm on Monday, Wednesday, and Friday of every other week, Dave leaves work and travels to the gym.*

There are several important attributes of the problem we are investigating that influence our work. First, our data source will be time-ordered input sequences that consist of occurrences of events with no natural points that indicate the start or stop of the pattern. Therefore, our approach must partition the input sequence by grouping together interactions that are related. Second, the nature of the patterns is not known a priori. As a result, the pattern-ordering characteristics (e.g., ordered, unordered) must also be discovered. Third, frequency is only one of the characteristics that should be evaluated. Additional features, such as pattern length and periodicity (occurs daily, weekly), should also be considered. Therefore, the approach will need to balance the values of more than one interestingness measurement.

With these attributes in mind, we have developed an Episode Discovery (ED) algorithm that processes a time-ordered input stream and identifies clusters of events, or episodes, that are closely related in time. These episodes are represented as an unordered set of events. Repeating symbolic patterns in the episodes are evaluated with an approach based on the Minimum Description Length principle. A pattern is described with an encoding, which may result in a reduction in the description length of the original input sequence. The symbolic-pattern encodings that result in the greatest compression of the input sequence are selected as interesting, and the associated patterns are presented to the user as interesting episodes. The details of our approach are presented in the next section.

Several works address mining sequential patterns. Agrawal and Srikant [1] present three techniques for mining sequential patterns from time-ordered transactions. Each transaction is a set of items, and variations of the Apriori property are used to find large sequences by first computing the large itemsets and then constructing the large sequences. Finally, selecting the sequences of largest length first and then pruning the subsets of that sequence yields the set of maximal sequences. Mannila, Toivonen, and Verkamo [6] consider the problem of discovering frequent episodes in an event sequence. A user-defined event window partitions the event stream into overlapping collections of events that are close to each other in time. These collections are examined to find frequent parallel and serial episodes by use of the Apriori property. All frequent episodes are output by the algorithm. Srikant and Agrawal [8] extend their previous work by supporting maximum and minimum time gaps between sequence elements, a sliding time window (event window) across transaction sets, and user-defined taxonomies. The support used for the Apriori-based algorithm is supplanted with additional measures that account for the new features.

3. EPISODE DISCOVERY

We view our data-mining task as the process of describing a time-ordered input sequence with encodings that represent the interesting characteristics that may be present in the sequence. Rissanen's work [7] on the Minimum Description Length (MDL) principle, which proposes searching for models and model classes with the shortest description length, serves as the underlying theory for our work. Rissanen reasons that by using an encoding of a dataset to reduce its description length, constraints are applied to the data that reduce the uncertainty about the nature of the data. Here, encoding is used as a general term to mean an exact representation. The resulting "minimum-encoded" description captures the properties that provide the most likely explanation for the data. Stated another way, the model defines a distribution that assigns the maximum probability to the observed data. Therefore, we will define encodings that encompass interesting characteristics, and use these encodings to find a model that exhibits the minimum description length. The end result will be descriptions that provide a likely explanation of the input.

We reason that if a pattern exhibits periodicity, then describing the pattern with terms that include its periodicity would reduce the overall description length of the input sequence because the occurrences of the periodic pattern can be replaced by the description. In addition, because patterns in the sequence may not be periodic, an encoding will need to be defined that describes non-periodic patterns. Finally, we consider pattern length and frequency to also be interesting characteristics, so the encodings should also describe these features. We anticipate that "more frequent" and "greater length" encodings will also reduce the description length of the input sequence. We now present our MDL-based approach that makes use of encodings to discover if patterns present in a time-ordered input sequence exhibit these interesting characteristics.

3.1 Terminology

We start our discussion by defining the input, processing, and output of our approach. The time-ordered input sequence is defined as follows. We will let Π represent the set of all possible symbols or interactions that could appear in an input stream (the domain alphabet), and let s represent any symbol that is a member of Π. An event δ is denoted as the pair (s, t), where t is a timestamp consisting of time and date information. An event sequence O is defined as an ordered sequence of events, $O = (\delta_1, \delta_2, \ldots, \delta_n)$. The event sequence consists of all events that have occurred up to the point in time represented by t_n. The sequence must be in non-descending order based on the timestamp value, such that $t_i \leq t_{i+1}$. Events in which the timestamp values are the same can be placed in any order.

Our objective is to discover the symbolic patterns in O that are considered interesting. In order to find these patterns, we will collect sets of events from the input sequence into an ordered collection, which we refer to as episodes. Formally, we define an episode ε to be a sequence of event occurrences, $\varepsilon = (\delta_j, \delta_{j+1}, \delta_{j+2}, \ldots, \delta_{j+m})$, which is a sub-sequence of O starting at time stamp t_j and ending at t_{j+m}. Upon partitioning the input sequence into overlapping episodes, the algorithm will evaluate patterns found in multiple episodes by encoding a description of the pattern occurrences. The encodings that contribute to a minimum

description length of the input sequence will be selected as interesting.

ED will evaluate if a pattern occurs at repeatable time intervals. An interval may consist of just a single value or a sequence of values. In order to explain an interval consisting of a single value, assume the existence of symbols *a* and *b*. After partitioning *O* into episodes, ED discovers that the symbols *a* and *b* appear together in multiple episodes, and that these episodes occur every 24 hours. Intuitively, then, this represents a situation in which the symbols *{a, b}* occur on a daily basis.

The interval may also be an ordered sequence of values, such as 48 hours, 48 hours, and 72 hours. Once again using the symbols *a* and *b*, assume ED has discovered a situation in which these symbols appear together in multiple episodes, and that the time interval between the episodes repeats the pattern 48 hours, 48 hours, and 72 hours over and over. We will assume the first episode occurs on Monday. There is 48 hours until the next occurrence of an episode containing *{a, b}*, which now makes it Wednesday. After an additional 48 hours passes, now making it Friday, another episode occurs that contains *{a, b}*. Finally, 72 hours elapses until another occurrence of an episode containing *{a, b}*, which makes it Monday again. Thus, *{a, b}* occurs on Monday, Wednesday, and Friday of every week.

ED will output a collection of interesting episodes, ω, that are discovered in an input sequence *O*. We will let λ, called a symbolic pattern, represent an unordered collection of symbols, $\{s_1, s_2, \ldots, s_n\}$. In the previous discussion, λ would be the set *{a, b}*. The symbolic pattern will be output as part of the discovered knowledge. Because we will treat the pattern as an unordered collection, a set of symbolic patterns that represent the permutations of λ that actually occur in the dataset will be provided as part of the output, represented as $\Lambda=\{\lambda^1, \lambda^2, \ldots, \lambda^m\}$. An example of such a set is $\Lambda=\{(a,b), (b,a)\}$, where $\lambda^1=(a, b)$ and $\lambda^2=(b, a)$. We will let $\mu=(\mu_1, \ldots, \mu_i)$ represent the repeating interval sequence. In the example above, the interval sequence would be $\mu=(48, 48, 72)$. If the interesting episode follows no repeating interval, then the sequence will be empty. Finally, $\Psi=\{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_j\}$ represents the episodes that contain the symbols defined by λ. With this notation, we define an interesting episode as $\omega = \{\lambda, \Lambda, \mu, \Psi\}$.

3.2 Pattern Encodings

We now turn our attention to the encodings that will be used. When encoding a symbolic pattern, we are interested in determining if the episodes containing the pattern repeat according to an interval sequence μ. We will evaluate two types of repeating interval sequences: fined-grained and course-grained. In order to present an example of a fine-grained interval, assume the symbols *{a, b}* occur periodically in multiple episodes on Mondays, Wednesdays, and Fridays. We will assume that *a* always occurs at 12:00:00pm, and that *b* always occurs at 12:00:30pm. An input sequence of events consisting of symbols *a* and *b*, and a possible collection of associated episodes, is shown in Table 1. We will let *x* represent a candidate episode that describes this situation based on the number of hours between the episode occurrences. Upon inspection of the table, it can be seen that *x* has the following characteristics:

$$\lambda_x = \{a, b\},$$
$$\Lambda_x = \{(a, b)\},$$
$$\mu_x = (48, 48, 72), \text{ and}$$
$$\Psi_x = \{\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4, \varepsilon_5, \varepsilon_6, \varepsilon_7\}.$$

In some situations, episodes containing common symbol patterns may not occur at exactly the same exact time interval, but can still be represented by an interval sequence. For example, it may be that an individual washes their clothes every Sunday, but not necessarily at the same time each Sunday. Consider once again the example shown in Table 1. A course-grained interval is computed by partitioning the timestamps into calendar days, which results in $\mu=(2, 2, 3)$. The interval between two timestamps that occur on the same day is zero. The time-based partition could be other categories, such as month or even year. We have chosen to use days because it is a common course-grained interval that balances the granularity of the partitions.

Table 1. Example Input Sequence of *{a, b}*.

Events	Episode	Interval
{*a*, 4/26/04,12:00:00pm} {*b*, 4/26/04,12:00:30pm}	ε_1	
{*a*, 4/28/04,12:00:00pm} {*b*, 4/28/04,12:00:30pm}	ε_2	48 hours 2 days
{*a*, 4/30/04,12:00:00pm} {*b*, 4/30/04,12:00:30pm}	ε_3	48 hours 2 days
{*a*, 5/3/04,12:00:00pm} {*b*, 5/3/04,12:00:30pm}	ε_4	72 hours 3 days
{*a*, 5/5/04,12:00:00pm} {*b*, 5/5/04,12:00:30pm}	ε_5	48 hours 2 days
{*a*, 5/7/04,12:00:00pm} {*b*, 5/7/04, 12:00:30pm}	ε_6	48 hours 2 days
{*a*, 5/10/04, 12:00:00pm} {*b*, 5/10/04, 12:00:30pm}	ε_7	72 hours 3 days

When describing patterns that follow an interval sequence, the encodings must account for mistakes that occur when the episodes do not completely follow the interval sequence. The greater the number of mistakes, the less predictable behavior the pattern displays, and the less the periodic description compresses the original input sequence. Other mistakes, such as a difference in the included events, are not part of the encoding. For example, if the pattern is *{a, b, c}* and the input data contains an occurrence of *{a, b}*, that occurrence of *{a, b}* will be evaluated as a separate pattern. Only if the mistake is a variation in start time is it counted as an encoding mistake.

We have chosen to use three potential encodings for each pattern candidate: one that reflects a fine-grained periodicity, one that reflects a course-grained periodicity, and one that reflects frequency. The periodic encodings include length, frequency, periodicity, and mistakes in periodicity. The encoding for frequency includes frequency and length characteristics, and represents a non-periodic description of the pattern.

Rissanen notes there is no way to determine if an encoding is optimal [7], so we make no guarantee that these encodings are optimal. However, we rely on this premise of the MDL principle:

> *By defining a description and searching for the minimum length using that description,*

the best model will remain as the best explanation of the observations [7].

By evaluating encodings and searching for a model that is the minimum description of the original input, at the very least the algorithm discovers a model that can be used for comparison with other models that may be discovered in the future. In addition, we will be selecting the most likely model based on our encoding, which will allow us to determine if patterns do indeed exhibit the characteristics our encodings describe.

Specifically, we will use the three encodings to compute a compression ratio for each candidate pattern. The compression ratio measures how much of the original input sequence is compressed by the encoding. A compression ratio of 2:1 implies that it takes just one unit in the compressed description to represent two units of the original input sequence. Of the three potential encodings, the encoding that results in the greatest compression ratio is selected as the encoding for that repeating pattern. The selected encoding will result in the greatest decrease in the description length of the original input sequence. Once all candidates are assigned an encoding, candidates with the greatest compression ratio will be selected as interesting episodes. By using these encodings, the description length of the input sequence will be reduced. Because candidates that provide the greatest compression are selected, the description with the minimum length will be discovered. If a fine-grained or course-grained encoding is used for an interesting episode, then we consider this candidate to be a *periodic episode*. When the frequency-based encoding is selected, then we use the term *frequent episode* to describe the interesting episode.

Using these encodings, a compression ratio was computed for the input sequence reflected by Table 1 for candidate episode *x*. These values are shown in Table 2. Based on these values, a fine-grained encoding would be selected for *x* because the encoding results in the greatest compression. We would consider *x* to be a periodic episode if it is chosen as an interesting candidate.

Table 2. Summary of Compression Ratios for *x*.

Encoding	Compression Ratio
Fine Grained	1.45
Course Grained	1.20
Frequency	1.27

3.3 Attributes of the Interestingness Measures

ED evaluates the patterns in the input sequence using encodings that incorporate length, frequency, and periodicity. Because of the encodings, the algorithm evaluates the characteristics in the following manner:

- Length - If two patterns have the same frequency and periodicity, the pattern that has a greater pattern length will be more interesting (greater compression ratio).
- Frequency - If two patterns have the same length and periodicity, the pattern that is more frequent will be more interesting.
- Periodicity - If two patterns have the same length and frequency, the pattern that contains fewer periodicity mistakes will be more interesting.

4. ALGORITHM DETAILS

The following summarizes the high-level steps of the algorithm. Given as input a stream *O* of event occurrences δ, ED:

1. Partitions the event sequence O into possibly overlapping maximal episodes, εi, by using an event-folding window W with a time span of tw and a capacity of cw.
2. Creates an initial set of candidate episodes, Ci, from the maximal episodes.
3. Creates additional candidate episodes from the subsets of the maximal episodes.
4. Computes a compression ratio for each Ci.
5. Identifies interesting episodes by evaluating the compression ratios of the candidate episodes. Additional candidate episodes may be generated when a candidate episode is selected as interesting.
6. Outputs a list of interesting episodes.

4.1 Step One: Partition the Input Sequence

ED partitions the input into maximal episodes by incrementally processing the events. An event-folding window collects the events and creates an episode when the time span or capacity of the window exceeds the corresponding parameter value. This is similar to the approach taken by Mannila, et al. [6]. The t_w parameter represents the time span of the window, and c_w the capacity. If $t_w=\infty$, then the window partitions the input sequences into episodes of size c_w. If $c_w=\infty$, then the window partitions the input sequence solely on the time span. The current time interval of the window is calculated based on the time stamp of the event being processed and t_w. Thus, if event δ_i is being added to the window, then the time interval of the window is t_i - t_w. The capacity is computed by counting the number of events currently contained in the window. When one or more events contained in the window are now outside of the specified parameter values due to the addition of the new event, those events are pruned from the window. The window contents prior to pruning are maximal for that particular window instance, and are used to generate a maximal episode. Our approach generates overlapping maximal episodes with potentially different lengths.

Table 3 shows an example of creating maximal episodes with $t_w=15$ and $c_w=\infty$. Five events are incrementally processed. We have simplified this example by considering the timestamp to be an integer value. Because events *(a, 1)*, *(b, 5)*, and *(c, 10)* all occur within the fifteen time-unit window, they are accumulated and kept as part of the episode window. When occurrence *(d, 20)* is processed, event *(a, 1)* must be removed from the window. At this point, the episode window contains the maximal episode *((a, 1), (b, 5)*, *(c, 10))*. When event *(e, 40)* is encountered, the occurrences *(b*, *5)*, *(c, 10)*, and *(d, 20)* must be removed. Thus, the window contains the maximal episode *((b, 5), (c, 10)*, (*d, 20))*. Assuming no other events are processed, the final maximal episode is *((e, 40))*.

Table 3. Creating a Maximal Episode, t_w=15 and c_w=∞.

Event	Episode Window	Start	Stop	Maximal Episodes
(a, 1)	((a, 1))	1	1	
(b, 5)	((a, 1), (b, 5))	1	5	
(c, 10)	((a, 1), (b, 5), (c, 10))	1	10	
(d, 20)	((b, 5), (c, 10), (d, 20))	5	20	((a, 1), (b, 5), (c, 10))
(e, 40)	((e, 40))	40	40	((b, 5), (c, 10), (d, 20))
<end>				((e, 40))

4.2 Steps 2 and 3: Create Candidates

The algorithm constructs the initial collection of candidate episode by creating a corresponding candidate for each maximal episode. However, additional symbolic patterns exist within each maximal episode that may need to be evaluated. For example, if the maximal episode contains the symbolic pattern *{a,b,c,d,e}*, then additional patterns that could be examined would be *{a,b,c,d}*, *{c,d,e}*, and so on. The power set of each symbolic pattern is the complete list of potential patterns that could be evaluated.

One possible approach to generating the additional candidates would be to generate the power set as new candidates. However, this would be intractable because it generates as 2^n candidates, where *n* is the number of symbols in the pattern. Thus, the candidate generation method must prune the complete set of potential candidates in a tractable manner, while ensuring that it does not eliminate any candidates that ultimately do represent interesting episodes.

The Apriori property prunes a search space by deleting non-frequent candidates, and then generating new candidates from the current list of candidates [3]. However, because frequency is not our only discriminator of interestingness, the Apriori approach does not work as a pruning technique. Nevertheless, it is possible to prune the candidate search space by selecting a subset of a symbolic pattern as an additional candidate based on one of the following conditions:

- The subset represents the intersection of a maximal episode with one or more other maximal episodes. Because the subset represents pattern occurrences in multiple episodes, it may be more significant than its parents.
- The subset represents the difference between a maximal episode and one of its episode subsets, which has been selected as an interesting pattern. In this situation, if a subset candidate is evaluated as interesting, then the remainder of the maximal episode must be evaluated to see if it is interesting.

Our approach relies on the following principle to prune the candidate space:

> *The subset candidates of a candidate episode that have the same episode occurrences as the parent episode (the episode sets of the candidate episodes are equivalent) do not need to be generated as candidates.*

Because these subset patterns are shorter in length, but have the same frequency and periodicity as their parent, their compression ratio cannot be greater than their parent's value. Our pruning method generates patterns of shorter lengths from longer ones, which is essentially the opposite of an Apriori approach [1] where larger itemsets are generated from smaller ones.

An example of the algorithm incrementally generating candidates is shown in Table 4. For simplicity, in the example it is assumed that one maximal episode is identified each day, and the timestamps have been omitted. The creation of the first maximal episode generates one candidate, *{a, b, c, d}*. The second maximal episode results in the generation of two additional candidates: *{a, b, c, e}* from the maximal episode, and *{a, b, c}* from the intersection of the two maximal episode. The last maximal episode results in the generation of four additional candidates. By inspection, it can be seen that *{a, b}* should be identified as a significant episode because it occurs every day. In the example, we see that *{a, b}* is indeed generated as a candidate. Notice that it is not necessary to generate *{a}* and *{b}* as candidates, because these subsets of *{a, b}* only occur in episodes that contain both *a* and *b*. The pruning technique effectively eliminates these and several other unnecessary candidates from consideration. It is also important to note that in Steps 2 and 3, only those candidates that are common across multiple episodes are generated. Candidates that need to be created because a candidate is selected as interesting are generated in Step 5.

Table 4. Generating Candidates.

Day	Maximal Episodes	List of Generated Candidates
1	(a, b, c, d)	{{a, b, c, d}}
2	(a, b, c, e)	{{a, b, c, d}, {a, b, c}, {a, b, c, e}}
3	(a, b, d, e)	{{a, b, c, d}, {a, b, c}, {a, b, c, e}, {a, b, d}, {a, b, e}, {a, b}, {a, b, d, e}}

4.3 Step 4: Compute compression ratios

The list of maximal episodes is walked, so that the episode set of each C_n is updated with those maximal episodes containing the symbolic pattern represented by the candidate. Once the episode assignments have been completed, the algorithm uses auto-correlation techniques to determine the best repeating interval sequence. In Table 1, we presented an example in which the interval sequence *{48, 48, 72}* repeats. By calculating auto-correlation values, it can be discovered that a pattern repeats and is of length three. The auto-correlation analysis is performed for the fine-grained and course-grained interval sequence. Once the search for an interval sequence has been completed, the individual entries are evaluated to identify any mistakes that occur in the repeating pattern. Then, a fine-grained, course-grained, and frequency compression ratio is computed. The ratio with the largest value is selected as the compression for that candidate. The algorithm steps are shown in Figure 1.

```
foreach maximal episode ε_i
   foreach candidate C_n that contains ε_i
      add ε_i to the episode set of
      candidate C_n
      foreach candidate C_m
         compute compression ratios
         select largest compression
         ratio
```

Figure 1. Steps for Computing Compression Ratios.

4.4 Step 5 and 6: Select and output interesting episodes

The candidates are sorted, and the algorithm greedily identifies an interesting episode by selecting from the sorted list the candidate with the largest compression ratio. The events represented by the interesting episode are marked. To avoid selecting overlapping candidates, the second and subsequent candidates are rejected if any of the events represented by the occurrences of the candidate are already marked. Once a candidate is selected, additional candidates are generated by subtracting the selected pattern from all remaining candidates containing the pattern, and generating the difference as additional candidates. This was discussed in Steps 2 and 3. In Table 4, we present an example of candidates that would be generated because they were subsets of multiple episodes. In Table 5, we show the additional candidates that would be generated if *{a,b}* were selected as interesting. These steps are repeated until all candidates have been processed. Candidates that are selected as interesting are output.

Table 5. Candidates Generated Upon Selecting *{a,b}*.

Selected Episode	Candidates	Additional Candidates
{a, b}	{{a,b,c,d}, {a,b,c}, {a,b,c,e}, {a,b,d}, {a,b,e}, {a,b}, {a,b,d,e}}	{{c,d}, {c}, {c,e}, {d}, {e}, {d,e}}

5. EPISODE DISCOVERY FEATURES

It has been empirically demonstrated [5] that ED detects symbolic patterns that exhibit periodicity, improves the performance of predictors in an intelligent environment, and operates efficiently. We now discuss additional algorithm features and other information output by ED.

5.1 Ordering Knowledge

ED can output ordering information about the pattern. Because ED tracks the different permutations of the pattern, statistics can be presented on how many permutations there are and how many times each permutation occurs. If a pattern were totally ordered, then ED would only output a single permutation. If it were unordered, then ED would output multiple patterns, with no pattern occurring significantly more often than another.

5.2 Discovering Event Folding Window Parameters

The compression ratios can be used to search for the optimal window span and capacity parameters. Based on our encodings, a window size search should start from a small window size and continue to the largest. As the window size is increased, patterns of increasing length will be evaluated. Because the encodings favor longer patterns, the overall compression should also increase because patterns of longer length will be discovered. At some point, the compression will stabilize as the optimum window size is discovered. Using a synthetic dataset representing inhabitant activities in an intelligent environment [5], we show the compression ratios plotted for various values of t_w in Figure 2. For this dataset, the compression ratio peaked at a window span of 20 minutes. The dataset was constructed such that all interesting interactions occur within a fifteen-minute time frame, which corresponds to the window span indicated by the plot. This same approach can also be used with the window capacity parameter.

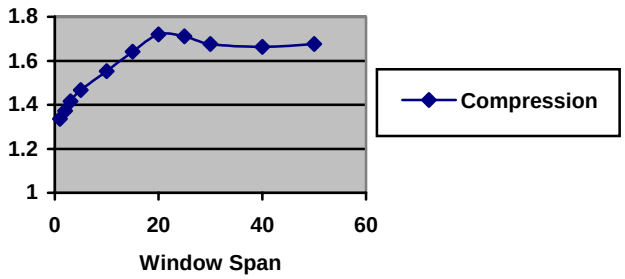

Figure 2. Window Span vs Compression Ratio.

5.3 Incremental Processing of Data and Knowledge Generation in Real-Time

Steps 1, 2, and 3 of the algorithm can be performed independently of steps 4, 5, and 6. Thus, the algorithm is able to incrementally process the events, and can be invoked when desired to produce the interesting episodes. This is important for a real-time system, because knowledge produce by the algorithm can be used at the same time the algorithm is collecting the necessary information to provide a knowledge update. In addition, it should be more efficient to generate knowledge based on an increment of data, rather than having to process the entire dataset.

To confirm our assumption, ED was run in an incremental manner by generating knowledge every month on a dataset consisting of intelligent environment events covering a nine-month period [5]. The following was observed:

- The exact same set of interesting episodes was generated once all of the input data was processed.
- It took thirty-four seconds to process the full nine months of activity data.

- It took eighteen seconds to incrementally process the ninth month. The processing time was reduced almost 50% by incrementally generating the knowledge.

5.4 Computing Interesting Episode Memberships

ED maintains the following statistical information on the interesting episodes:

1. For each interesting episode, ED maintains a list of the subsets that have been generated as candidates. Thus, we can quickly determine if a subset of a candidate has also been generated.
2. For each interesting episode, ED maintains a list of episodes that represent an occurrence of that symbolic pattern. Thus, we know how many occurrences there are of that symbolic pattern.
3. A collection of the episodes that have been generated by the event-folding window is maintained. Thus, we know the total number of episodes.

As events are processed, the contents of the current event-folding window maintained by ED can be retrieved. Using Bayes' rule and these statistics, it is possible to determine for each interesting episode the probability that the current event-folding window will contain this interesting episode. For example, if the window contains *{a,b,d,e}*, ED could report a probability of 82% that the window will eventually contain interesting episode *{a,b,c}*. Because ED incrementally processes the events, membership computation can be performed on-line as every event is processed.

6. INTEGRATING ED WITH A DECISION MAKER

We have implemented an intelligent environment architecture called MavHome [2]. The MavHome decision-making component uses the periodic episode and ordering knowledge output of ED to create a hierarchical hidden Markov model (HHMM) [3][9]. The system controls the environment by automating the most likely event based upon the current observation. This model is built from multiple passes through the dataset, which creates an increasing hierarchy of abstraction. Statistical information collected and derived through knowledge discovery is used to define the horizontal and vertical transitions of the model.

Experiments using simulated environment data have proven the ability of this approach to automatically create HHMMs from inhabitant interaction data. We have created datasets [5] containing months of inhabitant data based on stochastic simulation, organizing the data into human-perceived episodes (e.g., watching TV, entering room, exiting room, reading), and then using a simulator to distribute activities over specified periods of time. For example, we establish activity patterns that show someone entering the living room, watching TV, going to the kitchen, leaving, entering, and so forth all distributed over specified times of occurrence over a specified period. A test dataset was generated that contains twenty-three embedded behaviors. ED processed the data and found thirteen periodic episodes that correspond to those various environmental activities. A HHMM was automatically constructed, and it was manually verified that it correctly encoded the ED data. We are continuing our efforts in this area, and will eventually perform a comparison of our approach with other techniques.

7. CONCLUSIONS

In this paper, we have shown that the ED algorithm automatically detects regularity intervals, provides statistics on pattern ordering, and computes interesting episode membership values. We have also demonstrated that ED supports discovery of the parameters for the event-folding window, can be operated in an incremental manner to support real-time environments, and can be used to bootstrap the states of a decision maker.

In our future work, we intend to evaluate adding ordering as a characteristic of the encodings. We also are investigating using the compression ratios to understand drift and shift. In addition, we will investigate temporal aspects of the membership calculation in order to provide even more information on the likelihood of an episode occurrence. Finally, we will continue the integration of ED with a decision maker, and ultimately incorporate the technique into a setting with live inhabitants.

8. REFERENCES

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 11th International Conference Data Engineering (ICDE 1995)*, pp. 3-14, Taipei, Taiwan, March 1995.

[2] S. Das, D. Cook, A. Bhattacharaya, E. Heierman, and T. Lin. The Role of Prediction Algorithms in the MavHome Smart Home Architecture. *IEEE Wireless Communications*, vol. 9, no. 6, pp. 77-84, December 2002.

[3] Shai Fine, Yoram Singer, and Naftali Tishby. The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning*, 32(1):41– 62, 1998.

[4] J. Han and M. Kamber. *Data Mining*. Morgan Kaufman Publishers, 2001.

[5] E. Heierman and D. Cook. Improving home automation by discovering regularly occurring device usage patterns. In *Proc. 3rd International Conference on Data Mining (ICDM'03)*, pp. 537-540, Melbourne, FL, November 2003.

[6] H. Mannila, H. Toivonen, and A. Verkamo. Discovering frequent episodes in sequences. In *Proc. 1st International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pp. 210-215, Montreal, Canada, August 1995.

[7] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific Publishing Company, 1989.

[8] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the Fifth International Conference on Extending Database Technology* (EDBT-96), pp. 3-17, Avignon, France, 1996.

[9] G. Theocharous, K. Rohanimanesh, and S. Mahadevan. Learning Hierarchical Partially Observable Markov Decision Processes for Robot Navigation, 2001. IEEE Conference on Robotics and Automation.