Pattern Directed Mining of Sequence Data

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

Abstract

Sequence data arise naturally in many applications, and can be viewed as an ordering of events, where each event has an associated time of occurrence. An important characteristic of event sequences is the occurrence of *episodes*, i.e. a collection of events occurring in a certain pattern. 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 mining of 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* (SP 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. Finally, we show how the SP Tree can be optimized by sharing common conditions, and evaluating each such expression only once. We present the results of an evaluation of the proposed techniques.

Introduction

Sequence data arise 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 (Mannila, Toivonen, & Verkamo 1995). 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 (Agrawal & Srikant 1995). The authors showed how their association rule algorithm for unordered data (Agrawal & Srikant 1994) 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 (Srikant & Agrawal 1995).

Mannila et al (Mannila, Toivonen, & Verkamo 1995) 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 (Srikant & Agrawal 1995) approach, the algorithm would mine for *all* frequent episodes. Mannila et al subsequently generalized and optimized their techniques (Mannila & Toivonen 1996a). This team also introduced the idea of using *frequent sets* as an efficient representation for this problem (Mannila & Toivonen 1996b).

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 (Mannila & Toivonen 1996a). Second, we describe a novel data structure, called the *sequential pattern tree* (SP 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, and present its experimental results. Next, we show how the SP Tree can be optimized by sharing common conditions, and evaluating each such expression exactly once. We describe the experimental results of this optimization. Finally, we present the results of an initial study comparing our approach with Apriori-based approach.

Sequential Patterns

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

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*

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 Co.	Down	High	High

Table 1: Examples of Events in the Stock Market Domain

e *over* $\boldsymbol{A}$ *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, attributes could be company name, company type, stock movement direction, volatility of stock, and activeness of stock. In the example in Table 1 the beginning and ending times are equivalent and represent the date on which the stock transactions occured.

When mining for patterns in sequence data, usually the temporal order 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 order of events instead of total order. The mining algorithm generates episodes consisting of events that match the user-specified pattern.

Definition 2 *An* ***episode*** *is a collection of events which are consistent with a given partial order (Mannila, Toivonen, & Verkamo 1995).*

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 parallel 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:

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

In general, the user may not want to completely specify an event, while mining for patterns in a sequence, that is to specify the values of every attribute of the event. However, constraints may be imposed on some of the attributes.

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. An event can be specified by combining predicates into boolean expression, using negation, conjunction and disjunction. This makes our language more flexible than the language presented in (Mannila & Toivonen 1996a), which only allows conjunctive boolean conditions.

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:

$$e[(e.type = \text{'}computer\text{'} \vee e.type = \text{'}electronic\text{'}) \wedge e.movement\_direction = \text{'}down\text{'}]$$

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:

$$e[e.movement\_direction = \text{'}up\text{'}] \rightarrow [e.name = f.name] \; f[f.movement\_direction = \text{'}down\text{'}]$$

In general, by combining ordering and attribute constraints a complex sequential patterns of interest can be specified.

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

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,

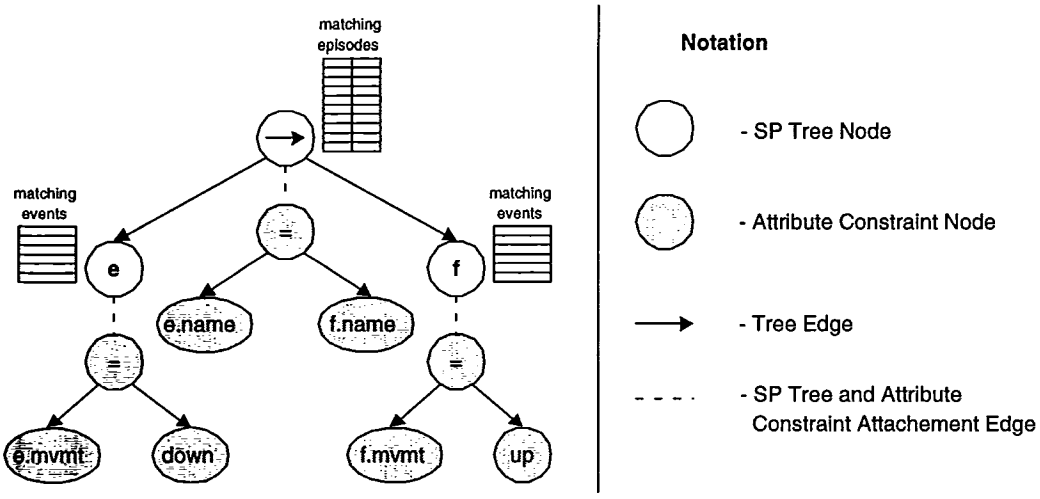

Figure 1: SP Tree for $e[e.mvmt = \text{'}up\text{'}] \rightarrow [e.name = f.name]f[f.mvmt = \text{'}down\text{'}]$ user specified pattern

we are interested in episodes that occur with sufficient frequency, since anything below this threshold is uninteresting. For experiments described in this paper, we used the following definition of frequency, first introduced in (Mannila, Toivonen, & Verkamo 1995):

Definition 8 *Given a sequence of events S, if an episode φ occurs in S we write $S \models \varphi$. The **frequency** of φ in the set $aw(S,w)$ over all windows of size w in S, is*

$$fr(\varphi, S, w) = \frac{|\{W \in aw(S,w) | W \models \varphi\}|}{|aw(S,w)|}$$

Our datamining algorithm does not depend on the definition of frequency. The metric should be based on the user or application needs. In general, there should be a library of metrics. Then, the application appropriate metric can be chosen by the user and plugged in the algorithm. Computation time of the algorithm will depend on the cost of computing the metric. However, usually data preprocessing can be done to make the calculations faster.

Problem Statement

Given a sequential pattern P, an input sequence of events S, a window width W, and a frequency threshold min_fr, the data mining algorithm has to find all episodes φ that match the user-specified pattern P within a given time window W and have frequency $fr(\varphi, S, W) \geq min\_fr$.

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 introduce an efficient data structure, called the *Sequential Pattern Tree (SP Tree).*

Sequence Relationship Specification

Each sequential pattern 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 *SP Tree*, improves the efficiency of process of discovering frequent episodes.

Definition 9 *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.*
4. *Associated with each node is a table of events matching the constraints of the node.*
5. *Attached to each node is a boolean expression tree representing the attribute constraints associated with this node.*

An example of SP Tree is shown in Figure 1.

Mining Algorithm

The basic idea in the data mining algorithm is to construct 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 constraint of the event specification, pruning out the ones which do not fit 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 fit the node specifications. Figures 2 and 3 give the desired algorithm. To limit the search space we used the sliding window technique(Mannila, Toivonen, & Verkamo 1995).

```
1.  Initialize queue Q to empty
2.  for(each leaf l in T) do begin
3.      Generate events from S that match constraints of l
4.      if(the parent p of l is not already in Q) then
5.          put p in Q
6.  end
7.  While (Q is not empty) do begin
8.      Remove node n from Q
9.      Generate_Events(n)
10.     if(for n's parent p another child was processed) then
11.         put p in Q
12. end
```

Figure 2: Bottom-up Algorithm to Generate Frequent Episodes

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

Figure 3: Generate-Events Algorithm

Following are the technical issues that we addressed:

- Our algorithm performs uniformly for disjunctive, conjunctive and negated logical combinations of attribute constraints.
- 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. Non-leaf 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.

Experimental Evaluation

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.

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 4 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. In this experiment the number of episodes grows linearly with the window size. Therefore, the increase in time to produce the episodes can be attributed to allowing more events to participate in the episodes generation.

Figure 5 plots how the algorithm behaves as the number of attribute constraints in the specified sequential pattern changes. In each successful pattern used in the experiment, an additional restrictive attribute constraint was added. The graph in Figure 5 shows that as we add more attribute constraints, the performance of the method dramatically improves. In each successive run, the child node generates less events, and as a result

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

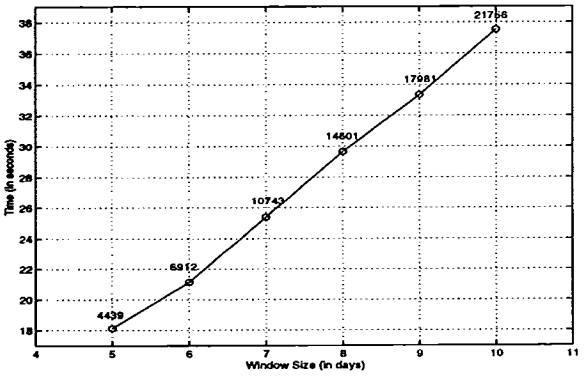

Figure 4: Minimum Frequency = 0.8.

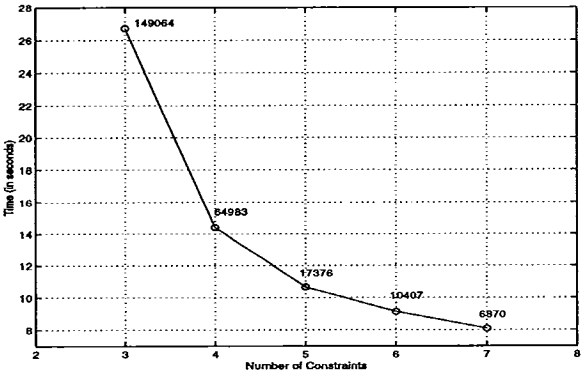

Figure 5: Minimum Frequency = 0.8. Window Size = 5.

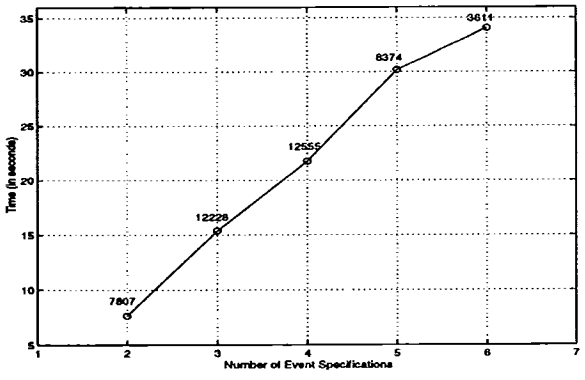

Figure 6: Minimum Frequency = 0.5. Window Size = 11.

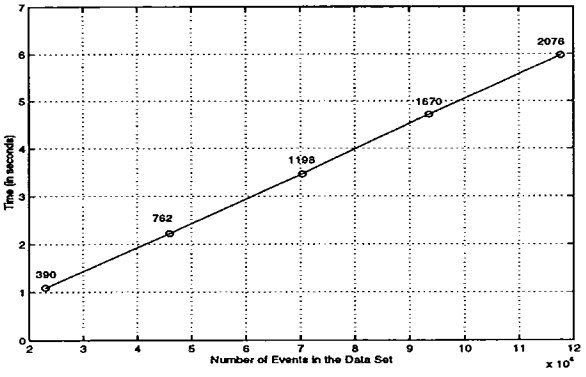

Figure 7: Minimum Frequency = 0.7. Window Size = 5.

the parent node has to make less iterations through the data.

Figure 6 shows how the algorithm behaves as the number of event specifications in the sequential pattern increases. As the number of event specifications in sequential pattern increases, the time spent producing frequent episodes increases linearly. In this experiment, initially the number of frequent episodes increases with an increase in the number of event specifications, because each successive pattern allows more combination of events matching that pattern. Then, the number of episodes decreases with an increase in the number of event specifications in the pattern, because less patterns occur within a specified time window. Therefore the increase in time to generate episodes can be attributed to the time spent on each additional event specification and order constraint.

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 7 and show that time increases linearly with the data set size. Here the number of episodes also increases linearly with the data set size.

SP Tree Optimization

Sharing of Nodes

The efficiency of our initial algorithm has been further improved by optimizing the SP Tree structure. Thus, if two event (leaf) nodes represent the same event, i.e. the attribute constraints are the same, then only one of the nodes should participate in producing appropriate events. Furthermore, if two ordering (interior) nodes have the same join constraints, and they both have right children representing the same events, and have left children representing the same events, then only one of the nodes should participate in producing appropriate events. In the optimized data structure some of the nodes will share events matching the node constraints. The optimized SP Tree allows speed up of the frequent episode generation, as well as conserves memory. A technique similar to the value-number method (Aho, Sethi, & Ulman 1986) was used to generate the optimized SP Tree. Since this problem is essentially one of common subexpression analy-

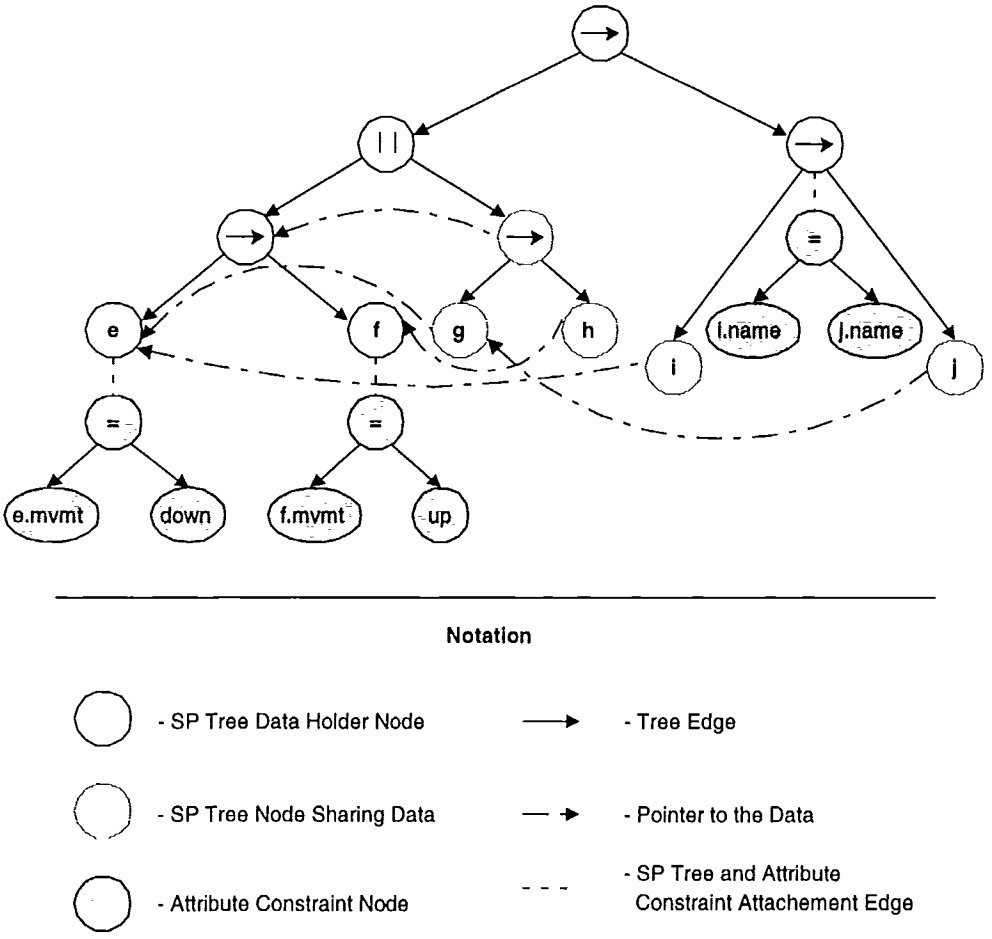

Figure 8: Optimized SP Tree representation of $((e[e.mvmt = \text{'}up\text{'}] \rightarrow f[f.mvmt = \text{'}down\text{'}])||(g[g.mvmt = \text{'}up\text{'}] \rightarrow h[h.mvmt = \text{'}down\text{'}])) \rightarrow (i[i.mvmt = \text{'}up\text{'}] \rightarrow [i.name = j.name]j[j.mvmt = \text{'}down\text{'}])$

sis, we are currently examining the techniques proposed in the compiler (Aho, Sethi, & Ulman 1986; Fischer & LeBlanc 1991) and database (Grant & Minker 1982; Finkelstein 1982; Sellis 1988) literature.

An example of an optimized SP tree is shown in Figure 8.

Performance Evaluation

In order to demonstrate the benefits of the optimized SP Tree we ran experiments, where we varied the degree of shareability. The degree of shareability is defined as follows:

$$1 - \frac{number\ of\ data\ holder\ nodes}{total\ number\ of\ nodes}$$

The results of our experiment are shown in Figure 9. As sharing increases, the performance of the algorithm improves linearly. This is because, each node in SP Tree produces subepisodes in linear time.

Comparison with Apriori-based approach

The presented approach scales well. However, we wanted to see if our approach is as efficient as other approaches that are based on Apriori ideas. For this purpose we chose the approach described in (Mannila & Toivonen 1996a) as a point of comparison. Their implementation was not available, so we implemented their algorithm based on the description in (Mannila & Toivonen 1996a). For these tests frequency threshold was not calculated. In these experiments we only used patterns that can be expressed in the language described in (Mannila & Toivonen 1996a).

With episodes consisting of one or two events, both algorithms make the same number of iterations. For episodes consisting of three or more events, our algorithm makes fewer iterations. Nevertheless, our algorithm performs slower than Apriori-based algorithm for episodes with a small number of events. However, the gap in execution times reduces as the number of events in episodes increases. This gap reduces even faster when

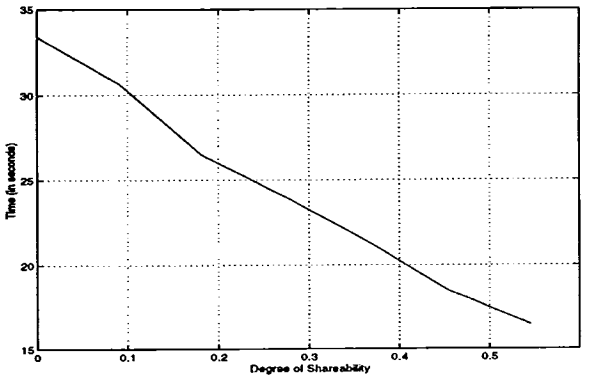

Figure 9: Minimum Frequency = 0.7. Window Size = 11

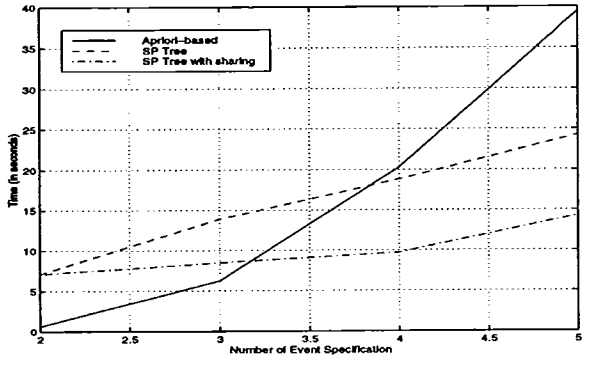

Figure 10: Minimum Frequency = 0.5, Window Size = 11

sharing is used. Results of a preliminary comparison are shown in Figure 10.

The additional expressive power that our language provides is offset by the rise in computational difficulty. This is why our algorithm has worse performance for episodes with a very small number of events.

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 collection of data.

The presented approach to pattern mining for frequent episodes is flexible, robust and efficient. The approach allows a user to specify a complex pattern 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. Moreover, the algorithm can be optimized for the sequential patterns with shared subpatterns.

In our ongoing research we are considering different SP Tree optimization strategies (such as shared subconditions sharing) to represent sequential patterns and investigating the impact of those representations on the performance of our algorithm. Furthermore, the tree structure lends itself easily to parallelization. We plan to investigate the parallelization of the proposed algorithm.

Acknowledgements

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

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

References

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

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

Aho, A.; Sethi, R.; and Ulman, J. 1986. *Compilers: Principles, Techniques, and Tools.* Reading, Massachusetts: Addison–Wesley.

Finkelstein, S. 1982. Common subexpression analysis in database application. *ACM SIGMOD.*

Fischer, C., and LeBlanc, R. 1991. *Crafting a Compiler with C.* The Benjamin/Cummings Publishing Company, Inc.

Grant, J., and Minker, J. 1982. On optimizing the evaluation of a set of expressions. *Int'l Journal of Computer and Information Science.*

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

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

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

Sellis, T. 1988. Multiple query optimization. *ACM TODS* 13(1):23–52.

Srikant, R., and Agrawal, R. 1995. Mining generalized association rules. In *Proc. of the 21th VLDB Conference*, 407–419.

An Efficient Approach to Clustering in Large Multimedia Databases with Noise

Alexander Hinneburg, Daniel A. Keim

Institute of Computer Science, University of Halle, Germany
{hinneburg, keim}@informatik.uni-halle.de

Abstract

Several clustering algorithms can be applied to clustering in large multimedia databases. The effectiveness and efficiency of the existing algorithms, however, is somewhat limited, since clustering in multimedia databases requires clustering high-dimensional feature vectors and since multimedia databases often contain large amounts of noise. In this paper, we therefore introduce a new algorithm to clustering in large multimedia databases called DENCLUE (DENsity-based CLUstEring). The basic idea of our new approach is to model the overall point density analytically as the sum of influence functions of the data points. Clusters can then be identified by determining density-attractors and clusters of arbitrary shape can be easily described by a simple equation based on the overall density function. The advantages of our new approach are (1) it has a firm mathematical basis, (2) it has good clustering properties in data sets with large amounts of noise, (3) it allows a compact mathematical description of arbitrarily shaped clusters in high-dimensional data sets and (4) it is significantly faster than existing algorithms. To demonstrate the effectiveness and efficiency of DENCLUE, we perform a series of experiments on a number of different data sets from CAD and molecular biology. A comparison with DBSCAN shows the superiority of our new approach.

Keywords: Clustering Algorithms, Density-based Clustering, Clustering of High-dimensional Data, Clustering in Multimedia Databases, Clustering in the Presence of Noise

1 Introduction

Because of the fast technological progress, the amount of data which is stored in databases increases very fast. The types of data which are stored in the computer become increasingly complex. In addition to numerical data, complex 2D and 3D multimedia data such as image, CAD, geographic, and molecular biology data are stored in databases. For an efficient retrieval, the complex data is usually transformed into high-dimensional feature vectors. Examples of feature vectors are color histograms [SH94], shape descriptors [Jag91, MG95], Fourier vectors [WW80], text descriptors [Kuk92], etc. In many of the mentioned applications, the databases are very large and consist of millions of data objects with several tens to a few hundreds of dimensions.

Automated knowledge discovery in large multimedia databases is an increasingly important research issue. Clustering and trend detection in such databases, however, is difficult since the databases often contain large amounts of noise and sometimes only a small portion of the large databases accounts for the clustering. In addition, most of the known algorithms do not work efficiently on high-dimensional data.The methods which are applicable to databases of high-dimensional feature vectors are the methods which are known from the area of spatial data mining. The most prominent representatives are partitioning algorithms such as CLARANS [NH94], hierarchical clustering algorithms,and locality-based clustering algorithms such as (G)DBSCAN [EKSX96, EKSX97] and DBCLASD [XEKS98]. The basic idea of *partitioning algorithms* is to partition the database into k clusters which are represented by the gravity of the cluster (k-means) or by one representative object of the cluster (k-medoid). Each object is assigned to the closest cluster. A well-known partitioning algorithm is CLARANS which uses a randomized and bounded search strategy to improve the performance. *Hierarchical clustering algorithms* decompose the database into several levels of partitionings which are usually represented by a dendrogram - a tree which splits the database recursively into smaller subsets. The dendrogram can be created top-down (divisive) or bottom-up (agglomerative). Although hierarchical clustering algorithms can be very effective in knowledge discovery, the costs of creating the dendrograms is prohibitively expensive for large data sets since the algorithms are usually at least quatratic in the number of data objects. More efficient are locality-based clustering algorithms since they usually group neighboring data elements into clusters based on local conditions and therefore allow the clustering to be performed in one scan of the database. DBSCAN, for example, uses a density-based notion of clusters and allows the discovery of arbitrarily shaped clusters. The basic idea is that for each point of a cluster the density of data points in the neighborhood has to exceed some threshold. DBCLASD also works locality-based but in contrast to DBSCAN assumes that the points inside of the clusters are randomly distributed, allowing DBCLASD to work without any input parameters. A performance comparison [XEKS98] shows that DBSCAN is slightly faster than DBCLASD and both, DBSCAN and DBCLASD are much faster than hierarchical clustering algorithms and partitioning algorithms such as CLARANS. To improve the efficiency, optimized clustering techniques have been proposed. Examples include R*-Tree-based Sampling [EKX95], Gridfile-based clustering [Sch96], BIRCH [ZRL96] which is based on the Cluster-Feature-tree, and STING which uses a quadtree-like structure containing additional statistical information [WYM97].

A problem of the existing approaches in the context of clustering multimedia data is that most algorithms are not designed for clustering high-dimensional feature vectors and therefore, the performance of ex-