CONSTRAINT-BASED MINING OF FREQUENT ARRANGEMENTS OF TEMPORAL INTERVALS

PANAGIOTIS PAPAPETROU

Thesis submitted in partial fulfillment

of the requirements for the degree of

Master of Arts

BOSTON

UNIVERSITY

BOSTON UNIVERSITY

GRADUATE SCHOOL OF ARTS AND SCIENCES

Thesis

CONSTRAINT-BASED MINING OF FREQUENT ARRANGEMENTS OF TEMPORAL INTERVALS

by

PANAGIOTIS PAPAPETROU

B.Sc., University of Ioannina, Greece, 2003

Submitted in partial fulfillment of the

requirements for the degree of

Master of Arts

2007

Approved by

First Reader

_____

George Kollios, PhD
Assistant Professor of Computer Science

Second Reader

_____

Stan Sclaroff, PhD
Associate Professor of Computer Science

Not everything that can be counted counts;
and not everything that counts can be counted;
Albert Einstein (1879-1955)

CONSTRAINT-BASED MINING OF FREQUENT ARRANGEMENTS OF TEMPORAL INTERVALS

PANAGIOTIS PAPAPETROU

ABSTRACT

The problem of discovering frequent arrangements of temporal intervals is studied. It is assumed that the database consists of sequences of events, where an event occurs during a time-interval. The goal is to mine temporal arrangements of event intervals that appear frequently in the database. The motivation of this work is the observation that in practice most events are not instantaneous but occur over a period of time and different events may occur concurrently. Thus, there are many practical applications that require mining such temporal correlations between intervals including the linguistic analysis of annotated data from American Sign Language as well as network and biological data. Two efficient methods to find frequent arrangements of temporal intervals are described; the first one is tree-based and uses depth first search to mine the set of frequent arrangements, whereas the second one is prefix-based. The above methods apply efficient pruning techniques that include a set of constraints consisting of regular expressions and gap constraints that add user-controlled focus into the mining process. Moreover, based on the extracted patterns a standard method for mining association rules is employed that applies different interestingness measures to evaluate the significance of the discovered patterns and rules. The performance of the proposed algorithms is evaluated and compared with other approaches on real (American Sign Language annotations and network data) and large synthetic datasets.

Contents

List of Figures

List of Abbreviations

ASL   . . . .   American Sign Language

DNA   . . . .   DeoxyriboNucleic Acid

ISIdList   . . . .   Interval Sequence Id List

BFS   . . . .   Breadth First Search

DFS   . . . .   Depth First Search

ODFlow   . . . .   Origin Destination Flow

IP   . . . .   Internet Protocol

Chapter 1

Introduction

Sequential pattern mining has received particular attention in the last decade (Agrawal and Srikant, 1994; Agrawal and Srikant, 1995; Ayres et al., 2002; Bayardo, 1998; Zaki, 2001; Fei et al., 2001; Pei et al., 2002; Han et al., 2000a; Seno and Karypis, 2002; Yan et al., 2003; Leleu et al., 2003; Han et al., 2000b; Wang and Han, 2004). The objective is to extract patterns from a set of sequences of instantaneous events which satisfy some user-specified constraints. These constraints can vary from just a support threshold, that defines frequency, to a set of gap, window (Zaki, 2000; Srikant and Agrawal, 1996), or regular expression constraints (Garofalakis et al., 1999), that push more user-controlled focus into the mining process. Despite advances in this area, nearly all proposed algorithms concentrate on the case where events occur at single time instants. However, in many applications events are not instantaneous; they instead occur over a time interval. Furthermore, since different temporal events may occur concurrently, it would be useful to extract frequent temporal patterns of these events. In this paper the goal is to develop methods that discover temporal arrangements of correlated event intervals which occur frequently in a database.

There are many applications that require mining such temporal relations. One potential application is for analysis of the multiple gestures that occur, in parallel, on the hands and on the face and upper body, to express linguistic information. In signed languages, lexical information is expressed primarily through movements of the hands and arms, whereas critical grammatical information is expressed non-manually, through such

> Who drove the car, who?

(Wh-Word)   **(Wh-Word)**
(Lowered eyebrows)   **(Lowered eyebrows)**
(Rapid head shake)   **(Rapid head shake)**
(Wh-Question)

time
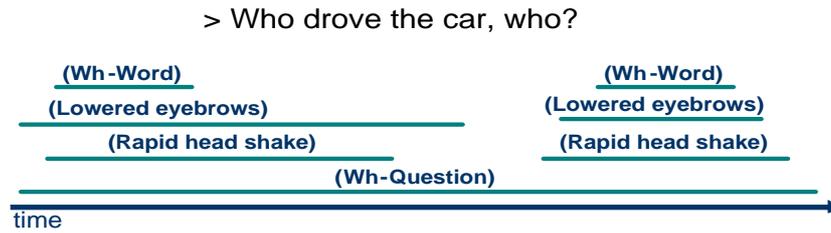
Figure 1·1: *An ASL example.*

behaviors as raised or lowered eyebrows, modifications in eye aperture or gaze, repeated head gestures (nods, shakes) or head tilt, as well as expressions of the nose or mouth. For example, the canonical marking of a wh-question (a question containing a word such as 'who', 'what', 'when', 'where', or 'why') includes lowered brows slightly squinted eyes occurring over a predictable domain (either the question sign or the whole clause constituting the question), and there is frequently a slight rapid head-shake co-occurring with the wh-phrase (Neidle and Lee, 2006). Although much is known about the linguistic significance of certain non-manual markings carrying critical syntactic information, there are others whose functions remain to be studied and more fully understood. Pattern detection could ultimately contribute to discovery of the significance of some of these non-manual behaviors. The annotated ASL corpus used for this research was produced by linguists as part of the American Sign Language Linguistic Research Project ((Neidle et al., 2000; Neidle, 2003; Neidle and Lee, 2006)) using SignStream(TM) ((Neidle et al., 2001; Neidle, 2002a)). The annotations identify start and end times for: the manual ASL signs (represented by English-based glosses), part of speech for those signs, plus grammatical interpretive labels indicating clusters of non-manual expressions that serve to mark particular syntactic functions (such as wh-questions, negation, etc.) as well as the gestures themselves (e.g., raised eyebrows, wrinkled nose, rapid head shake). See ((Neidle, 2002b)) for further information about the annotation conventions that were used.

Another application is in network monitoring, where the goal is to analyze packet and router logs. Consider Figure 1·2 for example, which shows two groups of machines

communicating with each other via two routers. In this case an event label is the source and destination IP and the event interval corresponds to the duration of the communication between the two machines. Multiple types of events occurring over certain time periods can be stored in a log, and the goal is to detect general temporal relations of these events that with high probability would describe regular patterns in the network, that could be used for prediction and intrusion detection.

Moreover, interval-based events can be identified in the human gene. More specifically, DNA is a sequence of items (nucleotides) defined over a four-letter alphabet, i.e. $\Sigma = \{A,\ C,\ G,\ T\}$. Regions of high occurrence of a nucleotide or combination of nucleotides, known as *poly-regions*, can be defined over DNA. The detection of frequently overlapping poly-regions could lead the biologists to a variety of useful observations concerning the evolution of different genes and their contribution to protein construction. To the best of our knowledge, the first general approach to mine frequent arrangements of poly-regions in DNA is introduced in (Papapetrou et al., 2006).

Most existing sequential pattern mining methods are hampered by the fact that they can only handle instantaneous events, not event intervals. Nonetheless, such algorithms could be retrofitted for the purpose, via converting a database of event intervals to a transactional database, by considering only the start and end points of every event interval. An existing sequential pattern mining algorithm could be applied to the converted database, and the extracted patterns could be post-processed to produce the desired set of frequent arrangements. However, an arrangement of k intervals corresponds to a sequence of length $2k$. Hence, this approach will produce up to 2^{2k} different sequential patterns. Moreover, post-processing will also be costly, since the extracted patterns consist of event start and end points, and for each event interval all the relations with the other event intervals must be determined. Therefore, it is essential to develop interval-based algorithms that can efficiently mine frequent patterns and rules from interval-based data.

The main contributions in this paper include:

- a robust definition of temporal relations between two event intervals that is noise
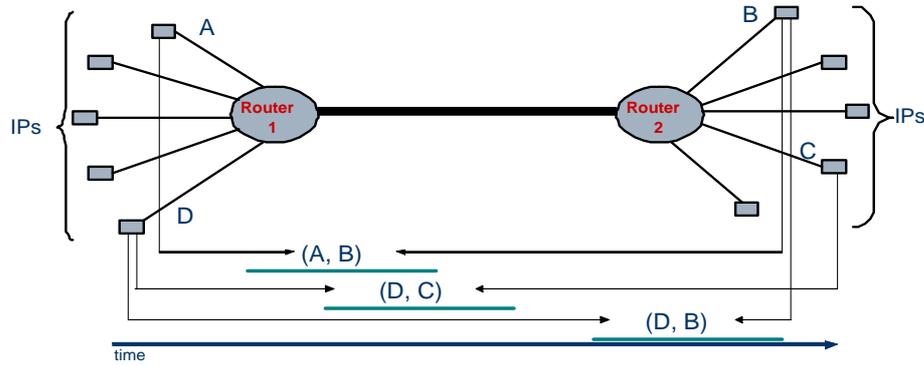
Figure 1·2: *A network example.*

tolerant and through the use of constraints eliminates the ambiguity of Allen's definitions (Allen and Ferguson, 1994),

- a formal definition for the problems of mining frequent temporal arrangements and arrangement rules of event intervals in an event interval database using temporal and structural constraints,

- a prefix-based approach and an efficient algorithm for mining frequent arrangements of temporally correlated events using depth first search in an enumeration tree of temporal arrangements,

- a further improvement of the mining process with the incorporation of temporal and structural constraints,

- an efficient algorithm for mining arrangement rules from the extracted patterns based on user-specified interestingness measures and constraints, and

- an extensive experimental evaluation of these techniques and a comparison with a standard sequential pattern mining method, SPAM (Ayres et al., 2002), using both real and synthetic data sets,

The remainder of this thesis is organized as follows: Chapter 3 presents the related work on sequential pattern mining, interval mining and temporal mining. Also it gives a brief

overview of the existing interestingness measures for association rules. Chapter 2 provides the problem formulation along with the appropriate definitions and background. Chapter 4 gives an extensive description of two tree-based approaches and a prefix-based approach for mining frequent arrangements of temporally correlated event intervals. Also, in the same Chapter, an efficient algorithm for extracting the set of top k arrangement rules is described. Chapter 5 describes the experimental evaluation, and Chapter 6 concludes the thesis, providing directions for future research.

Chapter 2

Background

Some basic definitions on temporal logic are presented, followed by a sufficient background on interestingness measures for association rules. Finally, the problems of constraint-based mining of frequent arrangements of temporal intervals and constraint-based mining of the top K interesting association rules from a database of interval-based events are formulated.

2.1 Event Interval Temporal Relations

Seven types of temporal relations between two event intervals are considered. Using these relations, general arrangements can be defined. However, the methods presented in this thesis are not limited to these relations and can be easily extended to include more types of temporal relations, as the ones described by Allen in (Allen and Ferguson, 1994) and also later on in (Freksa, 1992).

Consider two event-intervals A and B. Furthermore, assume that the user specifies a threshold ϵ used to define more flexible matchings between two time instants. The following relations are studied (see also Figure 2·1):

- **Meet**(A, B)**:** In this case, B follows A, with B starting at the time A terminates, i.e. $t_e(A) = t_s(B) \pm \epsilon$. This case is denoted as $A \sim B$ and we say that A *meets* B.

- **Match**(A, B)**:** In this case, A and B are parallel, beginning and ending at the same time, i.e. $t_s(A) = t_s(B) \pm \epsilon$ and $t_e(A) = t_e(B) \pm \epsilon$. This case is denoted as $A||B$ and we say that A *matches* B.
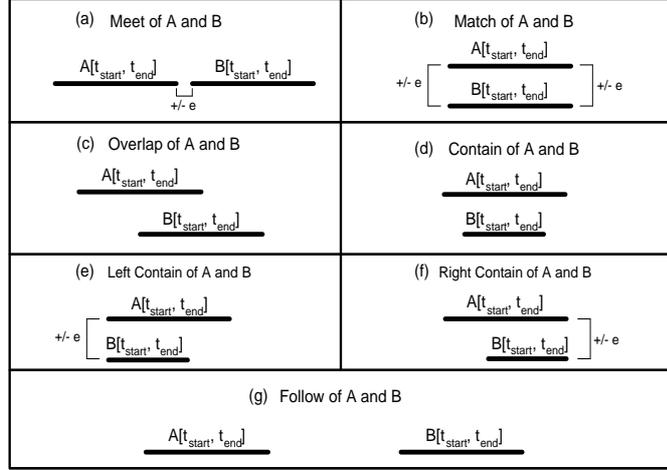
Figure 2·1: *Basic relations between two event-intervals: (a) Meet, (b) Match, (c) Overlap, (d) Contain, (e) Left-Contain, (f) Right-Contain, (g) Follow.*

- **Overlap**(A, B)**:** In this case, the start time of B occurs after the start time of A, and A terminates before B, i.e. $t_s(A) < t_s(B)$, $t_e(A) < t_e(B)$ and $t_s(B) < t_e(A)$. This case is denoted as $A|B$ and we say that A *overlaps* B.

- **Contain**(A, B)**:** In this case, the start time of B follows the start time of A and the termination of A occurs after the termination of B, i.e. $t_s(A) < t_s(B)$ *and* $t_e(A) > t_e(B)$. This case is denoted as $A > B$ and we say that A *contains* B.

- **Left-Contain**(A, B)**:** In this case, A and B start at the same time and A terminates after B, i.e. $t_s(A) = t_s(B) \pm \epsilon$ *and* $t_e(A) > t_e(B)$. This case is denoted as $A \mid > B$ and we say that A *left-contains* B.

- **Right-Contain**(A, B)**:** In this case, A and B end at the same time and the start time of A precedes that of B, i.e. $t_s(A) < t_s(B)$ *and* $t_e(A) = t_e(B) \pm \epsilon$. This case is denoted as $A > \mid B$ and we say that A *right-contains* B.

- **Follow**(A, B)**:** In this case, B occurs after A terminates, i.e. $t_e(A) < t_s(B)$. This case is denoted as $A \to B$ and we say that B *follows* A.
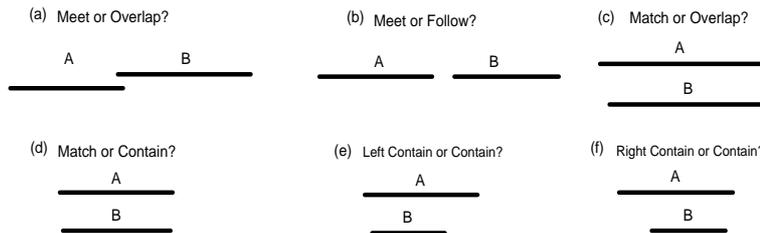
Figure 2·2: *Lack of Robustness in Allen's Relations.*

2.2 Robustness and Ambiguity Issues

Most existing interval-based mining algorithms use Allen's approach (Allen and Ferguson, 1994) to describe relations between event intervals. Because of the limit in the accuracy of demarcating the temporary boundaries of events, there can be variability in these boundaries. Unfortunately, Allen's relations are hampered by the fact that they cannot capture this variability, thus they are not robust and can be ambiguous. This issue has also been noted in (Moerchen, 2006) and is illustrated in Figure 2·2. Consider, for instance, the case where the actual relation between two event intervals is *meet*, but due to noise it appears as *overlap* as shown in Figures 2·2(a) and 2·2(b). Similarly, an actual *match* could appear as *overlap* (Figure 2·2(c)) or *contain* (Figure 2·2(d)), and also, a *left* or *right-contain* could show up as *contain* (Figures 2·2(e), 2·2(f)). Such errors can occur due to noisy data and may have a negative influence on the extracted patterns.

The aforementioned deficiency in Allen's relations is eliminated in our definitions by the use of an ϵ threshold that makes Allen's definitions more robust and noise tolerant. Two observations can be made regarding ϵ. Notice that the *meet* relation is a subset of the *follow* relation. To prevent any ambiguity, it is assumed that if two event intervals are within a user-defined ϵ threshold, then their relation is *meet*, otherwise it is *follow*. Similarly, *left-contain* and *right-contain* are subsets of *contain* and a clear distinction is achieved again through the ϵ threshold. Thus, the use of ϵ in the *meet*, *left-contain* and *right-contain* relations can efficiently handle "noisy" intervals.

In some applications, the user may not want to consider some of the above relations as

Table 2.1: Subsets of Event Interval Relations

Relation	Could also be counted as
meet	follow
match	overlap
	left-contain
	right-contain
	contain
left/right-contain	overlap
	contain
	overlap

mutually exclusive. Table 2.1 shows how these relations cannot be mutually exclusive. For example, a *match* could also be counted as a *left-contain*, *right-contain*, *contain* and/or *overlap*. Also, a *left-contain* or *right-contain* could be counted as a *contain* or an *overlap* as well. Finally, a *meet* could also be counted as a *follow* or *overlap*. Thus, depending on the application, a user might desire to: (1) collapse some relations, e.g. count *left-contain* and *right-contain* as *contain*, or count each *meet* as *follow*, etc., (2) count them multiple times, e.g. each *overlap* is also counted as *left-contain* and *right-contain*, or each *match* is also counted as *contain*, or each *meet* is also counted as *follow*, etc.

Thus, the user has flexibility with respect to which of these options get chosen and clearly it would be application specific. The user is given the aforementioned flexibility through the implementation of a graphical user interface, which is relatively straightforward.

2.3 Arrangements and Arrangement Rules

Let $\mathcal{E} = \{E_1, E_2, ..., E_m\}$ be an ordered set of event intervals, called *event interval sequence* or *e-sequence*. Each E_i is a triple $(e_i, t^i_{start}, t^i_{end})$, where e_i is an event label, t^i_{start} is the event start time and t^i_{end} is the end time. The events are ordered by the start time. If an occurrence of e_i is instantaneous, then $t^i_{start} = t^i_{end}$. An e-sequence of size k is called a *k-e-sequence*. For example, let us consider the 5-e-sequence shown in Figure 2·3. In this case the e-sequence can be represented as follows: $\mathcal{E} = \{(A, 1, 7), (B, 3, 19), (D, 4, 30), (C, 7, 15), (C, 23, 42)\}$. Finally, an *e-sequence database D*
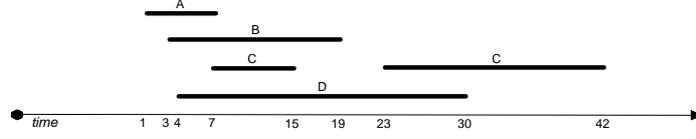
Figure 2·3: *An Example of an e-sequence.*

$= \{\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_k\}$ is a set of e-sequences.

In an e-sequence database there may be patterns of temporally correlated events; such patterns are called *arrangements*. The definitions given in Section 2.1 can describe temporal relations between two event intervals but they are insufficient for relations between more than two. Consider for example the two cases in Figure 2·4. Case (a) can be easily expressed using the current notation as: $A|B \rightarrow C$. This is sufficient to determine that A overlaps with B, C follows B and C follows A. On the other hand, the expression for case (b), i.e. $A|B > C$, is insufficient, since it gives no information about the relation between A and C. Thus, we need to add one more operand to express this relation concisely. In order to define an arrangement of more than two events we need to clearly specify the temporal relations between every pair of its events. This can be done by using the "AND" operand denoted by $\star$. Therefore, the above example can be sufficiently expressed as follows: $A|B \star A|C \star B > C$. Based on the previous analysis, we can efficiently express any kind of relation between any number of event intervals, using the set of operands: $\mathcal{R} = \{|, ||, >, | >, > |, \sim, \rightarrow\}$ and $\star$.

Consequently, an arrangement $\mathcal{A}$ of n events is defined as $\mathcal{A} = \{\mathcal{E}, R\}$, where $\mathcal{E}$ is the set of event intervals that occur in $\mathcal{A}$, with $|\mathcal{E}| = n$, and $R = \{R(E_1, E_2), R(E_1, E_3), ..., R(E_1, E_n), R(E_2, E_3), R(E_2, E_4), ..., R(E_2, E_n), R(E_{n-1}, E_n)\}$. R is the set or temporal relations between each pair (E_i, E_j), for $i = 1, ..., n$ and $j = i + 1, ..., n$, and $R(E_i, E_j) \in \mathcal{R}$ defines the temporal relation between E_i and E_j. The size of an arrangement $\mathcal{A} = \{\mathcal{E}, R\}$ is equal to $|\mathcal{E}|$. An arrangement of size k is called a *k-arrangement*. For example, consider arrangement S' of size 3 shown in Figure 2·4 (a). In this case $\mathcal{E} = \{A, B, C\}$ and $R = \{R(A, B) = |, R(A, C) = \rightarrow, R(B, C) = \rightarrow\}$. The *absolute support* of an arrangement in an e-sequence database is the number of e-sequences

in the database that contain the arrangement. The *relative support* of an arrangement is the percentage of e-sequences in the database that contain the arrangement. Given an e-sequence s, s *contains* an arrangement $\mathcal{A} = \{\mathcal{E}, R\}$, if all the events in $\mathcal{A}$ also appear in s with the same relations between them, as defined in R. Consider again arrangement S' in Figure 2·4(a) and e-sequence s in Figure 2·3. We can see that all the event intervals in S' appear in s and further, they are similarly correlated, i.e. Overlap (A,B), Follow (B,C), Follow (A,C). Thus, S' is contained in or *supported* by s. Given a minimum support threshold min_sup, an arrangement is *frequent* in an e-sequence database, if it occurs in at least min_sup e-sequences in the database.

Itemset association rules have been thoroughly studied in many previous works including (Srikant and Agrawal, 1996; Agrawal and Srikant, 1994). In these approaches an association rule was defined among items that belong to a frequent itemset. A similar definition was given in (Harms et al., 2002) for sequence association rules. Based on the above work, we are going to define association rules for arrangements.

Given two arrangements $\mathcal{A}_i$ and $\mathcal{A}_j$ that have been mined from an e-sequence database D, $r : \mathcal{A}_i \Rightarrow_{\lambda, D}^{R_{ij}} \mathcal{A}_j$ defines an *arrangement rule* between $\mathcal{A}_i$ and $\mathcal{A}_j$, based on an *interestingness measure* λ. This means that, given an arrangement $\mathcal{A} = \{\mathcal{E}, R\}$ that is frequent in D, we can break it into two arrangements $\mathcal{A}_i = \{\mathcal{E}_i, R_i\}$, $\mathcal{A}_j = \{\mathcal{E}_j, R_j\}$ and define a rule between them. Note that $\mathcal{E}$ is split into two sets $\mathcal{E}_i$ and $\mathcal{E}_j$, whereas R_i and R_j are defined based on R, and describe the temporal relations between the event intervals in $\mathcal{E}_i$ and $\mathcal{E}_j$ respectively. Also, R_{ij} defines the set of relations of the event labels $\mathcal{E}_i$ with those in $\mathcal{E}_j$.

2.4 Interestingness Measures

The use of interestingness measures, also known as quantitative measures, plays a very important role in the interpretation of the discovered arrangement rules. Many interestingness measures have been proposed and studied, each of them capturing different characteristics. In this section we give a brief overview of the most common quantitative measures and

show how they can be used for mining arrangement rules.

Given a rule $\mathcal{A} \Rightarrow_{\lambda,\ D}^{R_{AB}} \mathcal{B}$, the desired properties of λ are the following:

- **Property 1:** $\lambda = 0$, if $\mathcal{A}$ and $\mathcal{B}$ are statistically independent.

- **Property 2:** λ monotonically increases with $|(\mathcal{A}, \mathcal{B})|/|D|$, when $|\mathcal{A}|/|D|$ and $|\mathcal{B}|/|D|$ remain the same.

- **Property 3:** λ monotonically decreases with $|\mathcal{A}|/|D|$ (or $|\mathcal{B}|/|D|$) when the rest of the parameters, i.e. $|(\mathcal{A}, \mathcal{B})|/|D|$ and $|\mathcal{B}|/|D|$ or $|\mathcal{A}|/|D|$), remain unchanged.

The above properties are studied in more detail in (Kamber and Shinghal, 1996; Hilderman and Hamilton, 2001) and the extent to which interestingness measures satisfy these properties has been studied and is shown in (Tan et al., 2002). Two other properties of interestingness measures are: *monotonicity* and *anti-monotonicity* (Agrawal and Srikant, 1994):

1. **Monotonicity of an interestingness measure λ:** An interestingness measure λ is monotone, if for any two arrangements $\mathcal{A}$ and $\mathcal{B}$ (with $\mathcal{A} \subseteq \mathcal{B}$), $\lambda(\mathcal{A}) \leq \lambda(\mathcal{B})$.

2. **Anti-monotonicity of an interestingness measure λ:** An interestingness measure λ is anti-monotone, if for any two arrangements $\mathcal{A}$ and $\mathcal{B}$ (with $\mathcal{A} \subseteq \mathcal{B}$), $\lambda(\mathcal{B}) \leq \lambda(\mathcal{A})$.

Given an arrangement rule: $r : \mathcal{A} \Rightarrow_{\lambda,\ D}^{R_{AB}} \mathcal{B}$, we define $cover(\mathcal{A})$ to be the number of records in D that contain arrangement $\mathcal{A}$ over the size of the e-sequence database D, and $coverage(r)$ to be the cover of the antecedent arrangement $\mathcal{A}$. In this thesis, we focus on two anti-monotone interestingness measures: (1) support, (2) all-confidence, and four non anti-monotone: (1) confidence, (2) leverage, (3) lift, and (4) conviction.

2.4.1 Anti-monotone Interestingness Measures

Next, the definitions of two anti-monotone measures are given with respect to an arrangement $\mathcal{A}$ and an event interval database D. Due to the anti-monotonicity property, these measures can be applied on each node and can be used for efficient pruning.

- $supp(\mathcal{A}) = cover(\mathcal{A})$

  This is the most common quantitative measure among the frequent pattern mining algorithms. An arrangement with high support guarantees high co-occurrence of its event intervals in D and can produce interesting rules whose antecedent and consequent arrangements are frequent in D.

- $all\text{-}confidence(\mathcal{A}) = \frac{supp(\mathcal{A})}{max_{1 \leq k \leq m}\{supp(\mathcal{A}_k)\}}$

  The denominator is the maximum number of e-sequences in D that contain any sub-arrangement of $\mathcal{A}$. This states that all-confidence is in fact the smallest confidence of any rule inferred from $\mathcal{A}$.

2.4.2 Non Anti-monotone Interestingness Measures

There has been a great number of interestingness measures proposed and studied, that are not anti-monotone. In this thesis we consider four of them. Next, we give their definitions with respect to an arrangement rule r implied from an arrangement $\mathcal{A}$ that has been mined from an event interval database D. Note, that since these measures are not anti-monotone, they cannot be pushed "all the way" into the mining process. Thus, given an arrangement rule $r : \mathcal{A} \Rightarrow_{\lambda,\ D}^{R_{AB}} \mathcal{B}$, we have:

- $confidence(r) = \frac{supp(r)}{coverage(r)}$

  The confidence of a rule typically expresses the conditional probability of the occurrence of the consequent $\mathcal{B}$ in an e-sequence in D, given that the antecedent $\mathcal{A}$ also occurs in the e-sequence.

- $lift(r) = \frac{supp(r)}{supp(\mathcal{A}) \times supp(\mathcal{B})}$

  Lift is a traditional association rule measure, and it is the ratio of the observed joint frequency of $\mathcal{A}$ and $\mathcal{B}$, and the expected frequency if they were independent. The problem with this measure is the following: a rule with high lift, may be of little interest since it may have low coverage, meaning that it applies in very few records of D. In particular, since $coverage(r) = supp(\mathcal{A})$, we have $\frac{supp(r)}{supp(\mathcal{A}) \times supp(\mathcal{B})} = \frac{supp(r)}{coverage(r) \times supp(\mathcal{B})}$, and as $coverage(r) \downarrow$, $lift(r) \uparrow$.

- $leverage(r) = supp(r) - supp(\mathcal{A}) \times supp(\mathcal{B})$

  Leverage measures the difference between the observed joint frequency of $\mathcal{A}$ and $\mathcal{B}$ (i.e. support of r), and their expected frequency if they were independent. Some useful bounds on leverage have been introduced in (Webb and Zhang, 2005) and are used by the mining algorithms to efficiently prune the search space.

- $conviction(r) = \frac{1 - supp(\mathcal{B})}{1 - confidence(r)}$

  Conviction basically compares the probability of $\mathcal{A}$ appearing without $\mathcal{B}$, assuming independence, with the actual frequency of the appearance of $\mathcal{A}$ without $\mathcal{B}$. A very useful property of conviction is that it is monotone in confidence and lift, i.e.:

$$
\begin{aligned}
conviction(r) &= \frac{1 - supp(\mathcal{B})}{1 - confidence(r)} \\[2mm]
&= \frac{\frac{1}{supp(\mathcal{B})} - \frac{supp(\mathcal{B})}{supp(\mathcal{B})}}{\frac{1}{supp(\mathcal{B})} - \frac{confidence(r)}{supp(\mathcal{B})}} \\[2mm]
&= \frac{\frac{1}{supp(\mathcal{B})} - 1}{\frac{1}{supp(\mathcal{B})} - lift(r)}
\end{aligned}
$$

  and as we can see as $lift(r) \uparrow$, $conviction(r) \uparrow$.

2.5 Temporal and Structural Constraints

Frequency, however, does not always imply interestingness. A pattern can occur frequently in the database but it may not hold interesting information to every user. For example, if a user is very selective and wants to focus on certain patterns, the current formulation
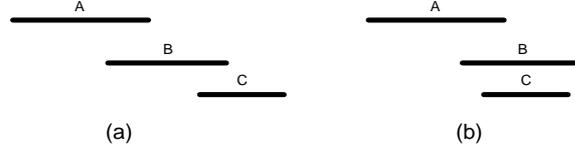
Figure 2·4: *(a) S′ can be expressed with four operands and (b) S″ cannot.*

will yield an extremely unfair computational cost. Thus, some *user-controlled focus* (Garofalakis et al., 1999; Zaki, 2000) needs to be added into the mining process. In addition to the support threshold, the user can also specify a set of constraints $\mathcal{C}$ that includes:

- **A set $\mathcal{R}_e$ of regular expression constraints:** the mined arrangements should follow the regular expressions defined in $\mathcal{R}_e$. Let $r_i \in \mathcal{R}_e$ be a regular expression of size n; then r_i is of the following form: $\mathcal{A}_1 * \mathcal{A}_2 * \mathcal{A}_3 * ... * \mathcal{A}_n$, where $\mathcal{A}_i$ is an arrangement and $*$ is a wildcard that stands for zero or more arrangements of any form.

- **A gap constraint C_g:** two event intervals that take part in a *follow* relation should be separated by at most C_g time units.

- **A pair of overlap constraints $C_o = \{C_o^l, C_o^u\}$:** the overlap of two event intervals that take part in an *overlap* relation, is limited by C_o. In fact, C_o^l, C_o^u can be seen as the lower and upper bound of an overlap relation. This means that if their overlap is less than $C_o^l\%$ then their relation is considered a *meet*; if their overlap exceeds $C_o^u\%$ then their relation is considered a *left-contain*. Given two event intervals $E_1 = (e_1, t_{start}^1, t_{end}^1)$ and $E_2 = (e_2, t_{start}^2, t_{end}^2)$, their *overlap* is equal to $t_{end}^1 - t_{start}^2$, if $t_{start}^1 < t_{start}^2 < t_{end}^1$, otherwise it is zero; and the *overlap percentage* is: $overlap\_percentage = \frac{overlap}{min\{t_{end}^1 - t_{start}^1, t_{end}^2 - t_{start}^2\}}$.

- **A pair of contain constraints $C_{ct} = \{C_{ct}^l, C_{ct}^u\}$:** two event intervals that take part in a *contain*, *left-contain* or *right-contain* relation, should have an overlap of at most $C_{ct}^u\%$. If their overlap exceeds this bound, their relation is considered a *match*, whereas if it is less than C_{ct}^l it is discarded.

- **A duration constraint** C_d**:** each event interval should have a duration of at most C_d units. If not, it is discarded.

2.6   Problem Formulation

Based on the above definitions we can now formulate the problem of *constraint-based mining of frequent arrangements of temporal intervals* as follows:

Problem I: Given an e-sequence database D, a set of constraints $\mathcal{C}$, and a support threshold min_sup, our task is to find set $F = \{\mathcal{A}_1, \mathcal{A}_2, \ ..., \mathcal{A}_n\}$, where $\mathcal{A}_i$ is a frequent arrangement in D and satisfies the constraints in $\mathcal{C}$.

We can further extend the previous formulation to extract arrangement rules given an interestingness measure λ. Also, a set of constraints $\mathcal{C}_R$ is added to the previously defined set of constraints $\mathcal{C}$. $\mathcal{C}_R$ contains a number of constraints for the arrangement rules:

1. a set of constraints for R_{ij} that restricts the relations that connect the antecedent and consequent arrangements.

2. a set of constraints for the set of event labels of the antecedent and consequent arrangements.

Incorporating interestingness measures and the aforementioned constraints to $\mathcal{C}$ (we define $\mathcal{C}' = \mathcal{C} \cup \mathcal{C}_R$), we can formulate the problem of *constraint-based mining of the top-K interesting association rules* as follows:

Problem II: Given a set $\{D, \mathcal{C}', \lambda, k, min\_sup\}$, where D is an e-sequence database, $\mathcal{C}'$ is a set of constraints, λ is an interestingness measure, k is an integer and min_sup is the minimum support threshold that implies frequency, we want to mine the top k frequent arrangement rules that satisfy $\mathcal{C}'$ and maximize λ.

The following Sections present a set of algorithms that deal with both problems. In particular, a set of algorithms to solve Problem I is proposed, and then used to efficiently solve Problem II by getting extended to include the new constraints and also mine rules given an interestingness measure.

Chapter 3

Related Work

Next, the existing work on sequential pattern mining and on temporal mining is presented and a brief overview of the existing interestingness measures that can be applied to the extracted patterns and association rules is given.

3.1 Sequential Pattern Mining Algorithms

Current sequential pattern mining algorithms can be classified to seven different classes with respect to: (1) the methods and data-structures used for the candidate sequence generation, (2) the pruning techniques used to accelerate the mining process, and (3) the final output set that the algorithms are targeting (closed, or non-closed sequential patterns).

3.1.1 Apriori-based Algorithms

The first and simplest family of sequential pattern mining algorithms are the *Apriori-based* algorithms and their main characteristic is that they apply the *Apriori principle* (Agrawal and Srikant, 1994). The problem of sequential pattern mining was introduced in (Agrawal and Srikant, 1995), along with three Apriori-based algorithms (AprioriAll, AprioriSome and DynamicSome). At each step k, a set of candidate frequent sequences $\mathcal{C}_k$ of size k is generated by performing a self-join on $\mathcal{F}_{k-1}$; $\mathcal{F}_k$ consists of all those sequences in $\mathcal{C}_k$ that satisfy a user-specified support threshold. The efficiency of support counting is improved by employing a *hash-tree* structure.

A similar approach, GSP (Generalized Sequential Patterns), was developed in (Srikant

18

and Agrawal, 1996) that pushes time constraints (maximum and minimum gaps between the events) as well as window constraints into the mining process, and was proved to be more efficient than its predecessors. At the same time, (Mannila et al., 1995) introduced the idea of mining frequent episodes, i.e. frequent sequential patterns in a single long input sequence, using a sliding window to cut the input sequence into smaller segments, and employing a mining algorithm similar to that of AprioriAll. Notice, however, that in our formulation we focus on finding frequent patterns across a set of input sequences (that constitute a sequence database) and not across a single sequence.

Discovering all frequent sequential patterns in large databases is a very challenging task since the search space is large. Consider for instance the case of a database with m attributes. If we are interested in finding all the frequent sequences of length k, there are $\mathrm{O}(m^k)$ potentially frequent ones. Increasing the number of objects might definitely lead to a paramount computational cost. Apriori-based algorithms employ a bottom-up search, enumerating every single frequent sequence. This implies that in order to produce a frequent sequence of length l, all 2^l subsequences have to be generated. It can be easily deduced that this exponential complexity is limiting all the Apriori-based algorithms to discover only short patterns, since they only implement *subset infrequency pruning* by removing any candidate sequence for which there exists a subsequence that does not belong to the set of frequent sequences.

3.1.2 Tree-based Algorithms

A faster and more efficient candidate production can be achieved using a tree-like structure (*set-enumeration tree*) (Bayardo, 1998) and traversing it in a depth-first search manner to enumerate all the candidate patterns applying both subset infrequency and superset frequency pruning.

The above idea was initially introduced for mining frequent itemsets, but was extended for sequential patterns. An efficient approach, SPAM (Ayres et al., 2002), employs a *sequence enumeration tree* to generate all the candidate frequent sequences given the set

of event labels. Each level k of the tree contains the complete set of sequences of size k (with each node representing one sequence) that can occur in the database. The nodes of each level are generated from the nodes of the previous level using two types of extensions: (1) *itemset extension* (the last itemset in the sequence is extended by adding one more item to the set), (2) *sequence extension* (a sequence is extended by adding a new itemset at the end of the sequence). The candidate sequences are enumerated by traversing the tree using depth-first search. If an infrequent sequence is reached, the subtree of the node representing that sequence is pruned (subset infrequency pruning). If a frequent sequence is reached, then all its subsequences have to be frequent, thus the tree nodes representing those sequences are skipped (superset frequency pruning). For efficient support counting, a bitmap representation of the database is used, which further improves performance over the lattice-based approaches (Zaki, 2001; Zaki, 2000; Leleu et al., 2003) discussed next.

3.1.3 Lattice-based Algorithms

Another class of sequential pattern mining algorithms includes those that use a lattice structure (Davey and Priestley, 2002) to enumerate the candidate sequences efficiently. Intuitively, a lattice can be seen as a "tree-like" structure where each node can have more than one "father". A node on the lattice that represents a sequence s, is connected to all the pairs of nodes on the previous level that can be joined to form s. This is illustrated in the following example: let $s = \{d, (bc), a\}$, then all the following nodes should be connected to s on the lattice: $\{(bc), a\}$, $\{d, b, a\}$, $\{d, (bc)\}$, $\{d, c, a\}$, since each pair of these subsequences can be joined to form s.

SPADE (Zaki, 2001) uses the above structure to efficiently enumerate the candidate sequences. The basic characteristics of SPADE are the following: (1) it employs a vertical representation of the database using *id-lists*, where each pattern is associated with a list of database sequences in which it occurs. All frequent sequences can be enumerated via temporal joins on the id-lists, (2) it uses a lattice-based approach to decompose the original search space into smaller subspaces, which can be processed independently in main memory,

(3) within each sub-lattice, two different search strategies (breadth-first and depth-first search) are used for enumerating the frequent sequences.

An extension of SPADE, called cSPADE, was proposed in (Zaki, 2000), which allowed a set of constraints to be placed on the mined sequences. These constraints were mainly: (1) length and width constraints (the maximum allowed length and width of a pattern is restricted), (2) gap and window constraints (similar to those of GSP), (3) item constraints (the mining task should return patterns that contain only certain items), and (4) class constraints (these constraints are applicable for classification of datasets where each input sequence has a class label).

A similar algorithm, GO-SPADE (Leleu et al., 2003), was proposed later on, where the idea of *generalized occurrences* was introduced. The intuition behind GO-SPADE is that in a sequence database certain items can occur in a consecutive way, i.e. they may appear in consecutive itemsets in the same sequence. To reduce the cost of the mining process, GO-SPADE mainly tries to compact all these consecutive occurrences by defining a generalized occurrence of a pattern p as a tuple $(sid, [min, max])$, where sid is the sequence id, and $[min, max]$ corresponds to the interval of the consecutive occurrences of the last event of p.

3.1.4   Algorithms with Regular Expression Constraints

Ignoring slight differences in the problem definition, the vast majority of the former algorithms aim the discovery of frequent sequential patterns based on only a support threshold, which limits the results to the most common or "famous" ones. Thus, a lack of *user-controlled focus* in the pattern mining process can be detected that may sometimes lead to an overwhelming volume of potentially useless patterns. A solution to this problem is proposed in (Garofalakis et al., 1999), where the mining process is restricted by not only a support threshold but also by user-specified constraints modelled by regular expressions.

More specifically, (Garofalakis et al., 1999) introduces the family of SPIRIT algorithms, where a set of constraints C is pushed into the mining process along with a sequence

database. Therefore, the minimum support requirement and a set of additional user-specified constraints are applied simultaneously restricting the set of candidate sequences produced during the mining process. To accomplish this, two different types of pruning techniques are used: *constraint-based* and *support-based* pruning. The first uses a *relaxation* C' of C ensuring that in every pass of the candidate generation all the candidate sequences satisfy C'. The second, tries to ensure that all the subsequences of a candidate sequence that satisfy C' are present in the current set of discovered frequent sequences.

Another characteristic of the SPIRIT algorithms concerns *anti-monotonicity*. Consider a given set of candidates C and a relaxation C' of C. In fact C' is a weaker constraint which is less restrictive, however all the sequences that satisfy C also satisfy C'. C' is *anti-monotone*, if all subsequences of a sequence satisfying C' are guaranteed to also satisfy C'. In such case, support-based pruning is maximized, since support information for every subsequence of a candidate sequence in C' can be used for pruning. Moreover, if C' is not anti-monotone, the efficiency of both support-based and constraint-based pruning depends on the relaxation C'.

3.1.5 Prefix-based Algorithms

Another class of sequential pattern mining algorithms includes the prefix-based ones (Fei et al., 2001; Wang and Han, 2004; Yan et al., 2003). In this case, the database is projected with respect to a frequent prefix sequence and based on the outcome of the projection, new frequent prefixes are identified and used for further projections until the support threshold constraint is violated.

The main steps of a prefix-based algorithm are the following: (1) scan the database for the frequent 1-sequences, (2) for each frequent 1-sequence s found in the previous step, project the database with respect to s, (3) scan the projected database for locally frequent items, (3) add each new frequent item to the end of the prefix and project the database with respect to the new prefix, (4) repeat steps 3-4 for each new prefix, until the projected database is of size less than the support threshold.

3.1.6 Algorithms for Mining Closed Sequential Patterns

All algorithms described so far, mine the complete set of frequent sequences including their subsequences. However, recent research and studies have presented convincing arguments that only closed frequent sequences should be mined targeting more compact results and higher efficiency (Zaki and Hsiao, 2002; Pei et al., 2000; Wang and Han, 2004; Yan et al., 2003; Pasquier et al., 1999). Two of the most efficient algorithms for mining frequent closed sequences BIDE (Wang and Han, 2004) and CloSpan (Yan et al., 2003) are based on the notion of the *projected database* and use special techniques to limit the number of frequent sequences and finally only keep the closed ones.

In particular, CloSpan follows the *candidate maintenance-and-test* approach, i.e. it first generates a set of closed sequence candidates which is stored in a hash-indexed tree structure and then prunes the search space using *Common Prefix* and *Backward Sub-Pattern pruning* (Yan et al., 2003). The main drawback of CloSpan is the fact that it consumes much memory when there are many closed frequent sequences, since pattern closure checking leads to a huge search space. Consequently, it does not scale very well with respect to the number of closed sequences. In order to face this weakness, BIDE employs a *BI-Directional Extension* paradigm for mining closed sequences, where a *forward directional extension* is used to grow the prefix patterns and check their closure and a *backward directional extension* is used to both check the closure of a prefix pattern and prune the search space. In overall, it has been shown that BIDE has surprisingly high efficiency, regarding speed (an order of magnitude faster than CloSpan) and scalability with respect to database size.

3.2 Temporal Mining and Association Rules

Up to this point, the events have been considered to be instantaneous. There have been several approaches on discovering intervals that occur frequently in a transactional database (Lin, 2003; Lin, 2002). In most cases, however, the intervals are unlabelled and no relations

between them are considered. (Villafane et al., 2000) extends the sequential approach by also including the *contain* relation introduced previously. To efficiently mine the arrangements, it employs a *containment graph* representation that imposes a partial order on the event intervals.

Extending earlier work on mining frequent episodes in a single sequence of events (Mannila et al., 1995; Mannila and Toivonen, 1996), there have been various approaches that consider interval-based events. (Hoeppner, 2001; Mooney and Roddick, 2004; Hoeppner and Klawonn, 2001) employ apriori-baeed techniques to find temporal patterns that occur frequently in the input event sequence. Along with the frequent patterns, they extract association rules and the latest applies some interestingness measures to evaluate their significance. These measures, however, are not pushed into the mining process; they are applied to the set of frequent patterns after the mining process has been completed.

Another approach that considers sequences of interval-based events in a database is discussed in (Kam and Fu, 2000). In this case, the extracted patterns are limited to some certain forms. Let A_i denote an interval-based event and rel_{ij} the temporal relation between events A_i and A_j, and let $A_i \ rel_{ij} \ A_j$ denote the temporal relation between A_i and A_j. In (Kam and Fu, 2000) the extracted patterns are of the following two forms:

- Form 1: $((... (A_1 \ rel_{12} \ A_2) \ rel_{23} \ A_3) \ ... \ rel_{(k-1)k} \ A_k)$.

- Form 2: let X be a temporal relation of size 2 and Y be a pattern of Form 2, then $X \ rel_{ij} \ Y$ is a temporal pattern of Form 2.

Notice that the aforementioned approaches are Apriori-based and do not consider any temporal or structural constraints for the extracted arrangements. Furthermore, the event interval relations used are not robust and cannot efficiently handle noisy data, i.e. noise at the start and end-points of the intervals. To the best of our knowledge, the first *tree-based* approach was proposed in (Papapetrou et al., 2005), where a *tree-like* structure was used to enumerate the set of arrangements and efficiently mine the frequent ones.

In the interim, there has been significant work on discovering association rules on sequential and temporal data. Association rules among items that belong to a frequent itemset are defined in (Srikant and Agrawal, 1996; Agrawal and Srikant, 1994). Similar definitions are given in (Harms et al., 2002) for sequence association rules, and in (Hoeppner, 2001; Hoeppner and Klawonn, 2001) for association rules among interval-based events. In the above works, the evaluation of the rules is achieved by the usage of interestingness measures. The most common ones (introduced in (Agrawal and Srikant, 1994)) are *support* and *confidence.* Using a non Apriori-based technique that avoids multiple database scans, (E.Winarko and J.F.Roddick, 2005) achieved to efficiently mine arrangements and rules in a temporal database. However, in their methods they do not consider any constraints for the temporal relations and do not examine any measures for their rules other than the traditional confidence. Temporal association rules combine traditional association rules with temporal aspects by using time stamps that describe the validity, periodicity, or change of an association. (Oezden et al., 1998) studies the problem of mining association rules that hold only during certain cyclic time intervals. It is argued that reducing the temporal granularity can lead to the extraction of more interesting rules. In a same fashion, (Chen and Petrounias, 1999; Abraham and Roddick, 1999) consider the discovery of association rules in temporal databases and thus the extraction of temporal features of associated items. The support of the rules is measured only during these intervals. Moreover, in (Ale and Rossi, 2000), the lifetime of an item is defined as the time between the first and the last occurrence and the temporal support is calculated with respect to this interval. In this way, the extracted rules are only active during a certain time, and outdated rules can be pruned by the user. Finally, (Lu et al., 1998) studies inter-transaction association rules by merging all itemsets within a sliding time window inside a transaction, whereas in (Tsoukatos and Gunopulos, 2001) efficient techniques for mining spatiotemporal patterns are proposed.

3.3   Interestingness Measures

There has been a variety of studies on other interestingness measures (Tan and Kumar, 2000) that provide more accurate results by removing redundancy and limiting the number of extracted rules to the most interesting ones. (Omiecinski, 2003) proposes alternative association rule measures for evaluating the importance of association rules in transactional databases, whereas (Kamber and Shinghal, 1996) introduces some efficient techniques for evaluating the interestingness of rules. (Hilderman and Hamilton, 1999) carried out a survey on the existing interestingness measures and their significance in association rule mining. In (Hilderman and Hamilton, 2001), a study on the performance of different association rule measures is presented, where different measures are being used to rank the extracted rules on various datasets and determine the appropriate measure for each dataset. Moreover, (Tan et al., 2002) provides the intuition behind each interestingness measure and gives the basic properties that effective rule measures should possess. It further presents an analysis of the main characteristics of the most common rule measures and suggests a technique for selecting the right one (most effective) for a given application. Recent work (Webb, 2006) has proposed generic techniques that provide effective control over the mining process that restricts the number of false rules. Finally, (Xin et al., 2006) presents two algorithms for discovering interesting patterns where the mining process is guided by the user's interactive feedback. In particular, they employ a so called *user-specific* interestingness measure thats consists of a ranking function and a model of prior knowledge that has been defined by the user. Despite all the aforementioned studies there has been yet no approach that considers interestingness measures on interval-based rules other than the traditional support and confidence.

Finally, there has been some work on constraint-based mining of frequent itemsets, where the goal is to mine the top k patterns that maximize an interestingness measure (other than the typical support threshold) and satisfy a set of constraints (Webb and Zhang, 2005). A similar approach is considered in this thesis; in our case however, we deal with sequences of interval-based events instead of itemsets.

To recap, there have been various approaches on mining frequent arrangements of temporal intervals; most of them however, are Apriori-based, in some cases (Kam and Fu, 2000) the extracted patterns are limited to certain forms, and no constraints are considered. Furthermore, in most cases the extraction of arrangement rules is performed after the detection of the frequent patterns and no attempt has been made to push it into the mining process. Also, no other measure is used, except for the traditional support and confidence, to evaluate the interestingness of each rule. Current algorithms target all rules that satisfy the desired measures and do not incorporate any constraints regarding the form of each rule. In this work, we present the first "tree-based" attempt to mine frequent arrangements of temporal intervals, where an efficient method is developed that employs a "tree-like" structure to enumerate the candidate set of arrangements. Furthermore, the problem of extracting arrangement rules is being considered, and in our case, efficient pruning techniques are applied and the notion of arrangement rules is generalized by including constraints and other interestingness measures except for the traditional support and confidence.

Chapter 4

Algorithms

A straightforward approach to mine frequent patterns from a database of e-sequences D is to reduce the problem to a sequential pattern mining problem by converting D to a transactional database D'. Without any loss of information, we can keep only the start and end time of each event interval. For example, for every event interval $(e_i,\ t_s,\ t_e)$ in D, that describes an event e_i starting at t_s and ending at t_e, we only keep t_s and t_e in D'. Now, we can apply an efficient existing sequential pattern mining algorithm, e.g., SPAM (Ayres et al., 2002), to generate the set of frequent sequences FS in D'. Every pattern in FS should be post-processed to be converted to an arrangement. However, this approach has two basic drawbacks, regarding cost and efficiency: (1) post-processing can be very costly, since in the worst case the number of frequent patterns in FS will be exponential ($O(2^{|N|})$), where N is the number of distinct items in the database, and the cost of converting every pattern f in FS to an arrangement is $O(|f|^2)$, (2) the patterns in FS will carry lots of redundant information; and this redundancy will still be present even if we apply an efficient closed sequential pattern mining algorithm (Wang and Han, 2004).

Next, we describe three efficient algorithms for mining frequent arrangements of temporal intervals that address the previous problems. The first two, employ a tree-based enumeration structure, like the one used in (Bayardo, 1998; Zaki, 2001; Ayres et al., 2002). The first algorithm uses BFS to generate the candidate arrangements, whereas the second uses DFS. Although the BFS-based approach is equivalent to Apriori, the algorithm is further extended to include temporal and structural constraints. The third algorithm em-

Database D	
id	e-sequence
1	A [1, 3], B [1, 3], A [6, 12], B [8, 11], C [9, 10]
2	A [1, 2], B [2, 6], A [10, 12], B [11, 15], C [14, 17]
3	B [1, 3], A [4, 7], A [9, 11], B [11, 12] , C [12, 14]
4	B [1, 5], A [6, 14], B [7, 10], C [8, 9]

Figure 4·1: *An e-sequence database D.*

ploys a prefix-growth approach, similar to (Fei et al., 2001). However, the cost of projection in the case of the event interval database is very high which makes the algorithm inefficient.

4.1   The Arrangement Enumeration Tree

The tree-based structure used by the first two algorithms is called *arrangement enumeration tree*. An arrangement enumeration tree is shown in Figure 4·2. Each level k consists of a set of nodes, denoted as $N(k)$, that hold the complete set of k-arrangements. Let n_i^k denote node i on level k, where i indicates the position of n_i^k in the k-th level based on the type of traversal used by the algorithm. For every node $n_i^k \in N(k)$, we consider the arrangement $\mathcal{A}=\{\mathcal{E}, R\}$ defined by the node, based on which, an intermediate set of nodes (as shown in Figure 4·2) is created, denoted as $M^k(n_i^k)$, linking to n_i^k. Each node in $M^k(n_i^k)$ represents a temporal relation in R. In the case shown in Figure 4·2, $\mathcal{E} = \{A, B, C\}$ and on level 1, $N(1) = \{\{A\}, \{B\}, \{C\}\}$, i.e. we have one node for every item in $\mathcal{E}$. Then, performing temporal joins on the nodes of level 1, the set of the 2-arrangements of Level 2 is generated, with $N(2) = \{\{A, A\}, \{A, B\}, \{A, C\}, \{B, A\}, \{B, B\}, \{B, C\}, \{C, A\}, \{C, B\}, \{C, C\}\}$, and for each node n_i^2 set $M^k(n_i^k)$ is defined. In general, on level k: (1) $N(k)$ is created by joining the nodes in $N(k$-1) with those in $N(1)$, (2) for every node n_i^k, $M^k(n_i^k)$ is defined and then linked to n_i^k. The arrangement enumeration tree is created as described above, using the set of operands defined in chapter 2 and it is traversed using either breadth-first or depth-first search.
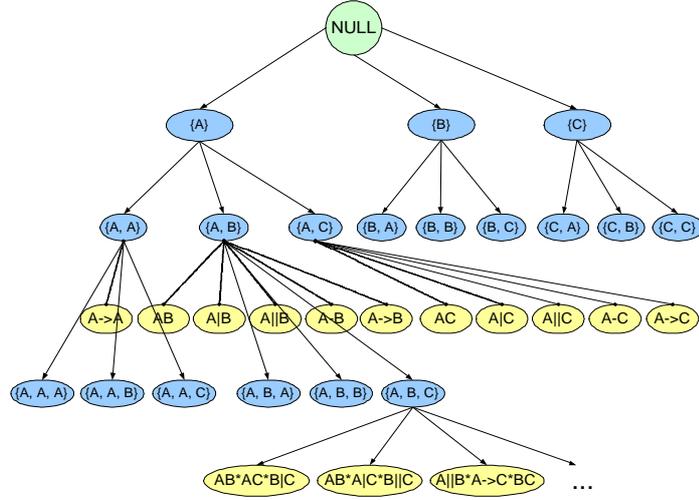
Figure 4·2: *An arrangement enumeration tree.*

4.2   BFS-based Approach

In this section we consider an event interval mining algorithm that uses the arrangement enumeration tree described above to generate the set of candidate arrangements and then prunes those that are not frequent or cannot lead to any frequent arrangement if expanded. The algorithm traverses the tree using breadth first search which is equivalent to the Apriori-based approaches described in section 3.1. The main characteristic of this algorithm is that a set of constraints has been incorporated into the mining process.

First, we introduce the *ISIdList* structure, that attains a compact representation of the intervals and a relatively low join cost. More specifically, an ISIdList is defined for every arrangement generated by this process. The head of the list is the representation of the arrangements using $\mathcal{R}$ and the event labels comprised in it; each record is of type (*id*, *intv-List*), where *id* is the e-sequence id in D that supports the arrangement, and *intv-List* is a double-linked list of all the time intervals during which the arrangement occurs in the corresponding e-sequence in D.

Consider, for example, an e-sequence database D with three unique items A, B and C, as in Figure 4·1. The ISIdLists of A and B is shown in Figure 4·3. Let F_k denote the complete set of frequent k-arrangements and C_k the set of candidate frequent k-arrangements.

A	
esid	Intv-List
1	[1, 3]
1	[7, 10]
2	[1, 2]
2	[10, 12]
3	[4, 7]
3	[9, 11]
4	[6, 14]

B	
esid	Intv-List
1	[1, 3]
1	[8, 9]
2	[2, 6]
2	[11, 15]
3	[1, 3]
3	[11, 12]
4	[1, 5]
4	[7, 10]

Figure 4·3: *ISId-Lists for items A and B.*

Our algorithm will first scan D to find F_1, i.e. the complete set of 1-arrangements. To achieve this, a scan will be performed on D for every event type e_i. If the number of e-sequences in D that contain an interval of e_i satisfies the support threshold, e_i will be added to F_1, and its ISIdList will be updated accordingly.

In order to generate the candidate 2-arrangements, we use the arrangement enumeration tree described above to get the nodes of level 2, along with the set of their corresponding intermediate nodes. Then, removing those that do not satisfy the support threshold constraint we get set F_2 of frequent 2-arrangements.

Moving to the next levels, i.e. generating the set of frequent k-arrangements, we traverse the nodes on level k-1. Note that these nodes correspond to the set of frequent $(k-1)$-arrangements. For every node n_i^{k-1}, a new node n_i^k is created on level k, along with the set of intermediate nodes $M^k(n_i^k)$, one for every type of correlation of the items in n_i^k. For every node in $M^k(n_i^k)$ an ISIdList is created that contains: (1) the set of items of n_i^k, (2) the types of 2-relations between them, (3) for every type of 2-relation a pointer to the intermediate nodes on Level 2 that correspond to that 2-relation. Also, note that if an arrangement is found to be infrequent, then the node in the tree that corresponds to that arrangement is no further expanded.

The above process is more clear through the following example: consider database D in Figure 4·1 and assume that $min\_sup = 2$. Scanning D and filtering with min_sup, we get $F_1 = \{\{A\}, \{B\}, \{C\}\}$. Based on F_1 and the enumeration tree, set F_2 of the frequent 2-arrangements is generated. In our case, we get all the possible pairs of the 1-arrangements in F_1, i.e. N(2), and for every pair of events in the arrangements, D is scanned to get all the

input : D: a database of e-sequences.

     min_sup: minimum support threshold.

     $\mathcal{C}$: a set of constraints.

     λ: an interestingness measure.

     k: an integer.

output : The set F of the frequent arrangements in D that satisfy $\mathcal{C}$.

     The set A_R of the top k rules that satisfy the constraint λ.

$F = \emptyset$;

foreach *event type e_i* **do**

  if *e_i exists in D* **then**

    | $C_1 = C_1 \cup e_i$;

  end

end

$F_1 = \{e_i \in C_1 \mid e_i.cupport \geq min\_sup , C_d \ is \ satisfied\}$;

while $F_{k-1} \neq \emptyset$ **do**

  $N(k) = generate\_candidates \ (N(k-1), N(2))$;

  // The next set of nodes on the tree is determined, following BFS traversal

  foreach *node $n_i^k \in N(k)$* **do**

    $M^k(n_i^k) = generate\_krelations()$;

    // this function generates the nodes in M^k, along with their ISIdLists.

    // for the case where $k = 2$, it ensures that C_{ct}, C_g and C_o are satisfied.

    $C_k = M^k$;

    foreach *candidate $c \in C_k$* **do**

      if *c.support $< min\_sup$ and c does not satisfy $\mathcal{R}_e$* **then**

        $C_k.remove(c)$; // removes c from C_k.

        $prune\_subtree(c)$; // prunes subtree(c).

      end

    end

    $F_k = C_k$;

    $extract\_rules(C_k, \lambda)$;

  end

end

Algorithm I: *A BFS-based algorithm for discovering the complete set of frequent temporal arrangements and the top k arrangement rules in a database of e-sequences given a set of constraints and an interestingness measure.*

types of relations between them, i.e. M^2. If these relations satisfy the support threshold they are added to F_2. Then we produce F_3 based on F_2. The algorithm first creates $N(3)$, following a breadth-first search traversal, along with the set of intermediate nodes. Every node in M^3 that satisfies min_sup is added to F_3, which in our case consists of only one arrangement: $\{(A, B, C), (>, >, >)\}$. F_1, F_2 and F_3 are shown in Figure 4·6. The main steps of this method are described in Algorithm I, considering an input database D, a minimum support threshold min_sup, an interestingness measure λ, a set of constraints $\mathcal{C}$ and an integer k.

Furthermore, during the above process, the set of constraints $\mathcal{C}$ described in chapter 2

is applied taking into account that the efficiency of pruning depends on the degree to which the constraints are pushed into the mining process. As regards the regular expression constraints and using the approach proposed in (Garofalakis et al., 1999), one option is to push the constraints all the way into the mining algorithm. In this case, the generation of the enumeration tree is limited by the regular expressions in $\mathcal{R}_e$, i.e. only the nodes that correspond to those expressions are created. Another option is to apply the regular expressions after the generation of each arrangement and before the application of any interestingness measure, i.e. when an arrangement is created, we first check whether it satisfies $\mathcal{R}_e$, and then apply the rest of the constraints along with the interestingness measure that the user required. As far as gap, overlap and contain constraints are concerned, they are applied at the second step of the algorithm, when the set of frequent 2-arrangements is created. Finally, the duration constraint is applied at the first step of the algorithm, when the set of frequent 1-arrangements is created.

Regarding the rule generation, two approaches can be followed: one is to extract the rules after the mining process is completed, i.e. given the complete set F of frequent arrangements and the user-specified interestingness measure λ, we apply a technique similar to the one proposed in (Agrawal and Srikant, 1994) to extract the rules implied from F and maximize λ. The second and more efficient approach is to push the interestingness measure into the mining process as "deep" as possible. This, however, depends on whether the interestingness measure satisfies the anti-monotonicity property. If so then it can be incorporated into the mining process; if not then the first approach is employed. More details on this step are given in section 4.6.

4.3   DFS-based Approach

In a BFS approach the arrangement enumeration tree is explored in a top-bottom manner, i.e. all the children of a node are processed before moving to the next level. On the other hand, when using a depth-first search approach, we must all sub-arrangements on a path must be explored before moving to the next one. A DFS-based approach for mining

frequent sequences has been proposed in (Tsoukatos and Gunopulos, 2001). Based on this, the previous algorithm can be easily modified to use a depth-first search candidate generation method. This can be done by adjusting function *generate_candidates*() so that it follows a depth-first search traversal. Consider the previous example: our algorithm will first generate node $n_1^1 = \{A\}$ followed by $M(n_1^1)$, then $n_1^2 = \{A, A\}$ followed by $M(n_1^2)$, and so on. Again, the constraints are applied in a similar fashion as in BFS, and the extraction of the rules is described in section 4.6.

The advantage of DFS over BFS is that DFS can very quickly reach large frequent arrangements and therefore, some expansions in the other paths in the tree can be avoided. For example, say that a k-arrangement $\mathcal{A}$ is found to be frequent. Then, the set of all sub-arrangements of $\mathcal{A}$ will also be frequent according to the Apriori principle. Thus, those expansions can be skipped, reducing the cost of computation. To do so, one more step is added to Algorithm 4.2: when a node is found to contain a frequent arrangement, each sub-arrangement is added to F and the corresponding expansions are made on the tree. However, in BFS there is more information available for pruning. For example, knowing the set of 2-arrangements before constructing the set of 3-arrangements can prevent us from making expansions that will lead to infrequent arrangements. This information, however, is not available in DFS.

4.4   Hybrid DFS-based Approach

A hybrid event interval mining approach is considered, based on the following observation: since the ISIdLists contain pointers to the nodes on the second Level of the tree, a DFS-based approach would be inappropriate since for every node n_i^k we would have to scan the database multiple times to detect the set of 2-relations among the items in that node. In the BFS-based approach these nodes will already be available, since they have been generated in the second step of the algorithm. Thus, we use a hybrid DFS approach that generates the first two levels of the tree using BFS and then follows DFS for the rest of the tree. This would compensate for the multiple database scans discussed above, since

the set of frequent 2-arrangements will already be available thereby eliminating the need for multiple database scans.

4.5 A Prefix-based Approach

A prefix-based algorithm for mining frequent arrangements of temporal intervals is presented. However, we will show that in the case of interval-based events, a prefix-growth approach is quite inefficient, especially when the size of the e-sequences is large and there is repetition of the same event labels in the same e-sequence.

Consider an arrangement $\mathcal{A} = \{\mathcal{E}, R\}$ and an e-sequence S. The *projection of S* with respect to $\mathcal{A}$ is the remaining part S' of S, if the *first occurrence* of $\mathcal{A}$ in S is removed. Figure 4·4 shows an example of a projection. Next, we define the projection of an e-sequences database with respect to an arrangement. Using the definition given in (Fei et al., 2001) for the sequential approach, we can define the *projection of an e-sequence database D* with respect to an arrangement $\mathcal{A}$ as the e-sequence database D' produced from D, if each record (e-sequence) in D is projected with respect to $\mathcal{A}$. However, this definition is incomplete. The problem is illustrated by the example shown in Figure 4·5, where an e-sequence database of two records is considered. Following the basic steps of the prefix-growth mining algorithms with support threshold $min\_sup = 2$, we have:

1. Scan D for frequent 1-arrangements: in our case we detect $\mathcal{A}$ and $\mathcal{C}$.

2. Project the database with respect to each of the arrangements found at Step 1.

3. The projection with respect to $\mathcal{A}$ is shown in Figure 4·5 and will yield one new locally frequent arrangement, $\mathcal{C}$, since the support threshold equal to 2.

4. The result of Step 3 is the detection of $A \rightarrow C$ in the first e-sequence and $A \mid C$ in the second.

5. Another projection follows with respect to $\mathcal{C}$, but it produces an empty e-sequence database and therefore the mining process is terminated.
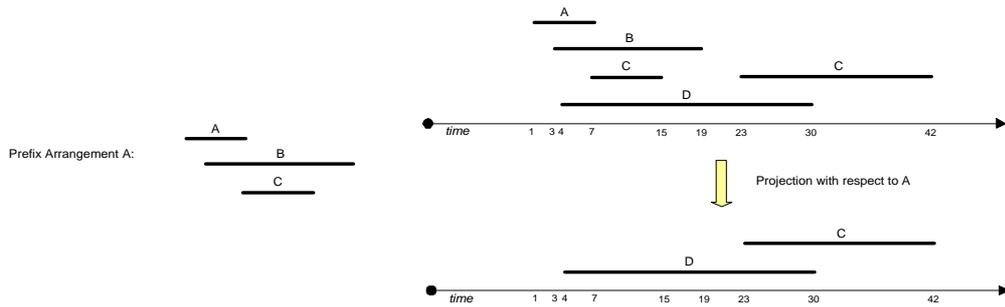
Figure 4·4: *An Example of a Projection.*

As it can be seen, we failed to get $A \mid C$ with support 2. In fact, $A \mid C$ was produced after the first projection, as shown in Figure 4·5, but it was not considered frequent since its support was erroneously calculated as 1. This example shows that when an e-sequence database is projected with respect to a prefix arrangement, finding only the first occurrence of the arrangement may hide some patterns and prevent the mining algorithm from detecting them.

Thus, given an e-sequence database D and an arrangement $\mathcal{A}$, the *projected e-sequence database D'* with respect to $\mathcal{A}$ can be obtained from D, if from each record in D we find *every* occurrence (not just the first one) of $\mathcal{A}$ and project with respect to each one of them. It can be seen that such an approach can lead to a huge computational cost, since for each database record, all the combinations of the occurrences of a prefix should be examined and not just the first one.

In our experimental evaluation we show the performance of the prefix-growth approach and compare it with the BFS and DFS approaches presented previously.

4.6   Applying Other Interestingness Measures

In the previous sections, three efficient methods are presented for mining the complete set F of frequent arrangements of an e-sequence database D. What remains to be done is to discover the set of top k arrangement rules that maximize the given interestingness measure λ in D. A very important issue here, is how deep we can push λ into the mining process.
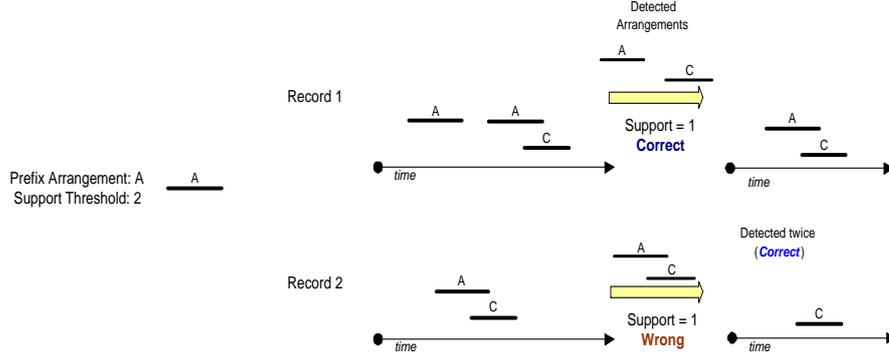
Figure 4·5: *An Example of an e-sequence database of two records and a projection that does not work.*

This, in fact, depends on the properties of λ and more specifically on anti-monotonicity. Next, we present two approaches to handle λ based on whether λ is anti-monotone or not.

4.6.1 Handling Non Anti-monotone Interestingness Measures

If an interestingness measure λ does not preserve the anti-monotonicity property, we have two options: (1) infer the arrangement rules from the extracted frequent arrangements after the mining process is completed (this process is similar to the one described in (Agrawal and Srikant, 1994) for mining association rules), (2) find an upper-bound for λ and push it into the mining process as much as possible.

The first option is quite straightforward. Given F and D, for each $\mathcal{A}_i \in F$, with $\mathcal{A}_i = \{\mathcal{E}, R\}$:

1. $\mathcal{E}$ is split into two sets $\mathcal{E}_1$ and $\mathcal{E}_2$, such that $\mathcal{E}_1 \cup \mathcal{E}_2 = \mathcal{E}$ and $\mathcal{E}_1 \cap \mathcal{E}_2 = NULL$. Based on R, two arrangements are defined: $\mathcal{A}_1 = \{\mathcal{E}_1, R_1\}$ and $\mathcal{A}_2 = \{\mathcal{E}_2, R_2\}$.

2. Apply λ on $r: \mathcal{A}_1 \Rightarrow_{\lambda, D}^{R_{12}} \mathcal{A}_2$.

3. If r satisfies λ and $\mathcal{C}_R$ of $\mathcal{C}'$, add it into the set of valid rules, else discard r.

4. If the set of rules has reached the desired size K, the rule with the smallest λ value (let it be λ_min) is removed and replaced by the new rule, as long as the new rule's

value is greater than λ_min. If not, then the new rule is discarded.

The above algorithm will produce the complete set of top K arrangement rules that maximize the constraint λ. Moreover, based on the mining process and the constraints applied during the extraction of the frequent patterns we have ensured that these rules will satisfy the set of constraints $\mathcal{C}' - \mathcal{C}_R$.

However, it would be more efficient if we could push λ into the mining process and use it for faster and more efficient pruning. The only problem is that λ is not anti-monotone. One way to overcome this issue and achieve some pruning is to find a bound value (upper or lower) $bound\_\lambda$ for λ, such that when an arrangement $\mathcal{A}$ is reached on the tree, if $\lambda(\mathcal{A}) < Upperbound\_\lambda$, then none of the rules implied by $\mathcal{A}$ can lead to an arrangement that satisfies λ, or if $\lambda(\mathcal{A}) > Lowerbound\_\lambda$ and $Lowerbound\_\lambda > \lambda$, then all rules implied by $\mathcal{A}$ can lead to an arrangement that satisfies λ. One such bound was used in (Bayardo et al., 1999) for association rules and the following claim extends it for the case of arrangement rules:

Claim 1: If $all\text{-}confidence(\mathcal{A}) \geq \lambda$, then all rules implied by $\mathcal{A}$ can lead to an arrangement that satisfies $confidence$, and thus they are not examined.

Proof: Since $all\text{-}confidence$ of an arrangement is the minimum confidence of any rule inferred from it, the Claim is straightforward.

Another sort of pruning would be to find a way to imply whether a rule satisfies λ by calculating a simpler form of the rule that reduces the computational cost. There have been several works on bounding interestingness measures for frequent itemset mining (Bayardo et al., 1999; Webb and Zhang, 2005; Omiecinski, 2003). These bounds were used to prune the search space during the rule extraction process. In this thesis we borrow some of those bounds and infer some new ones to incorporate some of the non anti-monotone measures into the mining process.

1. **Bounding Confidence:**

   Given an arrangement $\mathcal{A}$, for any rule $r : \mathcal{A}_1 \Rightarrow^{R_{12}}_{\lambda,\,D} \mathcal{A}_2$ implied from $\mathcal{A}$, where λ in this case stands for confidence, we have the following claim:

   Claim 2: If $\frac{cover(\mathcal{A}_2)}{cover(\mathcal{A}_1)} < min\_Confidence$, then r does not satisfy λ.

   Proof: Straightforward from the definition of confidence.

2. **Bounding Leverage:**

   Given an arrangement $\mathcal{A}$, for any rule $r : \mathcal{A}_1 \Rightarrow^{R_{12}}_{\lambda,\,D} \mathcal{A}_2$ implied from $\mathcal{A}$, where λ in this case stands for leverage, we have the following claim:

   Claim 3: If $cover(\mathcal{A}_1) > 1 - \frac{min\_Leverage}{cover(\mathcal{A}_1)}$ or $cover(\mathcal{A}_2) > 1 - \frac{min\_Leverage}{cover(\mathcal{A}_2)}$, then r does not satisfy λ.

   Proof: Shown in (Webb and Zhang, 2005) for association rules and it can be easily extended for arrangement rules.

4.6.2 Handling Anti-monotone Interestingness Measures

In this subsection we consider the case where the interestingness measure λ preserves the anti-monotonicity property. If λ is anti-monotone, it can be pushed much "deeper" into the mining process. When an arrangement $\mathcal{A}$ is reached on the enumeration tree, if there exists no rule r inferred from $\mathcal{A}$ satisfying λ, then the subtree of the node representing $\mathcal{A}$ is pruned, since it cannot produce any interesting rule. However, if there exists a set of rules $\mathcal{R}_{\mathcal{A}}$ for which λ holds, then the mining process continues with the subtree. In this

case though, the rules that are going to be discovered in the subtree depend on $\mathcal{R}_\mathcal{A}$, based on which, the rule extraction process can be accomplished faster by excluding those rules that will definitely not satisfy λ. Let $\mathcal{R}_\mathcal{A} = \{r_1,\ r_2,\ ..., r_n\}$ be the set of rules inferred from the arrangement at node n on the tree, where $r_i\ :\ \mathcal{A}_i\ \Rightarrow_{\lambda,\ D}^{R_{\mathcal{A}_i \mathcal{B}_i}}\ \mathcal{B}_i$. When n is expanded, for each new arrangement $\mathcal{C} = \{\mathcal{E},\ R\}$, we follow the same process as in the previous section to discover the new arrangement rules, however the search is limited by $R_\mathcal{A}$ as follows:

1. $\mathcal{E}$ is split into two sets $\mathcal{E}_1$ and $\mathcal{E}_2$.

2. If there is no arrangement $\mathcal{A}_i = \{\mathcal{E}_i,\ R_i\}$ in the antecedent part of any rule in $\mathcal{R}_\mathcal{A}$, such that $\mathcal{E}_1 \supseteq \mathcal{E}_i$, then due to the anti-monotonicity property, $\mathcal{E}_1$ cannot be the antecedent part of any rule inferred from $\mathcal{C}$; thus this split is skipped.

As it can be seen, an anti-monotone interestingness measure λ can be pushed into the mining process and used for more efficient pruning and thus lead to a faster rule extraction. Algorithm II shows how we can extract the set of top K rules, given set of frequent arrangements.

input   : D: a database of e-sequences.

              C_k: the set of candidate k-arrangements.

              λ: an interestingness measure.

output   : The set A_R of top k arrangement rules in D that satisfy λ.

$A_R = \emptyset$;

foreach $\mathcal{A}_i = \{\mathcal{E}, R\} \in C_k$ **do**

    if λ *is anti-monotone and* $\mathcal{A}_i$ *does not satisfy* λ **then**

        prune the subtree beyond $\mathcal{A}_i$;

        continue;

    end

    apply bounds on $\mathcal{A}_i$;

    if $\lambda(\mathcal{A}_i) \leq Upperbound\_\lambda$ **then**

        continue;

    end

    // apply lower bound, if such bound exists

    foreach $e \in \mathcal{E}$ **do**

        split $\mathcal{E}$ into two sets $\mathcal{E}_1$ and $\mathcal{E}_2$;

        $prune\_split()$;

        // based on the set of rules $\mathcal{R}_\mathcal{A}$ defined in the previous level

        // prune the split if necessary.

        based on R: define $\mathcal{A}_1 = \{\mathcal{E}_1, R_1\}$ and $\mathcal{A}_2 = \{\mathcal{E}_2, R_2\}$;

        apply λ on r : $\mathcal{A}_1 \Rightarrow_\lambda^D \mathcal{A}_2$;

        if r *does not satisfy* λ **then**

            continue;

        end

        if $|A_R| \geq k$ **then**

            min = minimum λ-value in A_R;

            if $r.\lambda \geq min$ **then**

                remove rule with $\lambda = min$;

                $A_R = A_R \cup r$;

            end

        end

    end

end

Algorithm II: *Extracting the set of arrangement rules given a set of candidate arrangements.*

{A, A}

Follow (A, A)	
esid	*Intv-List*
1	[1, 3] , [6, 12]
2	[1, 2] , [10, 12]
3	[4, 7] , [9, 11]

{A, B}

Meet (A, B)	
esid	*Intv-List*
2	[1, 2] , [2, 6]
3	[9, 11] , [11, 12]

Follow (A, B)	
esid	*Intv-List*
1	[1, 3] , [8, 9]
2	[1, 2] , [11, 15]
3	[4, 7] , [11, 12]

Contain (A, B)	
esid	*Intv-List*
1	[6, 12] , [8, 11]
4	[6, 14] , [7, 10]

A	
esid	*Intv-List*
1	[1, 3]
1	[7, 10]
2	[1, 2]
2	[10, 12]
3	[4, 7]
3	[9, 11]
4	[6, 14]

B	
esid	*Intv-List*
1	[1, 3]
1	[8, 9]
2	[2, 6]
2	[11, 15]
3	[1, 3]
3	[11, 12]
4	[1, 5]
4	[7, 10]

C	
esid	*Intv-List*
1	[9, 10]
2	[14, 17]
3	[12, 14]
4	[8, 9]

{B, A}

Follow (B, A)	
esid	*Intv-List*
1	[1, 3] , [6, 12]
2	[2, 6] , [10, 12]
3	[1, 3] , [3, 7]
3	[1, 3] , [9, 11]
4	[1, 5] , [6, 14]

{B, B}

Follow (B, B)	
esid	*Intv-List*
1	[1, 3] , [8, 11]
2	[2, 6] , [11, 15]
3	[1, 3] , [11, 12]
4	[1, 5] , [7, 10]

{B, C}

Contain (B, C)	
esid	*Intv-List*
1	[8, 11] , [9, 10]
4	[7, 10] , [8, 9]

Follow (B, C)	
esid	*Intv-List*
1	[1, 3] , [9, 10]
2	[2, 6] , [8, 9]
3	[1, 3] , [12, 14]
4	[1, 5] , [8, 9]

{A, C}

Contain (A, C)	
esid	*Intv-List*
1	[6, 12] , [9, 10]
4	[6, 14] , [8, 9]

Follow (A, C)	
esid	*Intv-List*
1	[1, 3] , [9, 10]
2	[1, 2] , [8, 9]
2	[10, 12] , [8, 9]
3	[4, 7] , [10, 12]
3	[9, 11] , [12, 14]

{A, B, C}

| Contain (A, B) * Contain (A, C) * Contain (B, C)		
esid		*Intv-List*
1		
4		

Figure 4·6: *The set of frequent 2 and 3-arrangements.*

Chapter 5

Experimental Evaluation

5.1 Experimental Setup

Experiments that compare the performance of our algorithms with SPAM (Ayres et al., 2002) are presented. [1] All experiments have been performed on a 2.8Ghz Intel(R) Pentium(R) 4 dual-processor machine with 2.5 gigabytes main memory, running Linux with kernel 2.4.20. The algorithms have been implemented in C++, compiled using g++ along with the -O3 flag, and their run time has been measured with the output turned off. Note that for SPAM, the post-processing time of converting the sequential patterns to arrangements has not been counted. Also, as mentioned earlier, SPAM is tuned as follows: for every event interval we keep only the start and end time; as for the postprocessing phase, the frequent arrangements are extracted from the sequential patterns as described in the same section. The patterns found by SPAM consist of a set of start and end points of event intervals, which are converted to arrangements at the postprocessing phase. SPAM manages to discover the same patterns extracted by our algorithms, but, as expected, it produces a great number of redundant patterns.

For our experimental evaluation we have used both real and synthetic datasets. Next, we present an analysis of our experimental evaluation, first by comparing our algorithms with respect to their run time and then by showing their performance for each of the interestingness measures described in Section 2.4.

---

[1] The code was obtained from: `http://himalaya-tools.sourceforge.net/Spam/`.

5.1.1 Experiments on Real Data

We have performed a series of experiments on two real datasets. One was an annotated database of ASL utterances, which is available online at: `http://www.bu.edu/asllrp/`. The other was a sample network dataset of ODFlows taken from Abilene, which is an Internet2 backbone network, connecting over 200 US universities and peering with research networks in Europe and Asia. Next we present a detailed description of our experiments on each dataset.

Experiments on the ASL SignStream Database

The first series of experiments have been performed on the ASL database created by the National Center for Sign Language and Gesture Resources at Boston University. The Sign-Stream(TM) database used in this experiment consists of a collection of 884 utterances, where each utterance associates a segment of video with a detailed transcription. Non-manual markings play a crucial role in the grammar of ASL (Baker-Shenk, 1983; Coulter, 1979; Liddell, 1980; Neidle et al., 2000), thus for our experiments we focused only on: specific non-manual gestures (e.g., raised eyebrows, head tilt forward), functional identification of clusters of these non-manual gestures that carry syntactic meaning (e.g., 'wh-question', 'negation'), and part-of-speech identifications of manual signs (e.g., verb, wh-word), each one occurring over a time interval. The overall list of field names and labels included in the database are given in Table 5.1.

We first tested our algorithms on subsets of sentences from the database: those that contained marking of a wh-question, and another that contained marking of negation. Our goal was to detect all frequent arrangements that occurred during wh-questions and negative sentences. In these two datasets, called *Dataset* 1 and *Dataset* 2 respectively, the number of e-sequences was 73 and 68, with an average number of items per sequence equal to 32 and 26 respectively. Since all four algorithms produce the same results, in our experiments we compare their run time. As shown in Figures 5·2(a) and 5·2(b), Hybrid

Dataset 1				Dataset 2				Dataset 3			
Pattern	Support	Pattern	Support	Pattern	Support	Pattern	Support	Pattern	Support	Pattern	Support
eye-aperture squint / wh-word	49%	wh-word / eye-aperture blink	52%	lowered eye-brows / verb	58%	verb / eye-aperture blink	57%	lowered eye-brows / verb	21%	verb / eye-aperture blink	57%
wh-word / lowered eye-brows	43%	lowered eye-brows / wh-word	55%	Head tilt: side / verb	46%	lowered eye-brows / negation	72%	eye-aperture wide / verb	24%	lowered eye-brows / wh-word / wh-question	28%
eye-aperture squint / wh-word / lowered eye-brows	32%	wh-word / lowered eye-brows / eye-aperture blink	37%	eye-aperture squint / lowered eye-brows / negation	43%	Head tilt: side / lowered eye-brows / negation	63%	eye-aperture squint / lowered eye-brows / negation	22%	wh-word / eye-aperture blink / lowered eye-brows / wh-question	23%

Figure 5·1: *Some Frequent Patterns of Datasets* 1, 2 *and* 3.

DFS outperformed both BFS and SPAM for supports less than 30%. On average, Hybrid DFS was approximately twice as fast as BFS and almost three times faster than SPAM. In many cases, the performance of the prefix-growth approach was very poor, as it was predicted in the analysis of Section 4.5. We tested the qualitative performance with respect to the setting of parameter ϵ, where varied between 2 and 6 instants. In our case, low values of ϵ gave the most meaningful results, since the amount of noise in the ASL dataset was limited to a small number of frames. For higher values of ϵ, the detected relation types changed, and as a result the number of extracted frequent arrangements decreased, hiding many of the interesting patterns. For all experiments presented in this section, $\epsilon = 3$.

Next, our algorithms were tested on the whole Signstream(TM) database that contained 884 utterances with an average e-sequence size of 29 items per e-sequence. We refer to this as *Dataset* 3. The algorithms have been tested for various supports and have been compared in terms of run time. The experimental results in Figure 5·2(c) show that in terms of run time, the Hybrid DFS-based approach outperforms the BFS-based especially in small supports. In both cases SPAM starts with a run time between that of BFS and Hybrid DFS and for small supports the run time increases dramatically. The prefix-growth approach is again very poor.

The results produced by our algorithms have been examined and evaluated by linguists who had been involved in collecting the ASL data and producing the annotations for this dataset [2]. According to their feedback, our algorithms managed to detect a set of

---

[2]Carol Neidle and Robert G. Lee, of Boston University.

Table 5.1: List of Field Names and Labels

Fields	Field Names	Field Labels
Head position	head pos: tilt fr/bk	hp: tilt fr/bk s
	head pos: turn	hp: turn
	head pos: tilt side	hp: tilt side
	head pos: jut	hp: jut
Head movement	head mvmt: nod	hm: nod
	head mvmt: shake	hm: shake
	head mvmt: side to side	hm: side$<->$side
	head mvmt: jut	hm: jut
Body	body lean	body lean
	body mvmt	body mvmt
	shoulders	shoulders
Eyes, Nose, and Mouth	eye brows	eye brows
	eye gaze	eye gaze
	eye aperture	eye apert
	nose	nose
	mouth	mouth
	English mouthing	English mouthing
	cheeks	cheeks
Neck	neck	neck
Grammatical information	negative	negative
	wh question	wh question
	yes-no question	yes-no question
	rhetorical question	rhq
	topic/focus	topic/focus
	conditional/when	cond/when
	relative clause	rel. clause
	role shift	role shift
	subject agreement	subj agr
	object agreement	obj agr
	adverbial	adv
Part of Speech	POS	POS
	Non-dominant POS	POS2
Gloss Fields	main gloss	main gloss
	non-dominant hand gloss	nd hand gloss
Text Fields	English translation	english

ASL patterns that have been already known: for example, the strong correlation between wh-question marking and lowered eyebrows. Similar correlations were found between the occurrence of what had been labelled as "negative" marking and the non-manual behaviors that comprise this marking (such as side-to-side head shake). Similarly, it was unsurprising that wh-words co-occurred with the non-manual markings associated with wh-questions. Nonetheless, it is good that our approaches independently found known correlations. Also, some other discovered patterns were considered to be trivial (e.g., that the onset of a behavior preceded the behavior itself) or in some cases, an artifact of the selection criterion used to define the sample of data under consideration. For example, within the set of negative sentences, verbs frequently co-occurred with marking of negation (whereas this would not be true of verbs in non-negative sentences). For example, a "verb" would always occur in a "negation", or a "wh-word" is always included in a "wh-question". Unfortunately, due to the limited size of the dataset, we did not manage to find any patterns not already known to the linguists; however, the above evaluation demonstrated the correctness of our algorithms. In Figure 5·1 we can see some of the most frequent arrangements detected in *Datasets* 1, 2, *and* 3.

Applying Interestingness Measures on the ASL Datasets

The main motivation behind the application of interestingness measures during the mining process was to reduce the number of extracted patterns to the most interesting ones (for the user), removing most of the trivial cases described previously. All six interestingness measures presented in Section 4.6 have been applied to the ASL Datasets, leading to the discovery of different sets of rules that maximize each one of them. The basic observation from the extracted patterns on the ASL database was that applying only the support threshold yielded a huge number of redundant patterns that do not provide any useful information. Therefore, except for the support we applied all the interestingness measures described previously. Table 5.2 shows the number of rules extracted by our algorithms for each of the three ASL Datasets. As we can see, lift, confidence and conviction produced

Table 5.2: Number of Extracted Rules from Datasets 1, 2, 3, and 4

λ	Support	λ	d/set 1	d/set 2	d/set 3	d/set 4
Lift	0.5	0.5	46	14	2	87
Lift	0.4	0.5	187	149	53	241
Lift	0.3	0.5	453	47	155	786
Conviction	0.5	0.5	29	47	13	68
Conviction	0.4	0.5	112	47	16	142
Confidence	0.5	0.5	46	14	2	81
Confidence	0.4	0.5	148	53	14	251
Leverage	0.4	0.3	6	1	6	25
Leverage	0.4	0.2	25	3	12	43
All-Confidence	0.5	0.5	46	15	15	84
All-Confidence	0.4	0.5	54	53	59	122
All-Confidence	0.3	0.5	67	56	64	143

an interesting number of rules, whereas the number of rules that maximize leverage and all-confidence was pretty small. In Figure 5·3 we present some of the top patterns in the ASL dataset with regard to each interestingness measure. Notice that the first column of the figure gives the complete arrangement that takes part in the rule, the second column shows the antecedent arrangement, the third column shows the consequent arrangement and the fourth column gives the value of each interestingness measure.

The application of interestingness measures managed to remove trivial cases and preserve those already known to the linguists. In our case *lift* and *conviction* had the best performance among all the measures we examined. *Leverage* also did a very good job in removing trivial rules, however it also removed a great number of known rules, and in many cases the number of extracted rules was extremely small. Our experimental evaluation showed that the combination of interestingness measures and the efficient arrangement mining algorithms can potentially provide more meaningful results. Nonetheless, an evaluation by experts is always needed to determine the most effective measure for a given application.

Experiments on Network Data

Our algorithms have also been tested on a network dataset of 960 e-sequences with an average e-sequence size of 100 items per e-sequence. The data has been obtained from

a collection of ODFlows obtained from Abilene, that consists of 11 Points of Presence (PoPs), spanning the continental US. Three weeks of sampled IP-level traffic flow data were collected from every PoP in Abilene for the period December 8, 2003 to December 28, 2003. We have selected two routers that were shown to have a high communication rate with each other, and have monitored the IP connections from one (LOSA: router in LA) to the other (ATLA: router in Altanta) for three days. An e-sequence in our dataset is the set of IP connections from LOSA to ATLA for every 15 minutes. Due to the huge number of IP addresses, we have selected 200 IPs that appear most frequently in these three days. The dataset that resulted from the above process is called *Dataset* 4.

Our experiments focused only on run time for the same reasons described earlier. In a qualitative experiment, the parameter ϵ was tuned to vary between 3 and 15. Due to the nature of the dataset, the number of extracted patterns was huge. As before, the number and type of extracted patterns varied with respect to ϵ; however, the most interesting ones were obtained for $\epsilon = 10$. In this case, "interestingness" means that the patterns were meaningful (from a network point of view), and described an expected communication behavior of the two routers. The run time comparison of the four algorithms is shown in Figure 5·2(d), and it is quite similar to that of the ASL datasets; again Hybrid DFS outperforms the other algorithms in low supports. In this case however, the number of extracted patterns is larger, due to the high averaage e-sequence size.

Applying Interestingness Measures on the Network Dataset

As far as the arrangement rules are concerned, all six interestingness measures have been applied and a great number of rules have been extracted, most of which were interesting (from the point of view described earlier). In Table 5.2 we can see the number of rules generated by each interestingness measure. The main observation here is that the number of rules generated for the network dataset by each measure is greater than that for the ASL datasets. This is expected due to the nature of the network dataset, i.e. the average e-sequence size is larger, and the number of relations (and thus arrangements) is greater,
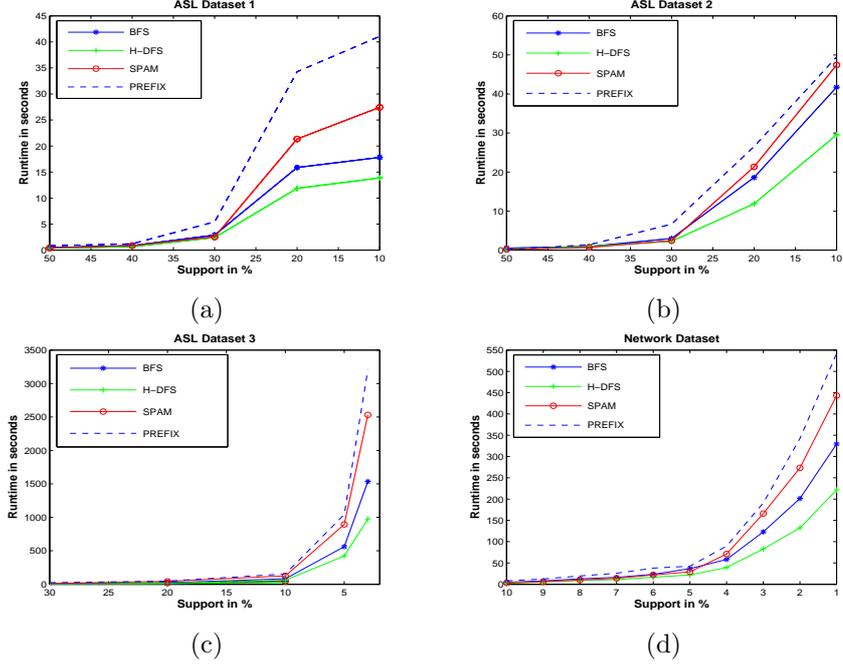
Figure 5·2: *Results on Real Datasets: (a) ASL Dataset* 1*: |S|: 73, |A|: 52, |E|: 400.; (b) ASL Dataset 2: |S|: 68, |A|: 26, |E|: 400.; (c) ASL Dataset 3: |S|: 884, |A|: 102, |E|: 400.; (d) Network Dataset: |S|: 960, |A|: 100, |E|: 200 (where |S| denotes the size of the dataset, |A| the average sequence size), and |E| the number of distinct items in the dataset.*

due to the high communication rate in the network.

5.1.2   Experiments on Synthetic Data

Due to the relatively small size of the current SignStream database, we have generated numerous synthetic datasets to test the efficiency of our algorithms.

Synthetic Data Generation

The following factors have been considered for the generation of the synthetic datasets: (1) number of e-sequences, (2) average e-sequence size, (3) number of distinct items, (4) density of frequent patterns. Using different variations of the above factors we have generated several datasets. In particular, our datasets were of sizes 200, 500, 1000, 2000, 5000 and 10000, with average sequence sizes of 3, 10, 50, 100 and 150 items per e-sequence.

Arrangement	Antecedent	Consequent	Lambda
wh-word eye-brow ONSET · eye-brow OFFSET	eye-brow ONSET	wh-word eye-brow OFFSET	**Lift: 0.97** **Leverage: 0.68** **Confidence: 0.92** **Conviction: 9.12** **All-Confidence: 0.54**
wh-word head mvmt: rapid · eye-aperture head shake · blink	wh-word head mvmt: rapid head shake	eye-aperture blink	**Lift: 0.72** **Leverage: 0.52** **Confidence: 0.85** **Conviction: 3.03** **All-Confidence: 0.51**
wh-word lowered eye-brows · eye-aperture blink wh-question	wh-word lowered eye-brows	eye-aperture blink wh-question	**Lift: 0.93** **Leverage: 0.51** **Confidence: 0.81** **Conviction: 3.64** **All-Confidence: 0.58**
negative neg · POS: verb head mvmt: rapid head shake negation	negative neg head mvmt: rapid head shake	POS: verb negation	**Lift: 0.79** **Leverage: 0.33** **Confidence: 0.96** **Conviction: 0.77** **All-Confidence: 0.51**
negative neg lowered eye-brows negation	negative neg raised eye-brows	negation	**Lift: 0.76** **Leverage: 0.45** **Confidence: 0.63** **Conviction: 3.88** **All-Confidence: 0.47**
wh-word eye-aperture squint · eye-aperture blink	wh-word eye-aperture squint	eye-aperture blink	**Lift: 0.96** **Leverage: 0.24** **Confidence: 0.95** **Conviction: 1.59** **All-Confidence: 0.53**
lowered eye-brows wh-word Body lean forward	lowered eye-brows wh-word	Body lean forward	**Lift: 1.0** **Leverage: 0.21** **Confidence: 0.33** **Conviction: 1.2** **All-Confidence: 0.21**
POS: noun negation	negation	POS: noun negation	**Lift: 0.88** **Leverage: 0.35** **Confidence: 0.82** **Conviction: 1.55** **All-Confidence: 0.44**
wh-word lowered eye-brows head mvmt: jut forward wh-question	wh-word lowered eye-brows wh-question	head mvmt: jut forward	**Lift: 0.41** **Leverage: 0.15** **Confidence: 0.82** **Conviction: 0.78** **All-Confidence: 0.46**

Figure 5·3: *Some of the discovered rules in Dataset 1 and Dataset 2.*

Moreover, we have tried various numbers of distinct items, i.e. 400, 600 and 800. Also, we have considered different densities of frequent patterns. We first created a certain number of frequent patterns that with medium support thresholds of 20% (sparse), 40% (medium density) and 60% (dense) would generate a lot of frequent patterns and then added random event intervals on the generated sequences.

Experimental Results

The experimental results have shown that Hybrid DFS clearly outperforms BFS, and especially in low support values and large database sizes Hybrid DFS is twice as fast as BFS. Regarding the performance of SPAM, we have concluded that in medium support values and small database sizes SPAM performs better than BFS but worse than Hybrid DFS, whereas in small support values and large datasets BFS outperforms SPAM. We compared the four algorithms on several small, medium and large datasets for various support values. The results of these tests are shown in Figure 5·4. As expected, SPAM performs poorly in large sequences and small supports. This behavior is expected since for every arrangement produced by BFS and Hybrid DFS, SPAM generates all the possible subsets of the start and end points of the events in that arrangement. As the database size grows along with the average e-sequence size, SPAM will be producing a great number of redundant frequent patterns that yield to a rapid increase of its run time. In all cases, the prefix-growth algorithm performs very poorly. In medium supports and small datasets it can still do better than SPAM, but in smaller supports and larger datasets its performance decreases dramatically.
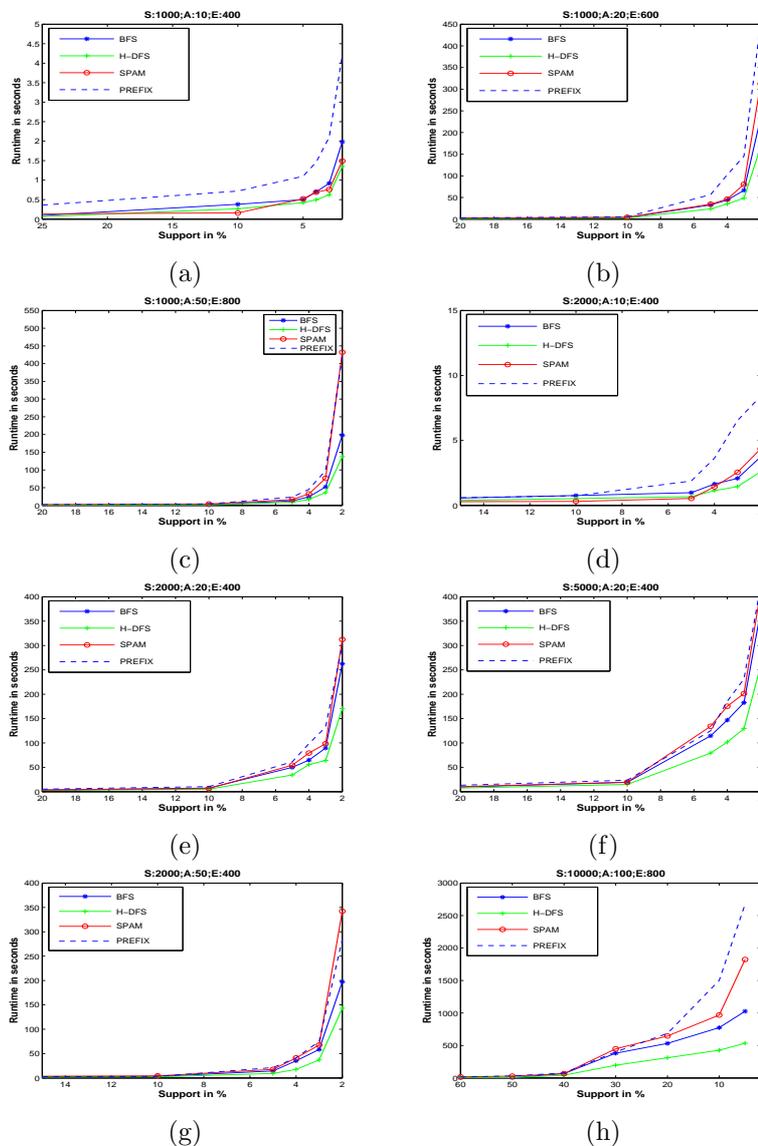
Figure 5·4: *Results on Synthetic Datasets: (a) Dataset 1: $|S|$: 1000, $|A|$: 10, $|\mathcal{E}|$: 400, frequent patterns of medium density.; (b) Dataset 2: $|S|$: 1000, $|A|$: 20, $|\mathcal{E}|$: 600, sparse frequent patterns.; (c) Dataset 3: $|S|$: 1000, $|A|$: 50, $|\mathcal{E}|$: 800, dense frequent patterns.;(d) Dataset 4: $|S|$: 2000, $|A|$: 10, $|\mathcal{E}|$: 400, frequent patterns of medium density.;(e) Dataset 5: $|S|$: 2000, $|A|$: 20, $|\mathcal{E}|$: 400, frequent patterns of medium density.;(f) Dataset 6: $|S|$: 5000, $|A|$: 20, $|\mathcal{E}|$: 400, dense frequent patterns.;(g) Dataset 7: $|S|$: 5000, $|A|$: 50, $|\mathcal{E}|$: 400, dense frequent patterns.;(h) Dataset 8: $|S|$: 10000, $|A|$: 100, $|\mathcal{E}|$: 800, dense frequent patterns.*

Chapter 6

Conclusions

We have formally defined the problem of constraint-based mining of frequent temporal arrangements of event interval sequences and presented three efficient methods to solve it. The first two approaches use an arrangement enumeration tree to discover the set of frequent arrangements. The DFS-based method further improves performance over BFS by reaching longer arrangements faster and hence eliminating the need for examining smaller subsets of these arrangements. The prefix-growth approach is poor in performance, since the number of projections can be really huge, especially when the input e-sequences have repetitions of the same event label. We further extended our algorithms by pushing constraints into the mining process. These constraints provide a more user-specified focus on the extracted patterns. Moreover, except for the support threshold, we have applied other interestingness measures and focused on mining the top k arrangement rules that maximize a given interestingness measure. Our experimental evaluation demonstrates the applicability and usefulness of our methods.

An interesting direction for future work is to develop an efficient algorithm for mining closed arrangements. In this case however, a prefix-based approach, like BIDE (Wang and Han, 2004), would be extremely costly. Therefore, we should come up with a method to produce the complete set of closed arrangements that will employ more efficient projections or use different techniques to prevent the paramount cost of multiple projections described previously. Furthermore, another direction for future work is to mine partial orders of temporal arrangements and closed temporal arrangements. The notion of mining partial

orders of sequential patterns has been introduced in (Mannila and Toivonen, 1996) and an interesting approach has been recently proposed for closed sequential patterns in (Casas-Garriga, 2005). However, these methods again assume that the events are instantaneous. Last but not least, our algorithms could be applied on biological data, such as genes of different organisms (Papapetrou et al., 2006). The ultimate goal would be to extract frequent arrangements of nucleotide regions and produce interesting rules. These patterns could be further used to determine various features of different groups of organisms and possibly detect mutations or tandem repeats.

Another extension is to consider e-sequences that include categorical domains. Suppose we have a set of event intervals that correspond to a certain treatment for a disease and also a categorical attribute that describes the result of the treatment, i.e. whether the patient was cured or not. Being able to mine arrangements in such e-sequences could give valuable information for the proper medical treatment of patients.

References

Abraham, T. and Roddick, J. F. (1999). Incremental meta-mining from large temporal data sets. In *Proceedings of the Workshop on Data Warehousing and Data Mining (DaWak)*, pages 41–54.

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules. In *Proc. of International Conference on Very Large Databases (VLDB)*, pages 487–499.

Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In *Proc. of IEEE International Conference on Data Engineering (ICDE)*, pages 3–14.

Ale, J. M. and Rossi, G. H. (2000). An approach to discovering temporal association rules. In *Proc. of Annual ACM Symposium on Applied Computing (SAC)*, pages 294–300.

Allen, J. and Ferguson, G. (1994). Actions and events in interval temporal logic. Technical Report 521, The University of Rochester.

Ayres, J., Gehrke, J., Yiu, T., and Flannick, J. (2002). Sequential pattern mining using a bitmap representation. In *Proc. of ACM Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 429–435.

Baker-Shenk, C. (1983). A micro-analysis of the nonmanual components of questions in American Sign Language. *Doctoral Dissertation*.

Bayardo, R., Agrawal, R., and Gunopulos, D. (1999). Constraint-based rule mining in large, dense databases. In *Proc. of IEEE International Conference on Data Engineering (ICDE)*, pages 188–197.

Bayardo, R. J. (1998). Efficiently mining long patterns from databases. In *Proc. of ACM International Conference on Management of Data (SIGMOD)*, pages 85–93.

Casas-Garriga, G. (2005). Summarizing sequential data with closed partial orders. In *Proc. of SIAM Data Mining (SDM)*, pages 380–391.

Chen, X. and Petrounias, I. (1999). Mining temporal features in association rules. In *Proc. of European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 295–300, London, UK. Springer-Verlag.

Coulter, G. R. (1979). American Sign Language typology. *Doctoral Dissertation.*

Davey, B. and Priestley, H. (2002). *Introduction to Lattices and Oder.* Cambridge University Press.

E.Winarko and J.F.Roddick (2005). Discovering richer temporal association rules from interval-based data. In *In Proc. of Data Warehousing and Knowledge Discovery (DaWaK)*, pages 315–325.

Fei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., and Hsu, M.-C. (2001). Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proc. of IEEE International Conference on Data Engineering (ICDE)*, pages 215–224.

Freksa, C. (1992). Temporal reasoning based on semi-intervals. *Artificial Intelligence*, 54(1):199–227.

Garofalakis, M., Rastogi, R., and Shim, K. (1999). Spirit: Sequential pattern mining with regular expression constraints. In *Proc. of International Conference on Very Large Databases (VLDB)*, pages 223–234.

Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., and Hsu, M. (2000a). Freespan: Frequent pattern-projected sequential pattern mining. In *Proc. of ACM Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 355–359.

Han, J., Pei, J., and Yin, Y. (2000b). Mining frequent patterns without candidate generation. In *Proc. of ACM International Conference on Management of Data (SIGMOD)*, pages 1–12.

Harms, S., Deogun, J., and Tadesse, T. (2002). Discovering sequential association rules with constraints and time lags in multiple sequences. In *International Symposium on Methodologies for Intelligent Systems (ISMIS)*, pages 432–442.

Hilderman, R. and Hamilton, H. (1999). Knowledge discovery and interestingness measures: A survey. Technical Report 99-04, Department of Computer Science, University of Regina.

Hilderman, R. J. and Hamilton, H. J. (2001). Evaluation of interestingness measures for ranking discovered knowledge. *Lecture Notes in Computer Science*, 2035:247–258.

Hoeppner, F. (2001). Discovery of temporal patterns - learning rules about the qualitative behaviour of time series. In *Proc. of European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 192–203.

Hoeppner, F. and Klawonn, F. (2001). Finding informative rules in interval sequences. In *Advances in Intelligent Data Analysis, Proc. of the 4th International Symposium*, pages 123–132.

Kam, P. and Fu, A. W. (2000). Discovering temporal patterns for interval-based events. In *DaWaK*, pages 317–326.

Kamber, M. and Shinghal, R. (1996). Evaluating the interestingness of characteristic rules. In *Proc. of ACM Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 263–266.

Leleu, M., Rigotti, C., Boulicaut, J., and Euvrard, G. (2003). Go-spade: Mining sequential patterns over databases with consecutive repetitions. In *Proc. of International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM)*, pages 293–306.

Liddell, S. K. (1980). American Sign Language syntax. *The Hague: Mouton.*

Lin, J.-L. (2002). Mining maximal frequent intervals. Technical report, Department of Information Management, Yuan Ze University.

Lin, J.-L. (2003). Mining maximal frequent intervals. In *Proc. of Annual ACM Symposium on Applied Computing (SAC)*, pages 624–629.

Lu, H., Han, J., and Feng, L. (1998). Stock movement prediction and n-dimensional inter-transaction association rules. In *Proc. of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 1–7.

Mannila, H. and Toivonen, H. (1996). Discovering generalized episodes using minimal occurences. In *Proc. of ACM Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 146–151.

Mannila, H., Toivonen, H., and Verkamo, A. (1995). Discovering frequent episodes in sequences. In *Proc. of ACM Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 210–215.

Moerchen, F. (2006). Algorithms for time series knowledge mining. In *Proc. of ACM Conference on Knowledge Discovery and Data Mining (SIGKDD)*.

Mooney, C. and Roddick, J. F. (2004). Mining relationships between interacting episodes. In *Proc. of SIAM Data Mining (SDM)*.

Neidle, C. (2002a). SignStream: A database tool for research on visual-gestural language. *Journal of Sign Language and Linguistics*, 4:203–214.

Neidle, C. (2002b). Signstream annotation: Conventions used for the American Sign Language linguistic research project. *American Sign Language Linguistic Research Project Report*, 11.

Neidle, C. (2003). Language across modalities: ASL focus and question constructions. *Linguistic Variation Yearbook*, 2:71–93.

Neidle, C., Kegl, J., MacLaughlin, D., Dawn, B., and Lee, R. G. (2000). The syntax of American Sign Language: Functional categories and hierarchical structure.

Neidle, C. and Lee, R. G. (2006). Syntactic agreement across language modalities. *Studies on Agreement*.

Neidle, C., Sclaroff, S., and Athitsos, V. (2001). SignStream: A tool for linguistic and computer vision research on visual-gestural language data. *Behavior Research Methods, Instruments, and Computers*, 33:311–320.

Oezden, B., Ramaswamy, S., and Silberschatz, A. (1998). Cyclic association rules. In *Proc. of IEEE International Conference on Data Engineering (ICDE)*, pages 412–421.

Omiecinski, E. R. (2003). Alternative interest measures for mining associations in databases. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(1):39–79.

Papapetrou, P., Benson, G., and Kollios, G. (2006). Discovering frequent polyregions in dna sequences. In *Proc. of the IEEE ICDM Workshop on Data Mining in Bioinformatics (IWI-DMB) [To Appear]*.

Papapetrou, P., Kollios, G., Sclaroff, S., and Gunopulos, D. (2005). Discovering frequent arrangements of temporal intervals. In *Proc. of IEEE International Conference on Data Mining (ICDM)*, pages 354–361.

Pasquier, N., Bastide, Y., Taouil, R., and Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. In *Proc. of International Conference on Database Theory (ICDT)*, pages 398–416.

Pei, J., Han, J., and Mao, R. (2000). Closet: An efficient algorithm for mining frequent closed itemsets. In *Proc. of ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*, pages 11–20.

Pei, J., Han, J., and Wang, W. (2002). Constraint-based sequential pattern mining in large databases. In *Proc. of ACM Conference on Information and Knowledge Management (CIKM)*, pages 18–25.

Seno, M. and Karypis, G. (2002). SLPMiner: An algorithm for finding frequent sequential patterns using length-decreasing support constraint. In *Proc. of IEEE International Conference on Data Mining (ICDM)*, pages 418–425.

Srikant, R. and Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In *Proc. of International Conference on Extending Database Technology (EDBT)*, pages 3–17.

Tan, P. and Kumar, V. (2000). Interestingness measures for association patterns: A perspective. Technical Report TR00-036, Department of Computer Science, University of Minnesota.

Tan, P., Kumar, V., and Srivastava, J. (2002). Selecting the right interestingness measure for association patterns. In *Proc. of ACM Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 183–192.

Tsoukatos, I. and Gunopulos, D. (2001). Efficient mining of spatiotemporal patterns. In *Proc. of the International Symposium on Spatial and Temporal Databases (SSTD)*, pages 425–442.

Villafane, R., Hua, K. A., Tran, D., and Maulik, B. (2000). Knowledge discovery from series of interval events. *Intelligent Information Systems*, 15(1):71–89.

Wang, J. and Han, J. (2004). Bide: Efficient mining of frequent closed sequences. In *Proc. of IEEE International Conference on Data Engineering (ICDE)*, pages 79–90.

Webb, G. I. (2006). Discovering significant rules. In *Proc. of ACM Conference on Knowledge Discovery and Data Mining (SIGKDD)*.

Webb, G. I. and Zhang, S. (2005). K-optimal rule discovery. *Data Min. Knowl. Discov.*, 10(1):39–79.

Xin, D., Shen, X., Mei, Q., and Han, J. (2006). Discovering interesting patterns through user's interactive feedback. In *Proc. of ACM Conference on Knowledge Discovery and Data Mining (SIGKDD)*.

Yan, X., Han, J., and Afshar, R. (2003). Clospan: Mining closed sequential patterns in large databases. In *Proc. of SIAM Data Mining (SDM)*.

Zaki, M. (2001). Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 40:31–60.

Zaki, M. and Hsiao, C. (2002). Charm: An efficient algorithm for closed itemset mining. In *Proc. of International Conference on Data Mining (SIAM)*, pages 457–473.

Zaki, M. J. (2000). Sequence mining in categorical domains: Incorporating constraints. In *Proc. of ACM Conference on Information and Knowledge Management (CIKM)*, pages 422–429.