Pattern Directed Mining for Frequent Episodes *

Valery Guralnik, Duminda Wijesekera, Jaideep Srivastava
{guralnik, wijesek, srivasta}@cs.umn.edu
Department of Computer Science
University of Minnesota
4-192 EECS Bldg., 200 Union St. SE
Minneapolis, MN 55455, USA

Abstract

Sequence data arises naturally in many applications. For example, marketing and sales data collected over a period of time provides sequences which can be analyzed and used for projections and forecasting. Abstractly, such a data collection can be viewed as a sequence of events, where each event has an associated time of occurrence. An important and interesting characteristic of event sequences is the occurrence of episodes, i.e. a collection of events occurring in a certain pattern [7]. Of special interest are frequent episodes, i.e. episodes occurring with a frequency above a certain threshold. In this paper, we study the problem of mining for frequent episodes in sequence data. We present a framework for efficient data mining frequent episodes which goes beyond previous work in a number of ways. First, we present a language for specifying episodes of interest. Second, we describe a novel data structure, called the sequential pattern tree, which captures the relationships specified in the patterns language in a very compact manner. Third, we show how this data structure can be used by a standard bottom-up mining algorithm to generate frequent episodes in an efficient manner. We present the results of a preliminary evaluation of the proposed techniques.

1. Introduction

Sequence data arises naturally in many applications. For example, process monitoring in process control applications generates sequences of sensor readings collected over a period of time. Similarly, marketing and sales data collected over a period of time provides sequences which can be analyzed and used for projections and forecasting. Abstractly, such a data collection can be viewed as a sequence of events, where each event has an associated time of occurrence. An important and interesting characteristic of event sequences is the occurrence of *episodes*, i.e. a collection of events occurring in a certain pattern [7]. Of special interest are *frequent episodes*, i.e. episodes occurring with a frequency above a certain threshold.

In this paper, we study the problem of mining for frequent episodes in sequence data, which was first introduced in Agrawal et al [2]. The authors showed how their association rule algorithm for unordered data [1] could be adapted to mine for frequent episodes in sequence data. The class of episodes being mined was generalized, and performance enhancements were presented in [10]. Mannila et al [7] also addressed this problem, and introduced a pattern language for specifying the episodes of interest, which would be helpful in guiding the mining algorithm. In [10] approach, the algorithm would mine for *all* frequent episodes. Mannila et al applied their techniques to discovering interesting episodes from telecommunication alarm databases [4] and subsequently generalized and optimized their techniques [5]. This team also introduced the idea of using *frequent sets* as an efficient representation for this problem [6]. In the context of temporal databases, this problem has been addressed in [9]. The basic techniques for mining frequent episodes have also been used to discover user browsing patterns from web log data [3, 8].

In this paper we present a framework for efficient data mining for frequent episodes which goes beyond previous work in a number of ways. First, we present a language for specifying episodes of interest which generalizes that of [5]. Second, we describe a novel data structure, called the *sequential pattern tree*, which captures the relationships specified in the pattern language in a very compact manner. Third, we show how this data structure can be used by a standard bottom-up mining algorithm to generate frequent episodes in an efficient manner.

The rest of this paper is organized as follows: Section 2 presents our model and language. Section 3 introduces the sequential pattern tree, and how expressions in the pat-

*This work was supported in part by NSF grant ASC-9554517

Event ID	Date	Company Type	Company Name	Movement	Volatility	Activeness
e_1	01/02/91	Computer	Microsoft Co.	Down	Low	High
e_2	01/03/91	Computer	Microsoft Co.	Up	Medium	High
e_3	01/02/91	Computer	Sun Microsystems Inc.	NoMvmt	High	Low
e_4	01/03/91	Computer	Oracle System Co.	Down	High	High

Table 1. Examples of Events in the Stock Market Domain

tern language are compiled into it. In addition, this section presents the pattern directed data mining algorithm. In Section 4 we present the experimental results, and Section 5 concludes the paper.

2. Sequential Patterns

In this section we introduce the model of knowledge being mined, and pattern language to specify user interest.

2.1. Pattern Language

Definition 1 *Given a set of event attributes* ***A*** *=*$\{A_1, A_2, \ldots, A_m\}$ *with domains* $D_1, D_2, \ldots, D_m$*, an* ***event*** e *over* ***A*** *is a (m+2)-tuple* ($a_1, a_2, \ldots, a_m, t_{begin}, t_{end}$) *where* $a_i \in D_i\ \forall i$. t_{begin} *and* t_{end} *are real numbers representing the begining and ending time of event* e.

In a domain such as the stock market, the appropriate attributes could be company name, company type, stock movement direction, volatility of stock, and activeness of stock. In this domain the beginning and ending times are equivalent and represent the date on which the stock transactions occured. A possible set of events could be the one shown in Table 1.

When mining for patterns in sequence data, usually the order of occurence of events is fully specified. However, in some cases the order in which some of the events occur is immaterial. Therefore, in general the patterns can specify a partial ordering of events instead of the total order.

Definition 2 *An* ***episode*** *is a collection of events that occur within a specified time window, and are consistent with a given partial order [7].*

Definition 3 *The order of occurence of two events* e *and* f *is said to be* ***serial*** *if either 1)* $e.t_{end} < f.t_{begin}$ *or 2)* $f.t_{end} < e.t_{begin}$. $e \rightarrow f$ *indicates that event* e *occured before event* f*, i.e.* $e.t_{end} < f.t_{begin}$

An example of serial occurence of events from Table 1 is $e_1 \rightarrow e_2$, but not $e_2 \rightarrow e_1$.

Definition 4 *The order of occurence of two events* e *and* f *is said to be* ***parallel*** *if no constraints are imposed on the times of occurence of* e *and* f*, indicated by* $e \| f$.

Examples of parrallel occurences of events in Table 1 are $e_1 \| e_2$ as well as $e_2 \| e_1$.

In this paper, the term ***ordering constraint*** refers to serial or parallel order among events. To specify more complex ordering among events, parallel and serial ordering constraints on the events can be combined into an expression. Parentheses can be used to impose precedence of the ordering constraints between events. Thus, to specify that two events occured before a third one, the following pattern is written:

$$(e \parallel f) \ \rightarrow \ g$$

In general, the user may not want to completely specify the event while mining for frequent episodes. However, constraints may be imposed on some of the attributes of events.

Definition 5 *A* ***selection constraint*** *on an event* e *is a unary predicate* $\alpha(e.a_i)$ *on domain* D_i*, where* a_i *is an attribute of* e.

For instance a user might be interested in all events affecting computer companies. This is specified by the following selection constraint:

$$e.type = \text{`}computer\text{`}$$

Definition 6 *A* ***join constraint*** *on events* e *and* f *is a binary predicate* $\beta(e.a_i, f.a_j)$ *on domain* $D_i \times D_j$*, where* a_i *and* a_j *are attributes of* e *and* f*, respectively.*

For example, to specify the join constraint that two events *e* and *f* affect different companies, the following expression should be used:

$$e.name \neq f.name$$

It is assumed that domains D_i and D_j allow comparison operators for equality and inequality. In this paper, the term ***attribute constraint*** refers to a selection constraint or a join constraint. A pattern can be specified by combining predicates into boolean expression, using negation, conjunction and disjunction. This makes our model more flexible than the model presented in [5], which only allows conjunctive boolean conditions.

2.2. Pattern Specification Language

In order to partially specify an event, a boolean expression comprising of selection constraints on the attributes of the event is used. For instance, in the stock market domain to specify the event that computer or electronic company's stocks went down, we write:

$$e\left[(e.type = \text{‘}computer\text{‘} \lor e.type = \text{‘}electronic\text{‘})\land \right.$$

$$\left. e.movement\_direction = \text{‘}down\text{‘}\right]$$

In order to specify that some characteristics of two events are the same (or different) a boolean expression of join constraints on the attributes is used. For example, to specify that the stock of a company went down after it went up, we write:

$$e\left[e.movement\_direction = \text{‘}up\text{‘}\right]$$

$$\rightarrow \left[e.name = f.name\right]$$

$$f\left[f.movement\_direction = \text{‘}down\text{‘}\right]$$

Definition 7 *A* ***sequential pattern*** *is a combination of partially ordered event specifications.*

In general, by combining language constructs introduced above complex sequential patterns of interest can be specified.

2.3. Frequent Episodes

We are often interested in episodes involving events which occur *"close"* to each other, i.e. events occurring within a user defined time window W. In addition, we are interested in episodes that occur with sufficient frequency, since anything below this threshold might be noise. We used definition of the frequency first introduced in [7]. The user should specify the minimum threshold frequency, min_fr, which the system uses to return an episode φ which has frequency $fr \geq min\_fr$.

Our datamining algorithm does not depend on the definition of the frequency. The metric appropriate to the datamining application can be chosen and plugged in the algorithm.

3. Data Structure and Algorithm

To help guide the mining algorithm, user-specified patterns should be translated into a computer-suitable form. For this purpose we introduced an efficient data structure, which we call the *Sequential Pattern Tree (SP Tree)*.

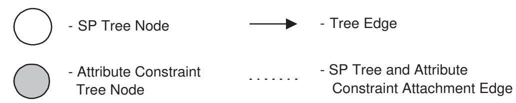

Figure 1. Notation to represent sequential pattern data structure

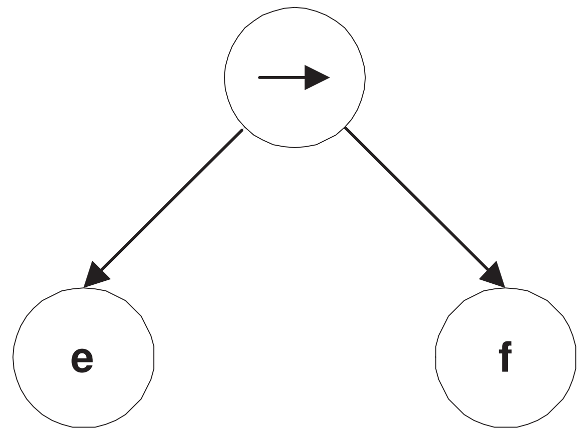

Figure 2. SP Tree for $e \rightarrow f$ user specified pattern.

3.1. Sequence Relationship Specification

Each episode specification can be translated into a tree data structure, where the leaves are unspecified events and intermediate nodes are parallel and serial ordering constraints. This tree data structure, called the *SP Tree*, guides how user-specified pattern assists in discovering episodes that match the pattern. To translate a user-specified pattern into the SP Tree a grammar with embedded semantic actions needs to be created. Once a pattern is specified, a parser can be used to translate the pattern into the SP Tree. For experiments described in this paper the Yacc parser generator was used.

Definition 8 *Given a user-specified sequential pattern, the* ***SP Tree*** *is a tree with the following properties:*

1. *A leaf node represents an event.*
2. *An interior node represents an ordering constraint.*
3. *If $\odot$ is an ordering constraint labeling some interior node, and e and f are the labels of the children of that node from left to right, then $e \odot f$ is a sequential pattern.*

```
Initialize queue Q to empty
for(each leaf l in T) do begin
      Generate events from S that match constraints of l
      if(the parent p of l is not already in Q) then
            put p in Q
end
While (Q is not empty) do begin
      Remove node n from Q
      Generate_Events(n)
      if(for n's parent p another child was processed) then
            put p in Q
end
```

Figure 4. Bottom-up Algorithm to Generate Frequent Episodes

```
for(each episode e from left child l of n) do begin
      for(each episode f from right child r of n) do begin
            if(node n is serial) then
                  if(e.t_end ≥ f.t_begin) then
                        continue
            if(events in e and f match the join constraint) then
                  form new episode g from events from e and f
      end
end
```

Figure 5. Generate-Events Algorithm

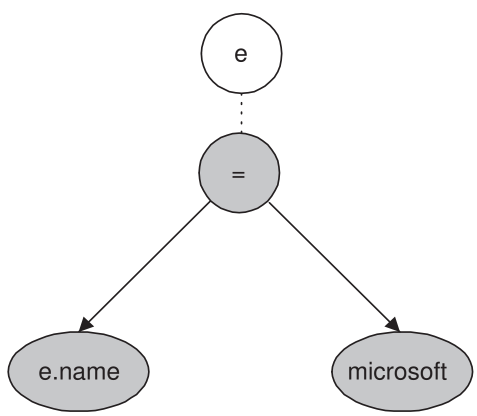

Figure 3. SP Tree for $e\{e.name = `microsoft`\}$ user specified pattern

4. *Each node will hold events matching constraints of that node.*

5. *Attached to each SP Tree node is a boolean expression representing the attribute constraints associated with this node.*

The notation used to represent the SP Tree data structure is shown in Figure 1. If a user specifies an episode $(e \rightarrow f)$, then the tree has an interior node labeled $\rightarrow$, with two children labeled e and f from left to right. This tree is shown in Figure 2. The constraints imposed on attributes are represented as an expression tree, which is an attribute of the node specifying the constraint. For example, to retrieve the event e associated with Microsoft the data structure shown in Figure 3 should be created.

3.2. Mining Algorithm

The basic idea in the data mining algorithm is to construct the frequent episodes in a bottom-up fashion. The algorithm takes the SP tree T and the sequence of events S as inputs. At the leaf level the algorithm matches events against the selection constraints of the event specification, pruning out the ones which do not match the selection constraint. Interior nodes merge events of left and right children according to the ordering and join constraints, pruning out ones which do not match the node specifications. Figures 4 and 5 give the desired algorithm.

The tree structure lends itself easily to parallelization. In general, whenever both node's children have finished their computation, the node can start generating frequent episodes without waiting for other (independent) nodes to complete their computation.

Following are the technical issues that we addressed:

- Our algorithm performs uniformly for conjunctive, disjunctive and negated logical combinations of attribute constraints. Model presented in [5] allows only conjunctive logical combinations.

- In addition, our algorithm performs uniformly for selection and join constraints, while [5] has an efficient algorithm only for selection constraints and requires exhaustive search when join constraints are present.

- Mining for sequential patterns often produces a large number of frequent episodes. Since each node of a SP tree should have a table of events matching the node specification, memory conservation is an issue. Generally, several copies of the same event specification take up more space than one event specification and several pointers to it. Therefore, in our implementation there is only one table of an actual event specification. At the leaf level, the nodes maintain pointers to the event specification matching the selection constraints. Nonleaf nodes for each frequent episode maintain pointers to the left and right children's pointers to the episode specification. This avoids extra copying as well as conserves space.

4. Experimental Evaluation

4.1. Experimental Setup

In order to experimentally evaluate our approach, we used stock data for S&P 500 companies. Our dataset contained 117649 events, which covered 15 months, from January 1991 to May 1992. The raw data included the date of stock transactions, company name, company type, opening stock price, closing stock price, maximum stock price during the day, minimum stock price, and the volume traded. The raw data was converted into a formatted data set, in which each event was described by 5 attributes: *company_name*, *company_type*, *stock_movement*, *stock_activeness* and *stock_volatility*. The volatility of stock is a measure of the price change and can take values HIGH, MEDIUM, and LOW. The activeness of stock is a measure of the quantity of stock traded and can take the values HIGH, MEDIUM, and LOW. In addition, each record contains the date on which the stock transaction took place. Mining for sequential patterns in this dataset may guide financial analysts in formulating strategies for trading stock.

4.2. Results

We performed several experiments to study how our method performed for different values of window size, number of attribute constraints, number of event specifications and data set size. The time to produce frequent episodes was used as a measure of the performance.

Figure 6 shows the change in the performance of the method as the size of the window changes. As shown, the time spent generating frequent episodes grows almost linearly with the size of the window.

Figure 7 plots how the algorithm behaves as the number of attribute constraints in the specified sequential pattern changes. In each successful pattern, an additional restrictive attribute constraint was added. The graph in Figure 7 shows that as we add more attribute constraints, the performance of the method dramatically improves.

Figure 8 shows how the algorithm behaves as the number of event specifications in the sequential pattern increases. As the number of event specifications increases, the time spent producing frequent episodes increases linearly. The time to produce frequent episodes consisting of 6 events is about 3 minutes, which shows that the algorithm has a relatively good performance.

In order to evaluate how the algorithm behaves with an increase in data set size, we used subsets of the original data set. Each successive subset had linearly increasing number of records. The results of the experiment are depicted in Figure 9 and show that time increases linearly with the data set size.

5. Conclusions

In this paper, we presented an approach for mining frequent episodes from sequence data. This includes a data mining language, an efficient data structure and a bottom-up algorithm to efficiently generate frequent episodes from a large collection of data.

The presented approach to pattern mining for frequent episodes is flexible, robust and efficient. The approach allows a user to specify complex patterns by combining parallel and serial ordering timing constraints as well as event attribute constraints. In addition, the proposed method scales well for different values of window size, complexity of the sequential pattern, and size of the data set. Also, the algorithm gives relatively good performance.

Mining for episodes is a computationally intensive task. However, given its data-parallel nature it is also highly parallelizable. In the future, we plan to investigate the parallelization of the proposed algorithm.

Acknowledgments

We would like to thank Robert Cooley for his helpful comments and suggestions to improve the presentation.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th VLDB Conference*, pages 487–499, Santiago, Chile, 1994.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of the 11th Int'l Conference on Data Engineering*, 1995.

The numbers on the graphs represent the number of frequent episodes matching the pattern.

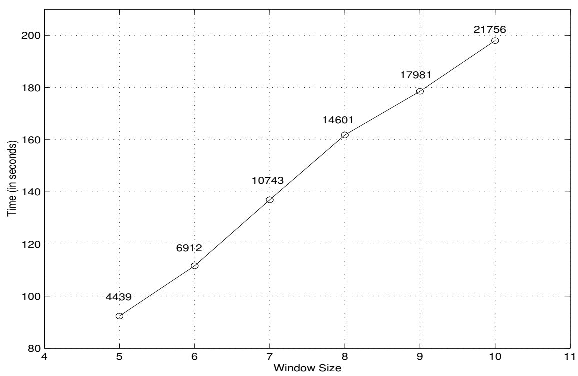

Figure 6. Minimum Frequency = 0.8.

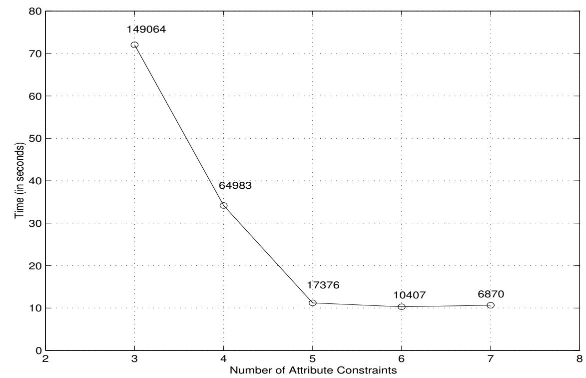

Figure 7. Minimum Frequency = 0.8. Window Size = 5.

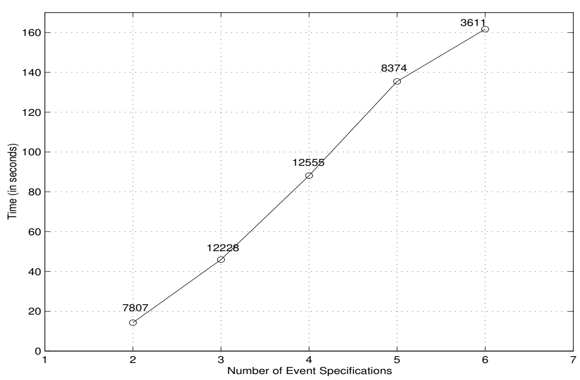

Figure 8. Minimum Frequency = 0.5. Window Size = 11.

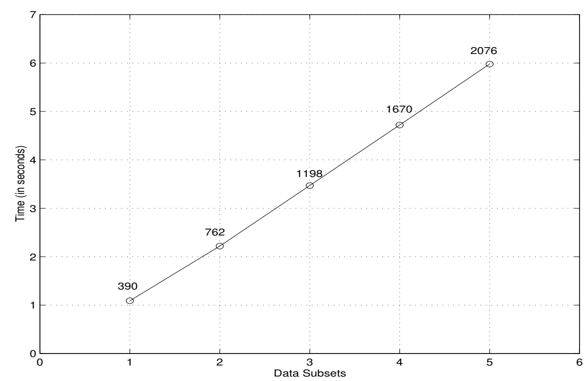

Figure 9. Minimum Frequency = 0.7. Window Size = 5.

[3] M. Chen, J. Park, and P. Yu. Data mining for path traversal patterns in a web environment. In *Proceedings of the 16th International Conference on Distributed Computing Systems*, pages 385–392, 1996.

[4] K. Hatonen, M. Klemettineen, H. Mannila, P. Ronkainen, and H. Toivonen. Knowledge discovery from telecommunication network alarm databases. In *Proc. of the 12th Int'l Conf. on Data Eng.*, pages 115–122, Kyoto, Japan, 1996.

[5] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proc. of 2nd Int'l Conference on Knowledge Discovery and Data Mining*, pages 146–151, Portland, Oregon, 1996.

[6] H. Mannila and H. Toivonen. Multiple uses of frequent sets and condensed representations. In *Proc. of 2nd Int'l Conference on Knowledge Discover and Data Mining*, pages 189–194, 1996.

[7] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *Proc. of the First Int'l Conference on Knowledge Discovery and Data Mining*, pages 210–215, Montreal, Quebec, 1995.

[8] B. Mobasher, N. Jain, E. Han, and J. Srivastava. Web mining: Pattern discovery from world wide web transactions. Technical Report TR 96-050, University of Minnesota, Dept. of Computer Science, Minneapolis, 1996.

[9] B. Padmanabham and A. Tuzhilin. Pattern discovery in temporal databases: A temporal logic approach. In *Proc. of 2nd Int'l Conference on Knowledge Discovery and Data Mining*, pages 351–354, 1996.

[10] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. of the 21th VLDB Conference*, pages 407–419, Zurich, Switzerland, 1995.