Chapter 1

Introduction

Since its inception by Agrawal and Srikant in 1995 the field of Sequence Mining has grown both in algorithmic maturity and in the breadth of application areas under consideration. With the amount of available data increasing at an exponential rate this trend, especially algorithmic development, must continue and indeed be enhanced with better methods for extracting and interpreting rules and displaying them in meaningful ways. In the area of sequential pattern mining, as is the case for all areas of data mining, what is of interest is the use of the discovered sequences to formulate rules that can be used for describing a phenomenon, or predicting future events and with the advancements in this area there is an increased need to attach more meaning to the rules that are generated. As a consequence, rules based on the temporal logic of Allen (1983) and extensions of this algebra by Freksa (1992), Badaloni and Giacomin (1999, 2002, 2006) and also Ohlbach (2004*b*) have now been considered by researchers.

This thesis is structured in the following manner. Chapter 2 comprises a survey of the area of sequence mining and is concluded with a discussion on rule inference. This is followed, in Chapter 3, by a survey of those temporal logics and extensions that are relevant to any rule determinations in sequence mining. Having laid this foundation, Chapter 4 introduces the formal concept of the Midpoint Interval (MI) algebra, both equal-length (ELMI) and variable-length (VLMI), sequences developed in this thesis. Chapter 5 presents a method for discovering interacting episodes from temporal sequences and analyses them using the temporal patterns presented in the earlier chapters. As an extension to this process Chapter 6 discusses relevant timing considerations and proposes a solution. All algorithms pertaining to both Chapter 5 and Chapter 6 are included in Appendix C. Chapter 7 investigates the use of transitivity tables for further reasoning between two or more sets of events thereby enabling more complex sets of rules to be elaborated. All transitivity tables that are referred to in this thesis are included in Appendix A. The types of rules generated by methods presented in this thesis can become quite complex, and to date, to the authors knowledge, have only been presented in a textual format. In an attempt to address this situation and present the rules in a manner that conveys the semantics in a more meaningful way, a visualization to perform this task has been developed and is included in Appendix B.

The areas of contribution made by this thesis and the areas of direct and indirect influence on this thesis are depicted in Figure 1.1.

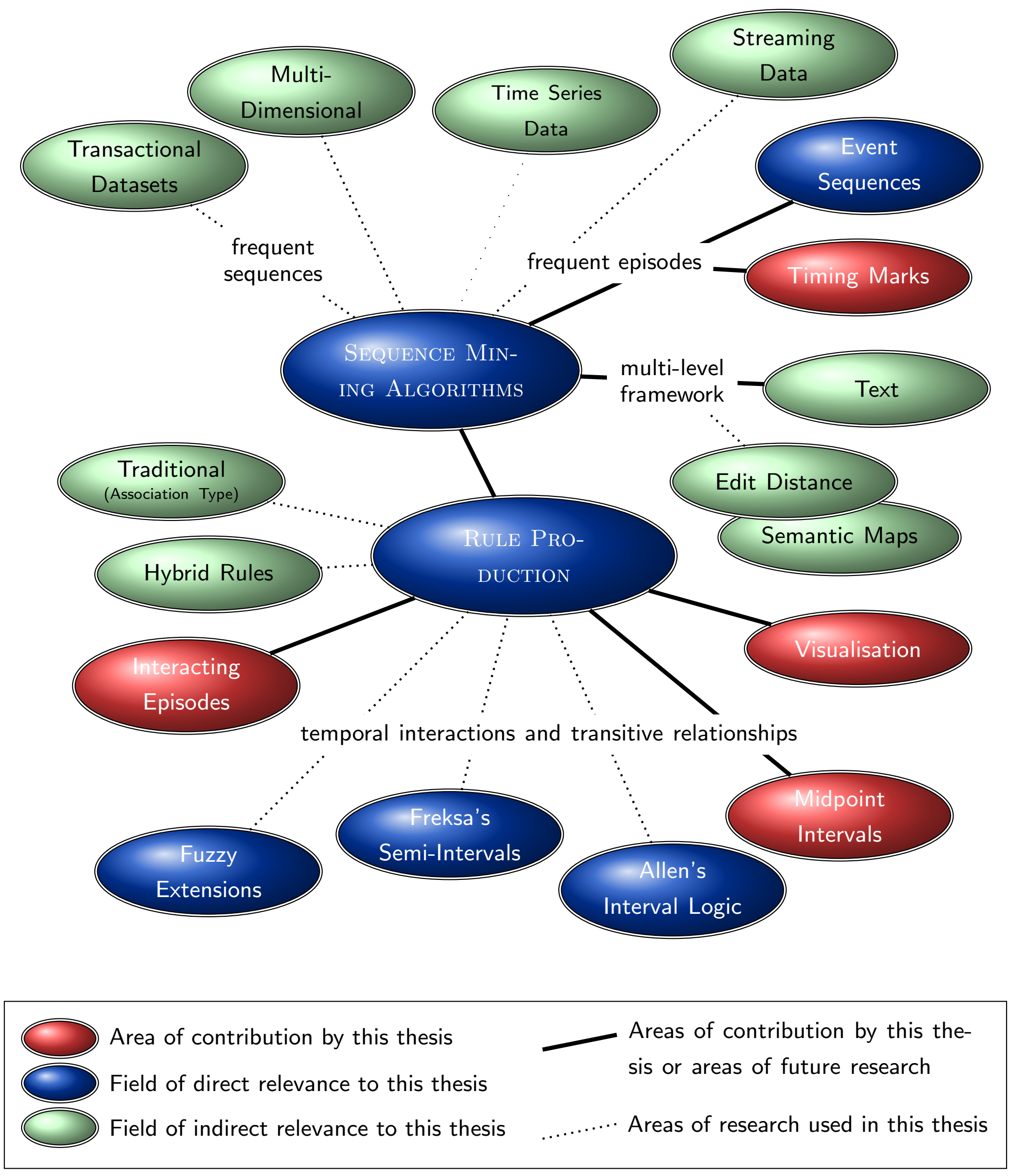

Streaming Data
Multi-Dimensional
Time Series Data
Transactional Datasets
Event Sequences
frequent sequences
frequent episodes
Timing Marks
Sequence Mining Algorithms
multi-level framework
Text
Traditional (Association Type)
Edit Distance
Semantic Maps
Rule Production
Hybrid Rules
Visualisation
Interacting Episodes
temporal interactions and transitive relationships
Midpoint Intervals
Freksa's Semi-Intervals
Fuzzy Extensions
Allen's Interval Logic
Area of contribution by this thesis
Field of direct relevance to this thesis
Field of indirect relevance to this thesis
Areas of contribution by this thesis or areas of future research
Areas of research used in this thesis

Figure 1.1: Structural Domain of this Thesis.

The following publications are attributed to the research conducted as part of this thesis or material presented within this thesis. However, where the latter is the case the material has been re-examined and revised as necessary. Publication 1 relates directly to Chapter 5 and has been revised and updated before inclusion. Publication 3 is a result of research carried out for the proposed MI algebra presented in Chapter 4 and Publication 4 is the software that supports the research. Chapter 6 dealing with timing is a revised version of Publication 5.

During the candidature the methodology was tested for use with text as the input sequence and this required modifications to the software. These modifications resulted in a framework for the analysis of text, as a sequence mining problem, and also Publication 2. This collaborative research, although related, is somewhat orthogonal to the main thrust of the research carried out as part of this thesis, and therefore discussion relating to this framework appears in Chapter 8 – Conclusions and Future Research.

1. Mooney, C. H. and Roddick, J. F. (2004), Mining relationships between interacting episodes, *in* M. W. Berry, U. Dayal, C. Kamath and D. Skillicorn, eds, '4th SIAM International Conference on Data Mining', SIAM, Lake Buena Vista, Florida.

2. Mooney, C. H., de Vries, D. and Roddick, J. F. (2004), A multi-level framework for the analysis of sequential data, *in* S. J. Simoff and G. J. Williams, eds, 'Australasian Data Mining Conference (AusDM'04)', Lecture Notes in Computer Science, Springer, Cairns, Qld, Australia, pp. 199–213.

3. Roddick, J. F. and Mooney, C. H. (2005), 'Linear temporal sequences and their interpretation using midpoint relationships', *IEEE Transactions on Knowledge and Data Engineering* 17(1), 133–135.

4. Mooney, C. H. (2005), Temporal mid-point demonstration applet, Technical Artefact SIE-05-004, School of Informatics and Engineering, Flinders University. http://www.infoeng.flinders.edu.au/research/techreps/SIE05004.zip.

5. Mooney, C. H. and Roddick, J. F. (2006), Marking time in sequence mining, *in* P. Christen, P. Kennedy, J. Li, S. Simoff and G. Williams, eds, 'Australasian Data Mining Conference (AusDM 2006)', Vol. 61, Conferences in Research and Practice in Information Technology (CRPIT), Sydney, Australia.

Chapter 2

Sequential Pattern Mining

The discovery and use of associations is a primary focus of data mining and has been the driving force for the development of algorithms that can both discover the associations and allow inferences to be made, by the miner or a suitable domain expert, from the discovered associations. This problem was first addressed by Agrawal, Imielinski and Swami (1993) and by Agrawal and Srikant (1994) who introduced the Apriori algorithm which was based on the *downward closure principle*: *if a length k pattern is not frequent in the database, then any length (k+1) super-pattern can never be frequent*, for discovery of such associations. Since that time there has been considerable research conducted in this field in the area of algorithmic development in an attempt to make the process more efficient and thus more tractable when applied to the ever-increasing size of available datasets. For a comprehensive survey on Association Mining see Ceglar and Roddick (2006). This general area is concerned with *intra-transaction* associations (associations between items in the same transaction) and imposes no ordering on the items that constitute a transaction[1]. However *inter-transaction* associations (associations between different transactions) can also provide further useful knowledge with respect to the data being mined. For example it may be useful for a video rental outlet to know that customers who hired "The Fellowship of the Ring" and "The Two Towers" in one transaction also hired "Return of the King" in a later transaction. Moreover, these temporal relationships might be used to predict future values in the data. The discovery of these types of associations is generally called Sequential Pattern Mining.

This chapter surveys the field of sequential pattern mining and discusses the algorithms that have been proposed to deal with the problem.

[1]Typically, in order to simplify the mining process, and with no loss of information, the items that make up a transaction are sorted lexicographically.

2.1 The Sequential Pattern Mining Problem

The sequential pattern mining problem was first addressed by Agrawal and Srikant (1995) and was defined to be:

> *"Given a database of sequences, where each sequence consists of a list of transactions ordered by transaction time and each transaction is a set of items, sequential pattern mining is to discover all sequential patterns with a user-specified minimum support, where the support of a pattern is the number of data-sequences that contain the pattern."*

Since this time there has been a growing number of researchers in this field and the problem definition has been formulated in a number of ways. For example: Garofalakis, Rastogi and Shim (1999) described it as

> *"Given a set of data sequences, the problem is to discover sub-sequences that are frequent, i.e. the percentage of data sequences containing them exceeds a user-specified minimum support"*,

while Masseglia, Poncelet and Teisseire (2000) describe it as

> *"... the discovery of temporal relations between facts embedded in a database"*,

and Zaki (2001*b*) as

> *"... to discover a set of attributes, shared across time among a large number of objects in a given database."*

Since there are varied forms of dataset (transactional, streams, time series etc.) algorithmic development in this area has been focused on the development and improvement for (in the main) specific domain data which has included medical, telecommunications (networks), defence, web, retailing (market-basket), and identification of plan failures. In the majority of cases the data has been stored as transactional datasets and similar techniques such as those used by association rule miners have been employed in the discovery process. However, the data used for sequence mining is not limited to data stored in overtly temporal or longitudinally maintained datasets – examples include genome searching, web logs, alarm data in telecommunications networks, population health data, etc. In such domains data can be viewed as a series of events, or episodes, occurring at specific times and therefore the problem becomes a search for collections of events that occur frequently together. Mannila, Toivonen and Verkamo (1997) described the problem as

> *"Sequences of events describing the behavior and actions of users or systems can be collected in several domains. We consider the problem of discovering frequently occurring episodes in such sequences. ... Once such episodes are known, one can produce rules for describing or predicting the behavior of the sequence."*

These types of dataset require different types of algorithms and will be described separately from those algorithms that are based on the more general transaction oriented datasets. Regardless of the format of the dataset, sequence mining algorithms can be categorized into one of three broad classes that perform the task (Pei, Han and Wang, 2002):

1. Apriori-based
   (a) horizontal database format
   (b) vertical database format
2. Projection-based
   (a) pattern growth

Improvements in algorithms and algorithmic development in general, has followed similar developments in the related field of association rule mining and have been motivated by the need to process more data at an increased speed with lower overheads.

The remainder of this chapter will begin with a discussion on the types of constraints used in sequence mining and counting techniques that are used as measures against user designated supports followed by a survey of the algorithms that are or have been used in sequence mining. This survey will be based firstly on the type of dataset that each algorithm is associated with and secondly, within that designation, on the broad class to which they belong. This will be followed by discussions on extensions dealing with closed, approximate and parallel algorithms and the area of incremental algorithms after which there will be a discussion of those algorithms that deal with streaming data. The survey will be completed with a discussion on other methods that have been employed and areas of related research.

2.2 Types of Constraints

Constraints to guide the mining process have been employed by many algorithms, not only in sequence mining but also in association mining (Fu and Han, 1995; Chakrabarti, Sarawagi and Dom, 1998; Bayardo and Agrawal, 1999; Ceglar, Roddick and Calder, 2003), but in general constraints that are imposed on sequential pattern mining, regardless of the algorithms employed, can be categorized into one of seven main types[2]

[2]This is not a complete list but categorises those that are of most interest and are currently in use.

(Pei et al., 2002). Other types of constraints will be elaborated as they arise, but in general a constraint C on a sequential pattern α is a boolean function of the form $C(\alpha)$. For the constraints listed below only the *duration* and *gap* constraints rely on a support threshold (*min_sup*) to confine the mined patterns whereas for the others, whether or not a given pattern satisfies a constraint can be determined by the pattern itself. These constraints rely on certain notions that are elaborated below.

Let $I = \{x_1, \ldots, x_n\}$ be a set of **items**, each possibly being associated with a set of **attributes**, such as value, price, or period, etc. The value on attribute A of item x is denoted as $x.A$. An **itemset** is a non-empty subset of items, and an itemset with k items is called a k-**itemset**. A **sequence** $\alpha = \langle X_1 \ldots X_l \rangle$ is an ordered list of itemsets. An itemset X_i (1 ≤ i ≤ l) in a sequence is called a **transaction**. A transaction X_i may have a special attribute, *times-tamp*, denoted as $X_i.time$, which registers the time when the transaction was executed. The number of transactions in a sequence is called the **length** of the sequence. A sequence α with length l is called an l-**sequence**, denoted as len(α) = l. The i-th itemset is denoted as α[i]. A sequence $\alpha = \langle X_1 \ldots X_n \rangle$ is called a **subsequence** of another sequence $\beta = \langle Y_1 \ldots Y_m \rangle$ ($n \leq m$), and β a **super-sequence** of α, denoted as $\alpha \sqsubseteq \beta$, if there exist integers $1 \leq i_1 < \ldots < i_n \leq m$ such that $X_1 \subseteq Y_{i_1}, \ldots, X_n \subseteq Y_{i_n}$. A **sequence database** *SDB* is a set of 2-tuples (*sid*, α), where *sid* is a **sequence-id** and α a sequence. A tuple (*sid*, α) in a sequence database *SDB* is said to contain a sequence γ if γ is a subsequence of α. The number of tuples in a sequence database *SDB* containing sequence γ is called the support of γ, denoted as *sup*(γ).(Pei et al., 2002)

1. **Item Constraint**: This type of constraint indicates which type of items, singular or groups, are to be included or removed from the mined patterns. The form is $C_{item}(\alpha) \equiv (\varphi i : 1 \leq i \leq len(\alpha),\ \alpha[i]\ \theta\ V)$, or $C_{item}(\alpha) \equiv (\varphi i : 1 \leq i \leq len(\alpha),\ \alpha[i] \cap V \neq \emptyset)$, where V is a subset of items, $\varphi \in \{\forall, \exists\}$ and $\theta \in \{\subseteq, \not\subseteq, \supseteq, \not\supseteq, \in, \notin\}$ (Pei et al., 2002).

   For example, when mining sequential patterns from medical data a user may only be interested in patterns that have a reference to a particular hospital. If H is the set of hospitals, the corresponding item constraint is $C_{hospital}(\alpha) \equiv (\forall i : 1 \leq i \leq len(\alpha), \alpha[i] \subseteq H)$.

2. **Length Constraint**: This type of constraint specifies the length of the patterns to be mined either as the number of elements or the number of transactions that comprises the pattern. This basic definition may also be modified to include only unique or distinct items.

   For example, a user may only be interested in discovering patterns of say length 20 elements that occur in the analysis of a sequence of web log clicks. This would be expressed as a length constraint $C_{len}(\alpha) \equiv (len(\alpha) \leq 20)$.

3. **Model-based Constraint**: These types of constraints are those which look for patterns that are either *sub-patterns* or *super-patterns* of some given pattern. A *super-pattern constraint* is in the form of $C_{pat}(\alpha) \equiv (\exists \gamma \in P \text{ s.t. } \gamma \sqsubseteq \alpha)$, where P is a given set of patterns. Simply stated this says find patterns that contain a particular set of patterns as sub-patterns (Pei et al., 2002).

   For example, an electronics shop may employ an analyst to mine their transaction database and in doing so may be interested in all patterns that contain first buying a PC then a digital camera and a photo-printer together. This can be expressed as $C_{pat}(\alpha) \equiv \langle (PC)(digital\_camera, photo\text{-}printer) \rangle \sqsubseteq \alpha$.

4. **Aggregate Constraint**: These constraints are imposed on an aggregate of items in a pattern, where the aggregate function can be, **avg**, **min**, **max**, etc.

   For example, the analyst in the electronics shop may also only be interested in patterns where the average price of all the items is greater than $500.

5. **Regular Expression Constraint**: This type of constraint C_{RE} are constraints specified using the established set of regular expression operators – disjunction, Kleene closure etc. For a sequential pattern to satisfy a particular regular expression constraint C_{RE} it must be accepted by its equivalent deterministic finite automata.

   For example, to discover sequential patterns about a patient who was admitted with measles and was given a particular treatment a regular expression of the form *Admitted(Measles | German Measles)(Treatment A | Treatment B | Treatment C)* where "|" indicates disjunction.

6. **Duration Constraint**: For these constraints to be defined, each and every transaction in the sequence database or each item in a temporal sequence must have an associated time-stamp. The duration is determined by the time difference between the first and last transaction, or first and last item, in the pattern and can be either longer or shorter than a given period. Formally, a duration constraint is in the form of $Dur(\alpha)\theta\ \Delta t$, where $\theta \in \{\leq, \geq\}$ and Δt is a given integer. A sequence α satisfies the constraint (checking satisfaction is achieved by examining the *SDB*) if and only if $|\{\beta \in SDB | \exists 1 \leq i_1 < \cdots < i_{len(\alpha)} \leq len(\beta) \text{ s.t. } \alpha[1] \sqsubseteq \beta[i_1], \ldots, \alpha[len(\alpha)] \sqsubseteq \beta[i_{len(\alpha)}]$, and $(\beta[i_{len(\alpha)}].time - \beta[i_1].time)\theta\ \Delta t\}| \geq min\_sup$ (Pei et al., 2002).

   In algorithms that have temporal sequences as their input this type of constraint is usually implemented as a 'sliding window' across the event set.

7. **Gap Constraint**: The patterns that are discovered using this type of constraint are similarly derived from sequence databases or temporal sequences that include time-stamps. The gap is determined by the time difference between adjacent items, or transactions and must be longer or shorter than a given gap. Formally,

a gap constraint is in the form of $Gap(\alpha)\theta\ \Delta t$, where $\theta \in \{\leq, \geq\}$ and Δt is a given integer. A sequence α satisfies the constraint (to check the satisfaction of this constraint the SDB must be examined) if and only if $|\{\beta \in SDB | \exists 1 \leq i_1 < \cdots < i_{len(\alpha)} \leq len(\beta)$ s.t. $\alpha[1] \sqsubseteq \beta[i_1], \ldots, \alpha[len(\alpha)] \sqsubseteq \beta[i_{len(\alpha)}]$, and for all $1 < j \leq len(\alpha), (\beta[i_j].time - \beta[i_{j-1}].time)\theta\ \Delta t\}| \geq min\_sup$ (Pei et al., 2002).

2.3 Counting Techniques

For a sequence to be deemed interesting it is required to meet some criteria in relation to the number of occurrences of it in the sequence with respect to any timing constraints that may have been imposed. As was indicated in the previous section only two constraints deal with timing – *duration* and *gap*, and so for the purpose of this discussion and with this in mind the following terminology will be used (modified from Joshi, Karypis and Kumar (1999)).

- *ms*: **maximum span** – maximum allowed time difference between the latest and earliest occurrences of events in the *entire* sequence.
- *ws*: **event-set window size** – maximum allowed time difference between the latest and earliest occurrences of events in any *event-set*. Here the term *event-set* is equivalent to the term *transaction* in Section 2.2
- *xg*: **maximum gap** – maximum allowed time difference between the latest occurrence of an event in and event-set and the earliest occurrence of an event in its immediately preceding event-set.
- *ng*: **maximum span** – minimum required time difference between the earliest occurrence of an event in and event-set and the latest occurrence of an event in its immediately preceding event-set.

These four parameters can be used to define five different counting techniques which can be divided into three groups. The first group CEVT (count event) searches for the given sequence in the entire sequence timeline. The second group deals with counting the number of windows (windows equate to the maximum span (*ms*)) in which the given sequence occurs and consists of CWIN (count windows) and CMINWIN (count minimum windows). The third group deals with distinct events within the window that is of size *ms* (maximum span) and comprises CDIST (count distinct) and CDIST_O (count distinct with the possibility of overlap). The use of any of these methods depends on the specific application area and on the users expertise in the domain of the area being mined. Figure 2.1 shows the differences between the counting methods, and highlights the need for careful consideration when choosing a counting method.

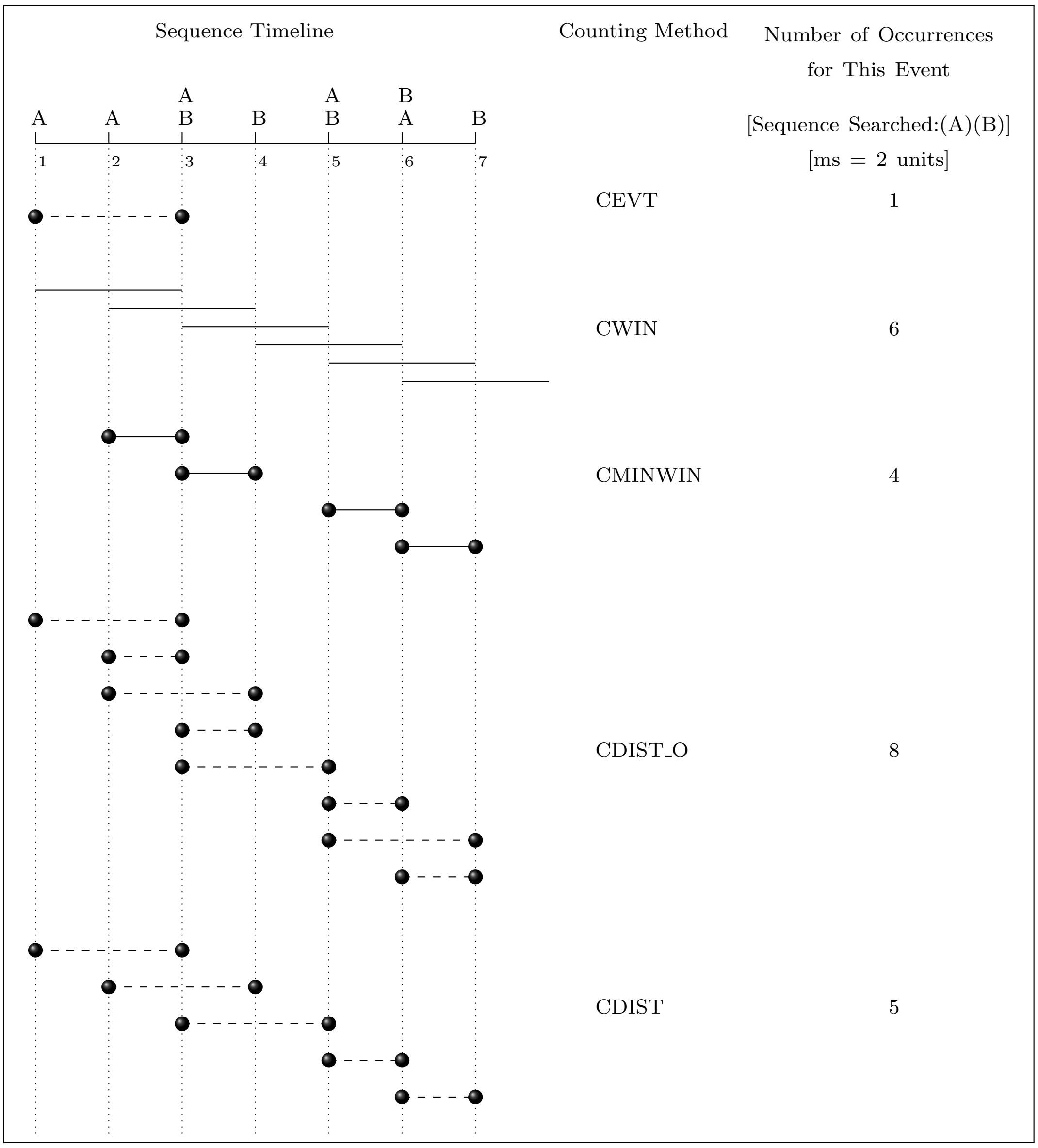


Figure 2.1: A comparison of different counting methods – Joshi et al. (1999).

2.4 Apriori-based Algorithms

The Apriori family of algorithms has typically been used to discover *intra-transaction* associations and then to generate rules about the discovered associations, however the *sequence* mining task is defined as discovering *inter-transaction* associations – sequential patterns – across the same, or similar data. It is not surprising then that the first algorithms to deal with this change in focus were based on the Apriori algorithm (Agrawal and Srikant, 1994) using transactional databases as their data source. The next section will outline the problem statement and notation that is typical of these types of algorithm. Before each section describing the algorithms a summary of the database format will be given.

2.4.1 Problem Statement and Notation

The problem of mining sequential patterns can be stated as follows:

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of literals called *items* which comprise the alphabet. An *event* is a non-empty unordered collection of items. It is assumed without loss of generality that items of an event are sorted in lexicographic order. A *sequence* is an ordered list of events. An event is denoted as $(i_1, i_2, \ldots, i_k)$, where i_j is an item. A sequence α is denoted as $\langle \alpha_1 \rightarrow \alpha_2 \rightarrow \cdots \rightarrow \alpha_q \rangle$, where α_i is an event. A sequence with k-events (transactions) ($k = \sum_j |\alpha_j|$) is called a *k-sequence.* For example, $\langle B \rightarrow AC \rangle$ is a 3-sequence. A sequence $\langle \alpha_1 \rightarrow \alpha_2 \ldots \rightarrow \alpha_n \rangle$ is a *subsequence* of another sequence $\langle \beta_1 \rightarrow \beta_2 \ldots \rightarrow \beta_m \rangle$ if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $\alpha_1 \subseteq \beta_{i_1}, \alpha_2 \subseteq \beta_{i_2}, \ldots, \alpha_n \subseteq \beta_{i_n}$. For example the sequence $\langle B \rightarrow AC \rangle$ is a subsequence of $\langle AB \rightarrow E \rightarrow ACD \rangle$, since $B \subseteq AB$ and $AC \subseteq ACD$, and the order of events is preserved. However, the sequence $AB \rightarrow E$ is not a subsequence of ABE and vice versa.

We are given a database $\mathcal{D}$ of input-sequences where each input-sequence in the database has the following fields: sequence-id, event-time and the items present in the event. It is assumed that no sequence has more that one event with the same time-stamp, so that the time-stamp may be used as the event identifier. In a general sense the *support* or *frequency* of a sequence, denoted $\sigma(\alpha, \mathcal{D})$, is defined as the total number of input-sequences in the database $\mathcal{D}$ that contain α. This general definition has been modified as algorithmic development has progressed and different methods for calculating support have been introduced, a summary of which is included in Section 2.3. Given a user-specified threshold called the *minimum support* (denoted *min_supp*), a sequence is said to be *frequent* if it occurs more than *min_supp* times and the set of frequent k-sequences is denoted as $\mathcal{F}_k$. Further a frequent sequence is deemed to be *maximal* if it is not a subsequence of any other frequent sequence. The task then becomes to find all frequent sequences from a database $\mathcal{D}$ of input-sequences and a user supplied *min_supp.*

This constitutes the problem definition for all sequence mining algorithms whose data are located in a transaction database or are transaction datasets. Further terminology, that is specific to an algorithm or set of algorithms, will be elaborated as required.

2.4.2 Horizontal Database Format

Horizontal formatting means that the original data, Table 2.1(a), is sorted first by Customer Id and then by Transaction Time which results in a transformed customer-sequence database, Table 2.1(b), where the timestamps from Table 2.1(a) are used to determine the order of events, which is used as the basis for mining. The mining is then carried out using a breadth-first approach.

Table 2.1: Horizontal Formatting Data Layout – adapted from Agrawal and Srikant (1995).

(a) Customer Transaction Database

Customer Id	Transaction Time	Items Bought
1	June 25 '03	30
1	June 30 '03	90
2	June 10 '03	10, 20
2	June 15 '03	30
2	June 20 '03	40, 60, 70
3	June 25 '03	30, 50, 70
4	June 25 '03	30
4	June 30 '03	40, 70
4	July 25 '03	90
5	June 12 '03	90

(b) Customer-Sequence version of the Database

Customer Id	Customer Sequence
1	$\langle$ (30) (90) $\rangle$
2	$\langle$ (10 20) (30) (40 60 70) $\rangle$
3	$\langle$ (30 50 70) $\rangle$
4	$\langle$ (30) (40 70) (90) $\rangle$
5	$\langle$ (90) $\rangle$

2.4.3 Horizontal Database Format Algorithms

2.4.3.1 AprioriAll, AprioriSome and DynamicSome

The first algorithms that were introduced were named AprioriAll, AprioriSome, and DynamicSome and were developed by Agrawal and Srikant (1995) using a 5-stage process:

1. *Sort Phase.* This phase effectively transforms the dataset from the original transaction database (Table 2.1(a)) to a customer sequence database (Table 2.1(b)) by sorting the dataset by customer_id and then by transaction_time.

2. *Litemset* (large itemset) *Phase.* The function of this phase is to find the set of all litemsets L (those that meet minimum support), which is in effect the set of all large 1-sequences since this is just $\{\langle l \rangle \mid l \in L\}$. Also at this stage optimisations for future comparisons are carried out by mapping the litemsets to a set of contiguous integers. For example, if the minimum support was given as 25%, using the data from Table 2.1(b), a possible mapping for the large itemsets is depicted in Table 2.2.

Table 2.2: Large Itemsets and a possible mapping – Agrawal and Srikant (1995).

Large Itemsets	Mapped To
(30)	1
(40)	2
(70)	3
(40 70)	4
(90)	5

3. *Transformation Phase.* Since there is a need to repeatedly determine which of a given set of long sequences are contained in a customer sequence, each customer sequence is transformed by replacing each transaction with the set of litemsets contained in that transaction. Transactions that do not contain any litemsets are not retained and a customer sequence that does not contain any litemsets is dropped. The customer sequences that are dropped do however still contribute to the total customer count. This is depicted in Table 2.3.

Table 2.3: The transformed database including the mappings – Agrawal and Srikant (1995).

C_Id	Original Customer Sequence	Transformed Customer Sequence	After Mapping
1	⟨(30) (90)⟩	⟨{(30)} {(90)}⟩	⟨{1} {5}⟩
2	⟨(10 20) (30) (40 60 70)⟩	⟨{(30)} {(40), (70), (40 70)}⟩	⟨{1} {2, 3, 4}⟩
3	⟨(30 50 70)⟩	⟨{(30), (70)}⟩	⟨{1, 3}⟩
4	⟨(30) (40 70) (90)⟩	⟨{(30)} {(40), (70), (40 70)} {(90)}⟩	⟨{1} {2, 3, 4} {5}⟩
5	⟨(90)⟩	⟨{(90)}⟩	⟨{5}⟩

4. *Sequence Phase.* This phase mines the set of litemsets to discover the sequences. Three algorithms are presented for this discovery all of which make multiple passes over the data. Each pass begins with a seed set for producing potential large sequences (candidates) and then the support for these candidates is calculated during the pass. Those that do not meet the minimum support threshold are pruned and then those that remain become the seed set for the next pass. The process begins with the large 1-sequences and terminates when either no candidates are generated or no candidates meet the minimum support criteria.

5. *Maximal Phase.* This is designed to find all maximal sequences among the set of large sequences. Although this phase is applicable to all of the algorithms, the AprioriSome and DynamicSome combine this with the sequence phase to save time by not counting non-maximal sequences. The process is similar in nature to the process of finding all subsets of a given itemset and as such the algorithm

for performing this task is also similar. An example can be found in Agrawal and Srikant (1994).

The difference in the three algorithms results from the methods of counting the sequences produced. Although all are based on the Apriori algorithm (Agrawal and Srikant, 1994) the AprioriAll algorithm counts all of the sequences whereas the Apriori-Some and DynamicSome are designed to only produce maximal sequences and therefore can take advantage of this by first counting longer sequences and only counting shorter ones which are not contained in longer ones. This is done by utilising a *forward* phase which finds all sequences of a certain length and a *backward* phase which finds the remaining long sequences not discovered during the forward phase.

This seminal work, however, had some limitations:

- Given that the output was the set of maximal frequent sequences, some of the inferences (rules) that could be made could be construed as being of no real value. For example a retail store would probably not be interested in knowing that a customer purchased product 'A' and then two years later purchased product 'B'.
- Items were constrained to appear in a single transaction limiting the inferences available and hence the potential value that the discovered sequences could elicit. In many domains it could be beneficial that transactions that occur within a certain time window (the time between the maximum and minimum transaction times) are viewed as a single transaction.
- Many datasets have a user-defined hierarchy associated with them and users may wish to find patterns that exist not only at one level, but across different levels of the hierarchy.

These limitations were addressed by the the algorithm called Generalised Sequential Patterns or GSP.

2.4.3.2 GSP: Generalised Sequential Patterns

In order to address the shortcomings of the original algorithms the apriori model was extended and resulted in the GSP (**G**eneralised **S**equential **P**atterns) algorithm (Srikant and Agrawal, 1996). The extensions included time constraints (minimum and maximum gap between transactions), sliding windows and taxonomies. The minimum and/or maximum gap between adjacent elements was included to reduce the number of 'trivial' rules that may be produced. For example, a video store owner probably does not care if someone hired "Fellowship of the Ring" and then a few years later hired "The Two Towers", but would if this sequence of events occurred within a few weeks. The sliding window enhances the timing constraints by allowing elements of a sequential pattern to be present in a set of transactions that occur within the user-specified time window. And finally the user-defined taxonomy (*is-a* hierarchy) which is present in many datasets allows sequential patterns to include elements from any level in the taxonomy.

The algorithm follows the classic *candidate generation and prune* paradigm where each subsequent set of candidates is generated from seed sets from the previous frequent pass (L_{k-1}) of the algorithm, where L_k is the set of frequent k-length sequences. This is accomplished in two steps:

1. *The join step.* This is done by joining L_{k-1} with L_{k-1} in the following way. A sequence s_1 is joined with s_2 if the subsequence obtained by removing the first item of s_1 is the same as the subsequence obtained by removing the last item of s_2. The candidate sequence is then generated by extending s_1 with the last item in s_2 and the added item becomes a separate element if it was a separate element in s_2 or becomes part of the last element of s_1 otherwise. For example if $s_1 = \langle(1,2)\ (3)\rangle$ and for the first case $s_2 = \langle(2)\ (3,4)\rangle$ and the second case $s_2 = \langle(2)\ (3)\ (4)\rangle$ then after the join the candidate would be $c = \langle(1,2)\ (3,4)\rangle$ and $c = \langle(1,2)\ (3)\ (4)\rangle$ respectively.

2. *The prune step.* Candidates are deleted if they contain a contiguous $(k-1)$ subsequence whose support count is less than the minimum specified support. A more rigid approach can be taken when there is no max-gap constraint resulting in any subsequence without minimum support being deleted.

The algorithm terminates when there are no frequent sequences from which to generate seeds, or there are no candidates generated.

To reduce the number of candidates that need to be checked an adapted version of the hash-tree data structure used by Agrawal and Srikant (1994) was adopted. The nodes of the hash-tree either contain a list of sequences as a *leaf* node or a hash table as an *interior* node. Each non-empty bucket of a hash table in an interior node points to another node. By utilising this structure, candidate checking is then performed using either a forward or backward approach. The forward approach deals with successive elements as long as the difference between the end time of the element just found and the start time of the previous element is less than max-gap. If this difference is more than max-gap then the algorithm switches to the backward approach where the algorithm moves backward until either the max-gap constraint between the element just pulled up and the previous element is satisfied or the root is reached. The algorithm then switches to the forward approach and this process continues, switching between the forward and backward approaches until all elements are found. These improvements in counting and pruning candidates led to improved speed over that of AprioriAll and although the introduction of constraints improved the functionality of the process (from a user perspective) a problem still existed with respect to the number of patterns that were generated. This is an inherent problem facing all types of algorithms and solutions that involve adding further constraints have typically been used. The major types of constraints that were added have already been discussed in Section 2.2.

2.4.3.3 PSP

The PSP algorithm by Masseglia, Cathala and Poncelet (1998) was inspired by GSP, but has improvements which make it possible to perform retrieval optimizations. The process uses transactional databases as its source of data and a candidate generation and scan approach for the discovery of frequent sequences. The difference lies in the way that the candidate sequences are organized. GSP and its predecessors use hash tables at each internal node of the candidate tree, whereas the PSP approach organizes the candidates in a prefix-tree according to their common elements which results in less memory overhead and faster retrievals. The tree structure used in this algorithm only stores initial sub-sequences common to several candidates only once and the terminal node of any branch stores the support of the sequence to any considered leaf inclusively. Adding to the support value of candidates is performed by navigating to each leaf in the tree and then simply incrementing the value, which is much faster than the GSP approach. A comparison of the tree structures is illustrated in Figure 2.2 using the following set of frequent 2-sequences: $L_2 = \langle(10)\ (30)\rangle, \langle(10)\ (40)\rangle, \langle(30)\ (20)\rangle, \langle(30)\ (40)\rangle, \langle(40\ 10)\rangle$. The illustration shows the state of the trees after generating the 3-candidates and clearly shows the reduced overhead of the PSP approach.

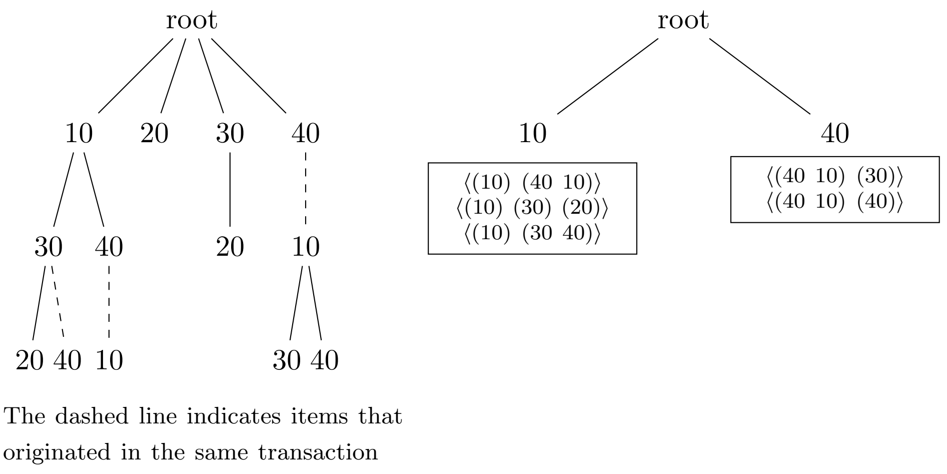


Figure 2.2: The prefix-tree of *PSP* (left tree) and the hash-tree of *GSP* (right tree) showing storage after candidate-3 generation – Masseglia et al. (1998).

2.4.3.4 SPIRIT: Sequential Pattern Mining with Regular Expression Constraints

To enable users to take advantage of a predefined requirement for output, a family of algorithms termed SPIRIT (**S**equential **P**attern m**I**ning with **R**egular express**I**on cons**T**raints) were developed by Garofalakis, Rastogi and Shim (1999). The choice of regular expression constraints was due to their expressiveness in allowing families of sequential patterns to be defined and the simple natural syntax that they provide. Apart from the reduction of potentially useless patterns the algorithms also gained significant performance by 'pushing' the constraints inside the mining algorithm, that is using the constraint-based pruning followed by support-based pruning and storing the regular expressions in finite state automata. This technique reduces to the candidate generation and pruning of GSP when the constraint, $\mathcal{C}$, is anti-monotone, but when this is not so (as is the case of the regular expressions used by SPIRIT) a relaxation of $\mathcal{C}$, that is a weaker or less restrictive constraint $\mathcal{C}'$, is used. Varying levels of relaxation of $\mathcal{C}$ gave rise to the SPIRIT family of algorithms that are ordered in the following way: SPIRIT(N)("N" for Naive), employs the weakest relaxation, followed by SPIRIT(L)("L" for Legal), SPIRIT(V)("V" for Valid), and SPIRIT(R)("R" for Regular). This decrease in relaxation impacts on the effectiveness of both the constraint-based and support-based pruning but has the potential to increase the performance of the algorithm by restricting the amount of candidates that are generated during each pass of the pattern mining loop.

2.4.3.5 MFS: Maximal Frequent Sequences

Zhang, Kao, Yip and Cheung (2001) proposed the algorithm MFS (**M**aximal **F**requent **S**equences) which uses a modified version of the GSP candidate generation function as its core candidate generation function. This modification allows for a significant reduction in any I/O requirements since the algorithm only checks candidates of various lengths in each database scan. The authors call this a *successive refinement* approach. The algorithm first computes a rough estimate of the set of all frequent sequences by using the results of a previous mining run if the database has been updated since the last mining run, or by mining a small sample of the database using GSP. This set of varying length frequent sequences is then used to generate the set of candidates which are checked against the database to determine which are in fact frequent. The *maximal* of these frequent sequences are kept and the process is repeated only on these *maximal* sequences again checking any candidates against the database. The process terminates when no more new frequent sequences are discovered in an iteration. The major source of efficiency over GSP is that the supports of longer sequences can be checked early in the process.

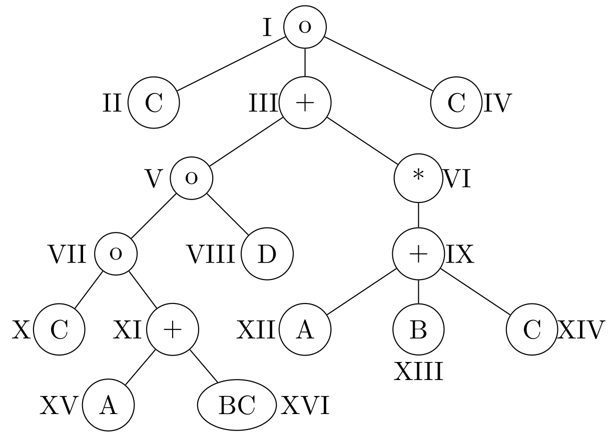


Figure 2.3: Hackle-tree for $C((C(A+BC)D)+(A+B+C)^{*})C$ – Albert-Lorincz and Boulicaut (2003*b*).

2.4.3.6 RE-Hackle: Regular Expression-Highly Adaptive Constrained Local Extractor

Using regular expressions and minimum frequency constraints combines both anti-monotonic (frequency) and non-anti-monotonic (regular expressions) pruning techniques and has already been discussed for the SPIRIT family of algorithms and is therefore not a new concept. The RE-Hackle algorithm (**R**egular **E**xpression-**H**ighly **A**daptive **C**onstrained **L**ocal **E**xtractor) by Albert-Lorincz and Boulicaut (2003*b*), however, uses a hierarchical representation of Regular Expressions which it stores in a Hackle-tree rather than the Finite State Automaton used in the SPIRIT algorithms. They build their RE-constraints for mining using RE's built over an alphabet using three operators: union (denoted +), concatenation (denoted by $\circ_k, \circ_o$ being the usual concatenation) and Kleeene closure (denoted *). A Hackle-tree, see Figure 2.3, is a form of Abstract Syntax Tree that encodes the structure of a RE-constraint and is structured as follows. Each inner node contains a operator and the leaves contain atomic sequences, in this manner the tree reflects the way in which the atomic sequences are assembled from the unions, concatenations and Kleene closures to form the the initial RE-constraint.

After the construction of the RE-constraint and the Hackle-tree has been built (this is termed the *Extraction phrase*) the extraction of begins. Extraction functions are applied to the nodes of the Hackle-tree and return the candidate sequences that need to be counted, those that are deemed to be frequent are used for the next generation. This is done by creating a new extraction phrase and a new Hackle-tree is then built and the process resumes. The process terminates when no more candidates are discovered.

2.4.3.7 MSPS: Maximal Sequential Patterns using Sampling

In the MSPS (**M**aximal **S**equential **P**atterns using **S**ampling) algorithm Luo and Chung (2004) combine the approach taken in GSP and the supersequence frequency pruning for mining maximal frequent sequences. Supersequence frequency based pruning is such that any subsequence of a frequent sequence is also frequent and therefore can be pruned from the set of candidates. This gives rise to the classic bottom-up breadth-first search strategy that includes, from the second pass over the database, mining a small random sample of the database to generate local maximal frequent sequences from the sample. After these have been verified in a top-down fashion against the original database, so that the longest frequent sequences covered by them can be collected, the bottom-up search is continued. In addition the authors use a signature technique to overcome any problems that may arise when a set of k-sequences will not fit into memory and candidate generation is required. The sampling that occurs is related to the work of Toivonen (1996) who used a lowered minimum support when mining the sample resulting in less chance that a frequent itemset is missed. In this work to make sampling more efficient, a theoretical method based on statistics to adjust the user-specified minimum support is used. The counting approach is similar to the PSP approach in that a prefix tree is used, however the tree used in this algorithm differs in the fact that it is used to count candidates of different sizes and the size of the tree is considerably smaller because of the supersequence frequency based pruning. The tree also has an associated bit vector whose size is that of the number of unique single items in the database where each position is initialised to zero. Setting the bit vector up to assist in counting requires that as each candidate is inserted into the prefix tree its corresponding bit is set to one. As each customer sequence is inserted into the tree it is checked against the bit vector first and those items that do not occur (the bit is set to zero) are trimmed accordingly. Figure 2.4 illustrates a Prefix tree for an incoming customer sequence of ABCD − ADEFG − B − DH trimmed against a bit vector of (10111001). The bit vector is derived as such, since the database contains the alphabet A,B,C,D,E,F,G,H, but there are no candidates containing B,F, and G and therefore should be ignored during counting. Each node may have two types of children S-extension (the child starts a new itemset) and I-extension (the child is in the same itemset with the item represented by its parent node). S-extensions are represented by a dashed line and I-extensions by a solid line.

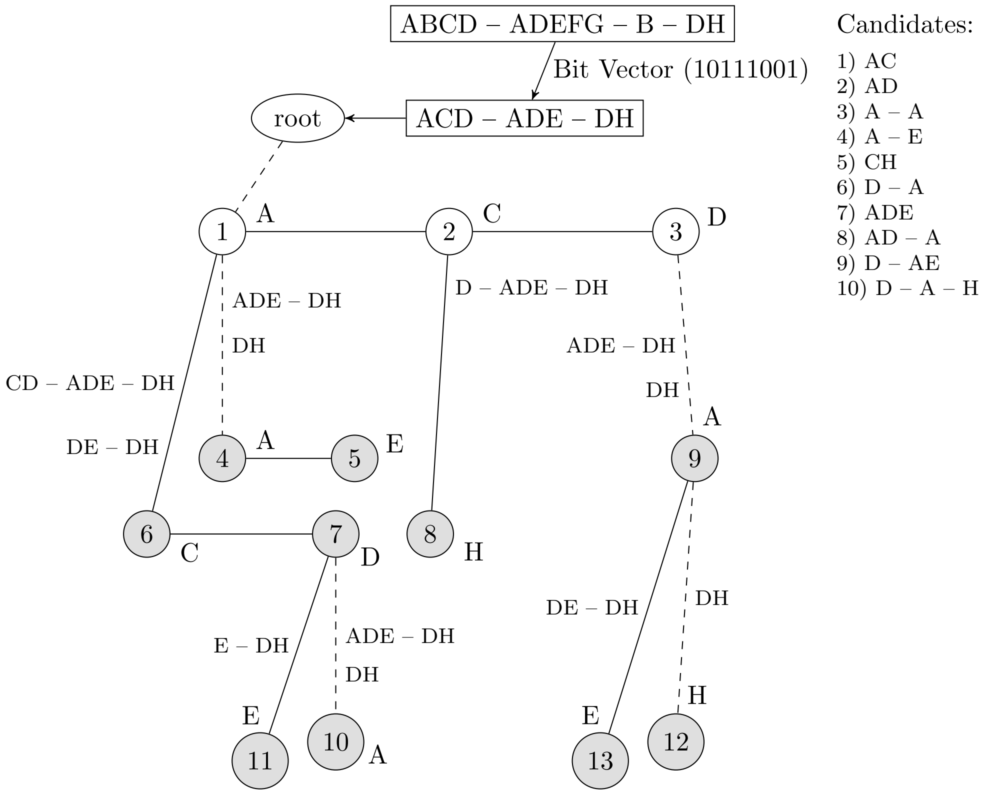


Figure 2.4: A Prefix Tree of MSPS – Luo and Chung (2004). The labels on the edges represent recursive calls on segments of the trimmed customer sequence.

2.4.4 Vertical Database Format

Algorithmic development in the sequence mining area, to a large extent, has mirrored that in the Association Rule Mining field and as improvements in performance were required they came about, in the first instance, by employing a depth-first approach to the mining, and later by using pattern growth methods (see Section 2.5). This shift required that the data be organised in an alternative fashion, which is called a vertical database format where the rows of the database consist of object-timestamped pairs associated with an event. This makes it easy to generate *idlists* for each event that consist of the object-timestamp rows of the events thus enabling all frequent sequences to be enumerated via simple temporal joins of the idlists. An example of this type of format is shown in Tables 2.4(a) and 2.4(b). Researchers such as Yang, Wang, Yu and Han (2002) have recognised that these methods generally perform better when the data is memory-resident and when the patterns are long, but also that the generation and counting of candidates becomes much easier. This shift in data layout brought about the introduction of algorithms based on a depth-first traversal of the search space and at the same time there was an increased focus on incorporating constraints into the mining process. This is due in part to the improvement in processing time but also as a reaction to the need to reduce the number of results.

Table 2.4: Vertical Formatting Data Layout – Zaki (2001*b*).

(a) Input-Sequence Database

Sequence Id	Time	Items
1	10	C D
1	15	A B C
1	20	A B F
1	25	A C D F
2	15	A B F
2	20	E
3	10	A B F
4	10	D G H
4	20	B F
4	25	A G H

(b) Id-Lists for the Items

A		**B**		**D**		**F**	
SID	EID	SID	EID	SID	EID	SID	EID
1	15	1	15	1	10	1	20
1	20	1	20	1	25	1	25
1	25	2	15	4	10	2	15
2	15	3	10			3	10
3	10	4	20			4	20
4	25						20

SID: Sequence Id
EID: Time

2.4.5 Vertical Database Format Algorithms

2.4.5.1 SPADE: Sequential Pattern Discovery using Equivalence Classes

The SPADE (**S**equential **PA**ttern **D**iscovery using **E**quivalence classes) algorithm (Zaki, 2001*b*) and its variant cSPADE (**c**onstrained SPADE) (Zaki, 2000) use combinatorial properties and lattice based search techniques and allow constraints to be placed on the mined sequences.

Key features of SPADE include the layout of the database which is a vertical id-list database format, the search space is decomposed into small pieces (sub-lattices) that can be processed independently in main memory thus enabling the database to be scanned only three times or just once on some pre-processed data. Two search strategies are proposed for finding sequences in the lattices:

1. Breadth-first search: the lattice of equivalence classes is explored in a bottom-up manner and all child classes at each level are processed before moving to the next.
2. Depth-first search: all equivalence classes for each path are processed before moving to the next path.

Using the vertical id-list database in Table 2.4(b) all frequent 1-sequences can be computed in one database scan. Computing the $\mathcal{F}_2$ can be achieved in one of two ways; by pre-processing and collecting all 2-sequences above a user specified lower bound, or by performing a vertical to horizontal transformation on the fly.

Once this has been completed the process continues by decomposing the 2-sequences into prefix-based parent equivalence classes followed by the enumeration of all other frequent sequences via either breadth-first or depth-first searches within each equivalence

Table 2.5: Computing Support using temporal id-list joins – adapted from Zaki (2001*b*).

(a) Intersect D and A, D and B, D and F.

$D \rightarrow A$		$D \rightarrow B$		$D \rightarrow F$	
SID	EID	SID	EID	SID	EID
1	15	1	15	1	20
1	20	1	20	1	25
1	25	4	20	4	20
2	15	3	10		
4	25				

(b) Intersect $D \rightarrow B$ and $D \rightarrow A$, $D \rightarrow B$ and $D \rightarrow F$.

$D \rightarrow B \rightarrow A$		$D \rightarrow BF$	
SID	EID	SID	EID
1	20	1	20
1	25	4	20
4	25		

(c) Intersect $D \rightarrow B \rightarrow A$ and $D \rightarrow BF$.

$D \rightarrow BF \rightarrow A$	
SID	EID
1	25
4	25

class. The enumeration of the frequent sequences can be performed by joining the id-lists in one of three ways (assume that A and B are items and S is a sequence):

1. *Itemset and Itemset*: joining AS and BS results in a new itemset ABS.
2. *Itemset and Sequence*: joining AS with $B \rightarrow S$ results in a new sequence $B \rightarrow AS$.
3. *Sequence and Sequence*: joining $A \rightarrow S$ with $B \rightarrow S$ gives rise to three possible results: a new itemset $AB \rightarrow S$, and two new sequences $A \rightarrow B \rightarrow S$ and $B \rightarrow A \rightarrow S$. One special case occurs when $A \rightarrow S$ is joined with itself resulting in $A \rightarrow A \rightarrow S$

The enumeration process is the union or join of a set of sequences or items and once these have been found their counts are calculated by performing an intersection of the id-lists of the elements that comprise the newly formed sequence. By proceeding in this manner it is only necessary to use the lexicographically first two subsequences at the last level to compute the support of a sequence at a given level (Zaki, 1998). This process for enumerating and computing the support for the sequence $D \rightarrow BF \rightarrow A$ is shown in Table 2.5 using the data supplied in Table 2.4(b).

The cSPADE algorithm (Zaki, 2000) is the same as SPADE except that it incorporates one or more of the the following syntactic constraints as checks during the mining process:

1. Length or width limitations on the sequences; allows for highly structured data, for example DNA sequence databases, to be mined without having the problem of an exponential explosion in the number of discovered frequent sequences.
2. Minimum or maximum gap constraints on consecutive sequence elements to enable the discovery of sequences that occur after a certain minimum amount of time, and no longer than a maximum time ahead.

3. Applying a time window on allowable sequences, which means the entire sequence must occur within the window. This differs from minimum and maximum gaps in that it deals with the entire sequence not the time between sequence elements.
4. Incorporating item constraints. Since the generation of individual equivalence classes is achieved easily by using the equivalence class approach and a vertical database format, exclusion of an item becomes just a check for the particular item in the class and the removal from those classes where it occurs. Further expansion of these classes will never contain these items.
5. Finding sequences predictive of one or more classes (even rare ones). This is only applicable to certain datasets (classification) where each input sequence has a class label.

2.4.5.2 SPAM: Sequential Pattern Mining using a Bitmap Representation

SPAM (**S**equential **PA**ttern **M**ining using A Bitmap Representation) (Ayres, Flannick, Gehrke and Yiu, 2002) uses a novel depth-first traversal of the the search space with effective pruning mechanisms and a vertical bitmap representation of the database which enables efficient support counting. A vertical bitmap for each item in the database is constructed while scanning the database for the first time and each bitmap has a bit corresponding to each element of the sequences in the database. One potential limiting factor on its usefulness is the SPAM algorithm requirement that all of the data fit into main memory.

The candidates are stored in a structure called a lexicographic sequence lattice or tree (the same type as used in PSP), which enables the candidates to be extended in one of two ways: Sequence extended using an *S-step* process and Itemset extended using an *I-step* process. This process is exactly the same as the approach taken in GSP (Srikant and Agrawal, 1996) and PSP (Masseglia et al., 1998) in that an item becomes a separate element if it was a separate element in the sequence it came from or becomes part of the last element otherwise. These processes are carried out using the bitmaps for the sequences or items in question by *ANDing* them to produce the result. The *S-step* process requires that a *transformed bitmap* first be created by setting all bits less than or equal to the item in question for any transaction to zero and all others to one. This transformed bitmap is then used for ANDing with the item to be appended. Table 2.6(c) and Table 2.6(d) illustrate this using the data in Table 2.6(a) and bitmap representation in Table 2.6(b).

The method of pruning candidates is based on downward closure and is conducted on both *S-extension* and *I-extension* candidates of a node in the tree using a depth-first search which guarantees all nodes are visited. However, if the support for a sequence $s < min\_supp$ at a particular node then no more depth-first search is required on s due to downward closure.

Table 2.6: SPAM data representation – Ayres et al. (2002).

(a) Data sorted by CID and TID.

Customer ID (CID)	**TID**	**Itemset**
1	1	$\{a, b, c\}$
1	3	$\{b, c, d\}$
1	6	$\{b, c, d\}$
2	2	$\{b\}$
2	4	$\{a, b, c\}$
3	5	$\{a, b\}$
3	7	$\{b, c, d\}$

(b) Bitmap representation of the dataset in (a)

CID	**TID**	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$
1	1	1	1	0	1
1	3	0	1	1	1
1	6	0	1	1	1
-	-	0	0	0	0
2	2	0	1	0	0
2	2	1	1	1	0
-	-	0	0	0	0
-	-	0	0	0	0
3	5	1	1	0	0
3	7	0	1	1	1
-	-	0	0	0	0
-	-	0	0	0	0

(c) S-Step processing

$(\{a\})$		$(\{a\})_s$		$\{b\}$		$(\{a\}, \{b\})$
1		0		1		0
0		1		1		1
0		1		1		1
0		1		0		0
0	S-step	0		1	result	0
1	$\longrightarrow$	1	&	1	$\longrightarrow$	0
0	process	1		0		0
0		1		0		0
1		0		1		0
0		1		1		1
0		1		0		0
0		1		0		0

(d) I-step processing

$(\{a\}, \{b\})$		$\{d\}$		$(\{a\}, \{b, d\})$
0		1		0
1		1		1
1		1		1
0		0		0
0		0	result	0
0	&	0	$\longrightarrow$	0
0		0		0
0		0		0
0		0		0
1		1		1
0		0		0
0		0		0

2.4.5.3 CCSM: Cache-based Constrained Sequence Miner

The CCSM (**C**ache-based **C**onstrained **S**equence **M**iner) algorithm of Orlando, Perego and Silvestri (2004) uses a level-wise approach initially but overcomes most problems associated with this type of algorithm. This is achieved by using k-way intersections of *id-lists* to compute the support of candidates (the same as SPADE (Zaki, 2001*b*)) combined with a *cache* that stores intermediate id-lists for future reuse.

The algorithm is similar to GSP (Srikant and Agrawal, 1996) because it adopts a level-wise bottom-up approach in visiting the sequential patterns in the tree but it differs since after extracting the frequent $\mathcal{F}_1$ and $\mathcal{F}_2$ from the horizontal database, this pruned database is transformed into a vertical one resulting in the same configuration as SPADE (Zaki, 2001*b*). The major difference is the use of a *cache* to store intermediate id-lists for use in speeding up support counting. This is achieved in the following way. When a new sequence is generated, and if a common prefix is contained in the cache, then the associated id-list is reused and subsequent lines of the cache are rewritten. This enables only a single equality join to be performed between the common prefix and the new item, after which the result of the join is added to cache.

2.4.5.4 IBM: Indexed Bit Map

The application goal of IBM (**I**ndexed **B**it **M**ap for Mining Frequent Sequences) (Savary and Zeitouni, 2005) is to find chains of activities that characterise a group of entities and where the input can be composed of single items. Their approach consists of two phases the first of which is data encoding and compression and the second is frequent sequence generation which is itself comprised of candidate generation and support checking. Four data structures are used for the encoding and compression as follows: A Bit Map that is a binary matrix representing the distinct sequences, an SV vector to encode all of the ordered combinations of sequences, an index on the Bit Map to facilitate direct access to sequences and an NB table also associated with the Bit Map to hold the frequencies of the sequences. The algorithm only makes one scan of the database to collect the number of distinct sequences, their frequencies and the number of sequences by size and in doing so allows for the computing of support for each generated sequence. Candidate generation is conducted in the same manner as GSP (Srikant and Agrawal, 1996), PSP (Masseglia et al., 1998) and SPAM (Ayres et al., 2002), except here there is only a need to use the I-extension process since the data has been encoded to single item values. Upon completion of candidate generation, support is determined by first accessing the IBM at the cell where the size of the sequence in question is encoded and then using the SV vector to determine if the candidate is contained in subsequent lines of the IBM. The candidate is then accepted as frequent if the count is larger than a user specified support. The process terminates under the same conditions as the other algorithms, that is when either no candidates can be generated or there are no frequent sequences obtained.

2.4.5.5 LAPIN-SPAM: Last Position Induction Sequential Pattern Mining

Yang and Kitsuregawa (2005) base LAPIN-SPAM (**L**ast **P**osition **IN**duction **S**equential **PA**ttern **M**ining) on the same principles as SPAM (Ayres et al., 2002) with the exception of the methods for candidate verification and counting. Whereas SPAM uses many ANDing operations the authors of the LAPIN strategy avoid this by making the observation that if the last position of item α is smaller than, or equal to, the position of the last item in a sequence s, then item α cannot be appended to s as a $(k+1)$-length sequence extension in the same sequence (Yang, Wang and Kitsuregawa, 2005). This transfers similarly to the I-step extensions. In order to exploit this observation the algorithm maintains an ITEM_IS_EXIST_TABLE in which the last position information is recorded for each specific position and during each iteration of the algorithm there only needs to be a check into this table to ascertain whether the candidate is behind the current position.

2.4.6 Summary of Apriori-based Algorithms

Table 2.7: A summary of apriori-based algorithms.

Algorithm Name	Author	Year	Notes
Candidate Generation: Horizontal Database Format			
Apriori (All, Some, Dynamic Some)	Agrawal and Srikant	1995	
Generalised **S**equential **P**atterns (GSP)	Srikant and Agrawal	1996	Max/Min Gap, Window, Taxonomies
PSP	Masseglia, Cathala and Poncelet	1998	Retrieval optimisations
Sequential **P**attern m**I**ning with **R**egular express**I**on cons**T**raints (SPIRIT)	Garofalakis, Rastogi and Shim	1999	Regular Expressions
Maximal **F**requent **S**equences (MFS)	Zhang, Kao, Yip and Cheung	2001	Based on GSP, uses Sampling
Regular **E**xpression-**H**ighly **A**daptive **C**onstrained **L**ocal **E**xtractor (RE-Hackle)	Albert-Lorincz and Boulicaut	2003	Regular Expressions, similar to SPIRIT
Maximal **S**equential **P**atterns using **S**ampling (MSPS)	Luo and Chung	2004	Sampling
Candidate Generation: Vertical Database Format			
Sequential **PA**ttern **D**iscovery using **E**quivalence classes (SPADE)	Zaki	2001	Equivalence Classes
Sequential **PA**ttern **M**ining (SPAM)	Ayres, Flannick, Gehrke and Yiu	2002	Bitmap representation
LAst **P**osition **IN**duction (LAPIN)	Yang and Kitsuregawa	2004	Uses last position
Cache-based **C**onstrained **S**equence **M**iner (CCSM)	Orlando, Perego and Silvestri	2004	k-way intersections, cache
Indexed **B**it **M**ap (IBM)	Savary and Zeitouni	2005	Bit Map, Sequence Vector, Index, NB table
LAst **P**osition **IN**duction **S**equential **PA**ttern **M**ining (LAPIN-SPAM)	Yang and Kitsuregawa	2005	Uses SPAM, Uses last position

2.5 Projection-based Algorithms

It was recognised early on that the number of candidates that were generated using Apriori type algorithms was exponential in number e.g. if there was one frequent sequence that was 100 long then $2^{100} \approx 10^{30}$ candidates had to be generated to find such a sequence. Although methods have been introduced to alleviate this problem, for example constraints, the candidate generation and prune method suffers greatly under circumstances when the datasets are large. Another inherent problem is the repeated scanning of the dataset to check a large set of candidates by some method of pattern matching. The recognition of these problems in the first instance in Association Mining gave rise to, in that domain, the frequent pattern growth paradigm and the FP-Growth algorithm (Han and Pei, 2000). This was similarly recognised by researchers in the sequence mining domain and algorithms were developed to exploit this methodology.

The frequent pattern growth paradigm is one which removes the need for the candidate generation and prune steps that occur in the Apriori type algorithms and does so by compressing the database representing the frequent sequences into a frequent pattern tree and then dividing this tree into a set of projected databases, which are mined separately (Han and Kamber, 2001).

2.5.1 Pattern Growth

The sequence database shown in Table 2.8 will be used as a running example for both FreeSpan and PrefixSpan.

Table 2.8: A sequence database, S, for use with examples for FreeSpan and PrefixSpan – Pei et al. (2001).

Sequence_id	**Sequence**
10	$\langle a(abc)(ac)d(cf)\rangle$
30	$\langle (ad)c(bc)(ae)\rangle$
30	$\langle (ef)(ab)(df)cb\rangle$
40	$\langle eg(af)cbc\rangle$

2.5.1.1 FreeSpan: Frequent pattern-projected Sequential Pattern Mining

The authors of FreeSpan (**Fre**quent pattern-projected **S**equential **Pa**tter**n** Mining) (Han, Pei, Mortazavi-Asl, Chen, Dayal and Hsu, 2000) submit that the aim of FreeSpan is to integrate the mining of frequent sequences with that of frequent patterns and use projected sequence databases to confine the search and growth of the subsequence fragments (Han et al., 2000).

By using projected sequence databases the method greatly reduces the generation of candidate sub-sequences. The algorithm firstly generates the frequent 1-sequences, L_1, termed an *f_list*, by scanning the sequence database, and then sorts them into support descending order, for example from the sequence database in Table 2.8, f_list = $\langle a : 4, b : 4, c : 4, d : 3, e : 3, f : 3\rangle$. This set can be divided into six subsets: those having item f, those having item e but no f, those having item d but no e or f, and so on until b is reached. This is followed by the construction of a lower-triangular frequent item matrix that is used to generate the length-2 sequential patterns and a set of projected databases. Items that are infrequent such as g are removed and do not take part in the construction of projected databases. The projected databases are then used to generate length-3 and longer sequential patterns. This mining process is outlined below (Han et al., 2000; Pei et al., 2001):

To find the sequential patterns containing only item *a*. This is achieved by scanning the database once, and the two patterns that are found containing only item a are $\langle a\rangle$ and $\langle aa\rangle$

To find the sequential patterns containing the item *b* but no item after *b* in f_list. By constructing the $\{b\}$-projected database and for a sequence α in S containing item b a subsequence α' is derived by removing from α all items after b in f_list. Next α' is inserted into the $\{b\}$-projected database resulting in the $\{b\}$-projected database containing the following four sequences: $\langle a(ab)a\rangle$, $\langle aba\rangle$, $\langle (ab)b\rangle$ and $\langle ab\rangle$. By scanning the projected database one more time all frequent patterns containing b but no item after b in f_list are found, which are $\langle b\rangle$, $\langle ab\rangle$,$\langle ba\rangle$, $\langle (ab)\rangle$.

Finding other subsets of sequential patterns. This is achieved by using the same process as outlined above on the $\{c\}$-, $\{d\}$-, $\ldots$, $\{f\}$-projected databases.

All of the single projected databases are constructed on the first scan of the original database and the process outlined above is performed recursively on all projected databases while there are still longer candidate patterns to be mined.

2.5.1.2 PrefixSpan

The authors of PrefixSpan (**Prefix**-projected **S**equential **Pa**ttern Mining) (Pei, Han, Mortazavi-Asl, Pinto, Chen, Dayal and Hsu, 2001) build on the concept of FreeSpan but instead of projecting sequence databases its general idea is to examine only the prefix subsequences and project only their corresponding postfix subsequences into projected databases. Using the sequence database, S, in Table 2.8 with a $min\_sup = 2$ the mining is as follows.

The length-1 sequential patterns are the same as the f_list in FreeSpan, that is $\langle a\rangle : 4, \langle b\rangle : 4, \langle c\rangle : 4, \langle d\rangle : 3, \langle e\rangle : 3, \langle f\rangle : 3$ and $\langle pattern\rangle : count$ is the sequence and its associated frequency. Once again the complete set can be divided into six subsets according to the six prefixes, those having the prefix $\langle a \rangle$ to those having the prefix $\langle f \rangle$. These are then used to gather those sequences that have them as a prefix. Using the prefix a as an example, when performing this operation it is important to consider a subsequence prefixed by the first occurrence of a, i.e in the sequence $\langle (ef)(ab)(df)cb \rangle$, only the subsequence $\langle (\_b)(df)cb \rangle$ should be considered. $(\_b)$ means the last element in the prefix, in this case a, together with b form an element.

The sequences in S containing $\langle a \rangle$ are next projected with respect to $\langle a \rangle$ to form the $\langle a \rangle$-projected database, and then those with respect to $\langle b \rangle$ and so on. By way of example the $\langle b \rangle$-projected database consists of four postfix sequences: $\langle (\_c)(ac)d(cf) \rangle$, $\langle (\_c)(ae) \rangle$, $\langle (df)cb \rangle$ and $\langle \_c \rangle$. By scanning these projected databases all of the length-2 patterns having the prefix $\langle b \rangle$ can be found. These are $\langle ba \rangle$:2, $\langle bc \rangle$:3, $\langle (bc) \rangle$:3, $\langle bd \rangle$:2, and $\langle bf \rangle$:2. This process is then conducted recursively by partitioning the patterns as above to give those having prefix $\langle ba \rangle$, $\langle bc \rangle$ and so on, and these are mined by constructing projected databases and mining each of them recursively. This done for each of the remaining single prefixes, $\langle c \rangle$, $\langle d \rangle$, $\langle e \rangle$ and $\langle f \rangle$ respectively.

The major benefit of this approach is that, no candidate sequence needs to be generated or tested that does not exist in a projected database, that is PrefixSpan only grows longer sequential patterns from shorter frequent ones, thus making the search space much smaller. This results in the major cost being the construction of the projected databases, however this can be alleviated by two optimisations. The first, by using a bi-level projection method to reduce the size and number of the projected databases, and second a pseudo-projection method to reduce the cost when a projected database can be wholly contained in main memory.

2.5.1.3 SLPMiner

The SLPMiner algorithm by Seno and Karypis (2002) follows the projection-based approach for generating frequent sequences but uses a *length-decreasing support* constraint for the purpose of finding not only short sequences with high support but also long sequences with a lower support. To this end the authors extended their model introduced for length-decreasing support in association rule mining (Seno and Karypis, 2001), to use the length of a sequence not an itemset. This is formally defined as: Given a sequential database D and a function $f(l))$ that satisfies $1 \geq f(l) \geq f(l+1) \geq 0$ for any positive integer l, a sequence s is frequent if and only if $\sigma_D(s) \geq f(|s|)$. Under this constraint a sequence can be frequent while its subsequences are infrequent so pruning can not be solely performed using the downward closure principle and therefore three types of pruning are introduced which are derived from knowledge about the length at which an infrequent sequence becomes frequent. This knowledge about the increase in length is called the *smallest valid extension* property or *SVE* and uses the fact that if a line is drawn parallel to the x-axis at $y = \sigma_D(s)$ until it intersects the support curve

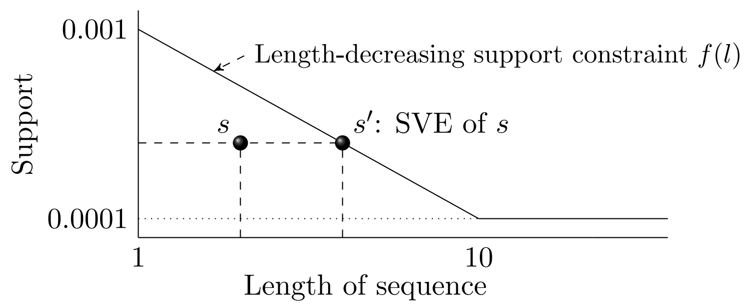


Figure 2.5: A typical length-decreasing support constraint and the smallest valid extension (SVE) property – Seno and Karypis (2002).

then the length of the extended sequence is the minimum length the original sequence must attain before it becomes frequent. Figure 2.5 shows both the length-decreasing support function and the SVE property.

The first of the three methods of pruning is called *sequence pruning* and removes certain sequences from the projected databases, which occur at every node in the prefix tree. A sequence s can be pruned if the value of the length-decreasing function $f(|s| + |p|)$, where p is the pattern represented at a particular node, is greater than the value of the support for p in the database D. The second method is called *item pruning* and essentially removes some of the infrequent items from short sequences. Since the inverse function of the length-decreasing support function yields the length of a particular sequence, an item i can be pruned by determining if the length of a sequence plus the length of the prefix at a node, $|s| + |p|$, is less than the value of the inverse function of the support for the item i in the projected database D' at the current node. The third method called *min-max pruning* eliminates a complete projected database. This is achieved by splitting the projected database into two subsets and determining if each of them is too small to be able to support any frequent sequential patterns. If this is the case then the entire projected database can be removed.

Although these pruning methods are elaborated for use with SLPMiner the authors do concede that most of the methods can be incorporated into other algorithms and cite PrefixSpan (Pei et al., 2001) and SPADE (Zaki, 2001*b*) as two possibilities.

2.5.2 Summary of Pattern Growth Algorithms

Table 2.9: A summary of pattern growth algorithms.

Algorithm Name	Author	Year	Notes
Pattern Growth			
FREqu**E**nt pattern-projected **S**equential **PA**tter**N** mining (FreeSpan)	Han, Pei, Mortazavi-Asl, Chen, Dayal and Hsu	2000	Projected sequence database
PREFIX-projected **S**equential **PA**tter**N** mining (PrefixSpan)	Pei, Han, Mortazavi-Asl, Pinto, Chen, Dayal and Hsu	2001	Projected prefix database
Sequential pattern mining with **L**ength-decreasing su**P**port (SLPMiner)	Seno and Karypis	2002	Length-decreasing support

2.6 Temporal Sequences

The data used for sequence mining is not limited to data stored in overtly temporal or longitudinally maintained datasets. Examples include genome searching, web logs, alarm data in telecommunications networks, population health data etc. In such domains data can be viewed as a series of events occurring at specific times and therefore the problem becomes a search for collections of events that occur frequently together. Solving this problem requires a different approach, and several types of algorithm have been proposed for different domains.

2.6.1 Problem Statement and Notation for Episode and Event-based Algorithms

The first algorithmic framework developed to mine datasets that were deemed to be episodic in nature was introduced by Mannila, Toivonen and Verkamo (1995). The task addressed in this work was to find all episodes that occur frequently in an event sequence, given a class of episodes and an input sequence of events. An episode was defined to be:

> *"... a collection of events that occur relatively close to each other in a given partial order*, and ... *frequent episodes as a recurrent combination of events" (Mannila et al., 1995)*

The notation used is as follows.

E is a set of event types and an event is a pair (A, t), where $A \in \mathbf{E}$ is an event type and t is an integer (time / occurrence) of the event. There are no restrictions on the number of attributes that an event type may contain, or be made up of, but the original work only considered single values with no loss of generality. An event sequence $\mathbf{s}$ on $\mathbf{E}$ is a triple (s, T_s, T_e), where $s = \langle (A_1, t_1), (A_2, t_2), \ldots, (A_n, t_n) \rangle$ is an ordered sequence of events such that $A_i \in \mathbf{E}$ for all $i = 1, \ldots, n$, and $t_i \leq t_{i+1}$ for all $i = 1, \ldots, n-1$. Further $T_s < T_e$ are integers, T_s is called the starting time and T_e the ending time, and $T_s \leq t_i < T_e$ for all $i = 1, \ldots, n$.

For example Figure 2.6 depicts the event sequence $\mathbf{s} = (s, 29, 68)$, where $s = \langle (A, 31), (F, 32), (B, 33), (D, 35), (C, 37), \ldots, (F, 67) \rangle$.

In order for episodes to be considered interesting they must occur close enough in time, which can be defined by the user through a time window of a certain width. These time windows are partially overlapping slices of the event sequence and the number of windows that an episode must occur in to be considered frequent, *min_freq*, is also defined by the user. Notationally a window on event sequence $\mathcal{S} = (s, T_s, T_e)$ is an event sequence $\mathbf{w} = (w, t_s, t_e)$ where $t_s < T_e$, $t_e > T_s$ and w consists of those pairs (A, t) from s where $t_s \leq t < t_e$. The time span $t_e - t_s$ is called the width of the window $\mathbf{w}$ denoted $\mathit{width}(\mathbf{w})$. Given an event sequence $\mathbf{s}$ and an integer *win*, the set of all windows $\mathbf{w}$ on $\mathbf{s}$ such that $\mathit{width}(\mathbf{w}) = \mathit{win}$ is denoted by $\mathcal{W}(\mathbf{s}, \mathit{win})$. Given the event sequence $\mathbf{s} = (s, T_s, T_e)$ and window width *win* the number of windows in $\mathcal{W}(\mathbf{s}, \mathit{win})$ is $T_e - T_s + \mathit{win} - 1$.

An *episode* $\varphi = (V, \leq, g)$ is a set of nodes V, a partial order $\leq$ on V, and a mapping $g : V \rightarrow E$ associating each node with an event type. The interpretation of an episode is that it has to occur in the order described by $\leq$.

There are two types of episodes considered:

Serial – where the partial order relation $\leq$ is a total order resulting in for example an event A preceding an event B which precedes event C. This is shown in Figure 2.7(a).

Parallel – where there are no constraints on the relative order of events, that is if the partial order relation $\leq$ is trivial: $(x \nleq y \; \forall \; x \neq y)$. This is shown in Figure 2.7(b).

The *frequency* of an episode is defined to be the number of windows in which the episode occurs. That is, given an event sequence $\mathbf{s}$ and a window width *win*, the

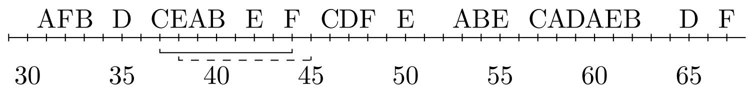


Figure 2.6: An example event sequence and two windows of width 7.

frequency of an episode φ in $\mathbf{s}$ is:

$$fr(\varphi, \mathbf{s}, win) = \frac{|\{\mathbf{w} \in \mathcal{W}(\mathbf{s}, win) | \varphi \ occurs \ in \ \mathbf{w}\}|}{|\mathcal{W}(\mathbf{s}, win)|}$$

So given a *frequency threshold min_freq*, φ is frequent if $fr(\varphi, \mathbf{s}, win) \geq min\_fr$. The task is to discover all frequent episodes (serial or parallel) from a given class $\mathcal{E}$ of episodes.

2.6.2 WINEPI

The algorithm named WINEPI was introduced by Mannila et al. (1995) and its task was: given an event sequence $\mathbf{s}$, a set $\mathcal{E}$ of episodes, a window width, *win*, and a frequency threshold *min_fr*, find the set of frequent episodes $\mathcal{F}$ with respect to $\mathbf{s}$, *win* and *min_freq* denoted as $\mathcal{F}(\mathbf{s}, win, min\_fr)$. It follows a traditional levelwise (breadth-first) search starting with the general episodes (one event). At each subsequent level the algorithm first computes a collection of candidate episodes, checks their frequency against *min_fr* and if it is greater it is added to a list of frequent episodes. This cycle continues until there are no more candidates generated or no candidates meet the minimum frequency. This is a typical Apriori-like algorithm under which the downward closure principal holds – if α is frequent then all sub-episodes $\beta \preceq \alpha$ are frequent.

In order to recognise episodes in sequences two methods are necessary; one for parallel episodes and one for serial episodes. However since both of these methods share a similar feature, namely that two adjacent windows are typically very similar to each other, the recognition can be done incrementally. For parallel episodes a count is maintained that indicates how many events are present in any particular window and when this count reaches the length of episode (at any given iteration of the algorithm) the index of the window is saved. When the count then decreases, indicating that the episode is not entirely in the window and occurrence field is incremented by the number of windows in which the episode was fully contained. Serial episodes are recognised using state automata that acccept the candidate episodes and ignore all other input (Mannila et al., 1995). A new instance of the automata is initialised for each serial

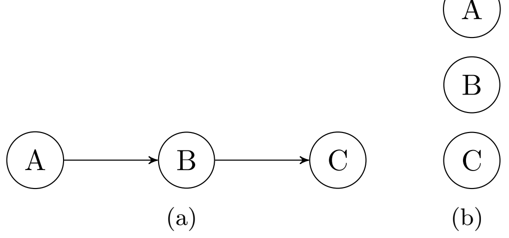


Figure 2.7: Depiction of a serial and parallel episode – Mannila et al. (1995).

episode every time the first event appears in the window, and the automata is removed when this same event leaves the window. To count the number of occurrences an automata is said to be in the accepting state when the entire episode is in the window and each time this occurs the automata is removed and its window index is saved. When there are no automata left for the particular episode the occurrence is incremented by the number of saved indexes.

A following paper by Mannila and Toivonen (1996) extended this work by only considering minimal occurrences of episodes, which are defined as follows. Given an episode φ and an event sequence $\mathbf{s}$, then the interval $[t_s, t_e)$ is a *minimal occurrence* of φ in $\mathbf{s}$ if (1) φ occurs in the window $\mathbf{w} = (w, t_s, t_e)$ on $\mathbf{s}$, and if (2) φ does not occur in any proper subwindow on $\mathbf{w}$, that is not in any window $\mathbf{w}' = (w', t'_s, t'_e)$ on $\mathbf{s}$ such that $t_s \leq t'_s, t'_e \leq t_e$ and $\mathit{width}(\mathbf{w}') < \mathit{width}(\mathbf{w})$.

The algorithm MINEPI that took advantage of this new formalism follows the same basic principles as WINEPI with the exception that minimal occurrences of candidate episodes are located during the candidate generation phase of the algorithm. This is performed by selecting from a candidate episode φ two subepisodes φ_1 and φ_2 such that φ_1 contains all events of φ except the last one and φ_2 contains all events except the first one. From these two subepisodes the minimal occurrences of φ are found using the following specification (Mannila and Toivonen, 1996):

$$\begin{aligned} mo(\varphi) = \{[t_s, u_e) \quad | \quad & \text{there are } [t_s, t_e) \in mo(\varphi_1) \text{ and } [u_s, u_e) \in mo(\varphi_2) \\ & \text{such that } t_s < u_s, t_e < u_e \text{ and } [t_s, u_e) \text{ is minimal}\} \end{aligned}$$

These minimal occurrences are then used to obtain confidences with respect to episode rules without the need to rescan the data. Both of these algorithms and also algorithms to perform the task of rule generation were brought together in a paper published by Mannila, Toivonen and Verkamo in 1997. Typically the WINEPI produces rules that are like those of Association Rules for example: if **ABC** is a frequent sequence then **AB** $\Rightarrow$ **C** (confidence γ) simply states that if **A** occurs followed by **B** then some time later **C** occurs. The MINEPI and its new formalism of episodes allows for more useful rule formulations for example: “Department Home Page”, “Semester I 2005” [15s] $\Rightarrow$ “Classes in Semester I 2005” [30s] (confidence 0.83). This is read as if a person navigated to the Department Home Page followed by the Semester I page within 15 seconds then within the next 30 seconds they would navigate to the Semester I Classes page 83% of the time. These algorithms represent the seminal work in the field of episode detection in linear temporal sequences in the data mining domain.

Table 2.10: Vertical layout of the event sequences for the PROWL algorithm – Huang et al. (2004).

Event	Time List	Projected Window List
A	1, 4, 7, 8, 11, 14	2, 5, 8, 9, 12, 15
B	3, 6, 9, 12, 16	4, 7, 10, 13
C	2, 10, 15	3, 11, 16
D	5, 13	6, 14

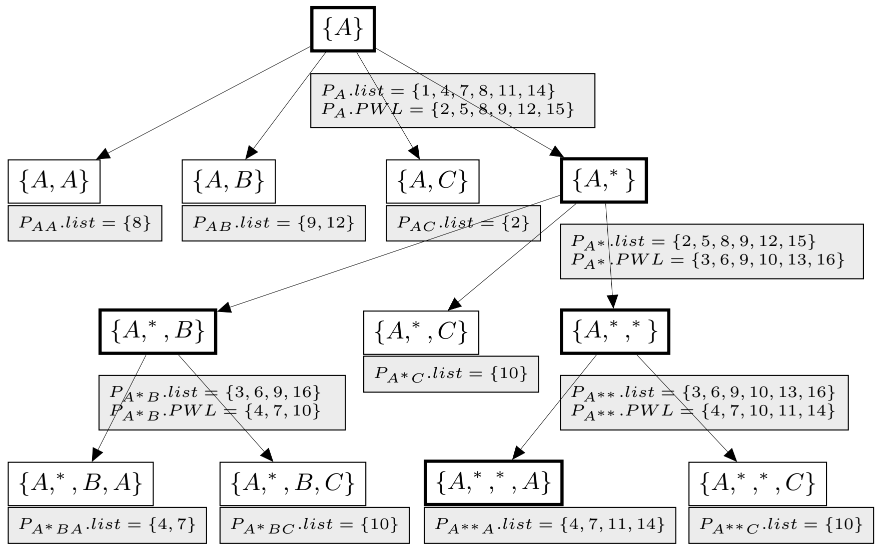
2.6.3 PROWL

Huang, Chang and Lin (2004) introduced the algorithm PROWL (**Pro**jected **W**indow **L**ists) to mine inter-transaction rules and used the term *frequent continuities* to distinguish their rules from those of intra-transaction or association rules. Further to this they introduced the symbol "*" for the *don't care* position in the sequence to allow partial periodicity. PROWL is a two phase algorithm for mining such frequent continuities and utilises a projected window list and a depth first enumeration of the search space.

The definition of a sequence follows the pattern from Section 2.6.1 however, the authors do not define the complete event sequence as a triple but only go as far as defining it as a tuple of the form (tid, x_i) where tid is a time instant and x_i is an event. The *continuity pattern* is defined to be a nonempty sequence with window W, $P = (p_1, p_2, \ldots, p_w)$ where p_1 is an event and the others can either be events or the * (*don't care*) token. A continuity pattern is called an *i-continuity* or has length i if exactly i positions in P contain an event. For example, $\{A,^*,^*\}$ is a 1-continuity and $\{A,^*,C\}$ is a 2-continuity of length 2. They define the problem to be the discovery of all patterns P with a window W where any subsequence of W in S supports P.

The algorithm uses a Projected Window List (PWL) to grow the sequences where a PWL is defined as $P = \{e_1, e_2, \ldots, e_k\}$ and $P.PWL = \{w_1, w_2, \ldots, w_k\}, w_i = e_i + 1$ for $1 \leq i \leq k$. By concatenating each event with the events in the PWL longer sequences can be generated and then the number of event in the concatenated list can be checked against the given support and accepted as frequent if it equals or exceeds the value. This process is applied recursively until the projected window lists become empty or the window of a continuity is greater than the maximum window. Table 2.10 shows the vertical layout of the event sequences and Figure 2.8 shows the recursive process for the continuity pattern $\{A\}$.

Support and confidence are defined in a similar fashion to association rule mining and therefore rules can be formed which are an implication of the form $X \Rightarrow Y$, where X and Y are continuity patterns with window w_1 and w_2 respectively and the concatenation $X \cdot Y$ is also a continuity pattern with window $w_1 + w_2$. This leads to support being equal to the support of the concatenation divided by the number of transactions in the event sequence, and confidence as the support of the concatenation divided by the support of either continuity depending on the required implication.



$\{A\}$
$P_A.list = \{1, 4, 7, 8, 11, 14\}$
$P_A.PWL = \{2, 5, 8, 9, 12, 15\}$
$\{A, A\}$
$P_{AA}.list = \{8\}$
$\{A, B\}$
$P_{AB}.list = \{9, 12\}$
$\{A, C\}$
$P_{AC}.list = \{2\}$
$\{A, ^*\}$
$P_{A*}.list = \{2, 5, 8, 9, 12, 15\}$
$P_{A*}.PWL = \{3, 6, 9, 10, 13, 16\}$
$\{A, ^*, B\}$
$P_{A*B}.list = \{3, 6, 9, 16\}$
$P_{A*B}.PWL = \{4, 7, 10\}$
$\{A, ^*, C\}$
$P_{A*C}.list = \{10\}$
$\{A, ^*, ^*\}$
$P_{A**}.list = \{3, 6, 9, 10, 13, 16\}$
$P_{A**}.PWL = \{4, 7, 10, 11, 14\}$
$\{A, ^*, B, A\}$
$P_{A*BA}.list = \{4, 7\}$
$\{A, ^*, B, C\}$
$P_{A*BC}.list = \{10\}$
$\{A, ^*, ^*, A\}$
$P_{A**A}.list = \{4, 7, 11, 14\}$
$\{A, ^*, ^*, C\}$
$P_{A**C}.list = \{10\}$

Figure 2.8: The recursive process for the continuity pattern $\{A\}$ – Huang et al. (2004).

2.6.4 Event-Oriented Patterns

Sun, Orlowska and Zhou (2003) treat the problem of mining sequential patterns by mining temporal event sequences that lead to a specific event called a *target event*, rather than finding all frequent patterns. They discuss two types of patterns; an *Existence pattern* α with a temporal constraint T that is a set of event values and a *Sequential pattern* β also with a temporal constraint T. Their method is used to produce rules of the type $r = \left\{LHS \xrightarrow{T} e\right\}$, where e is a *target event* value and LHS is a pattern of type α or β. T is a time interval that specifies both the temporal relationship between LHS and e and also the temporal constraint pattern of LHS. To find the LHS patterns they first locate all of the *target events* in the event sequence and create a timestamp set by using a T-sized window extending back from the target event. The sequence fragments (f_i) that are created from this process is called the *dataset* of target event e (see Figure 2.9 for an example).

The support for a rule is then given by the number of sequence fragments containing a specified pattern divided by the total number of sequence fragments in the event sequence. As the authors point out, this method finds frequent sequences that occur not only before target events but elsewhere in the event sequence and therefore it cannot be concluded that these patterns relate to the given target events. In order to prune these non-related patterns a confidence measure is introduced which evaluates the number of times the pattern actually leads to the target event divided by the total

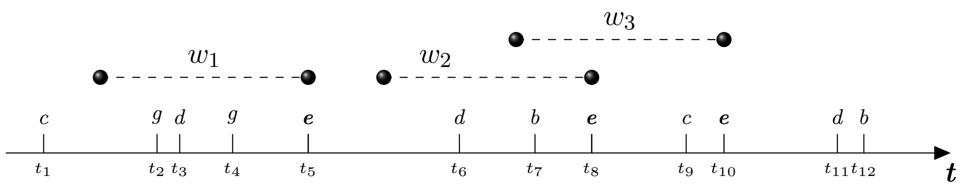


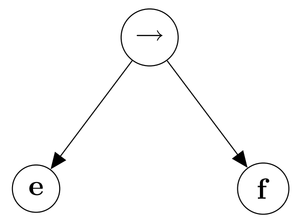
Figure 2.9: Sequence fragments of size T ($w_1 = \langle(g, t_2), (d, t_3), (g, t_4)\rangle$, $w_2 = \langle(d, t_6), (b, t_7)\rangle$, $w_3 = \langle(b, t_7), (e, t_8), (c, t_9)\rangle$) for the target event $\boldsymbol{e}$ in an event sequence s – Sun, Orlowska and Zhou (2003).

number of times the pattern occurs. Both of these values are defined as window sets. The formal definition of the problem is then: Given a sequence s, target event value e, window size T, two thresholds s_0 and c_0, find the complete set of rule $r = \left\{LHS \xrightarrow{T} e\right\}$ such that $supp(r) \geq s_0$ and $con(r) \geq c_0$.

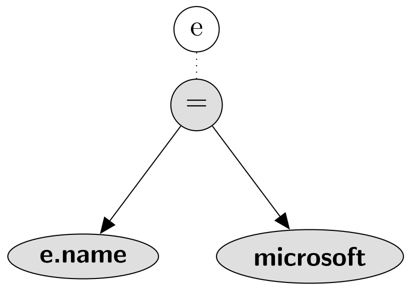
2.6.5 Pattern Directed Mining

Guralnik, Wijesekera and Srivastava (1998) present a framework for the mining of frequent episodes using a pattern language for specifying episodes of interest. A *sequential pattern tree* is used to store the relationships specified by the pattern language and then a standard bottom-up mining algorithm can be used to generate the frequent episodes. The specification for mining follows the notation described earlier (see Section 2.6.1) with the addition of a *selection constraint* on an event that is a unary predicate $\alpha(e, a_i)$ on a domain D_i where a_i is an attribute of e, and a *join constraint* on events e and f, which is a binary predicate $\beta(e.a_i, f.a_j)$ on a domain $D_i \times D_j$ where a_i and a_j are attributes of e and f respectively. They also define a *sequential pattern* as a combination of partially ordered event specifications constructed from both selection and join constraints. To facilitate the mining process the user-specified patterns are stored in a *Sequential Pattern Tree* (*SP Tree*) where the leaf nodes represent events and the interior nodes represent ordering constraints. Furthermore each node holds events matching constraints of that node and attached to the node is a boolean expression that represents the attribute constraints associated with the node (see Figure 2.10 for two examples).

The mining algorithm constructs the frequent episodes in a bottom-up fashion by taking an SP Tree T and a sequence of events S and at the leaf level matching events against any selection constraints and pruning out those that do not match. The interior nodes merge events of left and right children according to any ordering and join constraints, again pruning out those that do not match the node specifications. The process is continued recursively until all events in S have been visited.



(a) SP Tree for the user specified pattern $e \rightarrow f$.



(b) SP Tree for the user specified pattern $e\{e.name = \text{‘}microsoft\text{’}\}$.

Figure 2.10: Two examples of SP Trees – Guralnik et al. (1998).

2.6.6 Summary of Temporal Sequence Algorithms

Table 2.11: A summary of temporal sequence algorithms.

Algorithm Name	**Author**	**Year**	**Notes**
Candidate Generation: Episodes			
WINEPI	Mannila, Toivonen and Verkamo	1995, 1996	State automata, Window
WINEPI, MINEPI	Mannila, Toivonen and Verkamo	1997	State automata, Window, Maximal
Pattern Directed Mining	Guralnik, Wijesekera and Srivastava	1998	Pattern language
Event-Oriented Patterns	Sun, Orlowska and Zhou	2003	Target events
Pattern Growth: Episodes			
PROjected **W**indow **L**ists (PROWL)	Huang, Chang and Lin	2004	Projected window lists

2.7 Extensions

2.7.1 Closed Frequent Patterns

The mining of frequent patterns for both sequences and itemsets has proved to be valuable but in some cases, particularly when using candidate generation and test techniques, and when small supports are used the performance of algorithms can degrade dramatically. This has led, in the mining of frequent itemsets, to produce algorithms, such as CHARM (Zaki and Hsiao, 2002), CLOSET (Pei, Han and Mao, 2000), CLOSET+ (Wang, Han and Pei, 2003) and CARPENTER (Pan, Cong, Tung, Yang and Zaki, 2003), that produce *frequent closed itemsets*, which overcomes some of these difficulties and produces a far smaller yet still complete set of results. In the sequence mining field there are few algorithms that deal with this problem, however those that do are, to some degree, extensions of algorithms that mine the complete set of sequences with improvements in search space pruning and traversal techniques.

A closed subsequence is a sequence that contains no super-sequence with the same support. Formally this is defined to be: Given a set of **frequent sequences**, FS, which includes all of the sequences whose support is greater than or equal to *min_supp*, then the set of **closed sequences**, CS, is $CS = \{\varphi | \varphi \in FS \textit{ and } \nexists \gamma \in FS \textit{ such that } \varphi \sqsubseteq \gamma \textit{ and support}(\varphi) = \textit{support}(\gamma)\}$. This results in $CS \subseteq FS$ and the problem of **closed sequence mining** becomes to find CS above a minimum support.

2.7.1.1 CloSpan

Closed **S**equential **pa**tter**n** mining (Yan, Han and Afshar, 2003) follows the candidate generation and test paradigm and stores the generated sets of candidates in a hash-indexed result-tree following which post-pruning is conducted to produce the closed set of frequent sequences. The algorithm uses a lexicographic sequence tree to store the generated sequences using both I-extension and S-extension mechanisms and the same search tree structure as PrefixSpan to discover all of the frequent sequences (closed and non-closed). Pruning is conducted using *early termination by equivalence*, a *backward sub-pattern* and *backward super-pattern* method.

2.7.1.2 BIDE

BI-Directional **E**xtension based frequent closed sequence mining, an algorithm by Wang and Han (2004), mines frequent closed sequences without the need for candidate maintenance. Furthermore pruning is made more efficient by adopting a *BackScan* method and the *ScanSkip* optimisation technique. The terminology used to define the problem is identical to that of CloSpan. The mining is performed using a method called *BI-Directional Extension* where the forward directional extension is used to grow all of the prefix patterns and also check for closure of these patterns, while the backward directional extension is used to both check for closure and prune the search space.

2.7.1.3 ClosedPROWL

Closed PROjected **W**indow **L**ists (Huang, Chang and Lin, 2005) is an extension to the PROWL algorithm (Huang et al., 2004) that mines the set of closed frequent continuities. A **closed frequent continuity** is a continuity which has no proper super-continuity with the same support. This is formally defined as: Given two continuities

$P = [p_1, p_2, \ldots, p_u]$ and $P' = [p'_1, p'_2, \ldots, p'_v]$ then P is a **super-continuity** of P' (and therefore P' is a **sub-continuity** of P) if and only if, for each non-* pattern p'_j $(1 \leq j \leq w)$, $p'_j \subseteq p_{j+o}$ is true for some integer o and the integer o is called the offset of P. This added constraint marks the only difference in the mining compared to that of PROWL.

2.7.2 Hybrid Methods

2.7.2.1 DISC

DIrect **S**equence **C**omparison (Chiu, Wu and Chen, 2004) combines the candidate sequence and pruning strategy, database partitioning and projection with a strategy that recognises the frequent sequences of a specific length k without having to compute the support of the non-frequent sequences. This is achieved by using k-sorted databases to find all frequent k-sequences, which skips most of the non-frequent k-sequences by checking only the conditional k-minimum subsequences. The basic DISC strategy is as follows. The first position in a k-sorted database is called the *candidate k-sequence* and denoted by α_1, and given a minimum support count of δ, the k-minimum subsequence at the δ-th position is denoted α_δ. These two candidates are compared and if they are equal then α_1 is frequent since the first δ customer sequences in the k-sorted database take α_1 as their k-minimum subsequence and if $\alpha_1 < \alpha_\delta$ then α_1 is non-frequent and all subsequences up to and including α_δ can be skipped. This process is then repeated for the next k-minimum subsequence in the resorted k-sorted database. The proposed algorithm uses a partitioning method similar to SPADE (Zaki, 2001*b*), SPAM (Ayres et al., 2002) and PrefixSpan (Pei et al., 2001) for generating frequent 2-sequences and 3-sequences and then employs the DISC strategy to generate the remaining frequent sequences.

2.7.3 Approximate Methods

2.7.3.1 ApproxMAP

Approximate **M**ultiple **A**lignment **P**attern mining (Kum, Pei and Wang, 2002) mines consensus patterns from large databases by a two step strategy. A consensus pattern is one which is shared by many sequences and covers many short patterns but may not be exact. In the first step sequences are clustered by similarity and then from these clusters consensus patterns are mined directly through a process of multiple alignment. To enable the clustering of sequences a modified version of the *hierarchical edit distance* metric is used in a density-based clustering algorithm. Once this stage is completed, each cluster contains similar sequences and the summary of the general pattern of each cluster and a trend analysis begins. First the density for each sequence is calculated and the sequences in each cluster are sorted in density descending order, second the multiple alignment process is carried out using a weighted sequence, which records the statistics of the alignment of the sequences in the cluster. Consensus patterns are then selected based on the parts of the weighted sequence that are shared by most sequences in the cluster using a *strength threshold* (similar to support) to remove items whose strength is less than the threshold.

2.7.3.2 ProMFS

Probabilistic **M**ining **F**requent **S**equences, an algorithm by Tumasonis and Dzemyda (2004), is based on estimated statistical characteristics of the appearance of elements of the sequence being mined and their order. The main thrust of this technique is to generate a much smaller sequence based on these estimated characteristics and then make decisions about the original sequence based on analysis of the shorter one. Once the generation of the shorter sequence has concluded then analysis can be undertaken using GSP (Srikant and Agrawal, 1996) or any other suitable algorithm.

2.7.4 Parallel Algorithms

With the discovery of sequential patterns becoming increasingly useful and the size of the available datasets becoming increasingly large there is a growing need for both efficient and scalable algorithms. To this end a number of algorithms have been developed to take advantage of distributed memory parallel computer systems and in doing so their computational power.

Demiriz and Zaki (2002) developed an algorithm called webSPADE that modified the original SPADE algorithm (Zaki, 2001*b*) to enable it to be run on shared-memory parallel computers and was developed to analyse web log data that had been cleaned and stored in a data warehouse.

Guralnik and Karypis (2004) developed two static distribution algorithms, based on the tree projection algorithm for frequent itemset discovery by Agrawal, Aggarwal and Prasad (1999), to take advantage of either data- or task-parallelism. The data-parallel algorithm partitions the database across different processors whereas the task-parallel algorithm partitions the lattice of frequent patterns, and both take advantage of a dynamic load-balancing scheme.

The Par-CSP (**Par**allel **C**losed **S**equential **P**attern mining) algorithm developed by Cong, Han and Padua (2005) is based on the BIDE algorithm (Wang and Han, 2004) and mines closed sequential patterns on a distributed memory system. The process is divided into independent tasks to minimise inter-processor communication while using a dynamic scheduling scheme to reduce processor idle time. This algorithm also uses a load-balancing scheme.

2.7.5 Other Methods

Antunes and Oliveira (2003) use a modified string edit distance algorithm to detect whether a given sequence approximately matches a given constraint, expressed as a regular language, based on a *generation cost*. The algorithm to perform this is called ϵ-*accepts* and can be used with any algorithm that uses regular expressions as a constraint mechanism i.e. the SPIRIT (Garofalakis et al., 1999) family of algorithms .

Yang, Wang, Yu and Han (2002) have developed a method that uses a *compatibility matrix*, which is a conditional probability matrix that specifies the likelihood of symbol substitution, in conjunction with a *match* metric ("aggregated amount of occurrences") to discover long sequential patterns using primarily gene sequence data as the input.

Hingston (2002) viewed the problem of sequence mining as one of inducing a finite state automaton model that is a compact summary of the sequential data set in the form of a generative model or grammar and then using that model to answer queries about the data.

All of the algorithms discussed thus far are either 1- or 2-dimensional but some research has been conducted on multi-dimensional sequence mining. Yu and Chen (2005) introduce two algorithms, the first of which modifies the traditional Apriori algorithm (Agrawal and Srikant, 1994) and the second by modifying the PrefixSpan algorithm (Pei et al., 2001). Pinto, Han, Pei, Wang, Chen and Dayal (2001) propose a theme of multi-dimensional sequential pattern mining, which integrates both multi-dimensional analysis and sequential pattern mining and further explores feasible combinations of these two methods and makes recommendations on the proper selection with respect to particular datasets.

2.7.6 Time Series Mining

Data mining of time series datasets includes not only includes sequence mining but also clustering, classification, and association mining (Das, Lin, Mannila, Renganathan and Smyth, 1998; Guralnik and Srivastava, 1999; Höppner, 2001*a*; Keogh, Chu, Hart and Pazzani, 1993; Lin, Keogh, Lonardi and Chiu, 2003). As would be expected, the constraints available are those appropriate for the form of mining and the rules that emerge from this type of analysis are similarly aligned with the mining method chosen. For the case of sequences, typical rules are based on (*apriori* supplied) calendric, or cyclic patterns and have some similarity to those addressed in this thesis.

2.8 Incremental Mining Algorithms

The ever increasing amount of data being collected has also introduced the problem of how to handle the addition of new data and the possible non-relevance of older data, and in association rule mining there have been many methods proposed to deal with this. In this respect sequence mining is no different and similar techniques have also been developed. The following are the main contributors in this area.

2.8.1 Incremental Discovery of Sequential Patterns

This was an early algorithm produced by Wang and Tan (1996) where the database is considered to be a single long sequence of events in which a frequent sequential pattern is a subsequence of this and an update is either the insertion or deletion of a subsequence at either end, or the database is a collection of sequences and a frequent sequential pattern is a subsequence of the these and an update is the insertion or deletion of a complete sequence. The mining is performed using a *compact suffix tree*.

2.8.2 ISM: Interactive Sequence Mining

Interactive **S**equence **M**ining (Parthasarathy, Zaki, Ogihara and Dwarkadas, 1999) is built on the SPADE algorithm (Zaki, 2001*b*) and aims to minimise the I/O and computational requirements inherent in incremental updates and is concerned with both the addition of new customers and new transactions. It consists of two phases in which the first is for updating the supports of elements in *Negative Border*[3] (*NB*) and *Frequent Sequences* (*FS*) and the second is for adding to *NB* and *FS* beyond what was done in the first phase. The algorithm also maintains an *Incremental Sequence Lattice ISL*, which consists of all of the frequent sequences and all sequences in the *NB* in the original database, along with the supports for each.

2.8.3 ISE: Incremental Sequence Extraction

Incremental **S**equence **E**xtraction (Masseglia, Poncelet and Teisseire, 2000) is an algorithm for updating frequent sequences where new transactions are appended to customers who already exist in the original database. This is achieved by using information from the frequent sequences from a previous mining run. Firstly, if k is the longest frequent sequence in *DB* (the original database), find all new frequent sequences of size $j \leq (k+1)$ considering three type of sequences;

1. Those that are embedded in DB that become frequent due to an increase in support due to the increment database *db*,

[3]The *negative border* is the collection of all sequences that are not frequent but both of whose generating subsequences are frequent.

2. frequent sequences embedded in *db*,
   and
3. those sequences of *DB* that become frequent by the addition of an item from a transaction in *db*,

and secondly find all frequent sequences of size $j > (k + 1)$. In the first pass the algorithm finds frequent 1-sequences by considering all items in *db* that occur once and then by considering all items in *DB* to determine which items of *db* are frequent in U $(DB \cup db)$, which are denoted L_1^{db}. Once this is accomplished these frequent 1-sequences can be used as seeds to discover new frequent sequences of the following types:

1. previous frequent sequences in *DB* which can be extended by items of L_1^{db}
2. frequent sub-sequences in *DB* which are predecessor items in L_1^{db}
3. candidate sequences generated from L_1^{db}

This process is continued iteratively (since after the first step frequent 2-sequences are obtained to be used in the same manner as described above) until no more candidates are generated.

2.8.4 IUS/DUS: Incrementally/Decreasingly Updating Sequences

Incrementally/**D**ecreasingly **U**pdating **S**equences are two algorithms written by Zheng, Xu, Ma and Lv (2002) for discovering frequent sequences when new data is added to an original database and for deleting old sequences that are no longer frequent after the addition of new data respectively. The IUS algorithm makes use of the negative border sequences and the frequent sequences of the original database in a similar fashion to the ISM algorithm (Parthasarathy et al., 1999) but introduces an added support threshold for the negative border. This added support metric (*Min_nbd_supp*) means that only those sequences whose support is between the original *min_supp* value and *Min_nbd_supp* can be members of the *negative border* set. Both prefixes and suffixes of the original frequent sequences are extended to generate candidates for the updated database. DUS recomputes the *negative border* and frequent sequences in the updated database based on the results of a previous mining run and prunes accordingly.

2.8.5 GSP+ and MFS+

Zhang, Kao, Cheung and Yip (2002) created two algorithms based on the non-incremental versions from which they are named – GSP (Srikant and Agrawal, 1996) and MFS (Zhang et al., 2001). The problem of incremental maintenance here is one of updates via insertions of new sequences and deletions via removal of old sequences and follows a similar pattern to previous algorithms by taking advantage of the information from a previous mining run. The modifications to both GSP and MFS are based on three observations (Zhang et al., 2002):

1. The portion of the database that has not been changed, which in most cases is the majority, need not be processed since the support of a frequent sequence in the updated database can be deduced by only processing the inserted and deleted customer sequences.
2. If a sequence is infrequent in the old database then in order for it to become frequent in the updated database then its support in the inserted portion has to be large enough or small enough for the case of deletions and therefore it is possible to determine whether a candidate should be considered by just using the portion of the database that has changed.
3. If both the old database and the new updated database share a non-trivial set of common sequences, then the set of frequent sequences that have already been mined will give a good indication of likely frequent sequences in the updated database.

These observations are used to develop pruning tests in order to minimise the scanning of the old database to count the support of candidate sequences.

2.8.6 IncSpan: Incremental Sequential Pattern mining

Incremental **S**equential **Pa**tter**n** mining (Cheng, Yan and Han, 2004) is based on the non-incremental sequential pattern mining algorithm PrefixSpan (Pei et al., 2001) and uses a similar approach to Zheng et al. (2002) by buffering *semi-frequent sequences* (SFS) for use as candidates for newly appearing sequences in the updated database. This buffering is achieved by using a *buffer ratio* $\mu \leq 1$ to lower the *min_supp* and then maintaining the set of sequences, in the original database, that meet this modified *min_supp*. The assumption is that the majority of the frequent subsequences introduced by the updated part of the database (D') would come from these SFS or are already frequent in the original database (D) and therefore according to Cheng et al. (2004) the SFS' and FS' in D' are derived from the following cases:

1. A pattern which is frequent in D is still frequent in D'
2. A pattern which is semi-frequent in D becomes frequent in D'
3. A pattern which is semi-frequent in D is still semi-frequent in D'
4. An appended database Δdb brings new items (either frequent or semi-frequent)
5. A pattern which is infrequent in D becomes frequent in D'
6. A pattern which is infrequent in D becomes semi-frequent in D'

For Case (1)-(3) the information required to update the support and project D' to find all frequent/semi-frequent sequences is already in FS and SFS and is therefore a trivial exercise.

Case (4): **Property**: An item which does not appear in D and is brought by Δdb has no information in FS or SFS.

Solution: Scan the database LDB[4] for single items and then use any new frequent item as a prefix to construct a projected database and recursively apply the PrefixSpan method to discover frequent and semi-frequent sequences.

Case (5): **Property**: If an infrequent sequence p' in D becomes frequent in D', all of its prefix subsequences must also be frequent in D'. Then at least one of its prefix subsequences p is in FS.

Solution: Start from its frequent prefix p in FS and construct a p-projected database on D' and use the PrefixSpan method to discover p'.

Case (6): **Property**: If an infrequent sequence p' becomes semi-frequent in D', all of its prefix subsequences must be either frequent of semi-frequent. Then at least one of its prefix subsequences, p, is in FS or SFS.

Solution: Start from its prefix p in FS or SFS and construct a p-projected database on D' and use the PrefixSpan method to discover p'.

The task of the algorithm is then, given an original database D, and appended database D', a threshold *min_supp*, a buffer ratio μ, a set of frequent sequences FS and a set of semi-frequent sequences, SFS, to find all FS' in D' (Cheng et al., 2004). This is a two step process in which, first LDB is scanned for single items (Case (4)) and second, every pattern in FS and SFS in LDB are checked to adjust the support of them and if a pattern becomes frequent add it to FS'. If it meets the projection condition ($supp_{LDB}(p) \geq (1 - \mu) * min\_supp$) then use it as a prefix to project a database as in Case (5) and if the pattern is semi-frequent add it to SFS'. Optimisation techniques dealing with pattern matching and projected database generation are also applied.

2.8.7 Improvements of IncSpan

Nguyen, Sun and Orlowska (2005) in their paper titled *'Improvements of IncSpan: Incremental Mining of Sequential Patterns in Large Database'* found that in general IncSpan fails to mine the *complete* set of sequential patterns from an updated database. Furthermore, they clarify the weaknesses in the algorithm and provide a solution, IncSpan+ which rectifies them. The weaknesses identified are for Cases (4)-(6) (see pages 46 for details) and proofs are given, in the form of counter examples, that prove they are incorrect. For Case (4) they prove that not all single frequent/semi-frequent items can be found by scanning LDB and propose a solution that scans the entire D' for new single items. For Case (5) they show that it is possible that none of the prefix subsequences p of an infrequent sequence, p' in D that becomes frequent in D' are in FS and propose to not only mine the set of FS but also those of SFS. For Case (6) it is shown that it is possible that none of the prefix subsequences of p of an infrequent

[4]The LDB is the set of sequences in D' which are appended with items/itemsets

sequence, p' in D that becomes semi-frequent in D' are in FS or SFS and propose to not only mine the set of FS but also those of SFS based on the projection condition. By implementing these changes IncSpan+ not only guarantees the correctness of the incremental mining result, but also maintains the complete set of semi-frequent sequences for future updates (Nguyen et al., 2005).

2.9 Areas of related research

2.9.1 Streaming Data

Mining data streams is an area of related research that is still in its infancy, however there is a significant and growing interest in the creation of algorithms to accommodate different datasets. Among these researchers are Lin, Keogh, Lonardi and Chiu (2003) who look at a symbolic representation of time series and the implications with respect to streaming algorithms. Giannella, Han, Pei, Yan and Yu (2003) who mine frequent patterns in data stream using multiple time granularities, and Cai, Clutter, Pape, Han, Welge and Auvil (2004) and also Teng, Chen and Yu (2003) use multidimensional regression methods to solve the problem. Laur, Symphor, Nock and Poncelet (2005) evaluate a statistical technique which biases the estimation of the support of sequential patterns, so as to maximise either the precision or the recall and limits the degradation of the any other criteria and Marascu and Masseglia (2005) use a sequence alignment method similar to that used by Kum et al. (2002) in ApproxMAP and Hay, Wets and Vanhoof (2002) in their Web Usage Mining research. A review of the complete field of mining data streams can be found in *'Mining Data Streams: A Review'* by Gaber, Zaslavsky and Krishnaswamy (2005).

2.9.2 String Matching and Searching

A much older yet related field is that of approximate string matching and searching, where the strings are viewed as sequences of tokens. Research in this area has been ongoing for some time and includes not only algorithms for string matching using regular expressions (Hall and Dowling, 1980; Aho, 1990; Breslauer and Gąsieniec, 1995; Bentley and Sedgewick, 1997; Landau, Myers and Schmidt, 1998; Sankoff and Kruskal, 1999; Chan, Kao, Yip and Tang, 2002; Amir, Lewenstein and Porat, 2000), but also algorithms in the related area of edit distance (Wagner and Fischer, 1974; Tichy, 1984; Bunke and Csirik, 1992; Oommen and Loke, 1995; Oommen and Zhang, 1996; Cole and Hariharan, 1998; Arslan and Egecioglu, 1999, 2000; Cormode and Muthukrishnan, 2002; Batu, Ergün, Kilian, Magen, Raskhodnikova, Rubinfeld and Sami, 2003; Hyyrö, 2003). For an excellent survey on this field see *'A Guided Tour to Approximate String Matching'* by Navarro (2001).

There is however no reason why text cannot be viewed in the same way and Ahonen, Heinonen, Klemettinen and Verkamo (1997, 1998) and Rajman and Besançon (1998) have applied similar techniques, based on generalised episodes and episode rules, to text analysis tasks.

2.10 Rule Inference

The purpose of generating frequent sequences, no matter which techniques are used, is to be able to infer some type of *rule*, or *knowledge*, from the results. For example, given the sequence $A \ldots B \ldots C$ occurring in a string multiple times, rules such as: token (or event) A appears (occurs) then B and then C, can be expressed. It has been argued by Padmanabhan and Tuzhilin (1996) that these types of inference, and hence the temporal patterns, have a limited expressive power. For this reason mining for sequences and the generation of rules based on first-order temporal logic (FOTL), carried out by Padmanabhan and Tuzhilin (1996), has extended the work by Mannila et al. (1995) to include inferences of the type *Since, Until, Next, Always, Sometimes, Before, After and While*, by searching the database for specific patterns that satisfy a particular temporal logic formula (TLF). One disadvantage to this approach is that no intermediate results are obtained and thus any mining for a different pattern must be conducted on the complete database, which could incur a significant overhead. Höppner (2001*a*) and Höppner and Klawonn (2001) use modified rule semantics, J-Measure and rule specialisation to find temporal rules from a set of frequent patterns in a state sequence. The method described uses a windowing approach, similar to that which is used to discover frequent episodes, and then imposes Allen's interval logic (Allen, 1983) to describe rules that exist within these temporal patterns. Kam and Fu (2000) deal with temporal data for events that last over a period of time and introduce the concept of temporal representation and foster the view that this can be used to express relationships between interval based events, also using Allen's temporal logic. Sun, Orlowska and Zhou (2003) use 'Event-oriented patterns' in order to generate rules, with a given minimum confidence, that are in the form of $LHS \xrightarrow{T} e,\ conf(\theta)$.

2.11 Discussion

This chapter has surveyed the field of sequential pattern mining since its inception in 1995 by Agrawal and Srikant to the most recent advances. The focus has been on the different types of datasets and therefore the different algorithmic approaches required to meet those differences and also the inclusion of constraints, and counting methods for windowing based algorithms. While the rules produced from the majority of approaches are simple, in the sense they do not take into account the possibility of using interval logic algebras and their derivatives, some do take the that step and produce rules that are more expressive.

With the improvements in incremental sequential pattern mining and the growing field of stream data mining the need to express rules with a more expressive nature is becoming more appealing and has the potential to avail the sequence mining paradigm to more diverse and complex domains, for example the medical domain. This thesis presents a method for discovering interacting episodes from temporal sequences (see Chapter 5) and the analysis of them using temporal patterns. However, in order for that methodology to be seen in context the following two chapters will discuss temporal logic in general and a midpoint temporal logic, respectively.

Chapter 3

Temporal Logic

In Chapter 2 the various methods for mining sequences or sequential patterns and the various rules that may be inferred from those results was discussed. This chapter extends the discussion on the more traditional approaches to rule inference and introduces the algebras, that are more appropriately placed in the area of temporal logic, on which rule inference is now being undertaken by Kam and Fu (2000); Höppner and Klawonn (2001); Sun, Orlowska and Zhou (2003); Mooney and Roddick (2004); de Amo, Giacometti and Santana (2005).

The representation of temporal information in a logical framework has been referred to as Temporal Logic since the introduction of the modal-logic type approach by Prior (1957). This form of logic has since been widely developed by both logicians and computer scientists for a variety of applications including artificial intelligence, program execution, temporal expressions in natural language, and more recently data mining (Galton, 2003).

This chapter introduces the major contributing approaches to dealing with temporal logic in the data mining area, particularly sequences of events (Section 3.1), which includes the Point and Interval Algebras (Allen, 1981, 1983, 1984; Allen and Hayes, 1989; Vilain, 1982; Freksa, 1992; Guzmán and Rossi, 1995), in Section 3.2 and Section 3.3, as well as Fuzzy extensions (Fagin and Halpern, 1988; Badaloni and Giacomin, 1999; Ohlbach, 2004*a*,*b*; Sun, Orlowska and Li, 2003), in Section 3.4.

3.1 Temporal Logic Models

Time can be viewed as both discrete and linear in nature and, with the exception of Allen (1984), a logic of intervals can be constructed using points rather than from intervals themselves (Halpern and Shoham, 1991). For any two points in time there are only three possible operators that can exist between them: *before*, *after* or *equals*. These equate to the traditional set of binary operators that have been used for rule inference. Minor extensions to include the possibility of a *distance metric* to allow for example *k time units before/after* have since been included to enhance reasoning. Intervals, however, can be described as the time between two points and therefore the relationships between any two intervals can be expressed as a set of relationships between the start and end points of the given intervals (Allen, 1981; Vilain, 1982). If any two intervals are denoted as x and y with x^-, y^- as the start of x and y, and x^+ and y^+ as the end of x and y, then the set of relationships is as follows:

i.	x^- and y^-
ii.	x^- and y^+
iii.	x^+ and y^-
iv.	x^+ and y^+

If the relationships *before*, *after* and *equals* are now applied to intervals rather than as operators between points and the above set of constraints are invoked then these relationships can be expressed as a 4-tuple of the form ⟨ i ii iii iv ⟩ resulting in ⟨<< <<⟩, ⟨>> >>⟩ and ⟨=< >=⟩ respectively.

If it is assumed that an interval is an ordered set of points with the first point being less than/before the second point etc., then the valid combinations that can exist between these four point intervals yield a possible thirteen interval-interval relationships. Allen developed an algebra based on these thirteen intervals but he also argued

> "...that a model based on points {...} does not correspond to our intuitive notion of time." (Allen, 1983)

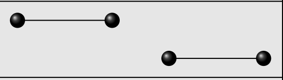
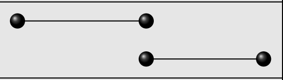
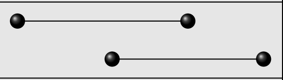
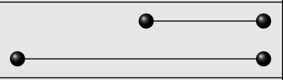
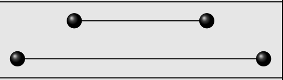
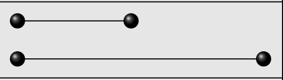
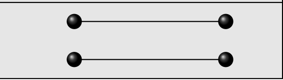
and as a consequence, he used the temporal interval as the primitive for his algebra. This, however, does not preclude the fact that each interval may have known end points. Moreover, this proposition has been used with certain relaxations to these endpoints and has resulted in the algebra of semi-intervals introduced by Freksa (1992).

The algebras that deal with fuzziness, do so mainly in two distinct ways:

1. by relaxing one or both of the endpoints, but not in the same way as Freksa (1992), as does Ohlbach (2004*b*)
   or,
2. by applying the fuzziness at the reasoning level, á la Badaloni and Giacomin (1999, 2002, 2006); Badaloni, Falda and Giacomin (2004).

These are primarily extensions that deal with the integration of flexibility and uncertainty into the classical Interval Algebra.

Table 3.1: Allen's thirteen temporal relationships.

Relationship	Label	Inverse	Schematically		Endpoint Constraints
x Before y	$<$		x		$<<$ $<<$
y After x		$>$	y		$>>$ $>>$
x Meets y	m		x		$<<$ $=<$
y is-Met-by x		mi	y		$>=$ $>>$
x Overlaps y	o		x		$<<$ $><$
y is-Overlapped-by x		oi	y		$><$ $>>$
x Finishes y	f		x		$><$ $>=$
y is-Finished-by x		fi	y		$<<$ $>=$
x During y	d		x		$><$ $><$
y Contains x		di	y		$<<$ $>>$
x Starts y	s		x		$=<$ $><$
y is-Started-by x		si	y		$=<$ $>>$
x Equals y	$=$		x		$=<$ $>=$
		$=$	y		

The black ball (●) indicates a known endpoint.

3.2 Allen's Interval Algebra

In 1983, Allen outlined a closed, non-overlapping set of 13 interval-interval relationships a set of which can be used to characterise the relative relationship between two temporal (or directional 1-D) intervals (Allen, 1983). An interval x is represented as a pair $[x^-, x^+]$ of real numbers with $x^- < x^+$, denoting the start and end points of the interval respectively (Krokhin, Jeavons and Jonsson, 2003; Drakengren and Jonsson, 1997*b*). These intervals are denoted: *Before* ($<$), *Meets* (**m**), *Overlaps* (**o**), *Finishes* (**f**), *During* (**d**), *Starts* (**s**), and their inverses, *After* ($>$), *is-Met-by* (**mi**), *is-Overlapped-by* (**oi**), *is-Finished-by* (**fi**), *Contains* (**di**), *is-Started-by* (**si**), and *Equals* ($=$) – see Table 3.1 for a complete description.

By considering the fact that there are thirteen basic relations, it follows that there are $2^{13} = 8192$ possible relations in the full algebra, which is denoted by $\mathcal{A}$. The complete set of subclasses of the full algebra are obtained by considering all subsets of $\mathcal{A}$ of which there are $2^{8192} \approx 10^{2466}$. Reasoning using this complete set has been shown to be NP-complete (Vilain and Kautz, 1986) and as such it has motivated the search for tractable subclasses where reasoning can be guaranteed among which are those of Nebel and Bürckert (1995), Drakengren and Jonsson (1997*c*) and Krokhin, Jeavons and Jonsson (2003). In the normal course of events, reasoning between such relationships is facilitated through a transitivity table (Table A.1) which, given the relationship between two intervals, X and Y and between Y and Z, provides the subset within which any relationship between X and Z must fall. Reasoning using this method has been extensively studied, since the publication of Allen's paper, with the major foci on plan creation and simulations (Miller, 1986; Weida, 1995), and in natural language processing. The work has also been extended in a number of ways including spatial and uncertain data, see Ladkin (1987) and Gennari (1998) for surveys, and also for the handling of metric time, as in Meiri's framework (1991), and the work of Drakengren and Jonsson (1997*a*). This research has necessitated the production of algorithms to solve these problems (van Beek, 1990; Vila, 1994; van Beek and Manchak, 1996; Oates, Schmill, Jensen and Cohen, 1997; Padmanabhan and Tuzhilin, 1996) for both the interval and point-based approaches.

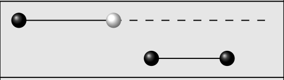
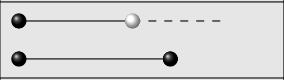
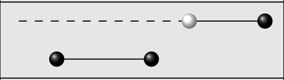
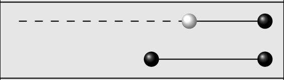
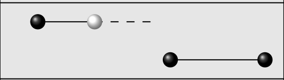
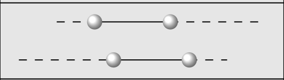
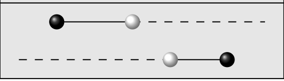
3.3 Freksa's Semi-Intervals

Freksa (1992) extended (generalised) Allen's work to incorporate the notion of 'conceptual neighbours' and 'conceptual neighbourhoods' to accommodate vagueness in one or more of the end-points. The intervals that are formed as a result of this vagueness were termed *semi-intervals*[1]. Freksa framed his work from a cognitive perspective and as such he used the 'beginnings' and 'endings' of intervals as representational primitives. The power of such a generalisation lies in the fact that if the relative position of only one end-point is known, some inference is often still possible, sometimes without loss of information. For example, although information may be known about the date of birth or death of a person but not both, this does not preclude some inferences being made regarding events in the person's life.

The generalisation to 'conceptual neighbour' results from the ability to transform one of the thirteen Allen relations (see Table 3.1) into another by continuous deformation, thus the relations *before* **(<)** and *meets* **(m)** are conceptual neighbours since they can be transformed into one another by lengthening one of the events. The relations *before* **(<)** and *overlaps* **(o)** are not conceptual neighbours since transformation by continuous deformation can only take place via the relation *meets* **(m)**. A 'conceptual neighbourhood' is a set of relations between pairs of events that are path-connected by virtue of 'conceptual neighbour' relations. *Before* **(<)**, *overlaps* **(o)** and *meets* **(m)** form a conceptual neighbourhood since they can be transformed into one another by a series of continuous deformations. Using these generalisations forms the basis of reasoning that is termed 'coarse knowledge' and it was Freksa's conclusion that temporal reasoning based on the thirteen Allen relations yields either complete knowledge - using well defined endpoints - or coarse knowledge - using beginnings and/or endings - but never scattered disjunctions. Freksa went on to define a series of eleven semi-intervals and conceptual neighbourhoods: *older* (**ol**), *younger* (**yo**), *head to head* (**hh**), *survives* (**sv**), *survived by* (**sb**), *tail to tail* (**tt**), *precedes* (**pr**), *succeeds* (**sd**), *contemporary of* (**ct**), *born before death of* (**bd**), and *died after birth of* (**db**) – see

[1]A semi-interval is equivalent to a 'nest' as described by Allen and Hayes (1985).

Table 3.2: Freksa's eleven semi-interval relationships.

Relationship	Label	Inverse	Schematically	Endpoint Constraints
x is Older than y y is Younger than x	ol	yo	x y 	<< ?? >? >?
x is Head to Head with y	hh	hh	x y 	=< >?
x Survives y y is Survived by x	sv	sb	x y 	?? >> ?< ?<
x is Tail to Tail with y	tt	tt	x y 	?< >=
x Precedes y y Succeeds x	pr	sd	x y 	<< ?< >? >>
x is a Contemporary of y	ct	ct	x y 	?< >?
x is Born before Death of y y Died after Birth of x	bd	db	x y 	?< ?? ?? <?

The black ball (●) indicates a known endpoint, the white ball (○) marks the end of the known event and the dashed lines (- - -) indicate either the continuation of the event in the same line or not. The length of the dashed lines indicates the number of alternative implementations of the given relation. The (?) in the endpoint constraints indicates the possibility of (<, > or =)[a].

[a]For the case of Precedes and Succeeds the (?) can only indicate the possibility of (< or >).

Table 3.2 for a complete description. Combinations of certain of these constraints also yield the following six relations: *older & survived by* (**ob**), *younger & survives* (**ys**), *older contemporary* (**oc**), *surviving contemporary* (**sc**), *survived by contemporary* (**bc**) and *younger contemporary* (**yc**).

As has been stated, temporal reasoning is usually conducted using a transitivity table, and in this respect Freksa was no different. However, by preserving the conceptual neighbourhoods in the arrangement of this table, Freksa was able to exploit certain outcomes, most notably the symmetry. This enabled a series of condensed representations or optimised transition tables for the purpose of performing the reasoning, the last of which contained only seven entries (Freksa, 1992). For ease of visual reference, Freksa also defined a series of icons (Figure 3.1) to facilitate the depiction of the neighbourhoods and their correspondences to Allen's relations, and used these in the construction of his tables for coarse reasoning. Fine-grained (Allen) reasoning can also be undertaken since conjunctions of inferences from coarse reasoning are performed to yield the result.

For example the fine relation X *starts* (**s**) Y corresponds to the two coarse relations X **bc** Y and X **hh** Y and the fine relationship Y *overlaps* (**o**) Z corresponds to the

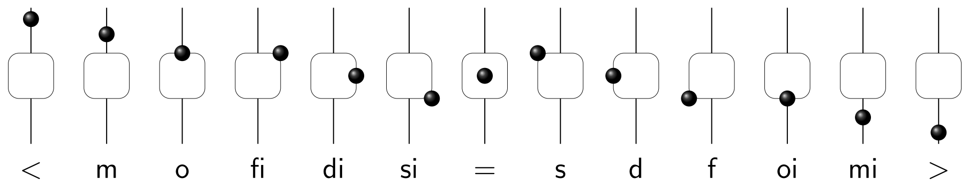
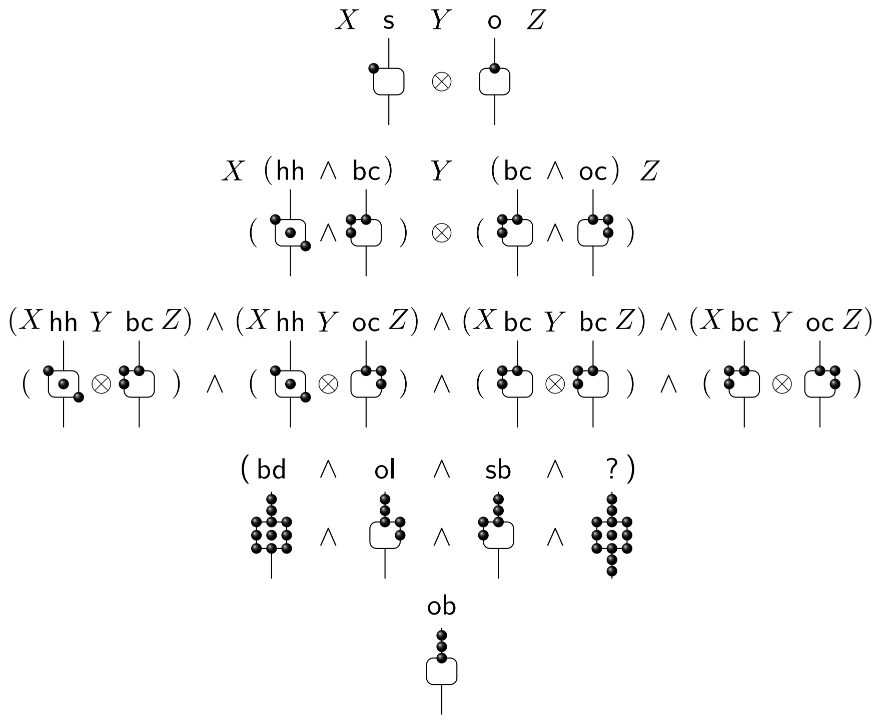


Figure 3.1: Freksa's iconic representation of Allen's relations in the order used to preserve conceptual neighbourhoods.

two coarse relations Y **bc** Z and Y **oc** Z. For these two relations to hold (X **s** Y **o** Z) it follows that the coarse relations must hold, and therefore the intersection of all coarse relations will yield the result. In this case the intersections are **bd**, **ol**, **sb** and **?**, of which the intersection is **ob**, see Figure 3.2. This iconic example has been elaborated here since it shows the relationship between coarse and fine reasoning and also since it is the basis for the back-end processing when dealing with Freksa outcomes in the **INT**eracting **E**pisode **M**iner with **T**iming **M**arks ($INTEM_{TM}$) application (see Appendix B).



X s Y o Z
X (hh ∧ bc) Y (bc ∧ oc) Z
(X hh Y bc Z) ∧ (X hh Y oc Z) ∧ (X bc Y bc Z) ∧ (X bc Y oc Z)
(bd ∧ ol ∧ sb ∧ ?)
ob

Figure 3.2: Fine-grained (Allen) reasoning using Freksa's coarse reasoning methods.

3.4 Extensions

Extensions to First Order Logic (FOL) have been used by Padmanabhan and Tuzhilin (1996) who introduce First Order Temporal Logic (FOTL) by adding the temporal operators *Since*, *Until*, *Next*, and *Previous*. Further to this, they also derive the operators *Always*, *Future Sometimes* and *Past Sometimes*, *Before*, *After* and *While* and the bounded versions $Until_k$ and $Since_k$. The FOTL language is used to express temporal patterns that are discovered from temporal databases and constitutes an extension of the work of Mannila et al. (1995) in the discovery of serial episodes. The use of the temporal operators *Until*, *Next*, *Always* and *Sometimes* have also been used by Bacchus and Kabanza (2000) who also include the derived operator *Eventually*, and by Cotofrei and Stoffel (2002).

The model introduced by Fagin and Halpern (1988) allows for the explicit mention of probabilities in the formulas describing reasoning about knowledge and probability together, for example "according to agent i, formula φ holds with probability at least b". They present a framework for the interpretation of the formulas and any interrelationships that may exist between the agents' probability assignments at different states. This approach is more appropriate in the area of a possible worlds scenario and for the analysis of program execution in distributed systems and cryptography. Methods that deal with fuzzy constraint networks such as those by Dubois and Prade (1989), Vila and Godo (1994), Godo and Vila (1995) and Marín, Cárdenas, Balsa and Sánchez (1997), are not directly related to the current thesis.

Other methods are primarily extensions that deal with the integration of flexibility and uncertainty into the classical Interval Algebra and can be grouped according to the method by which the extension is used.

The following two extensions are those that are most useful for application with sequence mining.

3.4.1 Fuzzy Time Intervals

The first approach by Ohlbach (2004*b*), introduces the notion of *fuzzy* time intervals as opposed to those that are *crisp* by means of a fuzzy value, which instead of sharply dropping at the end of the interval, gradually decrease allowing for a smooth degradation (see Figure 3.3). Their argument is that it is more likely that a question of the type "give me all of the performances that end *before* midnight" would not ideally exclude those that finish one minute or so after. Implementations of these extensions have been documented in the FuTIRe-system (**Fu**zzy **T**emporal **I**ntervals and **Re**lations) (Ohlbach, 2004*a*).

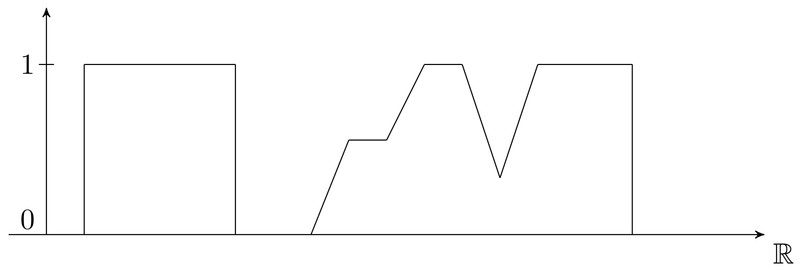


Figure 3.3: A depiction of a crisp and fuzzy interval – Ohlbach (2004*b*).

3.4.2 Fuzzy Interval Algebra

Badaloni and Giacomin (1999) take another approach and introduce the formalism IA^{fuz}. This formalism is introduced by way of the framework of Fuzzy Constraint Satisfaction Problem (FCSP). They extend the algebra of Allen by assigning a preference degree to each atomic relation. They define it on the set as follows:

$$\begin{aligned} I = \quad & \{b[\alpha_1], a[\alpha_2], m[\alpha_3], mi[\alpha_4], d[\alpha_5], di[\alpha_6], o[\alpha_7], \\ & oi[\alpha_8], s[\alpha_9], si[\alpha_{10}], f[\alpha_{11}], fi[\alpha_{12}], eq[\alpha_{13}]\}^a \\ where \quad & \alpha_i \in [0,1], \alpha_i \in \mathcal{R}, i = 1, \ldots, 13 \end{aligned}$$

[a]The atomic relation $\mathbf{b} \equiv <$ and $\mathbf{a} \equiv >$.

For example $\mathbf{I}$ $(o[0.3], d[0.7])$ $\mathbf{J}$ indicates that the *overlap* relation between **I** and **J** only holds to a degree of 0.3 and the *during* relation holds to a degree of 0.7 (see Figure 3.4 for a pictorial representation). They also include the possibility of adding a prioritized constraint to indicate how essential it is that a certain constraint be satisfied. Furthermore they have built a fuzzy Qualitative Algebra QA^{fuz}, an extension of the Qualitative Algebra of Meiri (1996) who defined the Point Algebra PA, the Interval Algebra IA, the Point-Interval and Interval-Point Algebras PI and IP, in which they consider the corresponding fuzzy extensions PA^{fuz} and IA^{fuz} (Badaloni and Giacomin, 2002, 2006) and also PI^{fuz} and IP^{fuz} (Badaloni et al., 2004).

Although FCSP has been applied to job-shop scheduling problems, the characterisation of ill-known diseases and has a view to be used in probabilistic planning systems, this thesis supports the view that it has value as a constraint in rule production from the results of sequence mining. Its use in such a situation is outlined in Section 5.3.3.2.

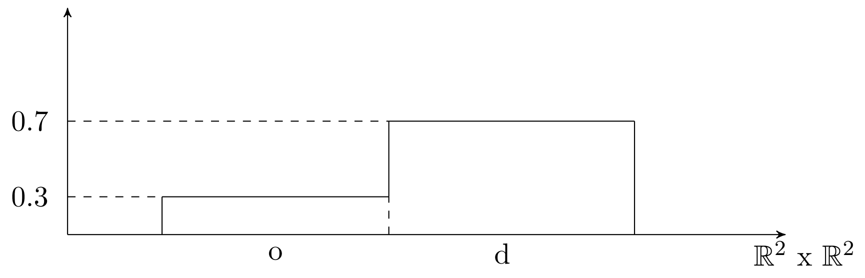


Figure 3.4: A depiction of an IA^{fuz} relation – Badaloni and Giacomin (1999).

3.5 Discussion

This chapter has introduced the area of temporal logic as it applies to sequence mining and includes discussions on both the point and interval based approaches and also extensions for different domains, for example fuzzy intervals. The Allen relationships, and to a lesser extent those of Freksa, have been widely used in research in both data modelling, particularly temporal data modelling and temporal databases, but to a lesser extent in data mining. In the data mining field, including association mining (Kam and Fu, 2000; Winarko and Roddick, 2006) and time series mining (Bettini, Wang and Jajodia, 1998; Bettini, Wang, Jajodia and Lin, 1998; Höppner, 2001*a*), research has been conducted using subsets of Allen's algebra, and although some work has been done using interval sequences (Höppner and Klawonn, 2001; Höppner, 2002; Sun, Orlowska and Zhou, 2003; Mooney and Roddick, 2004; de Amo et al., 2005) there remain significant contributions to be made in this area.

This chapter has provided the necessary background on temporal logic from which our method to discover 'interacting episodes', which incorporates the complete set of Allen's relations for rule production, can be presented in Chapter 5. Moreover, in conducting this research it was discovered that a far richer set of rules could be produced if the midpoints of the intervals were considered and consequently led to the development of the Midpoint Interval (MI) algebra that is presented in Chapter 4.

Chapter 4

Temporal Intervals with Midpoints

Chapter 3 discussed Allen's closed set of 13 interval-interval relationships, which are able to characterise the relative relationship between two temporal (or directional 1-D) intervals (Allen, 1983), and Freksa's semi-intervals (Freksa, 1992), which allow uncertainty in one or more of the endpoints. The Allen relations have subsequently been extended to spatial applications by extending the algebra to 2-D or higher dimensions and to equal-length intervals, which restricts the result set for specific applications (see Ladkin, 1987; Vila, 1994; Gennari, 1998, for surveys on the area).

The motivation for this aspect of the work stems from a need to make more specific interpretations of transitive relationships than those available using the Allen interval algebra when data regarding the midpoint of an interval are available. In many cases the 'natural' point of reference is the midpoint of an interval and it is therefore appropriate to develop a mechanism for reasoning between intervals when midpoint information is known. This is not an uncommon occurrence – for example, the following statements refer, at least implicitly, to the midpoint of an interval:

- "The parade will be during the second half of the festival",
- "The event occurred at 1.15pm, give or take a few minutes",
- "Maxwell was to be a big influence on Peter's work during the latter part of his life",
- "Midway through each quarter the opposition mounted a decisive challenge".

Furthermore, when dealing with linear temporal sequences certain problems can arise when the order of the tokens that comprise the stream is not explicitly coded. This problem can manifest itself in two ways. The first by an implication of order and the second an implication of simultaneity, both of which are shown to be solvable in the most part by an algebra that includes midpoints.

Within computing applications, midpoints are again not uncommon. For example, the midpoint is known implicitly or explicitly when an interval is given as an absolute time period, when the midpoint is supplied explicitly, and possibly when there is some granularity involved or when there is some uncertainty.

This chapter presents an algebra in which intervals can be characterised not only in relation to their endpoints but also their midpoints. The presentation described here complements the work in temporal constraint processing by Meiri (1996) and Schwalb and Vila (1998) in which temporal constraint networks are augmented with additional constraints. In this work the underlying primitive is enhanced with a greater expressive power but further constraints can still be added. Some of the research outlined in this chapter has appeared previously (Roddick and Mooney, 2005), but this has since been reviewed and revised before inclusion.

4.1 Midpoints in Relation to Existing Models

Two models will be presented in this chapter, which are collectively called the Midpoint Interval (MI) algebra. The first, in Section 4.4, is an extension of Allen's interval algebra (Allen, 1983) by refining the *overlaps* relationship to consider midpoints for equal-length intervals (Equal-Length Midpoint Interval – ELMI) and the second an extension of the first equal length model to consider the case of variable-length intervals (Variable-Length Midpoint Interval – VLMI), in Section 4.5. The relationship of both of these models to Allen's and Freksa's can be seen in Figure 4.1.

Furthermore, so that the variable-length model can be successfully integrated into the $INTEM_{TM}$ application, it has been necessary to extend Freksa's iconic representations to handle midpoints and this extension is presented in Section 4.7.1. The visualisation application itself is discussed in Section 7.4 and presented in Appendix B.

As has been stated in Chapter 3, extended reasoning using these interval algebras is performed using a transitivity table and it is the position of this thesis that both of the models presented here facilitate additional accuracy, in terms of information gain, when calculating transitive relationships using such a table. This discussion however is left until Chapter 7, as are examples that show that the expressive power of the MI algebra is richer and more powerful than that of Allen's.

4.2 Linear Temporal Sequences

Linear temporal sequences can be generated in a number of ways. For example telecommunications networks, web logs, sensor logs and so on. They can be mined statically

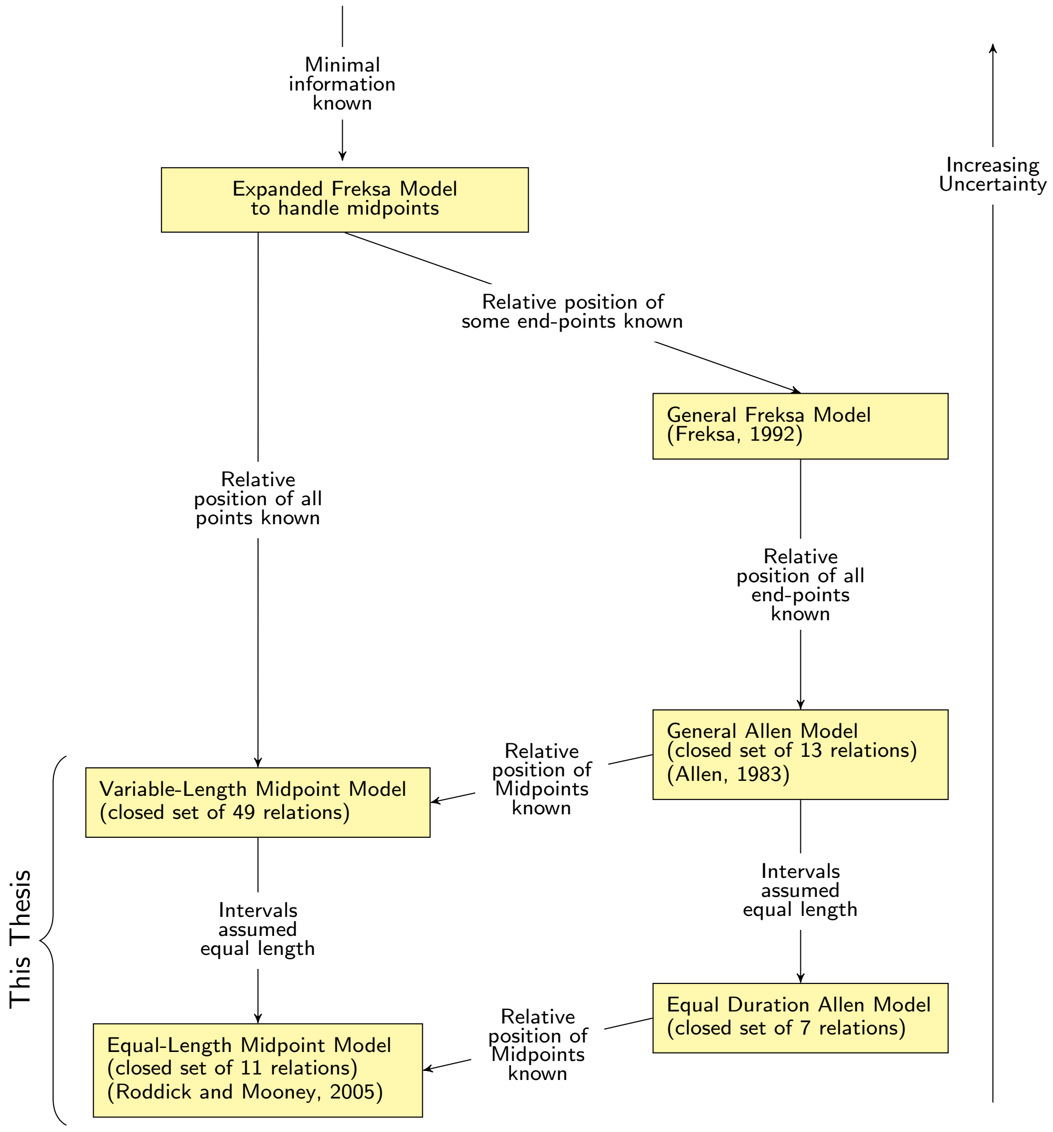

Minimal information known
Expanded Freksa Model to handle midpoints
Increasing Uncertainty
Relative position of some end-points known
General Freksa Model (Freksa, 1992)
Relative position of all points known
Relative position of all end-points known
General Allen Model (closed set of 13 relations) (Allen, 1983)
Relative position of Midpoints known
Variable-Length Midpoint Model (closed set of 49 relations)
This Thesis
Intervals assumed equal length
Intervals assumed equal length
Equal Duration Allen Model (closed set of 7 relations)
Relative position of Midpoints known
Equal-Length Midpoint Model (closed set of 11 relations) (Roddick and Mooney, 2005)

Figure 4.1: Models of temporal interval relations.

using sequence mining techniques (Mannila et al., 1997; Weiss and Hirsh, 1998; Huang et al., 2004; Vautier, Cordier and Quiniou, 2005; Dai and Wang, 2005) or dynamically using streaming data mining techniques (Giannella et al., 2003; Lin et al., 2003; Harada, 2004; Gaber et al., 2005). Regardless of the method, problems may arise if the sequence has been aggregated into unordered blocks, or when the granularity is larger than required. This will manifest itself by possibly requiring modifications to the relationships due to implied order (no time-stamps) or implied simultaneity. In the case of n polled sensors, for example, the order in which they are polled may have an influence on any resultant sequences of events that are discovered and therefore govern the set of temporal relationships that can be found. Moreover, if events are timestamped to the hour, two events recorded in consecutive hours can be as much as 120 minutes, or as little as a second apart. This problem can be exacerbated with larger granularities, but regardless of this, no ordering can be additionally implied for events within any chosen granularity, in this case the same hour. For this reason it is necessary to dispose of the strict ordering of events and consider the problem in terms of semi-ordered intervals (with the length possibly being the number of sensors) and then treat each frequent sequence discovered as having a start and end point accordingly, thus allowing for a mid-point to be used.

In many cases, there is no lack of knowledge in the Freksa sense, that is, the end-points of a particular 'block' are known, but it is often the inferred simultaneity of the events that is of primary concern. Even with non-timestamped data the situation can arise that 'blocks' infer the timestamp and information should be able to be gleaned even if the boundaries of those 'blocks' is shifting (through, for example, a moving window) (Roddick and Mooney, 2005).

As can be seen in Table 3.1 (page 52), four (*meets*[1], *equals*, *starts* and *finishes*) of the 13 interval-interval relationships (plus their inverses) require at least one set of the endpoints of the intervals to be simultaneous which, when presented with a single data stream, cannot occur – tokens are typically presented in an ordered sequence, even when, at a fine level of detail, that order is arbitrary. The problem becomes evident when this arbitrary ordering restricts, in the first instance, mining over the data, and the second, reasoning over the data.

4.2.1 Implied Order and Implied Simultaneity

In the case where data is originating from a single source and a need exists to either mine or reason over this data then implying order[2] is necessary to accomplish this task and does not pose a problem. The problem arises when the data either originates from multiple sources over a fixed period of time or when the data is reported with a larger than required granularity. The two related problems can be explained as follows.

1. In cases where tokens are provided as a linear (non-timestamped, unidentified) stream originating from n independent sensors, it must be assumed that any sequence of n tokens between two tokens in the stream originating from the same source took place simultaneously and that any apparent order is as a result of polling delays. This situation is depicted in Figure 4.2. This scenario is analogous to the way in which association mining treats the contents of each transaction arising from the same source, that is, the order in which they are put through the checkout may not be the order in which they were taken off the shelf and therefore any apparent order could be the result of the way in which the store

[1]In the discussion on detecting interacting episodes, Section 5.3, the position is adopted that a *meet*s relationship occurs when two episodes abut each other. This translates to consecutive events when viewed from the smallest granularity under consideration, therefore removing the need for simultaneity in this instance.

[2]Typically this could be a strict ordering, for example *less than (<)*, that would allow for an episodic or sequence mining algorithm to be used to elicit any information from the data.

was traversed. This is also analogous to the parallel episode discovery of Mannila and Toivonen (1996); Mannila et al. (1997) when using non-timestamped data.

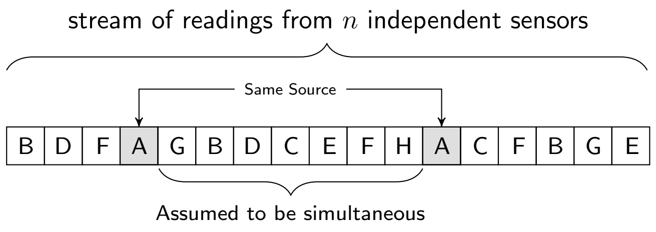

stream of readings from n independent sensors
Same Source
B D F A G B D C E F H A C F B G E
Assumed to be simultaneous

Figure 4.2: Data stream generated from n independent sensors.

2. Alternatively, when data are supplied with a larger than required granularity, it cannot be assumed that those tokens timestamped at t_i are necessarily all closer to each other than those timestamped at t_{i-1} or t_{i+1}. This applies also to events placed unordered in ordered blocks of input.

One solution to the first problem is to fragment the stream into sections of at least $2n + 1$. However, this does not eliminate the problem as this is a moving window, as shown in Figure 4.3, which can span any selected fragment size and therefore imply simultaneity for the entire stream. Moreover, a large fragment (as with a large granularity) may also serve to create the second problem and act to imply simultaneity where none exists.

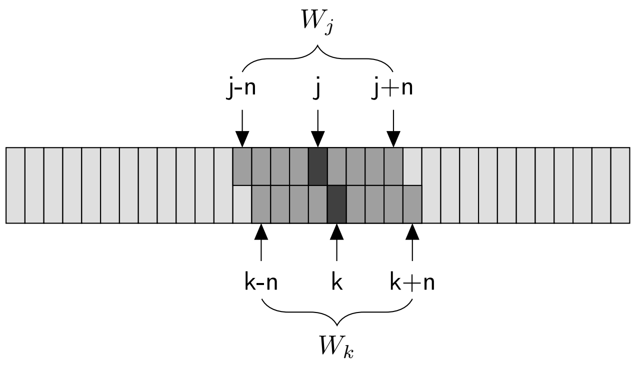


Figure 4.3: Moving window of potentially simultaneous tokens where $n = 4$.

In some cases, data are provided in blocks, in which events are unordered, even if the blocks themselves are ordered and in this case there is again a window as shown in Figure 4.4. Assuming n sensors, the ideal situation would be to use the moving window IW_k but in this case the larger window W_k must be used, which encompasses any block covered by IW_k.

For data with too large a granularity, there is again a window, this time of those events that have an implied simultaneity where none may exist. For example, in

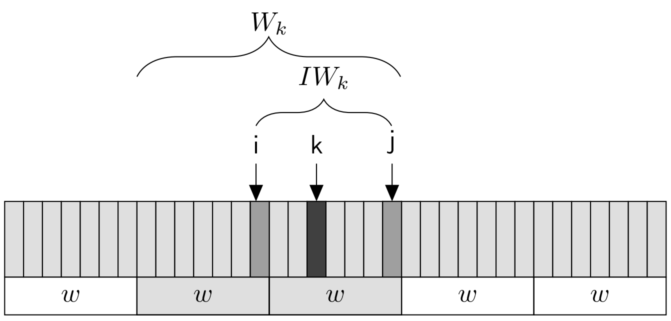
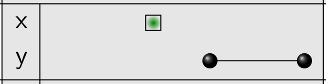
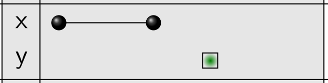
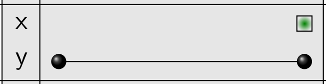
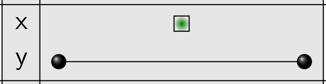
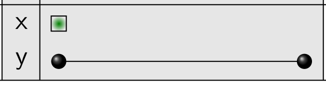


Figure 4.4: Moving window (w) over tokens with larger than required granularity (or tokens in blocks).

Figure 4.4 event i is closer to k than event j despite being given a different time-stamp (i.e. in a different block). This case can be considered as being analogous to the blocked events problem, albeit that the blocks may be sparse. In 1982, Vilain discussed five point-interval temporal relationships (see Table 4.1) and thus one solution is to consider two events as being simultaneous if any of the tokens j *starts*, *finishes* or is *during* the interval represented by the moving window W_k of the other. This must be performed twice, once for each end of W_k (Roddick and Mooney, 2005). The alternative as proposed by this thesis is to extend the interval algebra of Allen to consider not only the endpoints of the intervals but also their midpoints. This alternative does not present a true solution to the stated problems but rather represents a solution that yields on average better results. This alternative is expanded in the following sections.

Table 4.1: Vilains five point-interval temporal relationships.

Relationship	Label	Schematically	Endpoint Constraints
x Before y	$<$		$<<<<$
y After x	$>$		$>>>>$
x Finishes y	f		$>=>=$
x During y	d		$><><$
x Starts y	s		$=<=<$

The black ball (●) indicates a known endpoint of a interval and the green square (🟩) indicates a known point.

4.3 Midpoint Preliminaries

In Section 3.1 it was stated that intervals can be described as the time between two points and therefore the relationships between any two intervals can be expressed as a set of relationships between the endpoints of the given intervals. These relationships were shown to be,

i.	x^- and y^-
ii.	x^- and y^+
iii.	x^+ and y^-
iv.	x^+ and y^+

where x^- and y^- refer to the start of x and y, and x^+ and y^+ as the end of x and y.

When incorporating midpoints, two non-zero-length intervals x and y can be considered to have a temporal relationship based on the relative positions of their endpoints and midpoints. If the endpoints are denoted as above and the midpoints are denoted as x° and y° respectively, then the resulting relationships are as follows:

i.	x^- and y^-	iv.	x° and y^-	vii.	x^+ and y^-
ii.	x^- and y°	v.	x° and y°	viii.	x^+ and y°
iii.	x^- and y^+	vi.	x° and y^+	ix.	x^+ and y^+

Since it can be assumed that an interval is an ordered set of points such that $x^- < x^\circ < x^+$ and $y^- < y^\circ < y^+$, the $3^9 = 19683$ possible combinations are reduced to 49. This extension is discussed in Section 4.5 and the relationships are shown in Table 4.3.

Requiring all intervals to be equal in length introduces further constraints such as $x^- = y^- \rightarrow x^\circ = y^\circ \wedge x^+ = y^+$ which reduces the combinations to 11. This extension is discussed in Section 4.4 and the relationships are shown in Table 4.2. Both of these extensions can be viewed as a further restriction of Allen's relationships in the same way that Allen's are more restrictive than those of Freksa as depicted in Figure 4.1. This enables, in cases when midpoint data is unavailable, the MI algebra to be transformed without loss of information into the algebra given by Allen (see Section 4.6.1).

In Allen's model, closure is computed with a constraint propagation algorithm (see Figure 4.5). The operation of the algorithm that determines closure is driven by a queue that contains pairs of intervals $< i, j >$. Each time the relation between two intervals i and j is changed the pair is placed on the queue. A call to the procedure *Close* is made and each time a pair $< i, j >$ is removed from the queue the algorithm determines whether the relation between i and j can be used to constrain the relation between i and other intervals in the database, or between j and those other intervals.

```
{ Table is a two-dimensional array indexed by intervals, in which $Table[i, j]$ holds the relation between
intervals $i$ and $j$. $Table[i, j]$ is initialised to the additive identity vector consisting of all forty-nine
simple relations; except for $Table[i, i]$, which is initialised to (EQUALS).
$Queue$ is a FIFO data structure that keeps track of pairs of intervals whose relation has been changed.
$Intervals$ is a list of all intervals about which assertions have been made.
$+$ is defined as conjunction and $\times$ as composition. }
To $Add\ R_{i,j}$
    { $R_{i,j}$ is a relation being asserted between $i$ and $j$. }
    begin
        $Old \leftarrow Table[i, j]$;
        $Table[i, j] \leftarrow Table[i, j] + R_{i,j}$;
        if $Table[i, j] \neq Old$
            then Place pair $< i, j >$ on fifo $Queue$;
        $Intervals \leftarrow Intervals \approx \{ij\}$;
    end;
To $Close$
    { Compute the closure of assertions added to the database. }
    while $Queue$ is not empty do
        begin
            Get next $< i, j >$ from $Queue$;
            $Propagate(i, j)$;
        end;
To $Propagate(i, j)$
    { Propagates the changes to the relation between $i$ and $j$ to all other intervals. }
    for each interval $k$ in Intervals do
        begin
            $Temp \leftarrow Table[i, k] + (Table[i, j] \times Table[j, k])$;
            if $Temp = \Box$ { $\Box$ is the inconsistent vector. }
                then Signal contradiction;
            if $Table[i, k] \neq Temp$
                then Place pair $< i, k >$ on $Queue$;
            $Table[i, k] \leftarrow Temp$;
            $Temp \leftarrow Table[k, j] + (Table[k, i] \times Table[i, j])$;
            if $Temp = \Box$
                then Signal contradiction;
            if $Table[k, j] \neq Temp$
                then Place pair $< k, j >$ on $Queue$;
            $Table[k, j] \leftarrow Temp$;
        end;
```

Figure 4.5: Allen's constraint propagation algorithm (Vilain and Kautz, 1986; Vilain et al., 1989).

If a new relation can be successfully constrained, then the pair of intervals to which it relates is placed on the queue. The process terminates when no more relations can be constrained (Vilain and Kautz, 1986; Vilain, Kautz and van Beek, 1989). In the case of the MI algebra proposed here, the same algorithm can be used to show closure, since it is asserted that the MI algebra is an extension of the Allen algebra. Furthermore, since it will be shown that the MI algebra is an extension of the Allen algebra, proof of closure for the MI algebra can be accomplished using the same technique and is detailed below. The proof is adapted from proofs that are discussed by Vilain and Kautz (1986); Vilain et al. (1989) and van Beek (1989).

Theorem 4.1. *The problems of determining the satisfiability of assertions in the midpoint interval algebra and determining their closure are equivalent, in that there are* *polynomial-time mappings between them.*

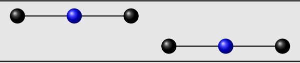
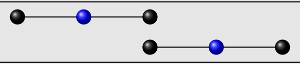
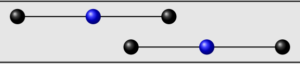
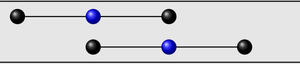
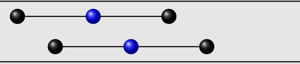
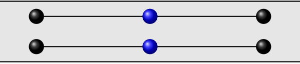
Proof: First, to show that determining closure follows readily from determining consistency, assume the existence of an oracle for determining the consistency of a set of assertions in the midpoint interval algebra. To determine the closure of assertions, run the oracle forty-nine times for each of the $O(n^2)$ pairs $< i, j >$ of intervals detailed in the assertions. Specifically, each time the oracle is run on a pair $< i, j >$, the oracle is provided with the original set of assertions and the additional assertion i (R) j, where R is one of the forty-nine simple relations. The relation vector between i and j is the one containing those simple relations that the oracle did not reject.

Second, to show that determining consistency follows from determining closure, assume the existence of an algorithm for closure. To see if a set of assertions is consistent run the algorithm, and inspect each of the $O(n^2)$ relations between the n intervals detailed in the assertions. The database is inconsistent if any of these relations is the inconsistent vector: this is the vector composed of no constituent simple relations. ■

4.4 Equal Length Intervals

Following Section 4.2.1 where W_j and W_k are identical in length (see Figure 4.3), of the 13 Allen relationships, 6 of them (*during*, *contains*, *starts*, *is-started-by*, *finishes* and *is-finished-by*) are not required. Instead, a closed set of 11 midpoint relationships can be created by extending the *Overlaps/is-Overlapped-by* relationship as shown in Table 4.2.

Table 4.2: The eleven equal-length interval-interval relationships with midpoints.

Relationship	Label	Inverse	Schematically		Constraints
x Before y y After x	$<$	$>$	x y		<<< <<< <<< >>> >>> >>>
x Meets y y is-Met-by x	m	mi	x y		<<< <<< =<< >>= >>> >>>
x SmallOverlap y y is-SmallOverlapped-by x	so	soi	x y		<<< <<< ><< >>< >>> >>>
x MediumOverlap y y is-MediumOverlapped-by x	mo	moi	x y		<<< =<< >=< >=< >>= >>>
x LargeOverlap y y is-LargeOverlapped-by x	lo	loi	x y		<<< ><< >>< ><< >>< >>>
x Equals y	$=$	$=$	x y		=<< >=< >>=

The black ball (●) indicates a known endpoint and the blue ball (🔵) indicates the midpoint.

As a result of this extension, in Table 4.2 it can be seen that in the context of a data stream, only *MediumOverlap*, *LargeOverlap* and *Equals* results in the midpoint of one interval being within the other and so for any given number of sensors n then the implied point-point relationship is *potentially Equals* implying possible simultaneity[3]. Again using this new algebra, it is noted that if the relationship between W_i and W_j for two tokens i and j, where W_i and W_j refer to windows containing i and j, (see Figure 4.4) is *SmallOverlap* for any given number of sensors n then the implied point-point relationship is *Before* as the endpoint of one does not overlap with the midpoint of the other.

This discussion on data streams can be concluded using the following pertinent example.

Example 4.1. Given the relationships $A \xrightarrow{rel} B \wedge B \xrightarrow{rel}$ C where $rel = so$,

in Allen's algebra

$A \xrightarrow{o} B \wedge B \xrightarrow{o} C \Rightarrow A \xrightarrow{o,m,<} C$

whereas the equal-length midpoint model can offer the ability to refine this to

$A \xrightarrow{so} B \wedge B \xrightarrow{so} C \Rightarrow A \xrightarrow{<} C$ □

This small example demonstrates that reasoning using information about midpoints can refine the set of relationships returned. This example will be more fully explored in Chapter 7.

4.5 Variable Length Intervals

The previous section extended Allen's interval-interval algebra to consider not only the endpoints of equal-length intervals but also their midpoints creating a closed set of 11 relationships (Roddick and Mooney, 2005). This section generalises this concept to variable length relationships. Figure 4.1 shows the relationship between existing algebras and the work discussed in this thesis. Effectively this process extends Allen's 13 relationships to 49 as shown in Table 4.3. The following section discusses the naming conventions and how they were derived.

4.5.1 Naming Conventions

To maintain consistency with the algebras currently in use, naming of the relationships is based, where possible, using an existing relationship as a stem and then a deductive selection from either a size factor or a midpoint position. Using this approach has resulted in the following rules.

[3]In the case of input as a linear sequence, the relationship *Equals* cannot itself occur.

1. Any unchanged Allen relationships remain the same, namely:

   Before ($<$), *Meets* (**m**), *is-Met-by* (**mi**), *After* ($>$) and *Equals* ($=$)

2. The *overlap* relationships use the stem **o** from the Allen *overlap* relationship as a postfix which is then prefixed by the nature of the overlapping intervals. This is required to be a two character prefix, the first denotes how much of the first interval overlaps the second and the second denotes how much of the second interval is overlapped by the first.

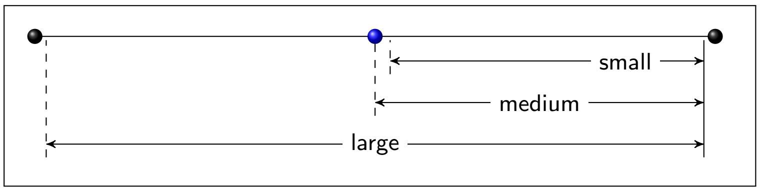


Figure 4.6: The sections of an interval that are representative of *small*, *medium* and *large* prefixes for the *overlap* relationships.

These prefixes can be one of the following (see Figure 4.6 for the section of the interval they pertain to):

- *small* (**s**), which involves the section of the interval up to but not including the midpoint and adheres to the constraints:

  $$(y^- < x^+ < y^\circ) \oplus (y^\circ < x^- < y^+)$$

- *medium* (**m**), which involves the section of the interval up to and including the midpoint and adheres to the constraints:

  $$(y^- < x^+ = y^\circ) \oplus (y^\circ = x^- < y^+)$$

- *large* (**l**), which involves the section of the interval up to but not including the endpoint and adheres to the constraints:

  $$(y^\circ < x^+ < y^+) \oplus (y^- < x^- < y^\circ)$$

For example if a *small* part of interval X overlaps a *large* part of interval Y then this results in a *SmallLargeOverlap* (**slo**), see Figure 4.7.

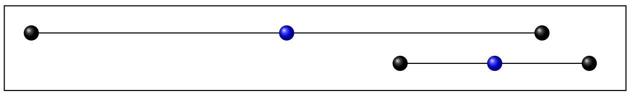


Figure 4.7: Depiction of a *SmallLargeOverlap* (**slo**) relationship.

1. The containment relationships are based on an endpoint or midpoint defining the relationship. Only four of the seven relationships use the stem **d** from the Allen *during* relationship, the other three are denoted more descriptively by using the midpoint of the containing interval.

This yields the following labels for use as prefixes and/or postfixes:

- *first* (**f**). If used as a prefix, it indicates an endpoint of an interval. If used as a postfix, it indicates the section of an interval (the same as in the *overlap* case).
- *last* (**l**). If used as a prefix, it indicates an endpoint of an interval. If used as a postfix, it indicates the section of an interval (the same as in the *overlap* case).
- *mid* (**m**). This always refers to the midpoint of an interval.
- *on* (**o**). This refers to an endpoint or midpoint being 'on' an endpoint or midpoint.

Combinations of these are used to indicate the nature of the containment, for example, if the endpoint of X (x^+) is during the first part of Y ($y^- < x^+ < y^\circ$) then the resultant relationship is a *LastDuringFirst* (**ldf**), see Figure 4.8.

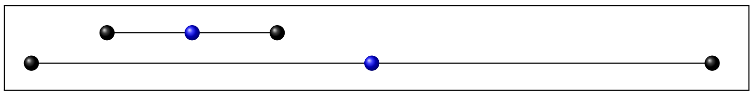


Figure 4.8: Depiction of a *LastDuringFirst* (**ldf**) relationship.

2. The starts and finishes relationships use the stem **s** or **f** from the Allen *start* and *finish* relationship as a prefix and this is then postfixed by the amount that one interval starts or finishes the other.

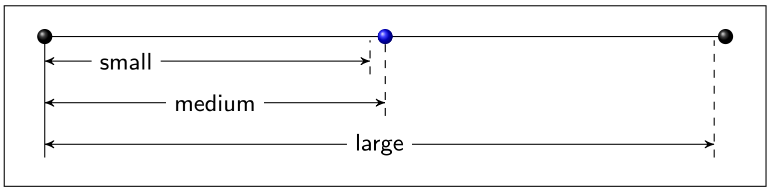


Figure 4.9: The sections of an interval that are representative of *small*, *medium* and *large* postfixes for the *starts*[a] relationships.

[a]The corresponding *finishes* sections are the inverse of the *starts* sections.

This determination is made in a similar way to that of the *overlap* relationship (see Figure 4.9 for the *starts* sections), however the constraints are different since both endpoints can be used.
The postfixes can be:

- *small* (**s**), which involves the section of the interval up to but not including the midpoint and adheres to the constraints:

$$(x^- = y^-) \wedge (x^+ < y^\circ)$$

- *medium* (**m**), which involves the section of the interval up to and including the midpoint and adheres to the constraints:

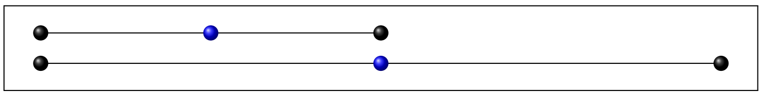


Figure 4.10: Depiction of a *StartsMedium* (**sm**) relationship.

$(x^- = y^-) \wedge (x^+ = y^\circ)$

- *large* (**l**), which involves the section of the interval up to but not including the endpoint and adheres to the constraints:

  $(x^- = y^-) \wedge (x^+ < y^+)$ where $(x^\circ > y^\circ)$

An example of a *StartsMedium* (**sm**) relationship is shown in Figure 4.10.

4.5.2 The Set of Variable-Length Midpoint Interval Relationships

Using the naming conventions derived in the previous section, it is now possible to show how the set of VLMI relationships divide the Allen relationships. This is performed in the following way:

***Overlap* – 9 different types**

SmallSmallOverlap (**sso**),	*SmallMediumOverlap* (**smo**),	*SmallLargeOverlap* (**slo**),
MediumSmallOverlap (**mso**),	*MediumMediumOverlap* (**mmo**),	*MediumLargeOverlap* (**mlo**),
LargeSmallOverlap (**lso**),	*LargeMediumOverlap* (**lmo**),	*LargeLargeOverlap* (**llo**)

***During* – 7 different types**

FirstDuringLast (**fdl**),	*FirstOnMid* (**fom**),	
MidDuringLast (**mdl**),	*MidOnMid* (**mom**),	*MidDuringFirst* (**mdf**),
LastOnMid (**lom**),	*LastDuringFirst* (**ldf**)	

***Starts* – 3 different types**

StartsSmall (**ss**), *StartsMedium* (**sm**), *StartsLarge* (**sl**)

***Finishes* – 3 different types**

FinishesSmall (**fs**), *FinishesMedium* (**fm**), *FinishesLarge* (**fl**)

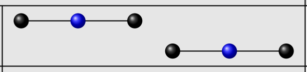
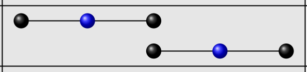
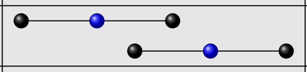
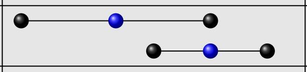
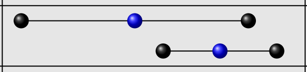
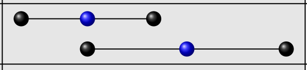
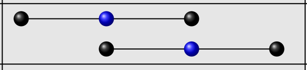
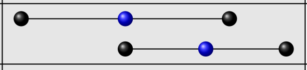
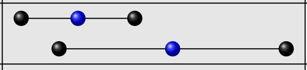
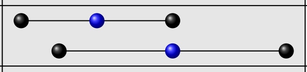
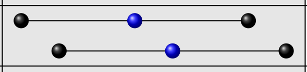
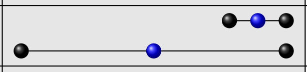
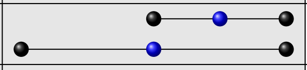
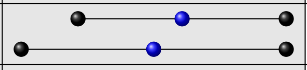
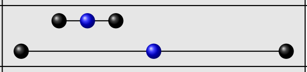
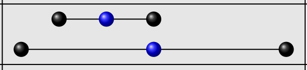
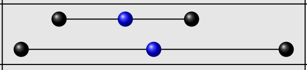
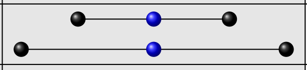
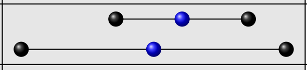
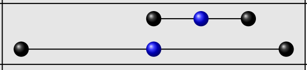
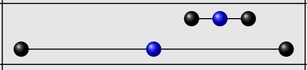
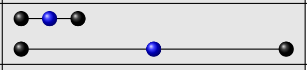
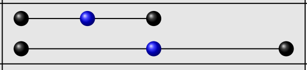
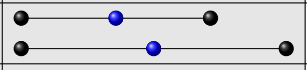
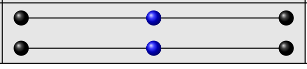
***unchanged* – 5 types**

Before (<), *After* (>), *Meets* (**m**), *is-Met-by* (**mi**), *Equals* (=)

which, including the inverses, makes a closed set of 49 variable length, interval-interval relationships with midpoints that are depicted in Table 4.3.

Harnessing the full power of the midpoint relations requires the examination of the possible transitive relationships and, in a similar manner to the Allen algebra, a transitivity table – this time 49 × 49 – can be constructed. Discussion of this is left until Chapter 7, and the table itself is included in Appendix A.

Table 4.3: The 49 VLMI relationships, showing the 9 overlap, 7 during, 3 start, 3 finish (with their inverses) and the 5 unchanged Allen relationships.

Relationship	Label	Inverse		Schematically	Constraints
x Before y y After x	<	>	x y		<<< <<< <<< >>> >>> >>>
x Meets y y is-Met-by x	m	mi	x y		<<< <<< =<< >>= >>> >>>
x SmallSmallOverlap y y is-SmallSmallOverlapped-by x	sso	ssoi	x y		<<< <<< ><< >>< >>> >>>
x SmallMediumOverlap y y is-SmallMediumOverlapped-by x	smo	smoi	x y		<<< <<< >=< >>< >>= >>>
x SmallLargeOverlap y y is-SmallLargeOverlapped-by x	slo	sloi	x y		<<< <<< >>< >>< >>< >>>
x MediumSmallOverlap y y is-MediumSmallOverlapped-by x	mso	msoi	x y		<<< =<< ><< >=< >>> >>>
x MediumMediumOverlap y y is-MediumMediumOverlapped-by x	mmo	mmoi	x y		<<< =<< >=< >=< >>= >>>
x MediumLargeOverlap y y is-MediumLargeOverlapped-by x	mlo	mloi	x y		<<< =<< >>< >=< >>< >>>
x LargeSmallOverlap y y is-LargeSmallOverlapped-by x	lso	lsoi	x y		<<< ><< ><< ><< >>> >>>
x LargeMediumOverlap y y is-LargeMediumOverlapped-by x	lmo	lmoi	x y		<<< ><< >=< ><< >>= >>>
x LargeLargeOverlap y y is-LargeLargeOverlapped-by x	llo	lloi	x y		<<< ><< >>< ><< >>< >>>
x FinishesSmall y y is-FinishedSmall-by x	fs	fsi	x y		>>< >>< >>= <<< <<< >>=
x FinishesMedium y y is-FinishedMedium-by x	fm	fmi	x y		>=< >>< >>= <<< =<< >>=
x FinishesLarge y y is-FinishedLarge-by x	fl	fli	x y		><< >>< >>= <<< ><< >>=
x LastDuringFirst y y FirstContainsLast-of x	ldf	ldfi	x y		><< ><< ><< <<< >>> >>>
x LastOnMid y y MidOnLast-of x	lom	lomi	x y		><< ><< >=< <<< >>= >>>
x MidDuringFirst y y FirstContainsMid-of x	mdf	mdfi	x y		><< ><< >>< <<< >>< >>>
x MidOnMid y y MidOnMid x	mom	momi	x y		><< >=< >>< <<< >=< >>>
x MidDuringLast y y LastContainsMid-of x	mdl	mdli	x y		><< >>< >>< <<< ><< >>>
x FirstOnMid y y MidOnFirst-of x	fom	fomi	x y		>=< >>< >>< <<< =<< >>>
x FirstDuringLast y y LastContainsFirst-of x	fdl	fdli	x y		>>< >>< >>< <<< <<< >>>
x StartsSmall y y is-StartedSmall-by x	ss	ssi	x y		=<< ><< ><< =<< >>> >>>
x StartsMedium y y is-StartedMedium-by x	sm	smi	x y		=<< ><< >=< =<< >>= >>>
x StartsLarge y y is-StartedLarge-by x	sl	sli	x y		=<< ><< >>< =<< >>< >>>
x Equals y	=	=	x y		=<< >=< >>=

The black ball (●) indicates a known endpoint and the blue ball (●) indicates the midpoint.

4.6 Transformations

Freksa's conceptual neighbourhoods have been widely accepted, in part because they represent a pragmatic grouping of sets of Allen relationships. That is, it may be the case that the results obtained from an analysis are too numerous and a more general outcome from a transitive relationship is sought. In this case it would be helpful if a more 'relaxed' set of possibilities were available. As can be seen in Figure 4.1 (page 61), the VLMI relationships transform either into the ELMI relationships when equal interval durations are applied as a restriction or into Allen's and further into Freksa's when midpoints or one endpoint are not known.

4.6.1 Conceptual Hierarchies

The results of any transformations from one state to another, as depicted in Figure 4.1 (page 61), are best described as a hierarchy and as such certain hierarchies must exist if the model as depicted is a correct representation of events. These hierarchies are descriptive of a transformation without loss of information from one state to another. However, it is not possible that all of the relationships are able to be transformed from one state to another. The most conspicuous example is the *during* relationship, which can be described by the Allen model, the VLMI model and take part in any Freksa neighbourhoods, but cannot be described using the ELMI or Equal-Length Allen models. The following sections further this concept, and in doing so validate the model, by describing the available hierarchies using the VLMI model as a base for any possible transformations.

4.6.1.1 Terminology for Describing Sets of Constraints

The following sections describe relationships using sets of constraints that were introduced earlier, see Section 4.3 (page 65), and are reproduced here for convenience.

Assume there exists intervals X and Y then:

For the MI algebra the relationships are depicted thus:

$$\langle <<<\ <<<\ <<< \rangle$$

The first of the three groups describes the relationship between: x^- and $y^-,\ y^\circ\ and\ y^+$; the second the relationship between: x° and $y^-,\ y^\circ\ and\ y^+$; and the third the relationship between: x^+ and $y^-,\ y^\circ\ and\ y^+$.

For the Allen algebra these relationships are depicted thus:

$$\langle <<\ << \rangle$$

Given the same two intervals X and Y then the first group describes the relationship between: x^- and $y^-\ and\ y^+$; and the second the relationship between: x^+ and $y^-\ and\ y^+$

4.6.1.2 The Overlap Hierarchy

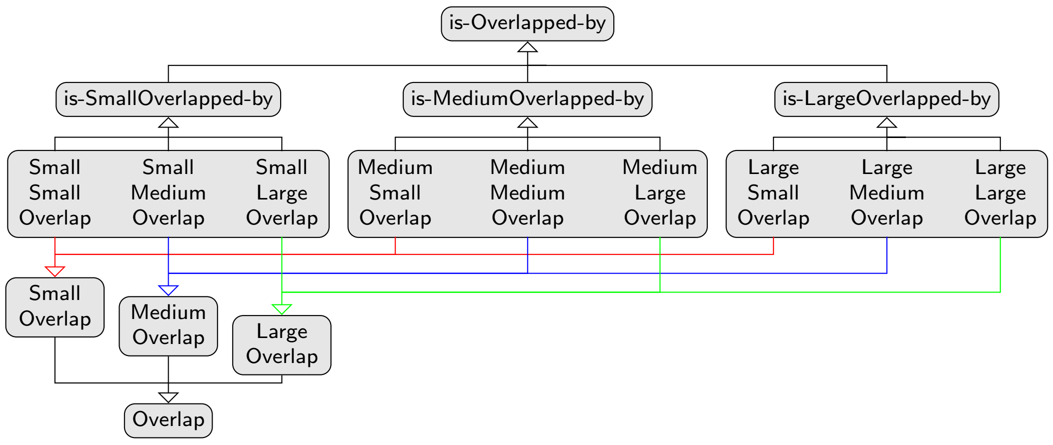


Figure 4.11: Hierarchical structure of the *overlap* relation.

The transformation from VLMI relationships through to ELMI midpoint relationships and then through to Allen's relationships is best illustrated by the *overlap* hierarchy, see Figure 4.11. The bottom section of the hierarchy shows the transformation into the Equal-Length relations (*SmallOverlap, MediumOverlap* and *LargeOverlap*) and then further into the Allen *overlap* relation from the perspective of an interval X that *overlaps* an interval Y. Using the three VLMI relationships, *SmallSmallOverlap*, *MediumSmallOverlap* and *LargeSmallOverlap* as an example, the transformation to the ELMI relationship *SmallOverlap* is evident since the endpoint of X, in all cases, is less-than the midpoint of Y – the condition for *SmallOverlap*. The top section of the hierarchy has the same property when viewed from the perspective of the interval Y being *overlapped* by an interval X. This can be formalised using the relative positions of the endpoints and midpoints as follows.

The defining constraints for any overlap relationship are that:

$(x^- < y^-) \wedge (x^+ > y^-) \wedge (x^+ < y^+)$

and this is regardless of whether midpoints are used. Therefore, the MI model for the overlap relationships can be proved by reducing them to the Allen model.

Proposition 4.1. *The MI overlap relationships are a subset of the Allen overlap relation.*

Proof: The set of Small Overlap midpoint relationships has the following variable-length members: **sso**, **mso** and **lso**, which have the following endpoint and midpoint constraints;

$\langle <<< \;\; <<< \;\; ><< \rangle$, $\langle <<< \;\; =<< \;\; ><< \rangle$ and $\langle <<< \;\; >< < \;\; ><< \rangle$

respectively. By removing any references to midpoints each of these is reduced to

$\langle << \;\; >< \rangle$, $\langle << \;\; >< \rangle$ and $\langle << \;\; >< \rangle$

which are precisely the constraints for the Allen *overlap* relation. ■

The proofs for the Medium and Large Midpoint Overlap relationships follow directly from this.

4.6.1.3 The During Hierarchy

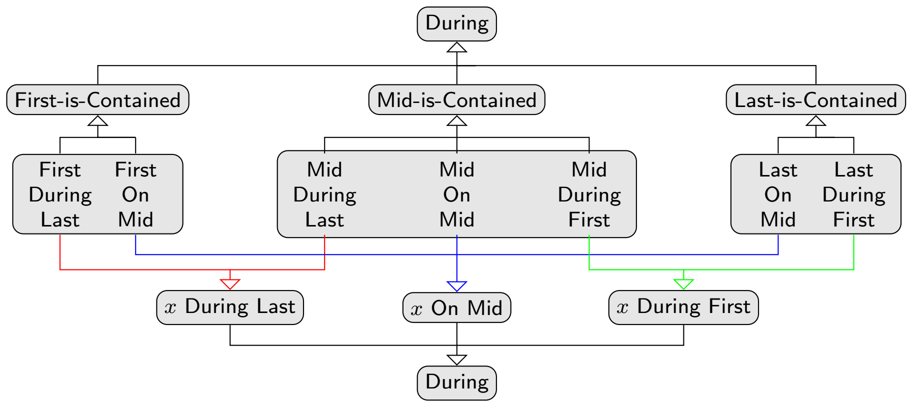


Figure 4.12: Hierarchical structure of the *during* relation.

The *during* hierarchy, see Figure 4.12, cannot be transformed into Equal-Length relationships (they can never occur), however some interesting features can still be extracted that may enable a more 'informed' outcome from any transitive relationships being investigated. The top section of the hierarchy deals with relations with respect to where the containment is situated allowing information to be gleaned regarding the general placement of one interval within another. For example "The parade will be during the second half of the festival", may be represented by a *First-is-Contained* relationship. The following is then evident:

- *Last-is-Contained* is concerned with the interval up to and including the midpoint, and therefore information regarding events that occur in the first half of the interval can be obtained.
- *First-is-Contained* is concerned with the interval from the midpoint to the end of the interval and therefore information regarding events that occur in the second half of the interval can be obtained.
- *Mid-is-Contained* is concerned with the 'middle' section of the interval and therefore information regarding events that occur over the complete interval can be obtained.

The bottom section is less definitive about what information is available but some insights are still possible. The x During First can be used to indicate that at the very least the first part of an interval X is contained completely in the first half of an interval Y, for example "At least the first half of the parade will be concluded by midday". A similar inference can be made using the x During Last relationship with respect to the last half of each interval. The x On Mid relationship is included for completeness, however, it does not yield any more information than using Allen's *during* relationship.

The defining constraints for any during relationship are that:
$(x^- > y^-) \wedge (x^+ < y^-)$

and this is regardless of whether midpoints are used. Therefore, the MI model for the during relationships can be proved by reducing them to the Allen model.

Proposition 4.2. *The MI during relationships are a subset of the Allen during relation.*

Proof: The set of Mid-is-Contained variable-length midpoint relationships has the following members: **mdl**, **mom** and **mdf**, which have the following endpoint and midpoint constraints;

$\langle >\!<\!< \; >\!<\!< \; >\!>\!< \rangle$, $\langle >\!<\!< \; >\!=\!< \; >\!>\!< \rangle$ and $\langle >\!<\!< \; >\!>\!< \; >\!>\!< \rangle$

respectively. By removing any references to midpoints each of these is reduced to

$\langle >\!< \; >\!< \rangle$, $\langle >\!< \; >\!< \rangle$ and $\langle >\!< \; >\!< \rangle$

which are precisely the constraints for the Allen *during* relation. ■

The proofs for the First-is-Contained and Last-is-Contained midpoint during relationships follow directly from this.

4.6.1.4 The Starts and Finishes Hierarchies

As with the *during* relation, it is not possible to transform the *starts* or *finishes* relations into ELMI relationships and therefore it is only a trivial transformation from a VLMI relationship to the Allen relationship that is possible. The information that can be gleaned from each of the Variable-Length relationships is inherent in its naming. The hierarchies are depicted in Figure 4.13.

The *starts* and *finishes* relationships can be seen as 'special' cases of the *during* relationship and their defining constraints are as follows:

Starts: $(x^- = y^-) \wedge (x^+ < y^-)$

and

Finishes: $(x^- > y^-) \wedge (x^+ = y^-)$

and this is regardless of whether midpoints are used. Therefore, the MI model for the starts and finishes relationships can be proved by reducing them to the Allen model.

Proposition 4.3. *The MI starts and finishes relationships are a subset of the Allen starts and finishes relations.*

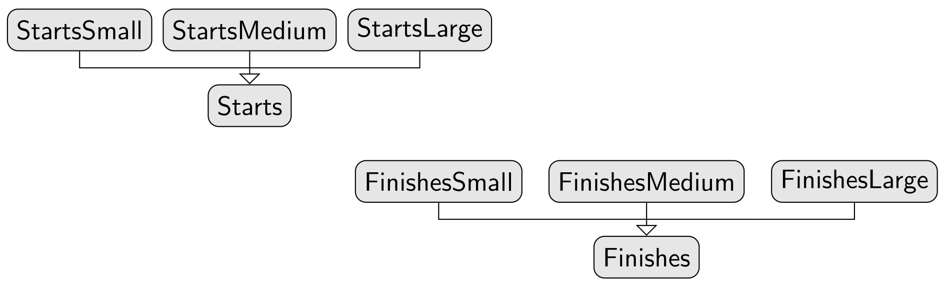


Figure 4.13: Hierarchical structure of the *starts* and *finishes* relations.

Proof: The set of Starts variable-length midpoint relationships has the following members: **ss**, **sm** and **sl**, which have the following endpoint and midpoint constraints;

$\langle =<< \;\; ><< \;\; ><< \rangle$, $\langle =<< \;\; ><< \;\; >=< \rangle$ and $\langle =<< \;\; ><< \;\; >>< \rangle$

respectively. By removing any references to midpoints each of these is reduced to

$\langle =< \;\; >< \rangle$, $\langle =< \;\; >< \rangle$ and $\langle =< \;\; >< \rangle$

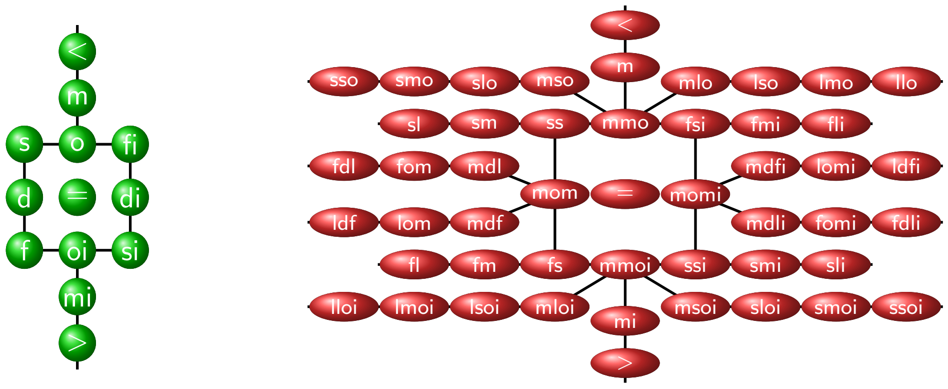
which are precisely the constraints for the Allen *starts* relation. ∎

The proof for the Finishes relationships follow directly from this.

4.7 Iconic Representation

4.7.1 Extensions to Freksa's Iconic Representations

Freksa's (1992) iconic representation of Allen's relations allowed for the expressed 'neighbourhoods' to be more readily interpreted. The icon was constructed in such a way so that conceptual neighbourhoods (the way in which a relation is linked to another through deformation) were preserved. The Freksa icon showing all of the thirteen Allen relations is depicted in Figure 4.14(a). For the VLMI relationships this iconic representation can be extended in a similar way resulting in the extended icon shown in Figure 4.14(b). By extending the representation in this way similar inferences to those using the Freksa semi-interval algebra can also be applied to the VLMI algebra. In addition, by extending the Freksa notation in this way it allows for a more consistent display of the output from the developed software (see Appendix B for details).



(a) The Freksa icon. (b) The extended midpoint icon.

Figure 4.14: Iconic representations of the Allen and VLMI relationships.

As for the Freksa icon, each branch represents a possible deformation and thus the icon facilitates a reading of the temporal relationship between intervals. The extended icons depicted in Table 4.4 represent the major divisions of the VLMI relationships. These equate to a finer granularity of the thirteen Allen relationships. The term finer granularity refers to the fact that the relationships are generated using midpoints and not just the endpoints, which equates to a finer decomposition of the interval.

Table 4.4: The major divisions of the VLMI relationships that equate to a fine grained depiction of Allens thirteen interval-interval relations.

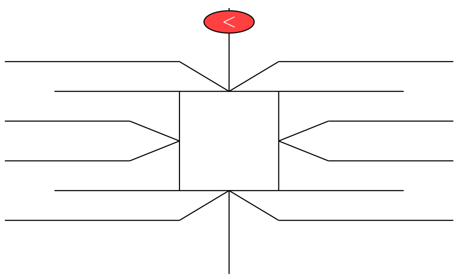
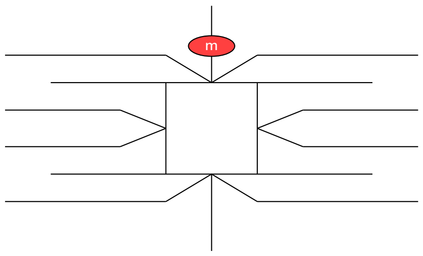
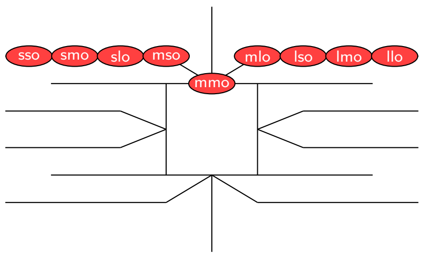
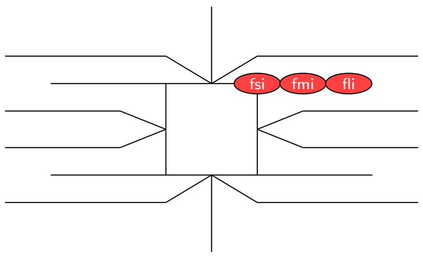
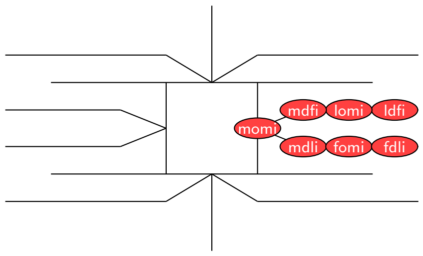
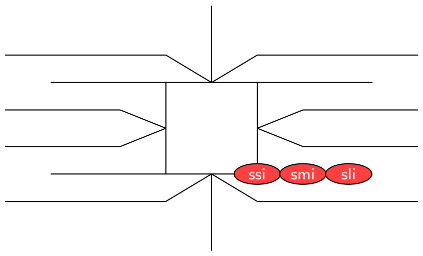
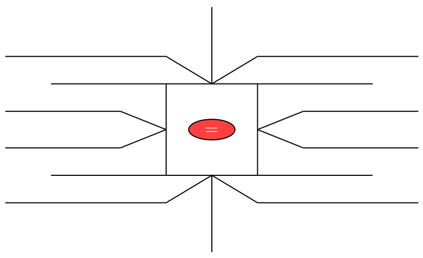
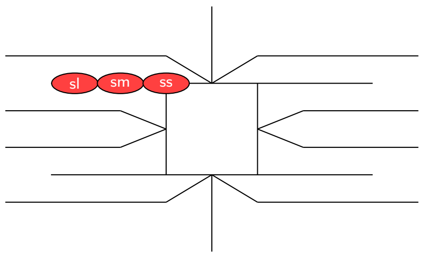
	
Before	Meets
	
Overlaps	is-Finished-by
	
Contains	is-Started-by

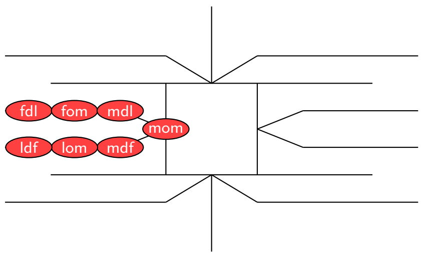
Table 4.4: Major divisions of the VLMI relationships – (continued).



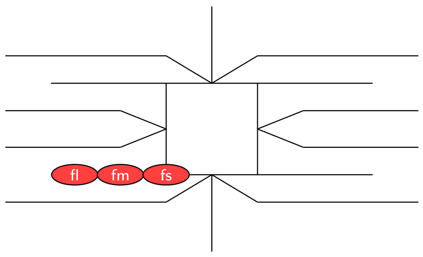
Equals



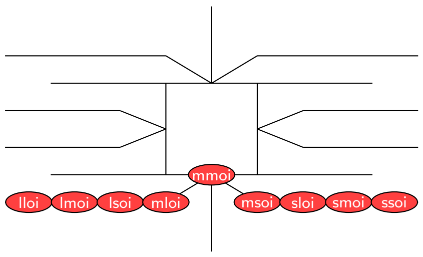
Starts



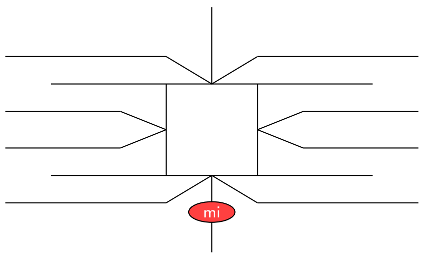
During



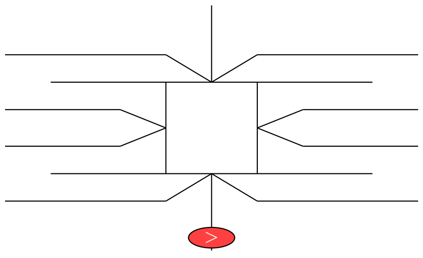
Finishes



is-Overlapped-by



is-Met-by



After

4.8 Discussion

This chapter has demonstrated that, first, an extension to Allen's interval-interval relationship algebra to refine the *overlaps* relationship to consider midpoints for equal length intervals can be used to accommodate linear (non-timestamped) sequences of tokens and allow more appropriate reasoning over stream data such as real-time sensor data. Second, although the equal-length model is sufficient for certain problems, the model has been generalised to a full set of variable-length, interval-interval relationships with midpoints. This model was shown to have properties that enabled transformations into the Allen relationships (without loss of information) and therefore its use is both appropriate for data that include midpoints or when the endpoints are given in absolute time, or for data for which this information is not given.

The application of the Allen and Freksa models has been widely applied (Ladkin, 1987; Vila, 1994; Gennari, 1998) and in principle, given the appropriate data, most of these applications should benefit from this extended midpoint model. Following an exposé on mining interacting episodes and the incorporation of timing marks into that process, Chapter 7 discusses the application of this extended model from the perspective of transitive relationships and will show that the expressive power of this model is at least equal and in many cases superior, in terms of refining the set of relationships returned, to both the Allen and Freksa models.

Chapter 5

Mining Interacting Episodes

In Chapter 2 it was stated that the reason for data mining in general was to generate rules or inferences that could be used as the basis for making decisions related to the datasets being mined. In the case of sequence mining it was shown that the majority of algorithms produce rules with limited expressive power; a typical example being that event X occurred *before/after* event Y. However, there is a growing number of researchers who express the rules as using one or more of the Allen intervals, (Höppner, 2001*a*; Höppner and Klawonn, 2001; Kam and Fu, 2000; de Amo et al., 2005) or some extension or modification of these (Padmanabhan and Tuzhilin, 1996) and a far richer or more expressive set of rules is possible. To enable the expression of these rules, an algorithm for discovery of the frequent sequences is either applied to the data and, as a result may directly produce temporal rules (Padmanabhan and Tuzhilin, 1996; Kam and Fu, 2000), or alternatively, a rule generation algorithm is used to for this production (Höppner and Klawonn, 2001). A further method has been introduced by Bettini, Wang and Jajodia (1996) where an *event structure* is specified and all frequent temporal patterns fitting the structure are discovered.

The purpose of this chapter is to present a framework for the discovery of both frequent sequences and rules, hereafter called *interacting episodes*, based on the interval algebra of Allen, and the Midpoint Interval (MI) algebra elaborated in Chapter 4. The supporting software, to enable the output from these algorithms to be viewed both in textual format, or by way of a visualisation of the discovered sequences and any *interactions* that they contain, is presented in Appendix B. Moreover, if these *interacting episodes* are to be expressed as Allen types, or as Midpoint types, then further reasoning between any such relationships can be facilitated through transitivity tables, the subject of which will be covered in Chapter 7.

Some of the research outlined in this chapter has appeared previously (Mooney and Roddick, 2004), but has since been revised.

5.1 The Framework

The problem of discovering interacting episodes is one that consists of two distinct phases:

Phase 1. mining the frequent episodes
and,
Phase 2. using the discovered frequent episodes as input for the discovery of the *interactions.*

It should be noted that these two processes are sufficiently different, as is the terminology used, and as such the mining problem will be defined separately from the discovery of the interacting episodes. These will be covered in Section 5.2 and Section 5.3 respectively. Algorithms for both phases are presented in Appendix C.

The application that has been developed that makes use of this framework is called $INTEM_{TM}$: **INT**eracting **E**pisode **M**iner with **T**iming **M**arks. This application allows for all constraints to be manipulated and adds a visualisation component that enables both text and pictorial output for both phases. Further additions to the application are highlighted in Chapter 6 and the application itself is presented in Appendix B.

5.1.1 Data considerations

Typically the data used for sequence mining has a structure that includes, as a minimum, an *id*, a *time-stamp* and associated *items* (Agrawal and Srikant, 1995). The data for episodic mining do not necessarily conform to these fields but are generally similar, being a collection of events that occur relatively close to each other in a given partial order (Mannila et al., 1995). These events may, however, have explicit or implicit information about the time or occurrence of the event and each event can have any number of attributes associated with it.

The data used in our experiments has been synthetically generated and consist of a contiguous string of tokens with the time-stamp of each token being implicitly determined by the order in which it occurs. That is, the first token, t_1, is at time-stamp 1 and the n^{th} token, t_n, is at time-stamp n, where $t_1 < t_2 < \cdots < t_n$. In addition each token has no explicit attributes assigned. An example of the type of data can be seen in Figure 5.2 (page 88).

5.2 Frequent Episode Discovery

5.2.1 Problem Definition

Definition 5.1. *Let the set of available input events (the alphabet), denoted T, be defined as $T = \langle t_1, \ldots, t_k \rangle \mid t_i \neq t_j, i \neq j, 1 \leq i, j \leq k$.* ■

Definition 5.2. *A* ***sequence*** *S is then defined as a time ordered sequence of input events and is denoted $S = \langle s_1, s_2, \ldots, s_m \rangle \mid s_i \in T, 1 \leq i \leq m$.* ■

Definition 5.3. *An* ***episode****, denoted E, is a sequence of events, $\langle s_n, s_{n+1}, \ldots, s_{n+k} \rangle$, where $E \subseteq S$ and they occur within a specified window width. The number of events in an episode is called the length of an episode and an episode with a length* ***k*** *is also called a* ***k****-episode. For example, $\langle BRAIJVE \rangle$ is a 7-episode. An episode $E_a = \langle a_1 a_2 \ldots a_n \rangle$ is contained*[1] *in another episode $E_b = \langle b_1 b_2 \ldots b_m \rangle$, if there exist integers $1 \leq i_1 < i_2 < \ldots < i_n \leq m$ such that $a_1 = b_{i_1}, a_2 = b_{i_2}, \ldots, a_n = b_{i_n}$. If episode E_a is contained in episode E_b, then E_a is called a sub-episode of E_b and E_b a super-episode of E_a, denoted as $E_a \sqsubseteq E_b$.* ■

The time at which an event occurs can either be fully specified in the input data (as would be the case with alarm detection data, patient medical data, or a domain such as the stock market), or be implied from the ordering of the input sequence (as would be the case for genomic data).

Definition 5.4. *For data that are temporally specified, the order of two events is* ***serial*** *if a strict time order is placed on the events. For data that are not temporally specified, then the serialisation is implied by the order of the events in the input sequence.* ■

By way of example, for fully specified data: two events α and β are considered serial if $\alpha_{end} < \beta_{start}$ or $\beta_{end} < \alpha_{start}$. For non fully specified data: given the sequence of events $\langle BRAIJVE \rangle$ then it is implied that $B < R < \ldots < V < E$.

For data that are fully temporally specified there exists the possibility that events may occur simultaneously (in parallel), that is, no constraints are placed on the time of occurrence (Mannila et al., 1997; Guralnik et al., 1998). Even in implied ordered datasets it is possible, with the introduction of constraints, to achieve a similar outcome. This and other related considerations will be addressed in Chapter 6, however, in the context of this current discussion, it can be assumed that the ordering of events is strictly *less than* ($<$).

Definition 5.5. *The user defined* ***lookahead****, l, defines the maximum length episode, where $|E| \leq l \leq |S|$.* ■

[1]Here the word *contained* is used in the sense of subset not the temporal logic sense.

This user defined *lookahead*, (which is similar to Mannila et al.'s (1997) window concept), has several advantages:

1. a domain expert may suggest a suitable value or have knowledge of the upper limit of episode length,
2. a user may wish to limit the scope of the search for data with small alphabets, which may otherwise produce large numbers of episodes, i.e. genome or DNA sequences are commonly modelled with the four character alphabet {A,C,G,T}, or as codons and tend to produce very long episodes, which may or may not be required for any analysis at hand.

Definition 5.6. *A **window**, denoted w, is defined as the length of any episode E, where $|E| \leq l$, this will start at one and increase until lookahead l is reached. The maximum number of windows at any time, max_win, then is given by $|S| - w + 1$.* ■

Definition 5.7. *The **frequency** of E in S, freq, denoted Δ, is defined as the number of windows in which E is contained.* ■

Definition 5.8. *The **minimum frequency** required for an episode to be reported, min_freq, denoted δ, is calculated using a user defined support, min_supp, denoted σ, multiplied by max_win.* ■

By calculating the minimum frequency in this manner a sliding scale is produced, similar to Seno and Karypis (2002), with the effect that potentially more interesting longer episodes can be reported at a lower threshold since there are fewer windows for longer episodes. Yang, Wang and Yu (2001) also recognised this phenomenon and introduced a probabilistic measure called *information gain* to achieve a similar result.

Definition 5.9. *If episode E_α is frequent, $\Delta(E_i) \geq \delta$, and there exists no proper super-episode of E_α with the same support i.e., $\nexists E_\beta$ such that $E_\alpha \sqsubset E_\beta$ and min_freq(E_α) = min_freq(E_β), E_α is called a **frequent closed episode**.* ■

The problem definition for Phase 1 can then be stated as:

Phase 1. *Find all frequent closed episodes*

$$E_i(1 < i \leq l) \text{ in } \{\mathbf{S} \mid \Delta(E_i) \geq \delta,\, \delta = (|S| - w + 1) \times \sigma\}$$

5.2.2 Algorithmic Considerations

The classic generation of frequent k length episodes, where $k \geq 2$, involves the self-join of the frequent *(k -1)* episodes, subsequent pruning in accordance with the *anti-monotone* Apriori heuristic (downward closure principle)[2] (Agrawal and Srikant, 1994),

[2]if any length k pattern is not frequent in the database, then its length (k+1) super-pattern can never be frequent.

and finally frequency derivation through a scan of the dataset. Other methods have now been proposed to make this process more efficient when dealing with episodic mining (Guralnik et al., 1998; Huang et al., 2004, 2005), however, since the generation of the sequences, although important, is not the main concern, the algorithm that has been implemented is a modified version of of the WINEPI algorithm (Mannila et al., 1997) for finding serial episodes and also incorporates some of the principles of the PROWL algorithm (Huang et al., 2004). The WINEPI algorithm uses a combination of the downward closure principle and a prefix lookup, which is also similar to the projected window lists of PROWL, only the latter of which has been used in $INTEM_{TM}$. The former is not applicable here since a decreasing sliding scale is used for support.

5.2.2.1 $INTEM_{TM}$ – Episode Discovery

The algorithm used for finding the frequent episodes, (see Algorithm C.1), is a breadth-first search of the input sequence starting with single token episodes. These are pruned according to *min_freq*, δ, and added to the set $\mathcal{F}_1$ of frequent episodes. The window width is now incremented by one and the 2-episodes are generated, which are again pruned according to *min_freq* and those that are frequent are added to the set $\mathcal{F}_2$. At this stage, for the purpose of minimising the size of the candidate sets on subsequent passes of the algorithm, those that are frequent are also added to an *included set* that is used as a prefix lookup. This set is maintained for each subsequent pass of the algorithm and the set of the k-prefixes of frequent episodes is used to improve valid candidate generation, that is, the first k tokens of the generated *(k+1)* candidates are checked against the k-prefixes and retained if there is a match. The *included set* that is maintained only contains the most recent k-prefixes in order to minimise any memory overheads. This candidate pool is then pruned and those candidates that meet the *min_freq* requirements are stored in $\mathcal{F}_{k+1}$[3]. This stage of the algorithm terminates when either the *lookahead*, l, is reached or $\mathcal{F}_{k+1} = \emptyset$. At this point a list of the frequent episodes is created for use in Phase 2 and then the closed set of episodes is generated for reporting.

[3]Since the frequent episodes are available after each pass of the input sequence, if required Phase 2 can be performed in parallel.

5.3 Interacting Episode Discovery

On completion of Phase 1, or if the requirement that interactions be discovered in parallel is enforced, the discovery of the interacting episodes begins. If the parallel option has been chosen this begins during the episode discovery in Algorithm C.1, at line 17, otherwise it is managed as a separate process. Since the process is the same regardless of where the user chooses this discovery to take place[4], the process is presented here as though it were being conducted at the conclusion of Phase 1.

5.3.1 Problem Definition

Definition 5.10. *Let $\mathcal{F}$ be the set of frequent episodes and $\mathcal{F}_k$ be the set of frequent k-length episodes.*

Definition 5.11. *Let C be the set of candidate episodes and C_k be the set of candidate k-length episodes.*

Definition 5.12. *Let an interaction be a temporal relationship between a frequent sub-episode e_i and a sub-episode e_j, contained in a frequent episode E, such that $|e_i|+|e_j| = |E|$, and where e_j is either:*

1. *frequent, $\in \mathcal{F}$, and denoted $\theta_r(e_i, e_j) \mid r \in \mathcal{R}$, (see Section 5.3.3.1)*
2. *infrequent but a candidate, $\in C$, and denoted $\psi^{w[p]}(e_i, e_j) \mid w \in \mathcal{R}$, (see Section 5.3.3.2)*
3. *only exists due to the discovery process, $\Delta = 0$, and denoted ${}_{[c]}\eta^d(e_i, e_j) \mid d \in \mathcal{R}$, (see Section 5.3.3.3)*

, where $\mathcal{R}$ is the set of temporal relationships as described by Allen (1983) (see Table 3.1, page 52), or MI relationships as described in Chapter 4. ■

Definition 5.13. *Let the interaction length be the length of either sub-episode e_i or e_j and further let the minimum interaction length, min_interaction_length, γ, be a user supplied value to limit the interaction length.* ■

Having this *min_interaction_length* enables a domain expert to minimise the number of interactions reported to those of potential interest.

Definition 5.14. *Let min_interaction_supp, φ, be the user defined level of minimum support.* ■

It is usually the case that rule generation produces an excessive number of candidates and therefore a suitable threshold value must be applied so that this may be minimised. This threshold value could be based on a combination of factors including:

[4]Smaller pieces are processed during the parallel option.

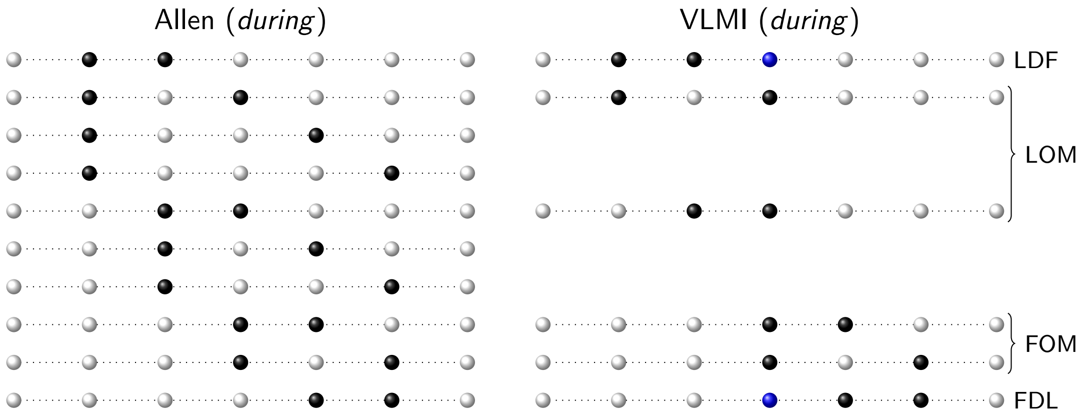


Figure 5.1: Possible positions for a sub-episode $e_2 \rightarrow$ ●, ($|e_2| = 2$) with a sub-episode, $e_1 \rightarrow$ ○, ($|e_1| = 5$).

1. the combined episode length: $|e_i| + |e_j|$,
2. the maximum number of possible positions that a sub-episode, e_i, can occur within an episode, E
   or,
3. the frequency (count) of an interaction expressed as a percentage of the total frequency of the frequent episodes of length $|e_i| + |e_j|$.

The determination of appropriate reporting thresholds is always a difficult task and can be very subjective. This research has found however, that since Item (3) takes into account not only the length of the combined sub-episodes (as does Item (1)) but also only those episodes of the length in question, it is the best metric to use.

Item (2) may have some merit if only Allen type interactions were to be discovered since there are more positions available for any sub-episode to occupy than would be the case for midpoint relationships. For example, see Figure 5.1, using the Allen *during* relationship and an episode E of length 7, comprising two sub-episodes e_1 and e_2 where $|e_1| = 5$ and $|e_2| = 2$, then e_2 has 10 possible positions with e_1. Given the same episode, E, and sub-episodes e_1 and e_2, using the VLMI *during* relationships, of which there are 7, the possibilities are much less; 1 each of *LastDuringFirst* and *FirstDuringLast* and 2 each of *FirstOnMid* and *LastOnMid*, and therefore potentially fewer interactions would be reported.

It should also be noted that any metric that is to be imposed will also be highly dependent and sensitive to the number of discovered frequent episodes, and therefore *min_support*, δ, see Definition 5.8.

Definition 5.15. *Let the interaction_count* $\xi = \left[\frac{frequency\ \theta_{r_k}}{\sum_k frequency\ |\theta_{r_k}|}\right]$, *where* $frequency\ \theta_{r_k}$ *is the count of any specific interaction of length k, and* $\sum_k frequency\ |\theta_{r_k}|$ *is the count of all interactions of length k. For an interaction to be reported* $\xi \geq \varphi$. ■

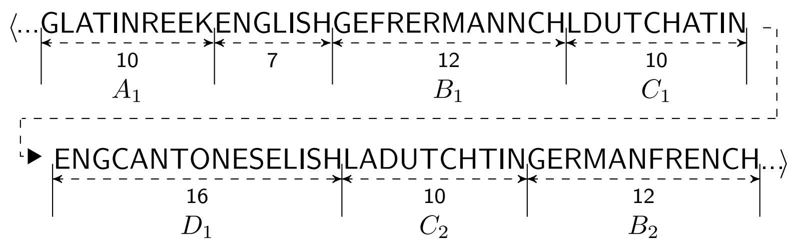

⟨...GLATINREEKENGLISHGEFRERMANNCHLDUTCHATIN
10
7
12
10
A_1
B_1
C_1
ENGCANTONESELISHLADUTCHTINGERMANFRENCH...⟩
16
10
12
D_1
C_2
B_2

Figure 5.2: Section of an input string showing varying window widths.

The method for performing this task is shown in Algorithm C.4.

The problem definition for Phase 2 can then be stated as:

Phase 2. *: given a list of frequent episodes, $\mathcal{F}$.*

$$\forall E_k \in \mathcal{F}^l_{k=\gamma} \textit{ find all } \theta_{r_k}(e_i, e_j) \textit{ in } \{E_k \mid |e_i|, |e_j| \geq \gamma\,, \, \xi \geq \varphi\,\}\,.$$

The following two examples serve to illustrate the nature of the problem.

Example 5.1. Given the frequent episodes E_1, E_2, E_3 where:

$E_1 = \langle B, R, I, J, A, V, E\rangle$,
$E_2 = \langle B, I, R, A, J, V, E\rangle$, and
$E_3 = \langle B, R, A, I, J, V, E\rangle$.

By inspection it can be seen that if $e_1 = \langle I, J\rangle$ and $e_2 = \langle B, R, A, V, E\rangle$ then:

1. For Allen relationships
   - all three episodes E_1, E_2,and E_3 can be described as IJ *during* BRAVE, denoted $\theta_d(e_1, e_2)$.
2. For VLMI relationships, however, inspection is more difficult but yields
   - Episode E_1 as an example of a *LastOnMid* relation, denoted $\theta_{lom}(e_1, e_2)$ – IJ *LastOnMid* BRAVE,
   - Episode E_2 as an example of a *MidDuringFirst* relation, denoted $\theta_{mdf}(e_1, e_2)$ – IJ *MidDuringFirst* BRAVE, and
   - Episode E_3 as an example of a *FirstOnMid* relation, denoted $\theta_{fom}(e_1, e_2)$ – IJ *FirstOnMid* BRAVE. □

A more complex example can be shown by using the section of input string in Figure 5.2 as the source for the discovery of the interactions where, A_1, B_1, B_2, C_1, C_2, and D_1 are assumed to be frequent. The input string itself, DAT15-650 (see Table B.1), was used as a synthetic data source for testing algorithms in the discovery process and contains languages for the purpose of readability and easy identification of any detected interactions.

Example 5.2. Given the following frequent episodes the task is to identify any relationships (both Allen and VLMI) that may exist within them:

$$
\begin{aligned}
A_1 &= \langle G, L, A, T, I, N, R, E, E, K \rangle \\
B_1 &= \langle G, E, F, R, E, R, M, A, N, N, C, H \rangle \\
B_2 &= \langle G, E, R, M, A, N, F, R, E, N, C, H \rangle \\
C_1 &= \langle L, D, U, T, C, H, A, T, I, N \rangle \\
C_2 &= \langle L, A, D, U, T, C, H, T, I, N \rangle \\
D_1 &= \langle E, N, G, C, A, N, T, O, N, E, S, E, L, I, S, H \rangle
\end{aligned}
$$

1. For Allen relationships
    - Episodes A_1, C_1, C_2, and D_1, are all examples of the *during* relation, denoted $\theta_d(e_1, e_2)$ – LATIN *during* GREEK, DUTCH *during* LATIN and CANTONESE *during* ENGLISH respectively,
    - Episode B_1 is an example of an *overlap* relation, denoted $\theta_o(e_1, e_2)$ – GERMAN *overlaps* FRENCH, and
    - Episode B_2 is an example of a *meets* relation, denoted $\theta_m(e_1, e_2)$ – GERMAN *meets* FRENCH.
2. For VLMI relationships
    - Although the Episodes A_1, C_1, C_2, and D_1, are all examples of some form of the *during* relation,
        - Episodes A_1 and C_1 are examples of the *MidDuringFirst* relation, denoted $\theta_{mdf}(e_1, e_2)$ – LATIN *MidDuringFirst* GREEK and DUTCH *MidDuringFirst* LATIN, and
        - Episodes C_2, and D_1, are examples of the *MidOnMid* relation, denoted $\theta_{mom}(e_1, e_2)$ – DUTCH *MidOnMid* LATIN and CANTONESE *MidOnMid* ENGLISH.
    - Episode B_1 is an example of the *LargeLargeOverlap* relation, denoted $\theta_{llo}(e_1, e_2)$ – GERMAN *LargeLargeOverlap* FRENCH, and
    - Episode B_2 remains unchanged. □

These examples serve to show that although one can easily detect Allen type relationships, this is not the case for MI relationships, and from experience, gained during this research, as the episodes get longer and more numerous, this task becomes increasingly difficult.

5.3.2 An Algorithm for Interaction Discovery

Chapters 3 and 4 have shown that intervals can be described using either the endpoint constraints, Table 3.1, or by using both the endpoint and midpoint constraints,

Table 4.3. This feature has been exploited in the approach to the discovery of interactions detailed here. Lookup tables that contain the endpoint constraints for the Allen relationships and the endpoint and midpoint constraints for the VLMI relationships, have been generated and as a result interactions can be found that conform to either of these two types. For the case of the Freksa semi-interval relationships only certain types of data are viable for interaction discovery, and the relationships can be vague in their information content. As such the interaction discovery process does not detect these relationships at this time. This does not suggest that they may not offer some valuable insights into the data.

5.3.2.1 $INTEM_{TM}$ – Interaction Discovery

Interaction candidate generation is conducted using an exhaustive search for each frequent k-episode in all frequent *(k+n)*-episodes. For each frequent episode this will result in the production of two sub-episodes, the first being the original frequent k-episode and the second being the sub-episode that remains from the frequent *(k+n)*-episode after the frequent k-episode has been removed from it. This will guarantee that all possible combinations of frequent episodes are used in the search and thus is complete with respect to the definition of Phase 2. Formally:

$$\text{search } \forall E_k \in \mathcal{F}_k \text{ in } \{\mathcal{F}_{k+n} \mid n = 1, \ldots, l-k\}$$

to yield

$$e_i \leftarrow E_k \in \mathcal{F}_k \text{ and } e_j \leftarrow (E_{k+n} - E_k) \in \mathcal{F}_n \text{ or } \in C_n \text{ or } \Delta = 0$$

where $\mathcal{F}_k$ is the set of frequent k-length episodes, $\mathcal{F}_n$ is the set of frequent n-length episodes, C_n is the set of candidate n-length episodes, and $\Delta = 0$ means that e_j exists only due to the discovery process.

The midpoint and endpoints are now determined for both e_i and e_j, with respect to the frequent *(k+n)*-episode from which they came, and a constraint pattern is generated. This pattern is generated by comparing the relative position of $e_i.start$ to $e_j.start, e_j.mid$ and $e_j.end$ and similarly for $e_i.mid$ and $e_i.end$. This process is summarised in Table 5.1.

Table 5.1: Constraint pattern propagation rules.

Position of		To				To yield		
$e_i.start$	$\mapsto$	$e_j.start,$	$e_j.mid,$	$e_j.end$	$\Rrightarrow$	$(\{<,=,>\}^*$	$\{<,=,>\}$	$\{<,=,>\}^*)$
$e_i.mid$	$\mapsto$	$e_j.start,$	$e_j.mid,$	$e_j.end$	$\Rrightarrow$	$(\{<,=,>\}$	$\{<,=,>\}$	$\{<,=,>\})$
$e_i.end$	$\mapsto$	$e_j.start$	$e_j.mid,$	$e_j.end$	$\Rrightarrow$	$(\{<,=,>\}^*$	$\{<,=,>\}$	$\{<,=,>\}^*)$

Generating the constraint pattern in this way enables both VLMI and Allen relationships to be discovered, since all nine constraints are used for VLMI relationships and only those that relate to endpoints (indicated by an asterisk, "*", in Table 5.1) for Allen relationships. These procedures are detailed in Algorithm C.2 and C.3. The following example will serve to clarify the procedure.

Example 5.3. Constraint Pattern Propagation
Given:
the frequent k-episode $E_k = \langle D, U, T, C, H\rangle$
the frequent *(k+n)*-episode $E_{k+n} = \langle L, A, D, U, T, C, H, T, I, N\rangle$
then
$e_i = \langle D, U, T, C, H\rangle$ and $e_j = \langle L, A, T, I, N\rangle$,
and therefore
$e_i.start = 3$, $e_i.mid = 5$, $e_i.end = 7$, and
$e_j.start = 1$, $e_j.mid = 5$[5], $e_j.end = 10$

This results in the following constraint patterns – based on the above propagation rules (see Table 5.1).

Allen	(><><)
Midpoint	(><< >=< >><)

and after performing a lookup into the relevant table, (Table 3.1 or Table 4.3), the relationships returned are *During* and *MidOnMid* and therefore $\theta_d(DUTCH, LATIN)$ and $\theta_{mom}(DUTCH, LATIN)$ respectively. □

The outcome from Example 5.3 assumes that both e_i and $e_j \in \mathcal{F}_k$, that is they are both frequent and reportable as episodes. This equates to the most desirable outcome, since it generates the strongest rules, but this may not always be the case. A further consequence of the generation process is that two other possible combinations of sub-episode may also be discovered. These combinations arise due to the frequency of e_j in the dataset; it may be a candidate that has not reached threshold, or it may not have even been considered as a candidate. The following section discusses these possibilities.

5.3.3 Interaction Classes

5.3.3.1 Strong Interactions

The class of strong interactions arise when both e_i and e_j are frequent and reportable as episodes, $(e_i, e_j \in \mathcal{F})$, and is the most desirable outcome, since it produces the strongest

[5]In this case the true midpoint is 5.5. The occurrence of a fractional midpoint is quite common and so that consistency of results is preserved the decision has been made to always round down.

rules. Given this combination the following relationships are possible: *meets*, *during*, *overlaps* and their inverses. The *equals* relationship is excluded from any discussion since it is not possible for it to be discovered using the procedure already outlined, see Section 5.3.2.1. The reasons for not reporting the remaining relationships, when this combination of sub-episodes is evident, follows.

For either *before* or *after* to be reported there is a requirement for some interval, or noise, between each of the sub-episodes. For example, the frequent episode $\langle\alpha_1\ \alpha_2\ \alpha_3\ \rho_1 \ldots \rho_n\ \beta_1\ \beta_2\ \beta_3\ \beta_4\rangle$, is made up of α_i and β_j which are the frequent sub-episodes e_i and e_j respectively and $\rho_{1\ldots n}$ which represents an interval, or noise, whichever is more appropriate. For the case of noise this could have been discovered from either e_i or e_j by way of further processing. This further processing could be conducted on a *meets* relationship[6], but given the procedure currently used for the discovery of interactions, this is precluded. Although further processing is not difficult, a recursive procedure can be implemented to further process either e_i or e_j, the overheads may be such that this becomes impractical and therefore a more suitable method may need to be found. A more practical solution is to report these as *meets* relationships and to leave any further processing for any infrequent e_j that may be considered as part of a *weak* (Section 5.3.3.2) or *dependent* (Section 5.3.3.3) *meets* interaction[7].

The *starts* and *finishes* relationships are also never reported under this scenario since the constraints can never be generated[8], but a domain expert may in fact transform a *meets* relationship into either a *starts* or *finishes* relationship. However, when e_j is infrequent, even though the constraints are still not generated, it is possible that these types can be reported by inference, see Section 5.3.3.2 for details.

Definition 5.16. *Let a strong interaction θ be a temporal relationship between frequent sub-episodes $(e_i, e_j) \mid |e_i| + |e_j| = |E|$, denoted $\theta_r(e_i, e_j) \mid r \in \mathcal{R}$, where $\mathcal{R}$ is the set of temporal relationships as described by Allen (1983) (see Table 3.1, page 52), or MI relationships as described in Chapter 4.* ■

5.3.3.2 Weak Interactions

Weak interactions arise when e_i is frequent and reportable as an episode ($e_i \in \mathcal{F}$) and e_j is a valid candidate episode ($e_j \in C$) but not frequent ($\Delta(e_j) < \delta \rightarrow e_j \notin \mathcal{F}$) and as such is not reportable as an episode. This situation can occur since the discovered frequent sequences do not adhere to the downward closure principle, and as such it is possible that during interaction discovery some of the sub-episodes, that have the potential to become part of a relationship, may be infrequent. This may occur for all possible relationship types, with the exception of *before* and *after* where more processing is required as previously explained.

[6]The position is adopted that a *meets* relationship occurs when two episodes abut each other.
[7]The best method for accomplishing this remains as a future research project.
[8]Both *starts* and *finishes* require that one of the endpoints be equal for both sub-episodes.

In a previously published paper, (Mooney and Roddick, 2004), two new types (*participant* and *container*) were defined for the *during* relationships under this scenario, since they were the only relationships in the *weak interaction* class that were discovered at that time. However, due to an improved algorithm, it is now possible to detect all relationship types, except *before*, *after* and *equals*. The result is that these type names have been replaced by a more consistent naming convention, outlined in Definition 5.17, although they are still used when describing them. Moreover, since this class of interaction does not comply with the definition of an interaction given earlier, see Definition 5.12, the need to define them becomes necessary.

Definition 5.17. *Let sub-episode $e_i \in \mathcal{F}$ be frequent, and sub-episode $e_j \in C$ be infrequent, $(\Delta(e_j) < \delta \rightarrow e_j \notin \mathcal{F})$. A weak interaction ψ is a temporal relationship between sub-episodes $(e_i, e_j) \mid |e_i| + |e_j| = |E|$, denoted $\psi^{w[p]}(e_i, e_j) \mid w \in \mathcal{R}$, where $\mathcal{R}$ is the set of temporal relationships as described by Allen (1983), or* VLMI *relationships as described in Chapter 4, and $[p]$ is a weighting calculated using the formula $p = \frac{\Delta(e_j)}{\delta}$.*

■

The following discusses the possible configurations sub-episodes and how they are denoted. The discussion outlines the more general Allen relationships, but these can be transferred (without loss of generality) to the VLMI relationships for which they (the Allen relationships) are a parent. All of the configurations assume that $e_i = \langle \alpha_1\ \alpha_2\ \ldots\ \alpha_n \rangle$ is frequent and $e_j = \langle \beta_1\ \beta_2\ \ldots\ \beta_m \rangle$ is not.

During: This relationship can be present in one of two forms.

- $\langle \boldsymbol{\alpha_1}\ \boldsymbol{\alpha_2}\ \beta_1\ \beta_2\ \beta_3\ \boldsymbol{\alpha_3}\ \boldsymbol{\alpha_4} \rangle$
- $\langle \beta_1\ \beta_2\ \boldsymbol{\alpha_1}\ \boldsymbol{\alpha_2}\ \boldsymbol{\alpha_3}\ \beta_3\ \beta_4 \rangle$

These are denoted: $\psi^{d[p]}(e_i, e_j)$ and $\psi^{di[p]}(e_i, e_j)$, respectively.

The first is called a *participant* interaction since the infrequent e_j participates in the *during* relationship and the second a *container* interaction since the infrequent e_j acts as a container to enable the *during* relationship.

Overlap: This relationship can be present in one of two forms.

- $\langle \boldsymbol{\alpha_1}\ \boldsymbol{\alpha_2}\ \beta_1\ \beta_2\ \boldsymbol{\alpha_3}\ \boldsymbol{\alpha_4}\ \beta_3 \rangle$
- $\langle \beta_1\ \beta_2\ \boldsymbol{\alpha_1}\ \boldsymbol{\alpha_2}\ \beta_3\ \beta_4\ \boldsymbol{\alpha_3} \rangle$

These types are denoted: $\psi^{o[p]}(e_i, e_j)$ and $\psi^{oi[p]}(e_i, e_j)$, respectively.

No alternative descriptive names have been coined for these two relationship types since they conform to the standard terminology of the overlap and overlap inverse relationships. However, they are only reported from the perspective of the dominant (frequent) sub-episode.

Start and Finish: This relationship can be present in only one form.

- $\langle \boldsymbol{\alpha_1}\ \boldsymbol{\alpha_2}\ \boldsymbol{\alpha_3}\ \boldsymbol{\alpha_4}\ \beta_1\ \beta_2 \rangle$

These relationship types are never captured when the requirement that both sub-episodes be frequent is enforced, rather they are reported as *meets* relationships. However, given that both sub-episodes are frequent, further analysis by a domain expert may in fact deem these (*meets*) relationships to be *starts* or *finishes* relationships.

Since the algorithm that generates the frequent sequences ensures that any prefix must be frequent, every relationship that is deemed *starts* or *finishes* will have the same structure – a frequent sub-episode followed by an infrequent sub-episode. This has the effect that only *starts* relationships are reported. Again, a domain expert may evaluate a selection of these to be in fact *finishes* and it is evident that each *finishes* relationship can be viewed as the inverse of a *starts* relationship and therefore be reported in that way. However, to enable these relationships to be reported using one that provides the most information, from a frequency perspective, the logical choice is to report them as *starts* relationships.

Under the conditions described above it is stated that:

e_i *starts* E, denoted $\psi^{s[p]}(e_i, E)$, or
e_j *finishes* E, denoted $\psi^{f[p]}(e_i, E)$

where $E = (e_i + e_j)$.

Meets: This relationship can be present in only one form.

- $\langle \boldsymbol{\alpha_1}\ \boldsymbol{\alpha_2}\ \boldsymbol{\alpha_3}\ \boldsymbol{\alpha_4}\ \beta_1\ \beta_2 \rangle$

As can be seen, this configuration is identical to the *starts* and *finishes* configuration and therefore using the arguments already outlined these types would never be reported as *meets* relationships but rather *starts* or *finishes.* Again, it may be possible that a domain expert deems that the configuration is in fact a *meets* relationship.

These types are denoted: $\psi^{m[p]}(e_i, e_j)$.

Before and After: These types of relationships are never reported due to the fact that more processing of e_j is required to ascertain whether they contain interactions. As has been stated earlier this recursive processing is left as a future research effort.

All member relationships of weak interactions are implemented using a weighting determined as a percentage of *min_sup*, which is stored with the relationship. By implementing them in this way they can be handled using the formalism described in Section 3.4.2 and expanded by Badaloni and Giacomin (1999, 2002, 2006). This weighting can then be used as a means of determining the strength of any relationship and also as an aid in any determination towards the strength of transitive relationships during extended reasoning.

5.3.3.3 Dependent Interactions

Dependent interactions arise when e_i is frequent and reportable as an episode, ($e_i \in \mathcal{F}$), and e_j only exists due to the process of discovery, ($\Delta(e_j) = 0 \rightarrow e_j \notin C$) and they represent the weakest class of relationship that can be detected. The types of relationships that may be formed under these circumstances are the same as for *weak interaction* class, but their meaning is somewhat different. Since one of the sub-episodes has not ever been considered a candidate, no frequency (count) is available other than it being the same as the episode from which it was detected, which in itself is frequent. The question then becomes whether the non-candidate episode should be reported as an anomaly or disregarded. In the current environment of $INTEM_{TM}$ the latter is the case, and therefore it has not been implemented.

For completeness the notation that would be used if and when these types are reported has been included.

Definition 5.18. *Let sub-episode $e_i \in \mathcal{F}$ be frequent, and sub-episode be detected due to the process of discovery, that is: $\Delta(e_j) = 0 \rightarrow e_j \notin C$. A dependent interaction η is a temporal relationship between sub-episodes $(e_i, e_j) \mid |e_i| + |e_j| = |E|$, denoted ${}_{[c]}\eta^d(e_i, e_j) \mid d \in \mathcal{R}$, where $\mathcal{R}$ is the set of temporal relationships as described by Allen (1983), or VLMI relationships as described in Chapter 4, and c is the frequency (count) of E.* ■

5.3.3.4 Interaction Class Summary

The results of interaction discovery can yield three different classes *strong*, *weak* and *dependent*, each of which contain a subset of the Allen or VLMI relationships. These are summarised in Table 5.2. A result of the discovery process is also that the same frequent episodes e_i and/or e_j could be involved in several *strong* interactions, and that a frequent e_i and an infrequent e_j could be involved in several *weak* or *dependent* interactions. If this were the case then this involvement would be denoted:

Strong Interactions – $\theta_{d,o}(e_i, e_j)$
Weak Interactions – $\psi^{s[0.6],o[0.3]}(e_i, e_j)$

and if dependent interactions were to be reported

Dependent Interactions – ${}_{[956,1056]}\eta^{m,s}(e_i, e_j)$

Table 5.2: Possible configurations of sub-episodes within a frequent episode and the resultant interaction class.

	e_i		e_j			E	
Interaction Class	*FR*	*NF*	*FR*	*NF*	*NC*	*FR*	*NF*
Strong	✓		✓			✓	
Weak	✓			✓		✓	
Dependent	✓				✓	✓	
(currently not reported)							
Never Reported		✓		✓			✓

FR: frequent, *NF*: non-frequent, *NC*: non-candidate

These interactions can then be used as a starting point for reasoning using either the Allen, Freksa or MI transitivity tables. It is also important to note the similarity of notation of both the *weak* and *dependent* interactions with the IA^{fuz} terminology of Badaloni and Giacomin (1999, 2006).

5.3.3.5 Interaction Pruning

Pruning of the candidate interactions is performed by calculating the frequency of the particular interaction as fraction of the total number of interactions of the same length initial episode and then comparing this with a user defined minimum interaction support, *min_interaction_supp*, φ. The algorithm to perform this is elaborated in Algorithm C.4. The $INTEM_{TM}$ application also includes a mechanism to constrain the reportable interactions by the length of the included sub-episodes, see Figure B.3 and Section B.1.3 for details of its use.

5.3.4 Common Tokens

The decision of how to deal with common tokens that occur within both sub-episodes that constitute a relationship, as in DUTCH *during* LATIN in Example 5.2 – with a common token T, is one which also needs to be addressed. This is handled directly by the interaction discovery process, because using the point based approach the only interest, initially, is in the start and end of the sub-episodes, and therefore the method locates the first token that results in a match for the sub-episode in question. This process, using endpoints, works regardless of the number of repeated tokens and is shown in Algorithm C.3.

5.3.5 Interruptions at different locations

Sub-episodes interrupting at different locations satisfy one of the three classes of interaction, *strong*, *weak*, or *dependent*, and therefore yield the same two sub-episodes,

e_i and e_j. Depending on whether Allen or VLMI relationships have been selected for discovery has a bearing on the final outcome of this scenario. Since there are multiple VLMI variations of some of the Allen relationships (7 during, 9 overlap, 3 starts etc...) the result may be a unique relationship, or may be all the same. However, with Allen types, and for those VLMI types that yield the same relationships, the count for that particular combination of sub-episodes can be incremented. This will result in an increased support and hence will indicate a stronger relationship.

For example, given frequent episodes:

$E_1 = \langle \alpha_1\ \alpha_2\ \beta_1\ \alpha_3\ \alpha_4\ \beta_2\ \beta_3\ \alpha_5\ \alpha_6 \rangle$,
$E_2 = \langle \alpha_1\ \beta_1\ \alpha_2\ \alpha_3\ \beta_2\ \beta_3\ \alpha_4\ \alpha_5\ \alpha_6 \rangle$,
$E_3 = \langle \alpha_1\ \alpha_2\ \alpha_3\ \alpha_4\ \beta_1\ \alpha_5\ \beta_2\ \beta_3\ \alpha_6 \rangle$,
$E_4 = \langle \alpha_1\ \alpha_2\ \alpha_3\ \beta_1\ \beta_2\ \alpha_4\ \alpha_5\ \beta_3\ \alpha_6 \rangle$, and
sub-episodes $e_i = \alpha_{1..n}$ and $e_j = \beta_{1..m}$

Assuming (without loss of generality) that the resultant relationships belong to a particular interaction class (*strong*), the following is possible when the type indicated is being used in the discovery process:

Allen Discovery:

$E_{1..4} \rightarrow \theta_d(e_i, e_j)$

and therefore the count could be incremented, resulting in a stronger rule.

Midpoint Discovery:

$E_1 \rightarrow \theta_{mom}(e_i, e_j)$
$E_2 \rightarrow \theta_{mdf}(e_i, e_j)$
$E_3 \rightarrow \theta_{fom}(e_i, e_j)$
$E_4 \rightarrow \theta_{mdl}(e_i, e_j)$

for which the count can not be incremented, but each of the discovered relationships may in fact offer stronger information content and therefore be just as valuable.

5.4 Discussion

This chapter discusses the development of a method for mining relationships between interacting episodes based on the Interval algebra of Allen (1983) and the MI algebra expressed in this thesis and lays down the foundation for the further processing of the discovered interactions by determining any transitive relationships in which they participate.

An important consideration has also been the reporting of episodes and interactions that are based on frequency metrics, imposed as user-defined constraints, and while this produces valuable results it may be necessary to alter this approach to cater for more diverse data in different domains. For example, the reporting heuristic for interactions (see Section 5.3.2), while adequate for some domain data, may prove to be lacking in datasets that generate numerous frequent episodes, or have small alphabets, and hence a different heuristic may be required. Methods that are not based solely on frequency metrics, for example, the concept of *information gain* as discussed by Yang, Wang and Yu (2001), and the application of string edit distance techniques (Arslan and Egecioglu, 2000; Cormode and Muthukrishnan, 2002) to the candidate episodes and interactions, both independently and in concert, may assist in overcoming the problems associated with numerous frequent episodes, and allow the reporting of interactions that are both frequent and of interest.

Chapter 6

Timing Considerations

While many sequences are associated with absolute time values, most sequence mining routines treat time in a relative sense, only returning patterns that can be described in terms of Allen-style relationships (or simpler). Sequence mining can be conducted over static and temporal datasets as well as over collections of events (episodes) and more recently, there has been a focus on streaming data and various algorithms have being proposed. As has been stated frequent-pattern (sequence) mining from static databases has been conducted over a number of years and algorithms for this form of mining are relatively mature (Srikant and Agrawal, 1996; Pei et al., 2001; Wang and Han, 2004; Yan et al., 2003). Transaction datasets commonly include a *time-stamp* for each transaction and it is this that can be used, in conjunction with a *transaction_id*, for constraining the mining activity with respect to time granularity.

However, sequence mining is not limited to data stored in transaction-structured datasets and there are other domains where an implicit time-stamp may or may not be included such as genome sequencing, web logs, alarm data in telecommunications networks, sensor data, and so on. In such domains data can be viewed as a series of events occurring at specific times and therefore the problem becomes a search for collections of events (episodes) that occur frequently together. Solving this problem requires a different approach, and several types of algorithm have been proposed for different domains (Mannila and Toivonen, 1996; Mannila et al., 1997; Spiliopoulou and Faulstich, 1998; Mooney and Roddick, 2004). Such datasets can also be very similar in nature to, or are themselves, streaming datasets, an area of research that is gaining significant interest at present (Gaber et al., 2005; Giannella et al., 2003; Lin et al., 2003). However, the datasets used in these domains do not always include a *time-stamp* and this reduces the problem to those that occur close to each other in the sequence. This changes the semantics of *frequent* and makes mining more problematic if time constraints are required, or if information relative to the pace of the activity is required.

The focus of the previous chapter was one in which the time-stamp of each token was implicitly determined by the order in which they (the tokens) occur, which is exactly the problem outlined above. This chapter addresses this problem by introducing the notion of a ***timing mark*** (or ***timing tick***) to accommodate absolute time within the sequence mining process. This allows the process not only to provide information relative to order and occurrence of sequences but also the pace at which they occurred. Some of the research outlined in this chapter has appeared previously in conference proceedings (Mooney and Roddick, 2006), but this has since been revised before inclusion.

6.1 Timing Marks

The concept of *timing marks* introduced here refers to embedded tokens that indicate the passage of time. They are not *time-stamps* in that they do not record absolute time values but rather *ticks* which can be referenced to determine the pace of events.

For example, the notion of polled data infers a (fixed) time interval during which the polling occurs. During this interval, it may be possible that not all sensors are read, some do not return data or that some need to be read. In addition to this, many sequences of events have a time-stamp, either inherently in how they are reported, or overlaid by a system that needs to interrogate the data. How this fixed time element is encoded in the data is of interest. In traditional sequence mining, time-series mining and web-log mining each element to be mined has a time-stamp associated with it and therefore encoding a time element or *timing mark* is not necessary. With sensor data and other data streams there is usually no time-stamp and therefore it is necessary to include a time-stamp or *timing mark* into the data.

The previous chapter on mining interacting episodes implicitly assumed (in common with other researchers) that each token (sensor reading) occurred for a fixed period of time and that the time between tokens was zero, or alternatively, that events are instantaneous and the time between tokens was constant. That is, a sequence of n tokens could be viewed as occurring over n time periods of equal length, no matter what the granularity of that period was (Mooney and Roddick, 2004). The presentation in this chapter relaxes this assumption. That is, although the time between events may remain unchanged – equal length intervals – the number of tokens (events) that occur within that time can vary. These two possible scenarios are depicted in Figure 6.1. To accommodate this assumption *timing marks* have been introduced into the data. These *timing marks* may have different properties depending on the data they are associated with and more generally *timing marks* can be viewed as having the following possibilities. In all examples that follow the period '.' is used as the notation of the *timing mark*.

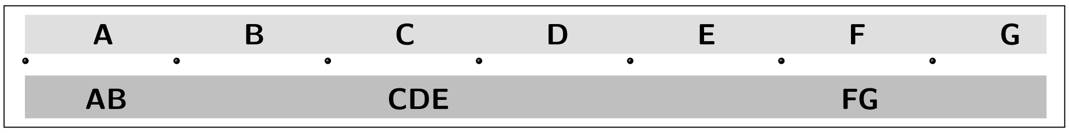


Figure 6.1: Possible structure of data when *timing marks* are included. The top section shows the structure used in Chapter 5 and the bottom section shows one possible scenario of tokens when *timing marks* are included.

6.1.1 Timing Marks as Tokens

One of the polled sensors is used as the *timing mark* which would mean that all time-referenced sequences would be reported with reference to this sensor. One problem with this option is that the sensor used as the *timing mark* may not fire regularly and as such any rules that are reported may or may not have value. If however the sensor is firing in every cycle then its usefulness from a reporting standpoint is valuable in the same way as if it were a delimiter.

6.1.2 Timing Marks Added as Delimiters

In this option, *timing marks* are added as additional tokens to the sequence. This is necessary where all other tokens are sporadic as is the case with many types of sequence, or when no timing information is available. For the purpose of dealing with data streams that carry no time-stamp information this is the preferred format. Possibilities exist under this scenario to choose a suitable granularity for particular datasets and assign a token, as a *timing mark*, to deal with that choice. If necessary two or more tokens may be chosen to represent different granularities and therefore enable a wider variety of rules to be generated.

6.1.3 Timing Marks as Absolute Time

In some cases, each token carries with it an absolute time stamp. In this case there is more information than is required for use with the algorithms proposed in this thesis and it would be trivial to convert such a sequence to one that contained *timing marks* as delimiters. A better solution would be use an algorithm that required the additional information and then if required use the rule generation algorithm proposed in this thesis (see Section 5.3 for details).

6.1.4 The Value of Timing Marks

The value of *timing marks* becomes apparent when queries can be issued and results reported with respect to *timing marks*. A given sequence, for example, could be deemed as “fast/bursty” when it contains none or very few *timing marks* or as “slow” when it contains more than a prescribed amount. For instance, the difference between the two sequences ⟨.ABC.⟩ and ⟨A...B..C⟩ may be significant even if they occur within the normal lookahead. Moreover, the semantics of the temporal rule $A - during \rightarrow B$ may change depending on the number of timing marks that have been encountered. More significantly, the semantics of rules using temporal relationships such as $A - during \rightarrow B$ or $A - meets \rightarrow B$ may change depending on the number of timing marks that have been encountered. For instance, to allow for recording latency, two intervals may be deemed to *meet* if they occur within n timing marks.

6.2 Rule semantics

Typically rules from sequence discovery are of the type that can described in terms of Allen-style (Allen, 1983) relationships (or simpler). This is the case not only for market-basket mining (Agrawal and Srikant, 1995; Garofalakis et al., 1999; Han et al., 2000; Ayres et al., 2002), but also episodic mining (Mannila et al., 1997; Mooney and Roddick, 2004; Huang et al., 2004). In the case of episodic mining both parallel and serial episodes yield these types of rules. When using *timing marks* as delimiters the following similar possibilities, to those of episodic mining, must be considered:

1. If the sequences occur within the interval of the *timing mark*, for example, ⟨.ABCDEF.⟩, then this may be analogous to parallel episodes. Of course this would be data dependent and would rely on whether the order within the *timing marks* is relevant.
2. If the sequences must occur within a certain number of *timing marks*, for example, ⟨.AB.CDE.F.⟩, then this is analogous to serial episodes since the order of the tokens is relevant.

In the first case above, the discovered sequences could be treated as transactions (if order is irrelevant) and therefore further processing may be conducted using other data mining methods, such as association rule mining. In the second case details about the 'speed' of discovered sequences can be obtained with respect to the number of timing marks that are contained, allowing for a better understanding of the data.

If the *timing mark* is viewed as a period of fixed length, for example 5 minutes or 1 hour, with no absolute time-stamp associated with it then a search can be conducted for sequences that occur under both of the above conditions with the semantics of any discovered sequences being that:

1. For sequences that occur over one time cycle (0 *timing marks*) in the following sequence – ⟨.ABCDEFG.HIJ..⟩ – with a maximum look ahead of 5: ⟨ABCDE⟩, ⟨BCDEF⟩, and ⟨CDEFG⟩ are all valid while ⟨FGHIJ⟩ is not. This may be useful to determine if certain sensors did not fire during a particular cycle.

2. For the sequences that occurred over a period of x time cycles or within a prescribed number of time cycles, in the sequence – ⟨.ABCDEFG.HIJ..⟩ – the sequence ⟨FGHIJ⟩ occurs over two time cycles.

Given this information, the knowledge of the position of the *timing mark* allows for added semantics to be attached to the sequence – not only can it be said that ⟨FGHIJ⟩ occurs over two time cycles but also that first cycle is ended with ⟨FG⟩ and the second is begun with ⟨HIJ⟩. This may have added interest, depending on the application, to any resulting output that may be derived.

6.3 Algorithmic Considerations

The presence or lack of *timing marks* in a data stream enables users to take the opportunity to include or exclude the *timing marks* in their search for frequent episodes. Consequently, the *timing marks* feature has necessarily been implemented as a constraint, thus allowing the user to select the token that is the *timing mark* and, in addition, choose whether to report those episodes that contain exactly the prescribed number of *timing marks* or all episodes up to and including the prescribed number of *timing marks*.

Since the user is provided with the choice, it makes sense for the implementation of this constraint to be post episode discovery. To further reinforce this decision, the token used for the *timing mark* may be one of the data tokens (see Section 6.1.1), not one that is orthogonal to the data (see Section 6.1.2), in which case the user may not wish to remove the token from those episodes that are reported. In order to facilitate the fact that the *timing mark* may be one of the data tokens, the input file is scanned upon selection to generate a list of those tokens available. This incurs no overhead since the file has to be read before further processing can be undertaken and since this is an added constraint, the impact on the existing algorithm is minimal.

The implementation of this constraint relies on the following two parameters:

- The **lookahead** (or window) parameter used in previous work (Mooney and Roddick, 2004) (similar to Mannila et al.'s (1997) window concept, defines the maximum length episode to be mined), together with
- a *timing mark count* (**tmc**), discussed above.

Since both of these measures can be used, for the purpose of "frequent", a sequence up to *lookahead* must also occur within the prescribed number of marks – that is, the cut-off is either the *lookahead* or *timing mark count* whichever is the smaller.

6.3.1 Timing Mark Pruning

If the user has selected to include the timing marks in their search then the following will occur after the frequent episodes have been discovered. Firstly pruning will be conducted on the frequent sequences so that only those that contain the prescribed number of timing marks, see Algorithm C.5, will remain. If the timing mark is not determined to be one of the tokens in the data, then removal of the timing marks from those remaining sequences will ensue, and finally reassignment of them to the correct output containers, see Algorithm C.6.

For timing marks to remain unambiguous to the user and therefore be consistent throughout the implemented application then the following is relevant:

1. Within one mark means that there are no *timing marks* allowed in the sequence. Algorithmically this can be described by – assuming the timing mark is "." –

   ```
   if (tmc = 1 $\wedge$ cand.indexOf(".") $\neq -1$) then
       set output to null
   end if
   return
   ```

   This also leads to an added pruning technique – for example given the case of one timing mark, if the search is for an x length sequence and the last item in the sequence is a *timing mark*, then the next x sequences are not viable candidates so can be eliminated from the search.

2. Within/During one or more *timing mark*s. During one timing marks is as described above while the remainder is as follows:

Within/During		**# of sections**[a]	**# of timing marks**
1 mark	$\Longrightarrow$	one	none
2 marks	$\Longrightarrow$	two	one
3 marks	$\Longrightarrow$	three	two
⋮	⋮	⋮	⋮
n marks	$\Longrightarrow$	**n**	**(n-1)**

[a] # of sections indicates the number of distinct sections the sequence would be broken up into by the included timing marks.

6.4 Discussion

This chapter has discussed the inclusion of timing marks for dealing with data that have no absolute time attached to the events to be mined. Different methods of implementing the *timing marks* has been discussed and the best outcome has been determined to be data dependent. Furthermore, it can be seen (Figure B.9, page 197) that the implementation of the algorithm incurs negligible added overhead and that any benefits associated with the rules that may be reported are important in terms of being able to determine the pace of a sequence.

Chapter 7

Transitive Relationships

This chapter investigates the use of transitivity tables for further reasoning between two or more sets of events. This technique has been used by both Allen (1983) and Freksa (1992) to accommodate both fine and coarse reasoning. This investigation however, will focus on the expressive power of the augmented transitivity table for ELMI relationships and the transitivity table for VLMI relationships, introduced in Chapter 4, and which appear in Appendix A. This exposé of further reasoning using the midpoint transitivity tables will be conducted using examples and therefore it will be necessary to assess the outcomes with a comparable alternative set of outcomes. When this has arisen the Allen transitive relations have been chosen. Any visualisation of the outcomes will be illustrated using the extended midpoint icon and the Freksa icon.

7.1 The Structure of Transitive Relationships

The use of a transitivity table enables a set of relationships that exist between two sets of events to be determined. For example, given the relationship between two events, A and B, $A \xrightarrow{rels} B$, and between B and C, $B \xrightarrow{rels} C$, the transitivity table provides the subset within which any relationship between A and C, $A \xrightarrow{rels} C$, must fall. This process is usually written in its longest form as $A \xrightarrow{rels} B \wedge B \xrightarrow{rels} C \Rightarrow A \xrightarrow{rels} C$, but from this point, without loss of information, it will be written as $A \xrightarrow{rels} B \xrightarrow{rels} C \Rightarrow A \xrightarrow{rels} C$.

The reasoning process associated with transitivity can be extended to accommodate not only relationships of the type above, but also additive structures, such as $A \xrightarrow{rels} B \xrightarrow{rels} C \xrightarrow{rels} D$ and thus $A \xrightarrow{rels} D$, or union types structures of the form $A \xrightarrow{rels} B \xrightarrow{rels} C$ and $A \xrightarrow{rels} D \xrightarrow{rels} C$ and thus $A \xrightarrow{rels} C$ using both paths for reasoning[1]. The latter of these two has the potential to incorporate fuzzy interval logic (see Section 3.4.2), since one path may have more influence than the other.

[1]In this case both A and C are different instances of the same events.

7.1.1 Terminology

Definition 7.1. *The set of relationships, RELS, that result from the application of a transitive process are termed the **outcome(s)** of that transitive process.* ■

Definition 7.2. *A transitive relationship that takes the form*

$$a_1 \xrightarrow{rels} a_2 \xrightarrow{rels} a_3 \xrightarrow{rels} \dots \xrightarrow{rels} a_{n-1} \xrightarrow{rels} a_n \Rightarrow a_1 \xrightarrow{RELS} a_n$$

*is termed an **additive** transitive relationship.* ■

For example the *outcome* for the *additive* transitive relationship described by
$a_1 \xrightarrow{oi} a_2 \xrightarrow{f} a_3 \xrightarrow{mi} a_4 \Rightarrow a_1 \xrightarrow{>} a_4$
is $>$ and therefore it can be said that a_1 is *after* a_4, this is illustrated in Figure 7.1(a).

Definition 7.3. *A transitive relationship that takes the form*

$$a_1 \xrightarrow{rels} a_2 \xrightarrow{rels} \dots \xrightarrow{rels} a_{n-1} \xrightarrow{rels} a_n \Rightarrow a_1 \xrightarrow{RELS} a_n$$

$$a_1 \xrightarrow{rels} b_1 \xrightarrow{rels} \dots \xrightarrow{rels} b_{m-1} \xrightarrow{rels} b_m \xrightarrow{rels} a_n \Rightarrow a_1 \xrightarrow{RELS} a_n$$

where at least one of $a_{2..n-1} \neq b_{1..m}$ *is termed a **union** transitive relationship.* ■

For example the *outcomes* for the *union* transitive relationship described by

$$a_1 \xrightarrow{o} a_2 \xrightarrow{m} A \Rightarrow a_1 \xrightarrow{<} A$$

$$a_1 \xrightarrow{f} b_1 \xrightarrow{o} A \Rightarrow a_1 \xrightarrow{o,s,d} A$$

(where a_1 and A are different instances of the same event) is the union of both sets of *outcomes* – $\{<\} \cup \{o, s, d\} \rightarrow \{<, o, s, d\}$ and therefore it can be said that a_1 is either *before, overlaps, starts* or *during* A, this is illustrated in Figure 7.1(b).

Outcomes for both of these structures are able to be determined using the software developed in support of this thesis, see Appendix B, Section B.2 for details.

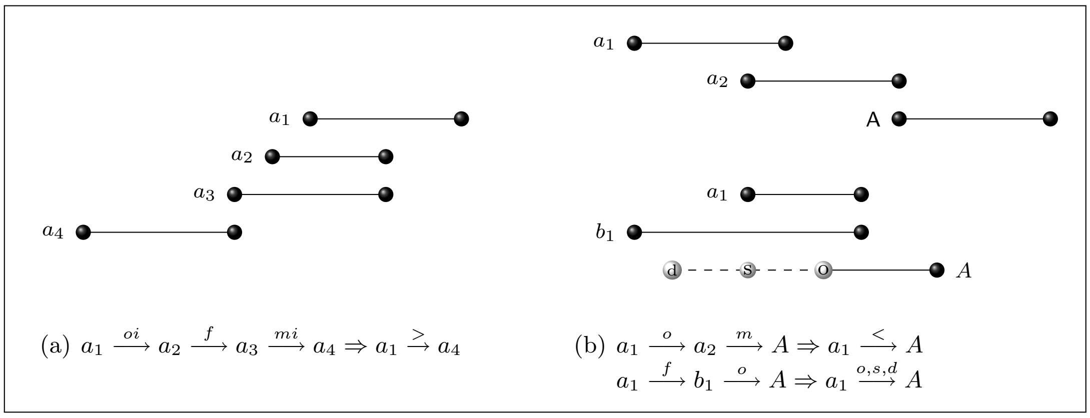

a_1
a_2
a_3
a_4
(a) $a_1 \xrightarrow{oi} a_2 \xrightarrow{f} a_3 \xrightarrow{mi} a_4 \Rightarrow a_1 \xrightarrow{>} a_4$
a_1
a_2
A
a_1
b_1
d
s
o
A
(b) $a_1 \xrightarrow{o} a_2 \xrightarrow{m} A \Rightarrow a_1 \xrightarrow{<} A$
$a_1 \xrightarrow{f} b_1 \xrightarrow{o} A \Rightarrow a_1 \xrightarrow{o,s,d} A$

Figure 7.1: Depiction of an additive (left) and union (right) transitive relationship.

7.2 Transitivity and Known Lengths

When the endpoints of a episode are known, a definite advantage can be gained (in terms of the reasoning available), since this implies that the midpoint is also known. Eliminating the knowledge of one or both of the endpoints reduces the reasoning to that described earlier as coarse reasoning and although significant information can be gained, the more information that is available the more definitive the answers. One example where this may make a significant difference is in the medical arena where more, rather than less, information could be crucial in saving or losing a life. The outcomes facilitated by the extra knowledge gained by knowing the midpoint is best explained using examples that are elaborated in the following section.

7.2.1 Transitivity for Variable-Length Intervals

Variable-length intervals (episodes) are generated as a result of sequence mining primarily when a windowing method is used as part of the mining. The relationships that are able to be expressed between these episodes are not limited in the same way as equal-length intervals, and therefore if the midpoint is known or can be inferred then the full set of interval-interval midpoint relationships can be used. The transitivity table (see Section A.3) for performing any reasoning between such intervals is in this instance a 49×49 table, since there are 49 midpoint interval relationships, as opposed to the 11×11 for the equal-length intervals.

Proposition 7.1. *The Variable-Length Midpoint Interval (VLMI) algebra is more powerful than the Allen algebra for determining the outcomes from transitive relationships.*

Proof: Since the Variable-Length Midpoint Interval (VLMI) algebra is a subset of the Allen algebra (see Section 4.6.1 for proofs) and it is a closed set, it follows that any transitive outcome will, in the worst case able to be transformed fully into the Allen equivalent, but in the majority of cases the outcomes will be a smaller set of the Allen equivalent. This last statement can be verified by constructing a 49×49 table and checking each entry, this can be done in $O(n^2)$ time. ■

The following will showcase two examples where the VLMI algebra produces superior outcomes to the Allen algebra and will serve to support Proposition 7.1. As a reference for these two examples, extracts from the VLMI and the Allen transitivity tables are included, see Table 7.1 and Table 7.2. The complete tables can be found in Appendix A.

The first example highlights the case where merely defining two meeting intervals does not have the semantic power of defining one interval with its midpoint even if this option is available.

Table 7.1: Extract from the VLMI transitivity table showing a selection of *Large* and *Medium* relationships and their inverses.

	...	**mso**	**mmo**	**mlo**	**lso**	**lmo**	**llo**	...
⋮				⋮				⋮
mloi	...	sl, sm, ss	sl	sli, =, sl	mdf, lom, ldf	mdf	mdl, mom, mdf, fl, lloi	⋯
mmoi	...	sl, sm, ss	=	sli	mdf, lom, ldf	mom	mdl, fl, lloi	
msoi	...	ssi, smi, sli, =, sl, sm, ss	ssi, smi, sli	ssi, smi, sli	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	mdl, fl, lloi, lmoi, lsoi	mdl, fl, lloi, lmoi, lsoi	...
⋮				⋮				⋮

Table 7.2: Extract from the Allen transitivity table showing the *meets, overlap, finishes, is-overlapped-by* and *is-met-by* relations.

	...	**m**	**o**	...	**f**	**oi**	**mi**	...
⋮				⋮				⋮
m	...	<	<	...	o, s, d	o, s, d	fi, =, f	...
o	...	<	<, m, o	...	o, s, d	o, fi, di, si, =, s, d, f, oi	di, si, oi	...
⋮				⋮				⋮
f	...	m	o, s, d	...	f	oi, mi, >	>	...
oi	...	o, fi, di	o, fi, di, si, =, s, d, f, oi	...	oi	oi, mi, >	>	...
mi	...	si, =, s	d, f, oi	...	mi	>	>	...
⋮				⋮				⋮

Example 7.1. Consider the relationships shown in Figure 7.2 in which the relationships a_1, a_2 meet and both have a relationship with B (a_1 finishes B and a_2 is met-by B), which in turn has a relationship with C (overlaps).

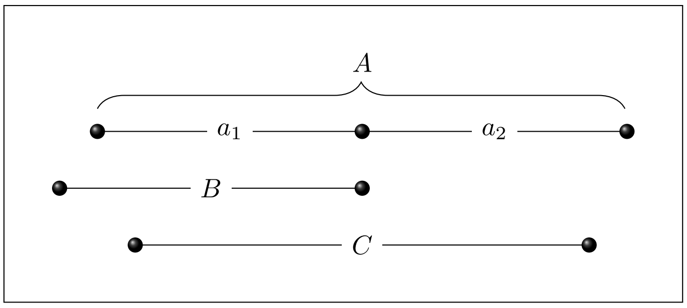


Figure 7.2: Expressive power example for two meeting intervals.

If it were necessary to find the relationship between A (split into a_1 and a_2) and C via B, then the following relationships are evident:

$$
\begin{aligned}
a_1 &\xrightarrow{m} a_2\\
a_1 &\xrightarrow{f} B\\
a_2 &\xrightarrow{mi} B\\
B &\xrightarrow{o} C
\end{aligned}
$$

Computing the relationship between a_1 and C, and therefore in part, A and C gives

$$a_1 \xrightarrow{f} B \xrightarrow{o} C$$

and hence (see Table 7.2)

$$a_1 \xrightarrow{o,\ s,\ d} C$$

Similarly, computing the relationship between a_2 and C and therefore, also in part, A and C gives

$$a_2 \xrightarrow{mi} B \xrightarrow{o} C$$

and hence (see Table 7.2)

$$a_2 \xrightarrow{d,\ f,\ oi} C$$

and therefore by performing a union of both sets of *outcomes*

$$A \xrightarrow{o,\ s,\ d,\ f,\ oi} C \tag{7.1}$$

However, given the Variable-Length Midpoint algebra it could be asserted

$$
\begin{aligned}
&A \xrightarrow{mloi} B\\
&\textit{and}\\
&B \xrightarrow{mmo,\ mlo,\ lmo,\ llo} C
\end{aligned}
$$

giving (see Table 7.1)

$$A \xrightarrow{sli,\ =,\ sl,\ mdl,\ mom,\ mdf,\ fl,\ lloi} C \tag{7.2}$$

Using the Allen transitivity table, with the same example and no midpoint on A would yield

$$A \xrightarrow{oi} B \xrightarrow{o} C$$

and therefore (see Table 7.2)

$$A \xrightarrow{o,\ fi,\ di,\ si,\ =,\ s,\ d,\ f,\ oi} C \tag{7.3}$$

If the three sets of *outcomes* (7.1), (7.2) and (7.3) are compared and the individual possibilities from the Midpoint outcome grouped to align with the corresponding Allen relationship then the following is apparent.

Table 7.3: Comparison of *outcomes* for Allen and midpoint transitive relationships for Example 7.1.

Outcome 7.1 – Allen Split:	o					s	d	f	oi
Outcome 7.2 – Midpoint:				sli	$=$	sl	$mdl,\ mom,\ mdf$	fl	$lloi$
Outcome 7.3 – Allen No Split:	o	fi	di	si	$=$	s	d	f	oi

Table 7.3 shows that the Midpoint *outcome* only returns 3 out of a possible 7 *during* constraints, 1 out of a possible 3 *finishes*, *starts* and *is-started-by* constraints, 1 out of a possible 9 *overlap* constraints and *equals*. This represents a far finer-grained set of constraints than both the spilt version and the non-split version of A both of which introduce constraints (o, fi, di) for the non-split version and (o) for the split version. Furthermore, the split version fails to report the possibility of an *is-started-by* or an *equals* constraint. The non-split *outcomes* are indicative of current real world reporting practices. □

Example 7.2. Consider also the following example, as illustrated in Figure 7.3:

$$A \xrightarrow{mmoi} B \xrightarrow{mmo} C$$

which results in the following *outcome*, see Table 7.1

$$A \xrightarrow{=} C \tag{7.4}$$

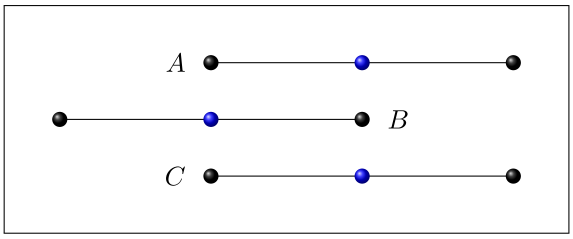


Figure 7.3: Expressive power example for two overlapping intervals.

Without introducing additional constraint handling to the algebra (as is the case with Meiri (1991) and Schwalb and Vila (1998)), splitting the intervals into two sub-intervals will not allow (within the Allen algebra) A to be seen as equal to C, which is the case when the midpoint is used because there is no way that the second half of A can be seen to be the same length as the second half of C. In fact the outcomes when A is split into a_1 and a_2 are considerably more numerous than when using the midpoint, and if B is also split into b_1 and b_2 then they are more numerous again. Both of these scenarios are shown below.

Scenario 1: A only is split, see Table 7.2.

$$
\begin{array}{lllll}
a_1 \xrightarrow{fi} B \xrightarrow{o} C & \Rightarrow & a_1 & \xrightarrow{o} & C \\
a_2 \xrightarrow{mi} B \xrightarrow{o} C & \Rightarrow & a_2 & \xrightarrow{d,\ f,\ oi} & C
\end{array}
$$

and when the union of these two sets of *outcomes* is performed then,

$$A \xrightarrow{o,\ d,\ f,\ oi} C \tag{7.5}$$

Scenario 2: Both A and B are split.

Firstly the relationships of both a_1 and a_2 with b_1 and b_2

$$
\begin{array}{lll}
a_1 \xrightarrow{mi} b_1 & \& & a_1 \xrightarrow{=} b_2 \\
a_2 \xrightarrow{>} b_1 & \& & a_2 \xrightarrow{mi} b_2
\end{array}
$$

secondly the relationships of b_1 and b_2 with C

$$
\begin{array}{l}
b_1 \xrightarrow{m} C \\
b_2 \xrightarrow{s} C
\end{array}
$$

which results in

$$A \xrightarrow{mi,\ =,\ >} B \xrightarrow{m,\ s} C$$

and therefore the set of *outcomes*, see Table 7.2

$$A \xrightarrow{m,\ si,\ =,\ s,\ d,\ f,\ oi,\ mi,\ >} C \tag{7.6}$$

Table 7.4: Comparison of *outcomes* for Allen and Midpoint transitive relationships for Example 7.2.

Outcome 7.4 – Midpoint:				$=$						
Outcome 7.5 – Scenario 1:		o				d	f	oi		
Outcome 7.6 – Scenario 2:	m		si	$=$	s	d	f	oi	mi	$>$

Table 7.4 shows that in this case the results are more conclusive with respect to the benefits of using the Midpoint model, since **Scenario 1** introduces constraints and fails to report the actual constraint. **Scenario 2** does however report the actual constraint but introduces even more constraints than **Scenario 1**. □

7.2.2 Transitivity for Equal-Length Intervals

Equal-length intervals can occur in many situations, for example when data has an associated time-stamp and the interest lies in episodes that occur within a fixed time period. This may still be accomplished even when data has no associated time-stamp since one can be imposed[2] and similar results obtained. The limitations on extended reasoning, with transitivity tables, using such intervals lie in the fact that *during*, *starts*, and *finishes* relationships can not occur. However, since the midpoint is known, either explicitly or implied, the possibility of gaining more refined information regarding any intervals is available using the augmented equal-length midpoint transitivity table.

Proposition 7.2. *The Equal-Length Midpoint Interval (ELMI) algebra is more powerful than the Allen algebra for determining the outcomes from transitive relationships.*

Proof: This proof follows directly from the proof for Proposition 7.1 since the equal-length interval model is a more restrictive case of the variable-length model. ■

The following example serves to support Proposition 7.2. The example is a more in depth treatment of Example 7.3 presented in Chapter 4, page 113.

Example 7.3. Given three episodes A, B, C of equal length and relationships between them that are some type of *overlap* relationship then the following is evident:

$$A \xrightarrow{o} B \xrightarrow{o} C \Rightarrow A \xrightarrow{RELS} C$$

If this was now reasoned to determine $A \xrightarrow{RELS} C$, using Allen's algebra the *outcome* is as follows and illustrated in Figure 7.4:

$$A \xrightarrow{o} B \xrightarrow{o} C \Rightarrow A \xrightarrow{<,m,o} C$$

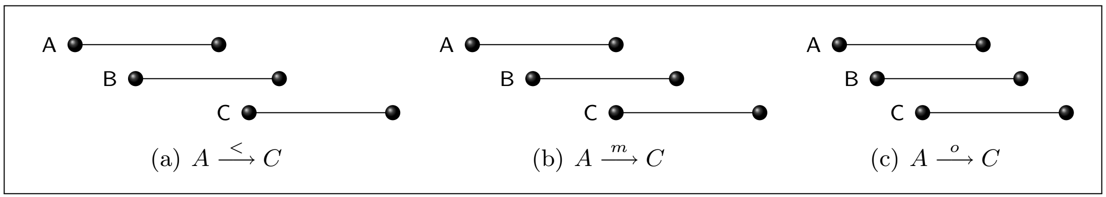

A
B
C
(a) $A \xrightarrow{<} C$
(b) $A \xrightarrow{m} C$
(c) $A \xrightarrow{o} C$

Figure 7.4: Allen *outcomes* for $A \rightarrow C$ when $A \xrightarrow{o} B \xrightarrow{o} C$.

However, since the midpoint of the intervals can be inferred by knowing the end-points the ELMI transitivity table can instead be used to determine the outcomes. In the most general case, which equates to not knowing any further information regarding the types of *overlap*, then there are nine possible combinations of *overlap* to consider[3].

[2]A method of doing this is presented in Chapter 6.

[3]For convenience *SmallOverlap*, *MediumOverlap* and *LargeOverlap* will be written as *so*, *mo*, and *lo* respectively.

These are: *(so, so)*, *(so, mo)*, *(so, lo)*, *(mo, so)*, *(mo, mo)*, *(mo, lo)*, *(lo, so)*, *(lo, mo)*, and *(lo, lo)*. However, some of these combinations; *(so, so)*, *(so, mo)* and *(mo, so)*; *(so, lo)* and *(lo, so)*; *(mo, lo)* and *(lo, mo)* result in the same *outcomes* and therefore there are five unique *outcomes* possible. These unique *outcomes* can be grouped into three broader categories that correspond to a grading of the information gained from the least to the most. These broad categories are outlined below and are illustrated in Figure 7.5, Figure 7.6 and Figure 7.7:

1. $(A \xrightarrow{so} B \xrightarrow{lo} C)$ OR $(A \xrightarrow{lo} B \xrightarrow{so} C) \Rightarrow A \xrightarrow{<,m,so} C$
   This corresponds to the least informative set of *outcomes*, but is still better than the Allen *outcomes* because the *overlap* that would be reported using the Allen transitivity table has been refined to a *SmallOverlap*.

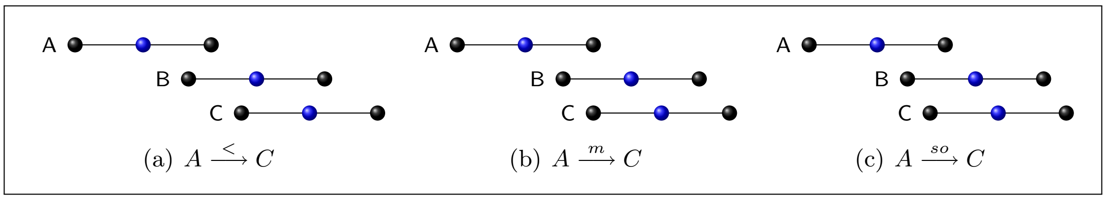

A
B
C
(a) $A \xrightarrow{<} C$
(b) $A \xrightarrow{m} C$
(c) $A \xrightarrow{so} C$

Figure 7.5: Equal-Length *outcomes* for $A \rightarrow C$ when $A \xrightarrow{so} B \xrightarrow{lo} C$ (and therefore $A \xrightarrow{<,m,so} C$).

2. $A \xrightarrow{lo} B \xrightarrow{lo} C \Rightarrow A \xrightarrow{so,mo,lo} C$
   The information gained here is more than that of the Allen *outcomes*, albeit there remain three possibilities, however all are *overlap* relationships – *before* and *meets* have been excluded. In the worst case scenario under this configuration of relationships the *outcome* would be reported as *overlap*, with further refinement one of the three equal-length midpoint relationships, *SmallOverlap*, *MediumOverlap*, or *LargeOverlap* could be reported.

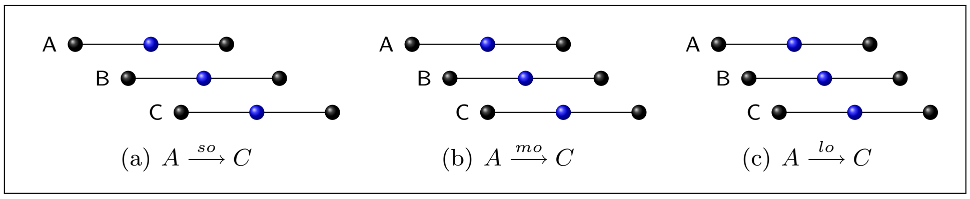
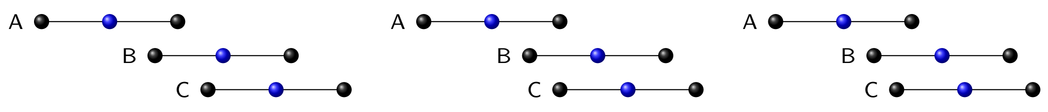

A
B
C
(a) $A \xrightarrow{so} C$
(b) $A \xrightarrow{mo} C$
(c) $A \xrightarrow{lo} C$

Figure 7.6: Equal-Length *outcomes* for $A \rightarrow C$ when $A \xrightarrow{lo} B \xrightarrow{lo} C$ (and therefore $A \xrightarrow{o} C$).

3. The following three sets of *outcomes* correspond to the best possible *outcomes* since they yield only one *outcome* in each set.

   (a) $(A \xrightarrow{so} B \xrightarrow{so,mo} C)$ OR $(A \xrightarrow{mo} B \xrightarrow{so} C) \Rightarrow A \xrightarrow{<} C$
   (b) $A \xrightarrow{mo} B \xrightarrow{mo} C \Rightarrow A \xrightarrow{m} C$

(c) $(A \xrightarrow{mo} B \xrightarrow{lo} C)$ OR $(A \xrightarrow{lo} B \xrightarrow{mo} C) \Rightarrow A \xrightarrow{so} C$



(a) $A \xrightarrow{so} B \xrightarrow{so,mo} C \Rightarrow A \xrightarrow{<} C$ (b) $A \xrightarrow{mo} B \xrightarrow{mo} C \Rightarrow A \xrightarrow{m} C$ (c) $A \xrightarrow{mo} B \xrightarrow{lo} C \Rightarrow A \xrightarrow{so} C$

Figure 7.7: Best possible Equal-Length *outcomes* for $A \rightarrow C$.

□

This example, Example 7.3, has served to show the possibilities for increase in information relating to any transitive relationships that occur between equal-length intervals when the midpoint is used, and has demonstrated that even using the worst set of *outcomes* this increase exists.

7.3 Rule Inference using Transitive Relationships

7.3.1 Rule Inference: An Overview

Rule inference on the results of data mining has been conducted in all major areas but not all are concerned with temporal inference. Those that have include; Association Mining resulting in temporal association rules (Chen, Petrounias and Heathfield, 1998; Chen and Petrounias, 2000; Rainsford and Roddick, 1999; Ale and Rossi, 2000; Höppner, 2001*b*, 2002; Winarko and Roddick, 2006), Time Series Mining (Povinelli, 2000; Höppner, 2001*a*; Mörchen, 2006) and Sequence Mining (Sun, Orlowska and Zhou, 2003; Sun, Orlowska and Li, 2005; Mooney and Roddick, 2004, 2006). The value of representing rules in this way is that further reasoning can be conducted by means of transitive relationships and the previous sections have shown the value of such reasoning in determining the outcomes for relationships that exist between two intervals, particularly when the midpoint is known. However, this extended reasoning is not in general performed and as such, knowledge associated with it (the extended reasoning) is not gained.

7.3.2 Limiting the Number of Itemsets, Sequences and Rules

As is the case in most forms of data mining, the number of rules produced from a mining run is large and therefore methods to limit this number, while still reporting rules of interest, is paramount. These methods include interestingness measures (Sahar, 1999; Bayardo and Agrawal, 1999; Spiliopoulou, 1999; Hilderman and Hamilton, 2001; Tan, Kumar and Srivastava, 2002; Shillabeer and Roddick, 2005) and interactive mining methods (Parthasarathy et al., 1999; Ceglar, Roddick and Calder, 2003; Lin and Lee, 2004). The former are usually implemented as metrics that are in effect from the start of the mining run until its completion and may influence the production of itemsets or sequences, or the rules produced from these. Any change in these metrics require the full mining run to be repeated using the new values. The latter allow for the user of the algorithm to 'interact' with it and make judgements as to how it should proceed, thus focussing the output towards the users' needs.

7.3.3 Outcome Discovery

In the recent past it was argued by Padmanabhan and Tuzhilin (1996) that the rules from sequence mining lacked expressive power and they introduced rule inference constructs based on First Order Temporal Logic (FOTL) that included *Since, Until, Next, Always, Sometimes, Before, After and While*, and from this, and the work of Kam and Fu (2000), Höppner (2001*b*), Sun et al. (2005) and Roddick and Mooney (2005), the number of constructs has been extended to include those in the algebra of Allen (1983). For the progression to continue it is the position of this thesis that the process of discovering *outcomes* from any transitive relationships within the results of a mining run is an area of research that has the potential to deliver knowledge that at present is under-utilised. One such application area that may benefit is that of medical data mining where the *outcomes*, from the mining of the presentation of symptoms and the progression of the illness in conjunction with the initial battery tests conducted, could reduce the number of tests and thus potentially save money (Imberman, Domanski and Thompson, 2002).

7.3.4 Presentation of Outcomes

The presentation of just the temporal relationships that exist between episodes, either as text or pictorially, does not harness the full power of the temporal logic used to generate the relationships. The preceding sections have shown that regardless of the algebra used, more information can be gained by discovering *outcomes*. For this reason the $INTEM_{TM}$ software supports the discovery and graphical display of sets of *outcomes* from a static set of inputs, an example of which is showcased in Section 7.4, however, an automated method for detecting these *outcomes* would be of benefit, but is left for further research. The implications for the implementation of this type of inference into the software are outlined in the following section.

7.3.5 Implications Arising from Interacting Episodes

Since there are three classes of interactions that may be discovered – strong, weak and dependent – the discovery of *outcomes* must reflect this breakdown in the first instance since the semantics of the *outcomes* can be readily inferred when each relationship is of the same class. However since the definition for both *weak* and *dependent* interactions requires that the first episode be frequent and the second either a viable candidate or existing due to the discovery process neither an *additive* or *union* transitive relationship is possible.

Proposition 7.3. *The definitions of weak and dependent interactions precludes them from participating in same interaction class transitive relationships.*

Proof: In its simplest form an *additive* transitive relationship takes the following form: $A \xrightarrow{rels} B \xrightarrow{rels} C \Rightarrow A \xrightarrow{RELS} C$. The definition of a *weak* interaction (see Definition 5.17) is denoted as follows: $\psi^{w[p]}(e_i, e_j) \mid w \in \mathcal{R}$ where e_i is frequent and e_j is only a valid candidate. The form of an *additive* transitive relationship requires that the first episode of each part, in the above case A and B, be frequent which cannot be the case since B would have to be both frequent and only a valid candidate simultaneously. The proof for *dependent* interactions follows directly from this. ■

This result then requires that if a transitive relationship includes a *weak* interaction then it must be the last interaction which means the valid combinations for any type of transitive relationship (*additive* or *union*) are reduced to the following:

1. All *strong* interactions, for example:
   $A \xrightarrow{o} B \xrightarrow{d} C \xrightarrow{f} D \Rightarrow A \xrightarrow{o,s,d} D$
2. A series of *strong* interactions followed by one *weak* interaction, for example:
   $A \xrightarrow{o} B \xrightarrow{d} C \xrightarrow{s[0.85]} D \Rightarrow A \xrightarrow{(o,s,d)[0.85]} D$
3. A series of *strong* interactions followed by one *dependent* interaction. This scenario will not arise at present since *dependent* interactions are currently not reported but any future reporting of these types will require careful consideration of the semantics of the resultant *outcomes*.

If an automated system for the discovery of *outcomes* is to be implemented then these valid combinations have to be taken into account as would the semantics of any *outcomes* produced.

7.4 Visualising the Outcomes from Transitive Relationships

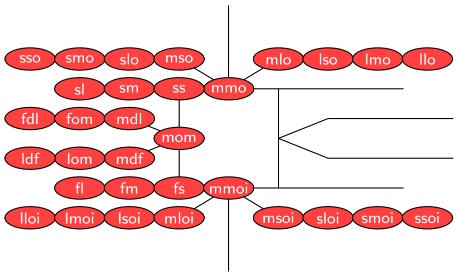
The *outcomes* from transitive relationships can be numerous and representing them solely in textual format can make their interpretation difficult. The introduction of an iconic representation of the Allen temporal intervals that preserved conceptual neighbours by Freksa (1992) enabled the depiction of fine-grained reasoning using coarse reasoning in an intuitive way, see Figure 3.2 (page 55). The extension of the Freksa icon as a result of this thesis, see Figure 4.14 (page 77), enables not only fine-grained (Allen) and coarse reasoning (Freksa) but also a finer-grained (Midpoint) reasoning. The process of allowing any combination of these types of reasoning for the purpose of generating *outcomes* has been incorporated into the $INTEM_{TM}$ application. The following example presents the *outcomes* for Example 7.1 (page 109), using the extended icon followed by the Freksa icon with the corresponding Freksa relationship if one exists. For the second case (VLMI algebra) the Allen/Freksa *outcomes* are depicted on the extended icon in green to highlight the finer granularity when using the midpoint model. This has not been done for the other two sets of *outcomes* since there is no gain in using the finer-grained model for coarser inputs.

Recalling the three possible scenarios in this example:

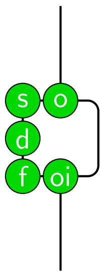
1. A was split into a_1 and a_2 which resulted in the following:
$$\begin{array}{ll} a_1 \xrightarrow{f} B \xrightarrow{o} C \;\Rightarrow & \\ & A \xrightarrow{o,s,d,f,oi} C \\ a_2 \xrightarrow{mi} B \xrightarrow{o} C \;\Rightarrow & \end{array}$$
and depicted using the extended icon in Figure 7.8(a) and the Freksa icon in Figure 7.8(b)

2. The Variable-Length Midpoint algebra was used resulting in the following:
$$A \xrightarrow{mloi} B \xrightarrow{mmo,mlo,lmo,llo} C \Rightarrow A \xrightarrow{sli,=,sl,mdl,mom,mdf,fl,lloi} C$$
and depicted using the extended icon in Figure 7.8(c) and the extended icon in Figure 7.8(d) to highlight the finer granularity

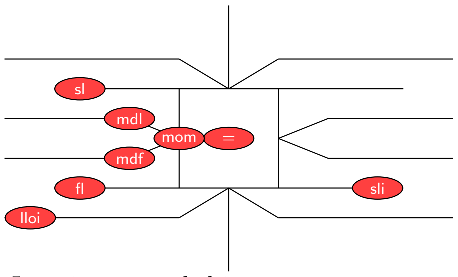
3. A was not split and no midpoints were used resulting in the following:
$$A \xrightarrow{oi} B \xrightarrow{o} C \Rightarrow A \xrightarrow{o,fi,di,si,=,s,d,f,oi} C$$
and depicted using the extended icon in Figure 7.8(e) and the Freksa icon in Figure 7.8(f)



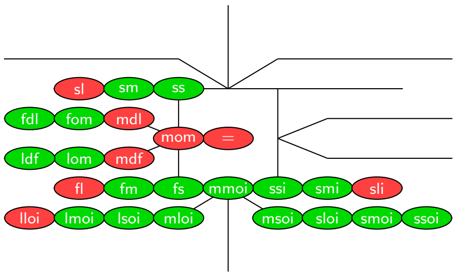
(a) Item i.- extended icon: $A \xrightarrow{o,s,d,f,oi} C$



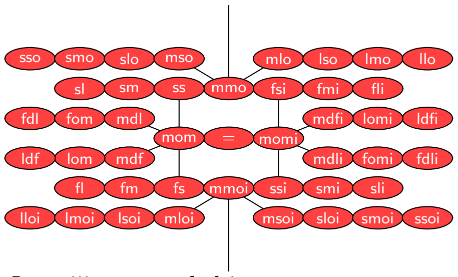
(b) Item i - Freksa icon: $A \xrightarrow{o,s,d,f,oi} C$



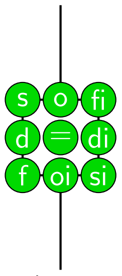
(c) Item ii. - extended icon: $A \xrightarrow{sli,=,sl,mdl,mom,mdf,fl,lloi} C$



(d) Item ii. - extended icon: $A \xrightarrow{si,=,s,d,f,oi} C$



(e) Item iii. - extended icon: $A \xrightarrow{oi} B \xrightarrow{o} C \Rightarrow A \xrightarrow{o,fi,di,si,=,s,d,f,oi} C$



(f) Item iii. - Freksa icon: $A \xrightarrow{oi} B \xrightarrow{o} C \Rightarrow A \xrightarrow{o,fi,di,si,=,s,d,f,oi} C$ Freksa's *contemporary of*

Figure 7.8: Extended and Freksa icon representations of the *outcomes* for Example 7.1.

7.5 Discussion

This chapter introduced the area of transitive relationships within the framework of enhancing rule semantics for the results of interacting episode mining, but also with a view to their use in other areas of data mining. This idea is a novel approach to enhancement of rule semantics and coupled with the MI algebra introduced in Chapter 4 it has been shown that the *outcomes* are more fine-grained and in all circumstances more powerful than when using, for example, Allen's algebra. These ideas and theoretical framework have not yet been integrated into any commercial systems but it has been shown that there is a potential to increase knowledge discovery and as such remains a significant area of research.

The implications arising from applying this process to interacting episodes also raises important issues. This is evident not only with respect to the meaning of any *outcomes* produced but also for the potential the integration of fuzzy interval algebras into the process may afford, as would be the case with *weak* and *dependent* relationships. Acting in concert with any knowledge discovery process is how the results are displayed and although this chapter has offered one alternative, refer to Appendix B (page 187), this area also remains one of significant future importance.

Chapter 8

Conclusions and Future Research

The areas of interest that have been explored in this thesis are represented in the Introduction as shown in Figure 1.1. This thesis explores a significant number of these areas, but further issues still remain both for the fields on the periphery and those at the heart of this thesis. Certain boundaries have also been highlighted throughout the thesis and these are considered now as possible future research tasks.

8.1 Mining Heuristics and Datasets

This thesis pointed out in Chapter 5 that an important consideration is the reporting of episodes and interactions that are based on frequency metrics, imposed as user-defined constraints. While this has been shown to produce valuable results it may be necessary to alter this approach to cater for more diverse data in different domains. This consideration as well as the introduction of methods to enable the mining of streaming data will also be of benefit and would enhance the software that has already been developed for this thesis. The further implementation of *timing marks*, as discussed in Chapter 6, would also be of benefit to the enhancements that have already been mentioned.

8.2 Transitive Relationships

This thesis introduced the area of transitive relationships within the framework of enhancing rule semantics for the results of interacting episode mining, but also with a view to their use in other areas of data mining. This idea is a novel approach to the enhancement of rule semantics and coupled with the MI algebra introduced in Chapter 4 enables complex rules to be discovered. The implications arising from applying this process to interacting episodes also raises important issues. This is evident not only with respect to the meaning of any *outcomes* produced but also for the potential the integration of fuzzy interval algebras into the process may afford, as would be the case with *weak* and *dependent* relationships. As has been stated these ideas and theoretical framework have not yet been integrated into any commercial systems but it has been shown that there is a potential to increase knowledge discovery and as such this remains a potential area of future research.

8.3 Development of Visualisation Tools

Acting in concert with any knowledge discovery process are methods for displaying the results and although this thesis has offered one alternative this area also remains one of significant future importance. This is very apparent since the lack of tools in this area has been highlighted in this thesis.

8.4 Application Areas Applicable to this Approach

The Introduction mentioned that one area that was explored as a potential avenue for this research was text mining, or more appropriately analysing text using sequence mining methods. Although a multi-level framework has been implemented (Mooney, de Vries and Roddick, 2004) there remains interesting areas for future research especially as an incremental and interactive mining problem and also in visualisation. One technique that could be useful to enhance the edit distance methods currently in use is the multiple alignment patterns of Kum et al. (2002), but also the use of projected window lists (Huang et al., 2004) to allow for a move away from only mining contiguous sequences would be of benefit.

8.5 Conclusion

This thesis has presented a useful method for the discovery of interacting episodes from temporal sequences and the analysis of them using temporal patterns. It has made significant contributions to the field of sequence mining, rule based temporal sequence analysis and the visualisation of these results. The use of the software implemented as part of this thesis to date has shown significant promise and this suggests that its application on real world datasets, for example medical records, will be similarly beneficial.

Appendix A

Transitivity Tables

The purpose of this appendix is act as a repository for the transitivity tables of Allen (1983), ELMI (Section 4.4) and VLMI (Section 4.5).

The Allen transitivity table, Table A.1, containing 13 entries is presented first. Second is the ELMI table, Table A.2, which contains eleven entries – 5 unchanged from Allen (*before* (<), *meets* (**m**), *equals* (=), *is-met-by* (**mi**), *after* (>)) – and the six equal-length overlap relationships - (*SmallOverlap* (**so**), *MediumOverlap* (**mo**), *LargeOverlap* (**lo**), *is-LargeOverlapped-by* (**loi**), *is-MediumOverlapped-by* (**moi**), *is-SmallOverlapped-by* (**soi**))

The transitivity table for the 49 VLMI relationships is very large and as such it has been necessary to split it up into manageable smaller pieces. The points at which it these splits occur are as follows:

Table A.3	–	*Before* (<) to *LargeLargeOverlap* (**llo**), (page 127)
Table A.4	–	*is-FinishedSmall-by* (**fsi**) to *LastContainsFirst-of* (**fdli**), (page 141)
Table A.5	–	*is-StartedSmall-by* (**ssi**) to *StartsSmall* (**ss**), (page 152)
Table A.6	–	*FirstDuringLast* (**fdl**) to *FinishesSmall* (**fs**), (page 158)
Table A.7	–	*is-LargeLargeOverlapped-by* (**lloi**) to *After* (>), (page 169)

The symmetry of the transitivity tables, when ordered to preserve conceptual neighbourhoods, was exploited by Freksa (1992) to facilitate coarse reasoning with incomplete knowledge. In order to verify the correctness of the complete 49 × 49 VLMI transitivity table, (Tables A.3 to A.7), this symmetric property has also been used. The verification table is shown in Figure A.1.

A.1 Allen's Transitivity Table

Table A.1: Allen's transitivity table.

	<	**m**	**o**	**fi**	**di**	**si**	**=**	**s**	**d**	**f**	**oi**	**mi**	>
<	<	<	<	<	<	<	<	<	<, m, o, s, d	<, m, o, s, d	<, m, o, s, d	<, m, o, s, d	No Info
m	<	<	<	<	<	m	m	m	o, s, d	o, s, d	o, s, d	fi, =, f	di, si, oi, mi, >
o	<	<	<, m, o	<, m, o	<, m, o, fi, di	o, fi, di	o	o	o, s, d	o, s, d	o, fi, di, si, =, s, d, f, oi	di, si, oi	di, si, oi, mi, >
fi	<	m	o	fi	di	di	fi	o	o, s, d	fi, =, f	di, si, oi	di, si, oi	di, si, oi, mi, >
di	<, m, o, fi, di	o, fi, di	o, fi, di	di	di	di	di	o, fi, di	o, fi, di, si, =, s, d, f, oi	di, si, oi	di, si, oi	di, si, oi	di, si, oi, mi, >
si	<, m, o, fi, di	o, fi, di	o, fi, di	di	di	si	si	si, =, s	d, f, oi	oi	oi	mi	>
=	<	m	o	fi	di	si	=	s	d	f	oi	mi	>

Table A.1: Allen's transitivity table – (continued).

	<	**m**	**o**	**fi**	**di**	**si**	=	**s**	**d**	**f**	**oi**	**mi**	>
s	<	<	<, m, o	<, m, o	<, m, o, fi, di	si, =, s	s	s	d	d	d, f, oi	mi	>
d	<	<	<, m, o, s, d	<, m, o, s, d	No Info	d, f, oi, mi, >	d	d	d	d	d, f, oi, mi, >	>	>
f	<	m	o, s, d	fi, =, f	di, si, oi, mi, >	oi, mi, >	f	d	d	f	oi, mi, >	>	>
oi	<, m, o, fi, di	o, fi, di	o, fi, di, si, =, s, d, f, oi	di, si, oi	di, si, oi, mi, >	oi, mi, >	oi	d, f, oi	d, f, oi	oi	oi, mi, >	>	>
mi	<, m, o, fi, di	si, =, s	d, f, oi	mi	>	>	mi	d, f, oi	d, f, oi	mi	>	>	>
>	No Info	d, f, oi, mi, >	d, f, oi, mi, >	>	>	>	>	d, f, oi, mi, >	d, f, oi, mi, >	>	>	>	>

A.2 Equal-Length Interval Midpoint Transitivity Table

Table A.2: Transitivity table for equal-length interval-interval relationships with midpoints.

	<	**m**	**so**	**mo**	**lo**	=	**loi**	**moi**	**soi**	**mi**	>
<	<	<	<	<	<	<	<, m, so	<, m, so	<, m, so, mo, lo	<, m, so, mo, lo	No Info
m	<	<	<	<	<	m	so	mo	lo	=	loi, moi, soi, mi, >
so	<	<	<	<	<, m, so	so	so, mo, lo	lo	lo, =, loi	loi	loi, moi, soi, mi, >
mo	<	<	<	m	so	mo	lo	=	loi	moi	soi, mi, >
lo	<	<	<, m, so	so	so, mo, lo	lo	lo, =, loi	loi	loi, moi, soi	soi	soi, mi, >
=	<	m	so	mo	lo	=	loi	moi	soi	mi	>
loi	<, m, so	so	so, mo, lo	lo	lo, =, loi	loi	loi, moi, soi	soi	soi, mi, >	>	>
moi	<, m, so	mo	lo	=	loi	moi	soi	mi	>	>	>
soi	<, m, so, mo, lo	lo	lo, =, loi	loi	loi, moi, soi	soi	soi, mi, >	>	>	>	>
mi	<, m, so, mo, lo	=	loi	moi	soi	mi	>	>	>	>	>
>	No Info	loi, moi, soi, mi, >	loi, moi, soi, mi, >	soi, mi, >	soi, mi, >	>	>	>	>	>	>

A.3 Variable-Length Interval Midpoint Transitivity Table

A.3.1 ***Before*** (<) to ***LargeLargeOverlap*** (llo)

Table A.3: *Before* (<) to *LargeLargeOverlap* (**llo**).

	<	**m**	**sso**	**smo**	**slo**	**mso**	**mmo**	**mlo**	**lso**	**lmo**	**llo**
<	<	<	<	<	<	<	<	<	<	<	<
m	<	<	<	<	<	<	<	<	<	<	<
sso	<	<	<	<	<	<	<	<	<, m, sso	<, m, sso	<, m, sso
smo	<	<	<	<	<	m	m	m	sso	sso	sso
slo	<	<	<, m, sso	<, m, sso	<, m, sso, smo, slo	sso	sso	sso, smo, slo	sso	sso	sso, smo, slo
mso	<	<	<	<	<	<	<	<	<, m, sso	<, m, sso	<, m, sso
mmo	<	<	<	<	<	m	m	m	sso	sso	sso
mlo	<	<	<, m, sso	<, m, sso	<, m, sso, smo, slo	sso	sso	sso, smo, slo	sso	sso	sso, smo, slo
lso	<	<	<	<	<	<	<	<	<, m, sso, mso, lso	<, m, sso, mso, lso	<, m, sso, mso, lso
lmo	<	<	<	<	<	m	m	m	sso, mso, lso	sso, mso, lso	sso, mso, lso

Table A.3: $<$ to **llo** – (continued).

	<	**m**	**sso**	**smo**	**slo**	**mso**	**mmo**	**mlo**	**lso**	**lmo**	**llo**
llo	<	<	<, m, sso	<, m, sso	<, m, sso, smo, slo	sso	sso	sso, smo, slo	sso, mso, lso	sso, mso, lso	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo
fsi	<	m	sso	smo	slo	sso	smo	slo	sso	smo	slo
fmi	<	m	sso	smo	slo	sso	smo	slo	sso	smo	slo
fli	<	m	sso	smo	slo	sso	smo	slo	sso, mso, lso	smo, mmo, lmo	slo, mlo, llo
ldfi	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, ldfi, lomi, mdfi, momi, mdli, fomi, fdli	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	ldfi, lomi, mdfi	ldfi, lomi, mdfi	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	ldfi, lomi, mdfi	ldfi, lomi, mdfi	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	ldfi, lomi, mdfi	ldfi, lomi, mdfi

Table A.3: $<$ to **llo** – (continued).

	<	**m**	**sso**	**smo**	**slo**	**mso**	**mmo**	**mlo**	**lso**	**lmo**	**llo**
lomi	<, m, sso, smo, slo, fsi, fdli	mso, mmo, mlo, fmi, fomi	lso, lmo, llo, fli, mdli	momi	mdfi	lso, lmo, llo, fli, mdli	momi	mdfi	lso, lmo, llo, fli, mdli	momi	mdfi
mdfi	<, m, sso, smo, slo, fsi, fdli	sso, smo, slo, fsi, fdli	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdfi, momi, mdli, fomi, fdli	lso, lmo, llo, fli, mdli	llo, fli, mdli	llo, fli, mdfi, momi, mdli	lso, lmo, llo, fli, mdli	llo, fli, mdli	llo, fli, mdfi, momi, mdli
momi	<, m, sso, smo, slo, fsi, fdli	sso, smo, slo, fsi, fdli	sso, smo, slo, fsi, fdli	slo, fsi, fdli	slo, fsi, fdli	mso, mmo, mlo, fmi, fomi	mlo, fmi, fomi	mlo, fmi, fomi	lso, lmo, llo, fli, mdli	llo, fli, mdli	llo, fli, mdli
mdli	<, m, sso, smo, slo, fsi, fdli	sso, smo, slo, fsi, fdli	sso, smo, slo, fsi, fdli	slo, fsi, fdli	slo, fsi, fdli	sso, smo, slo, fsi, fdli	slo, fsi, fdli	slo, fsi, fdli	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli

Table A.3: $<$ to **llo** – (continued).

	$<$	**m**	**sso**	**smo**	**slo**	**mso**	**mmo**	**mlo**	**lso**	**lmo**	**llo**
fomi	$<$, m, sso, smo, slo, fsi, fdli	sso, smo, slo, fsi, fdli	sso, smo, slo, fsi, fdli	slo, fsi, fdli	slo, fsi, fdli	sso, smo, slo, fsi, fdli	slo, fsi, fdli	slo, fsi, fdli	sso, smo, slo, fsi, fdli	slo, fsi, fdli	slo, fsi, fdli
fdli	$<$, m, sso, smo, slo, fsi, fdli	sso, smo, slo, fsi, fdli	sso, smo, slo, fsi, fdli	slo, fsi, fdli	slo, fsi, fdli	sso, smo, slo, fsi, fdli	slo, fsi, fdli	slo, fsi, fdli	sso, smo, slo, fsi, fdli	slo, fsi, fdli	slo, fsi, fdli
ssi	$<$, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, ldfi, lomi, mdfi, momi, mdli, fomi, fdli	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	ldfi, lomi, mdfi	ldfi, lomi, mdfi	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	ldfi, lomi, mdfi	ldfi, lomi, mdfi	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	ldfi, lomi, mdfi	ldfi, lomi, mdfi

Table A.3: $<$ to **llo** – (continued).

	<	**m**	**sso**	**smo**	**slo**	**mso**	**mmo**	**mlo**	**lso**	**lmo**	**llo**
smi	<, m, sso, smo, slo, fsi, fdli	mso, mmo, mlo, fmi, fomi	lso, lmo, llo, fli, mdli	momi	mdfi	lso, lmo, llo, fli, mdli	momi	mdfi	lso, lmo, llo, fli, mdli	momi	mdfi
sli	<, m, sso, smo, slo, fsi, fdli	sso, smo, slo, fsi, fdli	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdfi, momi, mdli, fomi, fdli	lso, lmo, llo, fli, mdli	llo, fli, mdli	llo, fli, mdfi, momi, mdli	lso, lmo, llo, fli, mdli	llo, fli, mdli	llo, fli, mdfi, momi, mdli
=	<	m	sso	smo	slo	mso	mmo	mlo	lso	lmo	llo
sl	<	<	<, m, sso	<, m, sso	<, m, sso, smo, slo	sso	sso	sso, smo, slo	sso, mso, lso	sso, mso, lso	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo
sm	<	<	<	<	<	m	m	m	sso, mso, lso	sso, mso, lso	sso, mso, lso
ss	<	<	<	<	<	<	<	<	<, m, sso, mso, lso	<, m, sso, mso, lso	<, m, sso, mso, lso

Table A.3: **<** to **llo** – (continued).

	<	**m**	**sso**	**smo**	**slo**	**mso**	**mmo**	**mlo**	**lso**	**lmo**	**llo**
fdl	<	<	<, m, sso, mso, lso, ss, ldf	<, m, sso, mso, lso, ss, ldf	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, fdl, fom, mdl, mom, mdf, lom, ldf	ldf	ldf	fdl, fom, mdl, mom, mdf, lom, ldf	ldf	ldf	fdl, fom, mdl, mom, mdf, lom, ldf
fom	<	<	<, m, sso, mso, lso	<, m, sso, mso, lso	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo	ss	ss	sl, sm, ss	ldf	ldf	mdl, mom, mdf, lom, ldf
mdl	<	<	<, m, sso, mso, lso	<, m, sso, mso, lso	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo	lso	lso	lso, lmo, llo	lso, ss, ldf	lso, ss, ldf	lso, lmo, llo, sl, sm, ss, mdl, mom, mdf, lom, ldf

Table A.3: **<** to **llo** – (continued).

	<	**m**	**sso**	**smo**	**slo**	**mso**	**mmo**	**mlo**	**lso**	**lmo**	**llo**
mom	<	<	<, m, sso	<, m, sso	<, m, sso, smo, slo	mso	mso	mso, mmo, mlo	lso, ss, ldf	lso, ss, ldf	lso, lmo, llo, sl, sm, ss, mdf, lom, ldf
mdf	<	<	<, m, sso	<, m, sso	<, m, sso, smo, slo	sso	sso	sso, smo, slo	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, mdf, lom, ldf
lom	<	<	<	<	<	m	m	m	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf
ldf	<	<	<	<	<	<	<	<	<, m, sso, mso, lso, ss, ldf	<, m, sso, mso, lso, ss, ldf	<, m, sso, mso, lso, ss, ldf
fl	<	m	sso, mso, lso	smo, mmo, lmo	slo, mlo, llo	lso	lmo	llo	lso, ss, ldf	lmo, sm, lom	llo, sl, mdl, mom, mdf
fm	<	m	sso, mso, lso	smo, mmo, lmo	slo, mlo, llo	ss	sm	sl	ldf	lom	mdl, mom, mdf

Table A.3: < to **llo** – (continued).

	<	**m**	**sso**	**smo**	**slo**	**mso**	**mmo**	**mlo**	**lso**	**lmo**	**llo**
fs	<	m	sso, mso, lso, ss, ldf	smo, mmo, lmo, sm, lom	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf	ldf	lom	fdl, fom, mdl, mom, mdf	ldf	lom	fdl, fom, mdl, mom, mdf
lloi	<, m, sso, smo, slo, fsi, fdli	sso, smo, slo, fsi, fdli	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdfi, momi, mdli, fomi, fdli	lso, lmo, llo, fli, mdli	llo, fli, mdli	llo, fli, mdfi, momi, mdli	lso, lmo, llo, fli, mdli, sl, sm, ss, mdf, lom, ldf	llo, fli, mdli, sl, mdf	llo, fli, mdfi, momi, mdli, sli, =, sl, mdl, mom, mdf, fl, lloi
lmoi	<, m, sso, smo, slo, fsi, fdli	mso, mmo, mlo, fmi, fomi	lso, lmo, llo, fli, mdli	momi	mdfi	lso, lmo, llo, fli, mdli	momi	mdfi	lso, lmo, llo, fli, mdli, sl, sm, ss, mdf, lom, ldf	momi, =, mom	mdfi, sli, mdl, fl, lloi

Table A.3: < to **llo** – (continued).

	<	**m**	**sso**	**smo**	**slo**	**mso**	**mmo**	**mlo**	**lso**	**lmo**	**llo**
lsoi	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, ldfi, lomi, mdfi, momi, mdli, fomi, fdli	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	ldfi, lomi, mdfi	ldfi, lomi, mdfi	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	ldfi, lomi, mdfi	ldfi, lomi, mdfi	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli, ssi, smi, sli, =, sl, sm, ss, mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	ldfi, lomi, mdfi, ssi, smi, sli, mdl, fl, lloi, lmoi, lsoi	ldfi, lomi, mdfi, ssi, smi, sli, mdl, fl, lloi, lmoi, lsoi
mloi	<, m, sso, smo, slo, fsi, fdli	sso, smo, slo, fsi, fdli	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdfi, momi, mdli, fomi, fdli	sl, sm, ss	sl	sli, =, sl	mdf, lom, ldf	mdf	mdl, mom, mdf, fl, lloi

Table A.3: $<$ to **llo** – (continued).

	<	**m**	**sso**	**smo**	**slo**	**mso**	**mmo**	**mlo**	**lso**	**lmo**	**llo**
mmoi	<, m, sso, smo, slo, fsi, fdli	mso, mmo, mlo, fmi, fomi	lso, lmo, llo, fli, mdli	momi	mdfi	sl, sm, ss	=	sli	mdf, lom, ldf	mom	mdl, fl, lloi
msoi	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, ldfi, lomi, mdfi, momi, mdli, fomi, fdli	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	ldfi, lomi, mdfi	ldfi, lomi, mdfi	ssi, smi, sli, =, sl, sm, ss	ssi, smi, sli	ssi, smi, sli	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	mdl, fl, lloi, lmoi, lsoi	mdl, fl, lloi, lmoi, lsoi

Table A.3: $<$ to **llo** – (continued).

	$<$	**m**	**sso**	**smo**	**slo**	**mso**	**mmo**	**mlo**	**lso**	**lmo**	**llo**
sloi	<, m, sso, smo, slo, fsi, fdli	sso, smo, slo, fsi, fdli	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, mdli, fomi, fdli, sl, sm, ss, mdf, lom, ldf	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli, sl, mdf	slo, mlo, llo, fsi, fmi, fli, mdfi, momi, mdli, fomi, fdli, sli, =, sl, fdl, fom, mdl, mom, mdf, fl, fm, fs, lloi, mloi, sloi	mdf, lom, ldf	mdf	fdl, fom, mdl, mom, mdf, fl, fm, fs, lloi, mloi, sloi	mdf, lom, ldf	mdf	fdl, fom, mdl, mom, mdf, fl, fm, fs, lloi, mloi, sloi

Table A.3: $<$ to **llo** – (continued).

	<	**m**	**sso**	**smo**	**slo**	**mso**	**mmo**	**mlo**	**lso**	**lmo**	**llo**
smoi	<, m, sso, smo, slo, fsi, fdli	mso, mmo, mlo, fmi, fomi	lso, lmo, llo, fli, mdli, sl, sm, ss, mdf, lom, ldf	momi, =, mom	mdfi, sli, fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	mdf, lom, ldf	mom	fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	mdf, lom, ldf	mom	fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi
ssoi	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, ldfi, lomi, mdfi, momi, mdli, fomi, fdli	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli, ssi, smi, sli, =, sl, sm, ss, mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	ldfi, lomi, mdfi, ssi, smi, sli, mdl, fl, lloi, lmoi, lsoi	ldfi, lomi, mdfi, ssi, smi, sli, fdl, fom, mdl, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	mdl, fl, lloi, lmoi, lsoi	fdl, fom, mdl, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	mdl, fl, lloi, lmoi, lsoi	fdl, fom, mdl, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi

Table A.3: $<$ to **llo** – (continued).

	$<$	**m**	**sso**	**smo**	**slo**	**mso**	**mmo**	**mlo**	**lso**	**lmo**	**llo**
mi	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, ldfi, lomi, mdfi, momi, mdli, fomi, fdli	ssi, smi, sli, =, sl, sm, ss	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	fom, fm, mloi, mmoi, msoi	fdl, fs, sloi, smoi, ssoi	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	fom, fm, mloi, mmoi, msoi	fdl, fs, sloi, smoi, ssoi	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	fom, fm, mloi, mmoi, msoi	fdl, fs, sloi, smoi, ssoi

Table A.3: $<$ to **llo** – (continued).

	$<$	**m**	**sso**	**smo**	**slo**	**mso**	**mmo**	**mlo**	**lso**	**lmo**	**llo**
$>$	No Info	fdl, fom, mdl, mom, mdf, lom, ldf, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, $>$	fdl, fom, mdl, mom, mdf, lom, ldf, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, $>$	fdl, fs, sloi, smoi, ssoi, mi, $>$	fdl, fs, sloi, smoi, ssoi, mi, $>$	fdl, fom, mdl, mom, mdf, lom, ldf, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, $>$	fdl, fs, sloi, smoi, ssoi, mi, $>$	fdl, fs, sloi, smoi, ssoi, mi, $>$	fdl, fom, mdl, mom, mdf, lom, ldf, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, $>$	fdl, fs, sloi, smoi, ssoi, mi, $>$	fdl, fs, sloi, smoi, ssoi, mi, $>$

A.3.2 *is-FinishedSmall-by* (fsi) to *LastContainsFirst-of* (fdli)

Table A.4: *is-FinishedSmall-by* (**fsi**) to *LastContainsFirst-of* (**fdli**).

	fsi	**fmi**	**fli**	**ldfi**	**lomi**	**mdfi**	**momi**	**mdli**	**fomi**	**fdli**
<	<	<	<	<	<	<	<	<	<	<
m	<	<	<	<	<	<	<	<	<	<
sso	<	<	<, m, sso	<, m, sso, smo, slo, fsi, fdli	<, m, sso, smo, slo	<, m, sso, smo, slo	<, m, sso	<, m, sso	<	<
smo	<	m	sso	fdli	fsi	slo	smo	sso	m	<
slo	<, m, sso, smo, slo	sso, smo, slo	sso, smo, slo	fdli	fdli	slo, fsi, fdli	slo, fsi, fdli	sso, smo, slo, fsi, fdli	sso, smo, slo, fsi, fdli	<, m, sso, smo, slo, fsi, fdli
mso	<	<	<, m, sso	<, m, sso, smo, slo, fsi, fdli	<, m, sso, smo, slo	<, m, sso, smo, slo	<, m, sso	<, m, sso	<	<
mmo	<	m	sso	fdli	fsi	slo	smo	sso	m	<
mlo	<, m, sso, smo, slo	sso, smo, slo	sso, smo, slo	fdli	fdli	slo, fsi, fdli	slo, fsi, fdli	sso, smo, slo, fsi, fdli	sso, smo, slo, fsi, fdli	<, m, sso, smo, slo, fsi, fdli

Table A.4: fsi to fdli – (continued).

	fsi	**fmi**	**fli**	**ldfi**	**lomi**	**mdfi**	**momi**	**mdli**	**fomi**	**fdli**
lso	<	<	<, m, sso, mso, lso	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, ldfi, lomi, mdfi, momi, mdli, fomi, fdli	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo	<, m, sso, mso, lso	<, m, sso, mso, lso	<	<
lmo	<	m	sso, mso, lso	ldfi, lomi, mdfi, momi, mdli, fomi, fdli	fsi, fmi, fli	slo, mlo, llo	smo, mmo, lmo	sso, mso, lso	m	<
llo	<, m, sso, smo, slo	sso, smo, slo	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo	ldfi, lomi, mdfi, momi, mdli, fomi, fdli	mdfi, momi, mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdfi, momi, mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, mdli, fomi, fdli	sso, smo, slo, fsi, fdli	<, m, sso, smo, slo, fsi, fdli

Table A.4: fsi to fdli – (continued).

	fsi	**fmi**	**fli**	**ldfi**	**lomi**	**mdfi**	**momi**	**mdli**	**fomi**	**fdli**
fsi	fsi	fsi	fsi	fdli	fdli	fdli	fdli	fdli	fdli	fdli
fmi	fsi	fsi	fsi	fdli	fdli	fdli	fdli	fdli	fdli	fdli
fli	fsi	fsi	fsi, fmi, fli	ldfi, lomi, mdfi, momi, mdli, fomi, fdli	mdfi, momi, mdli, fomi, fdli	mdfi, momi, mdli, fomi, fdli	mdli, fomi, fdli	mdli, fomi, fdli	fdli	fdli
ldfi	ldfi	ldfi	ldfi	ldfi	ldfi	ldfi	ldfi	ldfi	ldfi	ldfi
lomi	lomi	lomi	lomi	ldfi	ldfi	ldfi	ldfi	ldfi	ldfi	ldfi
mdfi	mdfi, momi, mdli, fomi, fdli	mdfi, momi, mdli	mdfi, momi, mdli	ldfi	ldfi	ldfi, lomi, mdfi	ldfi, lomi, mdfi	ldfi, lomi, mdfi, momi, mdli	ldfi, lomi, mdfi, momi, mdli	ldfi, lomi, mdfi, momi, mdli, fomi, fdli
momi	fdli	fomi	mdli	ldfi	lomi	mdfi	momi	mdli	fomi	fdli
mdli	fdli	fdli	mdli, fomi, fdli	ldfi, lomi, mdfi, momi, mdli, fomi, fdli	mdfi, momi, mdli, fomi, fdli	mdfi, momi, mdli, fomi, fdli	mdli, fomi, fdli	mdli, fomi, fdli	fdli	fdli

Table A.4: fsi to **fdli** – (continued).

	fsi	**fmi**	**fli**	**ldfi**	**lomi**	**mdfi**	**momi**	**mdli**	**fomi**	**fdli**
fomi	fdli	fdli	fdli	fdli	fdli	fdli	fdli	fdli	fdli	fdli
fdli	fdli	fdli	fdli	fdli	fdli	fdli	fdli	fdli	fdli	fdli
ssi	ldfi	ldfi	ldfi	ldfi	ldfi	ldfi	ldfi	ldfi	ldfi	ldfi
smi	lomi	lomi	lomi	ldfi	ldfi	ldfi	ldfi	ldfi	ldfi	ldfi
sli	mdfi, momi, mdli, fomi, fdli	mdfi, momi, mdli	mdfi, momi, mdli	ldfi	ldfi	ldfi, lomi, mdfi	ldfi, lomi, mdfi	ldfi, lomi, mdfi, momi, mdli	ldfi, lomi, mdfi, momi, mdli	ldfi, lomi, mdfi, momi, mdli, fomi, fdli
=	fsi	fmi	fli	ldfi	lomi	mdfi	momi	mdli	fomi	fdli
sl	<, m, sso, smo, slo	sso, smo, slo	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo	ldfi, lomi, mdfi, momi, mdli, fomi, fdli	mdfi, momi, mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdfi, momi, mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, mdli, fomi, fdli	sso, smo, slo, fsi, fdli	<, m, sso, smo, slo, fsi, fdli
sm	<	m	sso, mso, lso	ldfi, lomi, mdfi, momi, mdli, fomi, fdli	fsi, fmi, fli	slo, mlo, llo	smo, mmo, lmo	sso, mso, lso	m	<

Table A.4: fsi to **fdli** – (continued).

	fsi	**fmi**	**fli**	**ldfi**	**lomi**	**mdfi**	**momi**	**mdli**	**fomi**	**fdli**
ss	<	<	<, m, sso, mso, lso	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, ldfi, lomi, mdfi, momi, mdli, fomi, fdli	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo	<, m, sso, mso, lso	<, m, sso, mso, lso	<	<
fdl	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, fdl, fom, mdl, mom, mdf, lom, ldf	fdl, fom, mdl, mom, mdf, lom, ldf	fdl, fom, mdl, mom, mdf, lom, ldf	>	>	fdl, fs, sloi, smoi, ssoi, mi, >	fdl, fs, sloi, smoi, ssoi, mi, >	fdl, fom, mdl, mom, mdf, lom, ldf, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >	fdl, fom, mdl, mom, mdf, lom, ldf, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >	No Info

Table A.4: fsi to **fdli** – (continued).

	fsi	**fmi**	**fli**	**ldfi**	**lomi**	**mdfi**	**momi**	**mdli**	**fomi**	**fdli**
fom	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo	sl, sm, ss	mdl, mom, mdf, lom, ldf	>	mi	fdl, fs, sloi, smoi, ssoi	fom, fm, mloi, mmoi, msoi	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	ssi, smi, sli, =, sl, sm, ss	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, ldfi, lomi, mdfi, momi, mdli, fomi, fdli
mdl	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo	lso, lmo, llo	lso, lmo, llo, sl, sm, ss, mdl, mom, mdf, lom, ldf	ldfi, ssi, lsoi, msoi, ssoi, mi, >	ldfi, ssi, lsoi, msoi, ssoi	ldfi, lomi, mdfi, ssi, smi, sli, fdl, fom, mdl, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	ldfi, lomi, mdfi, ssi, smi, sli, mdl, fl, lloi, lmoi, lsoi	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli, ssi, smi, sli, =, sl, sm, ss, mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, ldfi, lomi, mdfi, momi, mdli, fomi, fdli

Table A.4: fsi to **fdli** – (continued).

	fsi	**fmi**	**fli**	**ldfi**	**lomi**	**mdfi**	**momi**	**mdli**	**fomi**	**fdli**
mom	<, m, sso, smo, slo	mso, mmo, mlo	lso, lmo, llo, sl, sm, ss, mdf, lom, ldf	ldfi, ssi, lsoi, msoi, ssoi, mi, >	lomi, smi, lmoi, mmoi, smoi	mdfi, sli, fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	momi, =, mom	lso, lmo, llo, fli, mdli, sl, sm, ss, mdf, lom, ldf	mso, mmo, mlo, fmi, fomi	<, m, sso, smo, slo, fsi, fdli
mdf	<, m, sso, smo, slo	sso, smo, slo	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, mdf, lom, ldf	ldfi, lomi, mdfi, momi, mdli, fomi, fdli, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	slo, mlo, llo, fsi, fmi, fli, mdfi, momi, mdli, fomi, fdli, sli, =, sl, fdl, fom, mdl, mom, mdf, fl, fm, fs, lloi, mloi, sloi	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli, sl, mdf	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, mdli, fomi, fdli, sl, sm, ss, mdf, lom, ldf	sso, smo, slo, fsi, fdli	<, m, sso, smo, slo, fsi, fdli

Table A.4: fsi to fdli – (continued).

	fsi	**fmi**	**fli**	**ldfi**	**lomi**	**mdfi**	**momi**	**mdli**	**fomi**	**fdli**
lom	$<$	m	sso, mso, lso, ss, ldf	ldfi, lomi, mdfi, momi, mdli, fomi, fdli, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, $>$	fsi, fmi, fli, =, fl, fm, fs	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf	smo, mmo, lmo, sm, lom	sso, mso, lso, ss, ldf	m	$<$
ldf	$<$	$<$	$<$, m, sso, mso, lso, ss, ldf	No Info	$<$, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, fdl, fom, mdl, mom, mdf, lom, ldf	$<$, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, fdl, fom, mdl, mom, mdf, lom, ldf	$<$, m, sso, mso, lso, ss, ldf	$<$, m, sso, mso, lso, ss, ldf	$<$	$<$

Table A.4: fsi to **fdli** – (continued).

	fsi	**fmi**	**fli**	**ldfi**	**lomi**	**mdfi**	**momi**	**mdli**	**fomi**	**fdli**
fl	fsi, fmi, fli	fli	fli, =, fl	ldfi, ssi, lsoi, msoi, ssoi, mi, >	ldfi, ssi, lsoi, msoi, ssoi	ldfi, lomi, mdfi, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	ldfi, lomi, mdfi, ssi, smi, sli, lloi, lmoi, lsoi	ldfi, lomi, mdfi, momi, mdli, ssi, smi, sli, lloi, lmoi, lsoi	ldfi, lomi, mdfi, momi, mdli	ldfi, lomi, mdfi, momi, mdli, fomi, fdli
fm	fsi, fmi, fli	=	fl	>	mi	sloi, smoi, ssoi	mloi, mmoi, msoi	lloi, lmoi, lsoi	ssi, smi, sli	ldfi, lomi, mdfi, momi, mdli, fomi, fdli
fs	fsi, fmi, fli, =, fl, fm, fs	fl, fm, fs	fl, fm, fs	>	>	sloi, smoi, ssoi, mi, >	sloi, smoi, ssoi, mi, >	lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >	lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >	ldfi, lomi, mdfi, momi, mdli, fomi, fdli, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >

Table A.4: fsi to **fdli** – (continued).

	fsi	**fmi**	**fli**	**ldfi**	**lomi**	**mdfi**	**momi**	**mdli**	**fomi**	**fdli**
lloi	mdfi, momi, mdli, fomi, fdli	mdfi, momi, mdli	mdfi, momi, mdli, sli, lloi	ldfi, ssi, lsoi, msoi, ssoi, mi, >	ldfi, ssi, lsoi, msoi, ssoi	ldfi, lomi, mdfi, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	ldfi, lomi, mdfi, ssi, smi, sli, lloi, lmoi, lsoi	ldfi, lomi, mdfi, momi, mdli, ssi, smi, sli, lloi, lmoi, lsoi	ldfi, lomi, mdfi, momi, mdli	ldfi, lomi, mdfi, momi, mdli, fomi, fdli
lmoi	lomi	lomi	lomi, smi, lmoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi	ldfi	ldfi
lsoi	ldfi	ldfi	ldfi, ssi, lsoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi	ldfi	ldfi
mloi	mdfi, momi, mdli, fomi, fdli	sli	lloi	>	mi	sloi, smoi, ssoi	mloi, mmoi, msoi	lloi, lmoi, lsoi	ssi, smi, sli	ldfi, lomi, mdfi, momi, mdli, fomi, fdli

Table A.4: fsi to fdli – (continued).

	fsi	**fmi**	**fli**	**ldfi**	**lomi**	**mdfi**	**momi**	**mdli**	**fomi**	**fdli**
mmoi	lomi	smi	lmoi	>	mi	ssoi	msoi	lsoi	ssi	ldfi
msoi	ldfi	ssi	lsoi	>	mi	ssoi	msoi	lsoi	ssi	ldfi
sloi	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	lloi, mloi, sloi	lloi, mloi, sloi	>	>	sloi, smoi, ssoi, mi, >	sloi, smoi, ssoi, mi, >	lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >	lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >	ldfi, lomi, mdfi, momi, mdli, fomi, fdli, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >
smoi	lomi, smi, lmoi, mmoi, smoi	lmoi, mmoi, smoi	lmoi, mmoi, smoi	>	>	ssoi, mi, >	ssoi, mi, >	lsoi, msoi, ssoi, mi, >	lsoi, msoi, ssoi, mi, >	ldfi, ssi, lsoi, msoi, ssoi, mi, >
ssoi	ldfi, ssi, lsoi, msoi, ssoi	lsoi, msoi, ssoi	lsoi, msoi, ssoi	>	>	ssoi, mi, >	ssoi, mi, >	lsoi, msoi, ssoi, mi, >	lsoi, msoi, ssoi, mi, >	ldfi, ssi, lsoi, msoi, ssoi, mi, >
mi	mi	mi	mi	>	>	>	>	>	>	>
>	>	>	>	>	>	>	>	>	>	>

A.3.3 *is-StartedSmall-by* (ssi) to *StartsSmall* (ss)

Table A.5: *is-StartedSmall-by* (**ssi**) to *StartsSmall* (**ss**).

	ssi	**smi**	**sli**	=	**sl**	**sm**	**ss**
<	<	<	<	<	<	<	<
m	m	m	m	m	m	m	m
sso	sso, smo, slo, fsi, fdli	sso, smo, slo	sso, smo, slo	sso	sso	sso	sso
smo	fdli	fsi	slo	smo	sso	sso	sso
slo	fdli	fdli	slo, fsi, fdli	slo	sso, smo, slo	sso	sso
mso	mso, mmo, mlo, fmi, fomi	mso, mmo, mlo	mso, mmo, mlo	mso	mso	mso	mso
mmo	fomi	fmi	mlo	mmo	mso	mso	mso
mlo	fomi	fomi	mlo, fmi, fomi	mlo	mso, mmo, mlo	mso	mso
lso	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli	lso, lmo, llo	lso, lmo, llo	lso	lso	lso	lso
lmo	ldfi, lomi, mdfi, momi, mdli	fli	llo	lmo	lso	lso	lso
llo	ldfi, lomi, mdfi, momi, mdli	mdfi, momi, mdli	llo, fli, mdfi, momi, mdli	llo	lso, lmo, llo	lso	lso
fsi	fdli	fdli	fdli	fsi	slo	smo	sso

Table A.5: ssi to **ss** – (continued).

	ssi	**smi**	**sli**	=	**sl**	**sm**	**ss**
fmi	fomi	fomi	fomi	fmi	mlo	mmo	mso
fli	ldfi, lomi, mdfi, momi, mdli	mdfi, momi, mdli	mdfi, momi, mdli	fli	llo	lmo	lso
ldfi	ldfi	ldfi	ldfi	ldfi	ldfi, lomi, mdfi	ldfi, lomi, mdfi	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli
lomi	ldfi	ldfi	ldfi	lomi	mdfi	momi	lso, lmo, llo, fli, mdli
mdfi	ldfi	ldfi	ldfi, lomi, mdfi	mdfi	llo, fli, mdfi, momi, mdli	llo, fli, mdli	lso, lmo, llo, fli, mdli
momi	ldfi	lomi	mdfi	momi	llo, fli, mdli	llo, fli, mdli	lso, lmo, llo, fli, mdli
mdli	ldfi, lomi, mdfi, momi, mdli	mdfi, momi, mdli	mdfi, momi, mdli	mdli	llo, fli, mdli	llo, fli, mdli	lso, lmo, llo, fli, mdli
fomi	fomi	fomi	fomi	fomi	mlo, fmi, fomi	mlo, fmi, fomi	mso, mmo, mlo, fmi, fomi
fdli	fdli	fdli	fdli	fdli	slo, fsi, fdli	slo, fsi, fdli	sso, smo, slo, fsi, fdli
ssi	ssi	ssi	ssi	ssi	ssi, smi, sli	ssi, smi, sli	ssi, smi, sli, =, sl, sm, ss
smi	ssi	ssi	ssi	smi	sli	=	sl, sm, ss
sli	ssi	ssi	ssi, smi, sli	sli	sli, =, sl	sl	sl, sm, ss

Table A.5: ssi to **ss** – (continued).

	ssi	**smi**	**sli**	**=**	**sl**	**sm**	**ss**
=	ssi	smi	sli	=	sl	sm	ss
sl	ssi, smi, sli	sli	sli, =, sl	sl	sl, sm, ss	ss	ss
sm	ssi, smi, sli	=	sl	sm	ss	ss	ss
ss	ssi, smi, sli, =, sl, sm, ss	sl, sm, ss	sl, sm, ss	ss	ss	ss	ss
fdl	>	>	fdl, fs, sloi, smoi, ssoi, mi, >	fdl	fdl, fom, mdl, mom, mdf, lom, ldf	ldf	ldf
fom	>	mi	fdl, fs, sloi, smoi, ssoi	fom	mdl, mom, mdf, lom, ldf	ldf	ldf
mdl	lsoi, msoi, ssoi, mi, >	lsoi, msoi, ssoi	fdl, fom, mdl, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	mdl	mdl, mom, mdf, lom, ldf	ldf	ldf
mom	lsoi, msoi, ssoi, mi, >	lmoi, mmoi, smoi	fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	mom	mdf, lom, ldf	ldf	ldf
mdf	lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >	lloi, mloi, sloi	fdl, fom, mdl, mom, mdf, fl, fm, fs, lloi, mloi, sloi	mdf	mdf, lom, ldf	ldf	ldf

Table A.5: ssi to **ss** – (continued).

	ssi	**smi**	**sli**	**=**	**sl**	**sm**	**ss**
lom	lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >	fl, fm, fs	fdl, fom, mdl, mom, mdf	lom	ldf	ldf	ldf
ldf	fdl, fom, mdl, mom, mdf, lom, ldf, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >	fdl, fom, mdl, mom, mdf, lom, ldf	fdl, fom, mdl, mom, mdf, lom, ldf	ldf	ldf	ldf	ldf
fl	lsoi, msoi, ssoi, mi, >	lsoi, msoi, ssoi	lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	fl	mdl, mom, mdf	lom	ldf
fm	>	mi	sloi, smoi, ssoi	fm	mdl, mom, mdf	lom	ldf
fs	>	>	sloi, smoi, ssoi, mi, >	fs	fdl, fom, mdl, mom, mdf	lom	ldf
lloi	lsoi, msoi, ssoi, mi, >	lsoi, msoi, ssoi	lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	lloi	mdl, mom, mdf, fl, lloi	mdf	mdf, lom, ldf

Table A.5: ssi to **ss** – (continued).

	ssi	**smi**	**sli**	=	**sl**	**sm**	**ss**
lmoi	lsoi, msoi, ssoi, mi, >	lsoi, msoi, ssoi	lsoi, msoi, ssoi	lmoi	mdl, fl, lloi	mom	mdf, lom, ldf
lsoi	lsoi, msoi, ssoi, mi, >	lsoi, msoi, ssoi	lsoi, msoi, ssoi	lsoi	mdl, fl, lloi, lmoi, lsoi	mdl, fl, lloi, lmoi, lsoi	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi
mloi	>	mi	sloi, smoi, ssoi	mloi	mdl, mom, mdf, fl, lloi	mdf	mdf, lom, ldf
mmoi	>	mi	ssoi	mmoi	mdl, fl, lloi	mom	mdf, lom, ldf
msoi	>	mi	ssoi	msoi	mdl, fl, lloi, lmoi, lsoi	mdl, fl, lloi, lmoi, lsoi	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi
sloi	>	>	sloi, smoi, ssoi, mi, >	sloi	fdl, fom, mdl, mom, mdf, fl, fm, fs, lloi, mloi, sloi	mdf	mdf, lom, ldf
smoi	>	>	ssoi, mi, >	smoi	fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	mom	mdf, lom, ldf
ssoi	>	>	ssoi, mi, >	ssoi	fdl, fom, mdl, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	mdl, fl, lloi, lmoi, lsoi	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi

Table A.5: ssi to **ss** – (continued).

	ssi	**smi**	**sli**	**=**	**sl**	**sm**	**ss**
mi	>	>	>	mi	fdl, fs, sloi, smoi, ssoi	fom, fm, mloi, mmoi, msoi	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi
>	>	>	>	>	fdl, fs, sloi, smoi, ssoi, mi, >	fdl, fs, sloi, smoi, ssoi, mi, >	fdl, fom, mdl, mom, mdf, lom, ldf, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >

A.3.4 ***FirstDuringLast*** (fdl) to ***FinishesSmall*** (fs)

Table A.6: *FirstDuringLast* (**fdl**) to *FinishesSmall* (**fs**).

	fdl	**fom**	**mdl**	**mom**	**mdf**	**lom**	**ldf**	**fl**	**fm**	**fs**
<	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, fdl, fom, mdl, mom, mdf, lom, ldf	<, m, sso, mso, lso, ss, ldf	<, m, sso, mso, lso, ss, ldf	<, m, sso, mso, lso, ss, ldf	<, m, sso, mso, lso, ss, ldf	<, m, sso, mso, lso, ss, ldf	<, m, sso, mso, lso, ss, ldf	<, m, sso, mso, lso, ss, ldf	<, m, sso, mso, lso, ss, ldf	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, fdl, fom, mdl, mom, mdf, lom, ldf
m	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf	smo, mmo, lmo, sm, lom	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf	smo, mmo, lmo, sm, lom	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf
sso	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf	slo, mlo, llo, sl, mdf	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, mdf, lom, ldf	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, mdf, lom, ldf	slo, mlo, llo, sl, mdf	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf

Table A.6: fdl to **fs** – (continued).

	fdl	**fom**	**mdl**	**mom**	**mdf**	**lom**	**ldf**	**fl**	**fm**	**fs**
smo	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf	slo, mlo, llo, sl, mdf	slo, mlo, llo, sl, mdf	smo, mmo, lmo, sm, lom	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf	slo, mlo, llo, sl, mdf	slo, mlo, llo, sl, mdf	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf
slo	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf	slo, mlo, llo, sl, mdf	slo, mlo, llo, sl, mdf	slo, mlo, llo, sl, mdf	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, mdf, lom, ldf	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf	slo, mlo, llo, sl, mdf	slo, mlo, llo, sl, mdf	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf
mso	fdl, fom, mdl	mom	lso, lmo, llo, sl, sm, ss, mdf, lom, ldf	lso, ss, ldf	lso, ss, ldf	lso, ss, ldf	lso, ss, ldf	lso, lmo, llo, sl, sm, ss, mdf, lom, ldf	mom	fdl, fom, mdl
mmo	fdl, fom, mdl	mom	llo, sl, mdf	lmo, sm, lom	lso, ss, ldf	lso, ss, ldf	lso, ss, ldf	llo, sl, mdf	mom	fdl, fom, mdl
mlo	fdl, fom, mdl	mom	llo, sl, mdf	llo, sl, mdf	lso, lmo, llo, sl, sm, ss, mdf, lom, ldf	lso, ss, ldf	lso, ss, ldf	llo, sl, mdf	mom	fdl, fom, mdl

Table A.6: fdl to **fs** – (continued).

	fdl	**fom**	**mdl**	**mom**	**mdf**	**lom**	**ldf**	**fl**	**fm**	**fs**
lso	fdl, fom, mdl	mdl	lso, lmo, llo, sl, sm, ss, mdl, mom, mdf, lom, ldf	lso, ss, ldf	lso, ss, ldf	lso, ss, ldf	lso, ss, ldf	lso, lmo, llo, sl, sm, ss, mdl, mom, mdf, lom, ldf	mdl	fdl, fom, mdl
lmo	fdl, fom, mdl	mdl	llo, sl, mdl, mom, mdf	lmo, sm, lom	lso, ss, ldf	lso, ss, ldf	lso, ss, ldf	llo, sl, mdl, mom, mdf	mdl	fdl, fom, mdl
llo	fdl, fom, mdl	mdl	llo, sl, mdl, mom, mdf	llo, sl, mdf	lso, lmo, llo, sl, sm, ss, mdf, lom, ldf	lso, ss, ldf	lso, ss, ldf	llo, sl, mdl, mom, mdf	mdl	fdl, fom, mdl
fsi	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf	slo, mlo, llo, sl, mdf	slo, mlo, llo, sl, mdf	slo, mlo, llo, sl, mdf	slo, mlo, llo, sl, mdf	smo, mmo, lmo, sm, lom	sso, mso, lso, ss, ldf	fsi, fmi, fli	fsi, fmi, fli	fsi, fmi, fli, =, fl, fm, fs
fmi	fdl, fom, mdl	mom	llo, sl, mdf	llo, sl, mdf	llo, sl, mdf	lmo, sm, lom	lso, ss, ldf	fli	=	fl, fm, fs
fli	fdl, fom, mdl	mdl	llo, sl, mdl, mom, mdf	llo, sl, mdf	llo, sl, mdf	lmo, sm, lom	lso, ss, ldf	fli, =, fl	fl	fl, fm, fs

Table A.6: fdl to **fs** – (continued).

	fdl	**fom**	**mdl**	**mom**	**mdf**	**lom**	**ldf**	**fl**	**fm**	**fs**
ldfi	ldfi, lomi, mdfi, ssi, smi, sli, fdl, fom, mdl, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	ldfi, lomi, mdfi, ssi, smi, sli, mdl, fl, lloi, lmoi, lsoi	ldfi, lomi, mdfi, ssi, smi, sli, mdl, fl, lloi, lmoi, lsoi	ldfi, lomi, mdfi, ssi, smi, sli, mdl, fl, lloi, lmoi, lsoi	ldfi, lomi, mdfi, ssi, smi, sli, mdl, fl, lloi, lmoi, lsoi	ldfi, lomi, mdfi, ssi, smi, sli, mdl, fl, lloi, lmoi, lsoi	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli, ssi, smi, sli, =, sl, sm, ss, mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi, msoi, ssoi
lomi	mdfi, sli, fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	mdfi, sli, mdl, fl, lloi	mdfi, sli, mdl, fl, lloi	mdfi, sli, mdl, fl, lloi	mdfi, sli, mdl, fl, lloi	momi, =, mom	lso, lmo, llo, fli, mdli, sl, sm, ss, mdf, lom, ldf	lomi, smi, lmoi	lomi, smi, lmoi	lomi, smi, lmoi, mmoi, smoi

Table A.6: fdl to **fs** – (continued).

	fdl	**fom**	**mdl**	**mom**	**mdf**	**lom**	**ldf**	**fl**	**fm**	**fs**
mdfi	mdfi, sli, fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	mdfi, sli, mdl, fl, lloi	mdfi, sli, mdl, fl, lloi	mdfi, sli, mdl, fl, lloi	llo, fli, mdfi, momi, mdli, sli, =, sl, mdl, mom, mdf, fl, lloi	llo, fli, mdli, sl, mdf	lso, lmo, llo, fli, mdli, sl, sm, ss, mdf, lom, ldf	mdfi, sli, lloi	mdfi, sli, lloi	mdfi, sli, lloi, mloi, sloi
momi	mdfi, sli, fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	mdfi, sli, mdl, fl, lloi	mdfi, sli, mdl, fl, lloi	momi, =, mom	llo, fli, mdli, sl, mdf	llo, fli, mdli, sl, mdf	lso, lmo, llo, fli, mdli, sl, sm, ss, mdf, lom, ldf	mdfi, sli, lloi	mdfi, sli, lloi	mdfi, sli, lloi, mloi, sloi
mdli	mdfi, sli, fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	mdfi, sli, mdl, fl, lloi	llo, fli, mdfi, momi, mdli, sli, =, sl, mdl, mom, mdf, fl, lloi	llo, fli, mdli, sl, mdf	llo, fli, mdli, sl, mdf	llo, fli, mdli, sl, mdf	lso, lmo, llo, fli, mdli, sl, sm, ss, mdf, lom, ldf	mdfi, momi, mdli, sli, lloi	mdfi, sli, lloi	mdfi, sli, lloi, mloi, sloi

Table A.6: fdl to **fs** – (continued).

	fdl	**fom**	**mdl**	**mom**	**mdf**	**lom**	**ldf**	**fl**	**fm**	**fs**
fomi	mdfi, sli, fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	momi, =, mom	llo, fli, mdli, sl, mdf	llo, fli, mdli, sl, mdf	llo, fli, mdli, sl, mdf	llo, fli, mdli, sl, mdf	lso, lmo, llo, fli, mdli, sl, sm, ss, mdf, lom, ldf	mdli	momi	mdfi, sli, lloi, mloi, sloi
fdli	slo, mlo, llo, fsi, fmi, fli, mdfi, momi, mdli, fomi, fdli, sli, =, sl, fdl, fom, mdl, mom, mdf, fl, fm, fs, lloi, mloi, sloi	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli, sl, mdf	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli, sl, mdf	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli, sl, mdf	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli, sl, mdf	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli, sl, mdf	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, mdli, fomi, fdli, sl, sm, ss, mdf, lom, ldf	mdli, fomi, fdli	mdli, fomi, fdli	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi

Table A.6: fdl to **fs** – (continued).

	fdl	**fom**	**mdl**	**mom**	**mdf**	**lom**	**ldf**	**fl**	**fm**	**fs**
ssi	fdl, fs, sloi, smoi, ssoi	fom, fm, mloi, mmoi, msoi	mdl, fl, lloi, lmoi, lsoi	mdl, fl, lloi, lmoi, lsoi	mdl, fl, lloi, lmoi, lsoi	mdl, fl, lloi, lmoi, lsoi	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	lsoi	msoi	ssoi
smi	fdl, fs, sloi	fom, fm, mloi	mdl, fl, lloi	mdl, fl, lloi	mdl, fl, lloi	mom	mdf, lom, ldf	lmoi	mmoi	smoi
sli	fdl, fs, sloi	fom, fm, mloi	mdl, fl, lloi	mdl, fl, lloi	mdl, mom, mdf, fl, lloi	mdf	mdf, lom, ldf	lloi	mloi	sloi
=	fdl	fom	mdl	mom	mdf	lom	ldf	fl	fm	fs
sl	fdl	fom	mdl, mom, mdf	mdf	mdf, lom, ldf	ldf	ldf	mdl, mom, mdf	fom	fdl
sm	fdl	fom	mdl, mom, mdf	lom	ldf	ldf	ldf	mdl, mom, mdf	fom	fdl
ss	fdl	fom	mdl, mom, mdf, lom, ldf	ldf	ldf	ldf	ldf	mdl, mom, mdf, lom, ldf	fom	fdl
fdl	fdl	fdl	fdl	fdl	fdl, fom, mdl, mom, mdf, lom, ldf	ldf	ldf	fdl	fdl	fdl

Table A.6: fdl to **fs** – (continued).

	fdl	**fom**	**mdl**	**mom**	**mdf**	**lom**	**ldf**	**fl**	**fm**	**fs**
fom	fdl	fdl	fdl	fom	mdl, mom, mdf, lom, ldf	ldf	ldf	fdl	fdl	fdl
mdl	fdl	fdl	fdl, fom, mdl	mdl	mdl, mom, mdf, lom, ldf	ldf	ldf	fdl, fom, mdl	fdl	fdl
mom	fdl	fdl	fdl, fom, mdl	mom	mdf, lom, ldf	ldf	ldf	fdl, fom, mdl	fdl	fdl
mdf	fdl	fdl	fdl, fom, mdl, mom, mdf	mdf	mdf, lom, ldf	ldf	ldf	fdl, fom, mdl, mom, mdf	fdl	fdl
lom	fdl	fdl	fdl, fom, mdl, mom, mdf	lom	ldf	ldf	ldf	fdl, fom, mdl, mom, mdf	fdl	fdl
ldf	fdl	fdl	fdl, fom, mdl, mom, mdf, lom, ldf	ldf	ldf	ldf	ldf	fdl, fom, mdl, mom, mdf, lom, ldf	fdl	fdl
fl	fdl	fdl	fdl, fom, mdl	mdl	mdl, mom, mdf	lom	ldf	fl, fm, fs	fs	fs
fm	fdl	fdl	fdl	fom	mdl, mom, mdf	lom	ldf	fs	fs	fs

Table A.6: fdl to **fs** – (continued).

	fdl	**fom**	**mdl**	**mom**	**mdf**	**lom**	**ldf**	**fl**	**fm**	**fs**
fs	fdl	fdl	fdl	fdl	fdl, fom, mdl, mom, mdf	lom	ldf	fs	fs	fs
lloi	fdl, fs, sloi	fdl, fs, sloi	fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	mdl, fl, lloi	mdl, mom, mdf, fl, lloi	mdf	mdf, lom, ldf	lloi, mloi, sloi	sloi	sloi
lmoi	fdl, fs, sloi	fdl, fs, sloi	fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	mdl, fl, lloi	mdl, fl, lloi	mom	mdf, lom, ldf	lmoi, mmoi, smoi	smoi	smoi
lsoi	fdl, fs, sloi, smoi, ssoi	fdl, fs, sloi, smoi, ssoi	fdl, fom, mdl, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	mdl, fl, lloi, lmoi, lsoi	mdl, fl, lloi, lmoi, lsoi	mdl, fl, lloi, lmoi, lsoi	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	lsoi, msoi, ssoi	ssoi	ssoi

Table A.6: fdl to **fs** – (continued).

	fdl	**fom**	**mdl**	**mom**	**mdf**	**lom**	**ldf**	**fl**	**fm**	**fs**
mloi	fdl, fs, sloi	fdl, fs, sloi	fdl, fs, sloi	fom, fm, mloi	mdl, mom, mdf, fl, lloi	mdf	mdf, lom, ldf	sloi	sloi	sloi
mmoi	fdl, fs, sloi	fdl, fs, sloi	fdl, fs, sloi	fom, fm, mloi	mdl, fl, lloi	mom	mdf, lom, ldf	smoi	smoi	smoi
msoi	fdl, fs, sloi, smoi, ssoi	fdl, fs, sloi, smoi, ssoi	fdl, fs, sloi, smoi, ssoi	fom, fm, mloi, mmoi, msoi	mdl, fl, lloi, lmoi, lsoi	mdl, fl, lloi, lmoi, lsoi	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	ssoi	ssoi	ssoi
sloi	fdl, fs, sloi	fdl, fs, sloi	fdl, fs, sloi	fdl, fs, sloi	fdl, fom, mdl, mom, mdf, fl, fm, fs, lloi, mloi, sloi	mdf	mdf, lom, ldf	sloi	sloi	sloi
smoi	fdl, fs, sloi	fdl, fs, sloi	fdl, fs, sloi	fdl, fs, sloi	fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	mom	mdf, lom, ldf	smoi	smoi	smoi
ssoi	fdl, fs, sloi, smoi, ssoi	fdl, fs, sloi, smoi, ssoi	fdl, fs, sloi, smoi, ssoi	fdl, fs, sloi, smoi, ssoi	fdl, fom, mdl, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	mdl, fl, lloi, lmoi, lsoi	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	ssoi	ssoi	ssoi

Table A.6: fdl to **fs** – (continued).

	fdl	**fom**	**mdl**	**mom**	**mdf**	**lom**	**ldf**	**fl**	**fm**	**fs**
mi	fdl, fs, sloi, smoi, ssoi	fdl, fs, sloi, smoi, ssoi	fdl, fs, sloi, smoi, ssoi	fdl, fs, sloi, smoi, ssoi	fdl, fs, sloi, smoi, ssoi	fom, fm, mloi, mmoi, msoi	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	mi	mi	mi
>	fdl, fs, sloi, smoi, ssoi, mi, >	fdl, fs, sloi, smoi, ssoi, mi, >	fdl, fs, sloi, smoi, ssoi, mi, >	fdl, fs, sloi, smoi, ssoi, mi, >	fdl, fs, sloi, smoi, ssoi, mi, >	fdl, fs, sloi, smoi, ssoi, mi, >	fdl, fom, mdl, mom, mdf, lom, ldf, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >	>	>	>

A.3.5 ***is-LargeLargeOverlapped-by*** (lloi) to ***After*** (>)

Table A.7: *is-LargeLargeOverlapped-by* (**lloi**) to *After* (>).

	lloi	**lmoi**	**lsoi**	**mloi**	**mmoi**	**msoi**	**sloi**	**smoi**	**ssoi**	**mi**	>
<	<, m, sso, mso, lso, ss, ldf	<, m, sso, mso, lso, ss, ldf	<, m, sso, mso, lso, ss, ldf	<, m, sso, mso, lso, ss, ldf	<, m, sso, mso, lso, ss, ldf	<, m, sso, mso, lso, ss, ldf	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, fdl, fom, mdl, mom, mdf, lom, ldf	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, fdl, fom, mdl, mom, mdf, lom, ldf	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, fdl, fom, mdl, mom, mdf, lom, ldf	<, m, sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, fdl, fom, mdl, mom, mdf, lom, ldf	No Info

Table A.7: lloi to > – (continued).

	lloi	**lmoi**	**lsoi**	**mloi**	**mmoi**	**msoi**	**sloi**	**smoi**	**ssoi**	**mi**	>
m	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf	sso, mso, lso, ss, ldf	smo, mmo, lmo, sm, lom	smo, mmo, lmo, sm, lom	smo, mmo, lmo, sm, lom	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf	fsi, fmi, fli, =, fl, fm, fs	ldfi, lomi, mdfi, momi, mdli, fomi, fdli, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >

Table A.7: lloi to > – (continued).

	lloi	**lmoi**	**lsoi**	**mloi**	**mmoi**	**msoi**	**sloi**	**smoi**	**ssoi**	**mi**	>
sso	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, mdf, lom, ldf	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, sl, sm, ss, mdf, lom, ldf	sso, smo, slo, mso, mmo, mlo, lso, lmo, llo, fsi, fmi, fli, mdli, fomi, fdli, sl, sm, ss, mdf, lom, ldf	slo, mlo, llo, sl, mdf	slo, mlo, llo, sl, mdf	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli, sl, mdf	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf	slo, mlo, llo, fsi, fmi, fli, mdfi, momi, mdli, fomi, fdli, sli, =, sl, fdl, fom, mdl, mom, mdf, fl, fm, fs, lloi, mloi, sloi	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	ldfi, lomi, mdfi, momi, mdli, fomi, fdli, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >
smo	slo, mlo, llo, sl, mdf	fsi, fmi, fli	mdli, fomi, fdli	slo, mlo, llo, sl, mdf	fsi, fmi, fli	mdli, fomi, fdli	slo, mlo, llo, sl, fdl, fom, mdl, mom, mdf	fsi, fmi, fli, =, fl, fm, fs	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	ldfi, lomi, mdfi, momi, mdli, fomi, fdli, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >

Table A.7: lloi to > – (continued).

	lloi	**lmoi**	**lsoi**	**mloi**	**mmoi**	**msoi**	**sloi**	**smoi**	**ssoi**	**mi**	>
slo	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli, sl, mdf	mdli, fomi, fdli	mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdli, fomi, fdli, sl, mdf	mdli, fomi, fdli	mdli, fomi, fdli	slo, mlo, llo, fsi, fmi, fli, mdfi, momi, mdli, fomi, fdli, sli, =, sl, fdl, fom, mdl, mom, mdf, fl, fm, fs, lloi, mloi, sloi	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	ldfi, lomi, mdfi, momi, mdli, fomi, fdli, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >

Table A.7: lloi to > – (continued).

	lloi	**lmoi**	**lsoi**	**mloi**	**mmoi**	**msoi**	**sloi**	**smoi**	**ssoi**	**mi**	>
mso	lso, lmo, llo, sl, sm, ss, mdf, lom, ldf	lso, lmo, llo, sl, sm, ss, mdf, lom, ldf	lso, lmo, llo, fli, mdli, sl, sm, ss, mdf, lom, ldf	mom	mom	momi, =, mom	fdl, fom, mdl	fdl, fom, mdl	mdfi, sli, fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	lomi, smi, lmoi, mmoi, smoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >
mmo	llo, sl, mdf	fli	mdli	mom	=	momi	fdl, fom, mdl	fl, fm, fs	mdfi, sli, lloi, mloi, sloi	lomi, smi, lmoi, mmoi, smoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >
mlo	llo, fli, mdli, sl, mdf	mdli	mdli	momi, =, mom	momi	momi	mdfi, sli, fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	mdfi, sli, lloi, mloi, sloi	mdfi, sli, lloi, mloi, sloi	lomi, smi, lmoi, mmoi, smoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >

Table A.7: lloi to > – (continued).

	lloi	**lmoi**	**lsoi**	**mloi**	**mmoi**	**msoi**	**sloi**	**smoi**	**ssoi**	**mi**	>
lso	lso, lmo, llo, sl, sm, ss, mdl, mom, mdf, lom, ldf	lso, lmo, llo, sl, sm, ss, mdl, mom, mdf, lom, ldf	lso, lmo, llo, fli, ldfi, lomi, mdfi, momi, mdli, ssi, smi, sli, =, sl, sm, ss, mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	mdl	mdl	ldfi, lomi, mdfi, ssi, smi, sli, mdl, fl, lloi, lmoi, lsoi	fdl, fom, mdl	fdl, fom, mdl	ldfi, lomi, mdfi, ssi, smi, sli, fdl, fom, mdl, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >
lmo	llo, sl, mdl, mom, mdf	fli, =, fl	ldfi, lomi, mdfi, momi, mdli, ssi, smi, sli, lloi, lmoi, lsoi	mdl	fl	ldfi, lomi, mdfi, ssi, smi, sli, lloi, lmoi, lsoi	fdl, fom, mdl	fl, fm, fs	ldfi, lomi, mdfi, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >

Table A.7: lloi to > – (continued).

	lloi	**lmoi**	**lsoi**	**mloi**	**mmoi**	**msoi**	**sloi**	**smoi**	**ssoi**	**mi**	>
llo	llo, fli, mdfi, momi, mdli, sli, =, sl, mdl, mom, mdf, fl, lloi	mdfi, momi, mdli, sli, lloi	ldfi, lomi, mdfi, momi, mdli, ssi, smi, sli, lloi, lmoi, lsoi	mdfi, sli, mdl, fl, lloi	mdfi, sli, lloi	ldfi, lomi, mdfi, ssi, smi, sli, lloi, lmoi, lsoi	mdfi, sli, fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	mdfi, sli, lloi, mloi, sloi	ldfi, lomi, mdfi, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >
fsi	mdli, fomi, fdli	mdli, fomi, fdli	mdli, fomi, fdli	mdli, fomi, fdli	mdli, fomi, fdli	mdli, fomi, fdli	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	ldfi, lomi, mdfi, momi, mdli, fomi, fdli, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >

Table A.7: lloi to > – (continued).

	lloi	**lmoi**	**lsoi**	**mloi**	**mmoi**	**msoi**	**sloi**	**smoi**	**ssoi**	**mi**	>
fmi	mdli	mdli	mdli	momi	momi	momi	mdfi, sli, lloi, mloi, sloi	mdfi, sli, lloi, mloi, sloi	mdfi, sli, lloi, mloi, sloi	lomi, smi, lmoi, mmoi, smoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >
fli	mdfi, momi, mdli, sli, lloi	mdfi, momi, mdli, sli, lloi	ldfi, lomi, mdfi, momi, mdli, ssi, smi, sli, lloi, lmoi, lsoi	mdfi, sli, lloi	mdfi, sli, lloi	ldfi, lomi, mdfi, ssi, smi, sli, lloi, lmoi, lsoi	mdfi, sli, lloi, mloi, sloi	mdfi, sli, lloi, mloi, sloi	ldfi, lomi, mdfi, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >
ldfi	ldfi, ssi, lsoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >
lomi	ldfi, ssi, lsoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >

Table A.7: lloi to > – (continued).

	lloi	**lmoi**	**lsoi**	**mloi**	**mmoi**	**msoi**	**sloi**	**smoi**	**ssoi**	**mi**	>
mdfi	ldfi, lomi, mdfi, ssi, smi, sli, lloi, lmoi, lsoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi	ldfi, lomi, mdfi, ssi, smi, sli, lloi, lmoi, lsoi	ldfi, ssi, lsoi	ldfi, ssi, lsoi	ldfi, lomi, mdfi, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >
momi	mdfi, sli, lloi	lomi, smi, lmoi	ldfi, ssi, lsoi	mdfi, sli, lloi	lomi, smi, lmoi	ldfi, ssi, lsoi	mdfi, sli, lloi, mloi, sloi	lomi, smi, lmoi, mmoi, smoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >
mdli	mdfi, momi, mdli, sli, lloi	mdfi, momi, mdli, sli, lloi	ldfi, lomi, mdfi, momi, mdli, ssi, smi, sli, lloi, lmoi, lsoi	mdfi, sli, lloi	mdfi, sli, lloi	ldfi, lomi, mdfi, ssi, smi, sli, lloi, lmoi, lsoi	mdfi, sli, lloi, mloi, sloi	mdfi, sli, lloi, mloi, sloi	ldfi, lomi, mdfi, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >

Table A.7: lloi to > – (continued).

	lloi	**lmoi**	**lsoi**	**mloi**	**mmoi**	**msoi**	**sloi**	**smoi**	**ssoi**	**mi**	>
fomi	mdli	mdli	mdli	momi	momi	momi	mdfi, sli, lloi, mloi, sloi	mdfi, sli, lloi, mloi, sloi	mdfi, sli, lloi, mloi, sloi	lomi, smi, lmoi, mmoi, smoi	ldfi, ssi, lsoi, msoi, ssoi, mi, >
fdli	mdli, fomi, fdli	mdli, fomi, fdli	mdli, fomi, fdli	mdli, fomi, fdli	mdli, fomi, fdli	mdli, fomi, fdli	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	mdfi, momi, mdli, fomi, fdli, sli, lloi, mloi, sloi	ldfi, lomi, mdfi, momi, mdli, fomi, fdli, ssi, smi, sli, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >

Table A.7: lloi to > – (continued).

	lloi	**lmoi**	**lsoi**	**mloi**	**mmoi**	**msoi**	**sloi**	**smoi**	**ssoi**	**mi**	>
ssi	lsoi	lsoi	lsoi	msoi	msoi	msoi	ssoi	ssoi	ssoi	mi	>
smi	lsoi	lsoi	lsoi	msoi	msoi	msoi	ssoi	ssoi	ssoi	mi	>
sli	lloi, lmoi, lsoi	lsoi	lsoi	mloi, mmoi, msoi	msoi	msoi	sloi, smoi, ssoi	ssoi	ssoi	mi	>
=	lloi	lmoi	lsoi	mloi	mmoi	msoi	sloi	smoi	ssoi	mi	>
sl	mdl, mom, mdf, fl, lloi	lloi	lloi, lmoi, lsoi	fom, fm, mloi	mloi	mloi, mmoi, msoi	fdl, fs, sloi	sloi	sloi, smoi, ssoi	mi	>
sm	mdl, mom, mdf	fl	lloi, lmoi, lsoi	fom	fm	mloi, mmoi, msoi	fdl	fs	sloi, smoi, ssoi	mi	>
ss	mdl, mom, mdf, lom, ldf	mdl, mom, mdf, lom, ldf	mdl, mom, mdf, lom, ldf, fl, lloi, lmoi, lsoi	fom	fom	fom, fm, mloi, mmoi, msoi	fdl	fdl	fdl, fs, sloi, smoi, ssoi	mi	>

Table A.7: lloi to > – (continued).

	lloi	**lmoi**	**lsoi**	**mloi**	**mmoi**	**msoi**	**sloi**	**smoi**	**ssoi**	**mi**	>
fdl	fdl, fs, sloi, smoi, ssoi, mi, >	>	>	fdl, fs, sloi, smoi, ssoi, mi, >	>	>	fdl, fs, sloi, smoi, ssoi, mi, >	>	>	>	>
fom	fdl, fs, sloi, smoi, ssoi	mi	>	fdl, fs, sloi, smoi, ssoi	mi	>	fdl, fs, sloi, smoi, ssoi	mi	>	>	>
mdl	fdl, fom, mdl, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	lsoi, msoi, ssoi	lsoi, msoi, ssoi, mi, >	fdl, fs, sloi, smoi, ssoi	ssoi	ssoi, mi, >	fdl, fs, sloi, smoi, ssoi	ssoi	ssoi, mi, >	>	>
mom	fdl, fom, mdl, fl, fm, fs, lloi, mloi, sloi	lmoi, mmoi, smoi	lsoi, msoi, ssoi, mi, >	fdl, fs, sloi	smoi	ssoi, mi, >	fdl, fs, sloi	smoi	ssoi, mi, >	>	>

Table A.7: lloi to > – (continued).

	lloi	**lmoi**	**lsoi**	**mloi**	**mmoi**	**msoi**	**sloi**	**smoi**	**ssoi**	**mi**	>
mdf	fdl, fom, mdl, mom, mdf, fl, fm, fs, lloi, mloi, sloi	lloi, mloi, sloi	lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >	fdl, fs, sloi	sloi	sloi, smoi, ssoi, mi, >	fdl, fs, sloi	sloi	sloi, smoi, ssoi, mi, >	>	>
lom	fdl, fom, mdl, mom, mdf	fl, fm, fs	lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >	fdl	fs	sloi, smoi, ssoi, mi, >	fdl	fs	sloi, smoi, ssoi, mi, >	>	>
ldf	fdl, fom, mdl, mom, mdf, lom, ldf	fdl, fom, mdl, mom, mdf, lom, ldf	fdl, fom, mdl, mom, mdf, lom, ldf, fl, fm, fs, lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi, mi, >	fdl	fdl	fdl, fs, sloi, smoi, ssoi, mi, >	fdl	fdl	fdl, fs, sloi, smoi, ssoi, mi, >	>	>

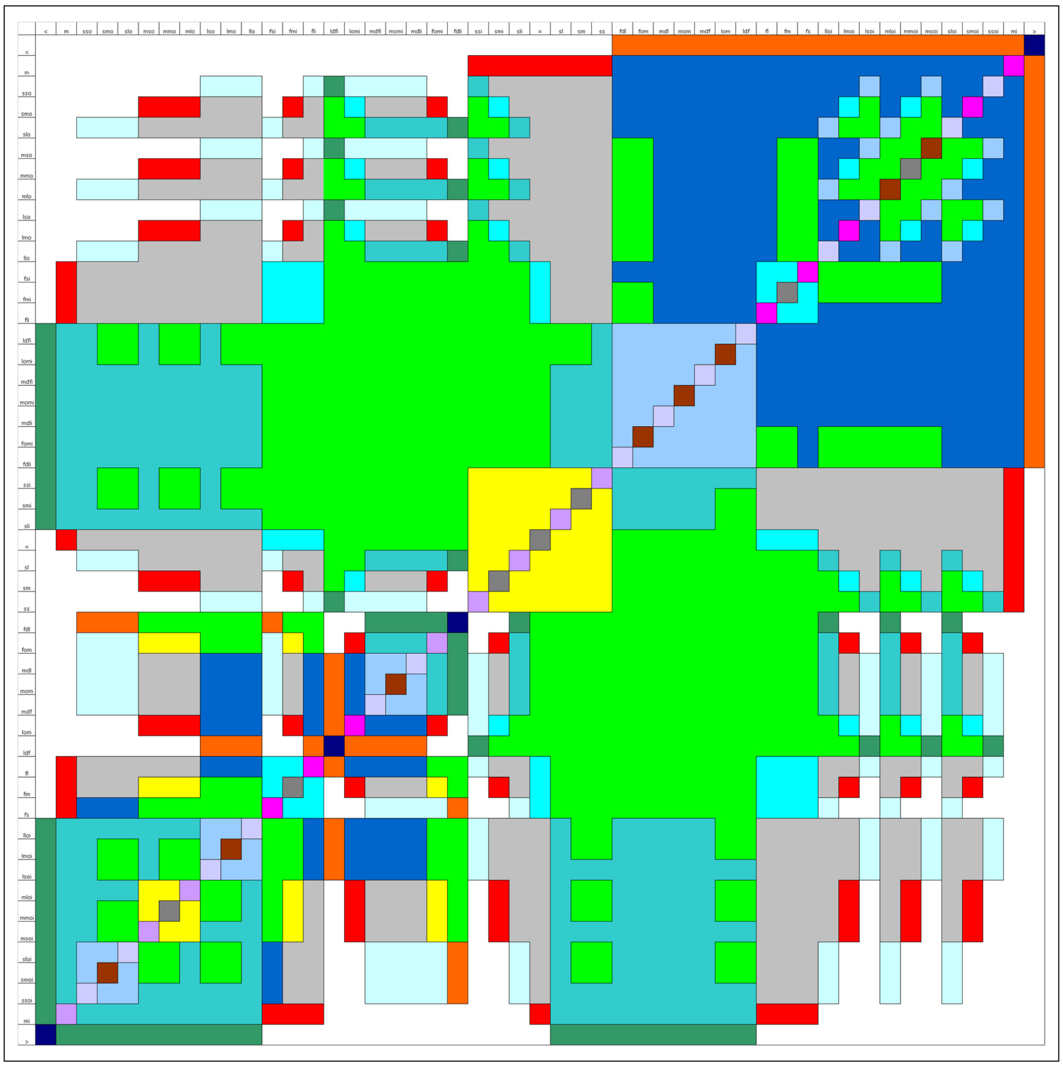
Table A.7: lloi to > – (continued).

	lloi	**lmoi**	**lsoi**	**mloi**	**mmoi**	**msoi**	**sloi**	**smoi**	**ssoi**	**mi**	**>**
fl	lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	lsoi, msoi, ssoi	lsoi, msoi, ssoi, mi, >	sloi, smoi, ssoi	ssoi	ssoi, mi, >	sloi, smoi, ssoi	ssoi	ssoi, mi, >	>	>
fm	sloi, smoi, ssoi	mi	>	sloi, smoi, ssoi	mi	>	sloi, smoi, ssoi	mi	>	>	>
fs	sloi, smoi, ssoi, mi, >	>	>	sloi, smoi, ssoi, mi, >	>	>	sloi, smoi, ssoi, mi, >	>	>	>	>
lloi	lloi, lmoi, lsoi, mloi, mmoi, msoi, sloi, smoi, ssoi	lsoi, msoi, ssoi	lsoi, msoi, ssoi, mi, >	sloi, smoi, ssoi	ssoi	ssoi, mi, >	sloi, smoi, ssoi	ssoi	ssoi, mi, >	>	>
lmoi	lsoi, msoi, ssoi	lsoi, msoi, ssoi	lsoi, msoi, ssoi, mi, >	ssoi	ssoi	ssoi, mi, >	ssoi	ssoi	ssoi, mi, >	>	>
lsoi	lsoi, msoi, ssoi	lsoi, msoi, ssoi	lsoi, msoi, ssoi, mi, >	ssoi	ssoi	ssoi, mi, >	ssoi	ssoi	ssoi, mi, >	>	>

Table A.7: lloi to > – (continued).

	lloi	**lmoi**	**lsoi**	**mloi**	**mmoi**	**msoi**	**sloi**	**smoi**	**ssoi**	**mi**	>
mloi	sloi, smoi, ssoi	mi	>	sloi, smoi, ssoi	mi	>	sloi, smoi, ssoi	mi	>	>	>
mmoi	ssoi	mi	>	ssoi	mi	>	ssoi	mi	>	>	>
msoi	ssoi	mi	>	ssoi	mi	>	ssoi	mi	>	>	>
sloi	sloi, smoi, ssoi, mi, >	>	>	sloi, smoi, ssoi, mi, >	>	>	sloi, smoi, ssoi, mi, >	>	>	>	>
smoi	ssoi, mi, >	>	>	ssoi, mi, >	>	>	ssoi, mi, >	>	>	>	>
ssoi	ssoi, mi, >	>	>	ssoi, mi, >	>	>	ssoi, mi, >	>	>	>	>
mi	>	>	>	>	>	>	>	>	>	>	>
>	>	>	>	>	>	>	>	>	>	>	>

A.3.6 Symmetric Verification Table



⬜	🟨	🟧	🟦	🟥	🟩	🟪	▨	🟪
<, >	si, s	<mods, >mioidisi	ods, oidisi	m, mi	di=f	di, d	=ooiddi ssiffi	s=si
🟥	⬜	⬛	🟦	🟦	🟦	🟩	🟦	⬛
fi=f	o, oi	=	<mo, >mioi	fi	odifi, oidf	<modifi, >mioidf	oddisfi, oididsif	No Info

Figure A.1: The 49 × 49 VLMI transitivity table[1]using the same colour for each inverse relationship to highlight the symmetry.

[1]The relationship order of the rows and columns is as follows: ⟨ <, m, [sso,smo,slo,mso,mmo,mlo,lso,lmo,llo], [fsi,fmi,fli], [ldfi,lomi,mdfi,momi,mdli,fomi,fdli], [ssi,smi,sli], =, [sl,sm,ss], [fdl,fom,mdl,mom,mdf,lom,ldf], [fl,fm,fs], [lloi,lmoi,lsoi,mloi,mmoi,msoi,sloi,smoi,ssoi], mi, >⟩

Appendix B

Software Application

This appendix presents the **INT**eracting **E**pisode **M**iner with **T**iming **M**arks ($INTEM_{TM}$) application that has been developed to facilitate algorithm use. The application is available for use and can be accessed at –

`http://www.infoeng.flinders.edu.au/research/techreps/SIE05004.zip` (Mooney, 2005).

Currently the application comprises two distinct modules:

Sequence Mining – this module encompasses both episode and interaction mining, refer to Chapter 5 for a detailed explanation and Section C.1 for algorithms.

Transitive Relationship Discovery – the module allows for variable input and output (Allen, Freksa, Midpoint or Mixed selections) for discernment of any transitive relationships, refer to Chapter 7 for a discussion.

A combination of both modules is available as an integrated implementation and the **Transitive Relationship Discovery** module is also available as a standalone Java™ Applet. The descriptions and screenshots that follow are from the integrated implementation.

B.1 Sequence Mining

This module deals with the Sequence Mining aspect of algorithmic development. It encompasses both the episodic mining – with and without timing marks – and the interaction discovery process.

B.1.1 Sequence Mining Interface

The sequence mining interface has been developed to facilitate algorithm use, and to view the results in a way that enables the user to more easily select those episodes and interactions that are of most interest. Furthermore a common problem identified with data mining routines is that the number of results produced can be large and difficult to interpret and hence methods for constraining the output have been implemented. In accordance with this position the user interface for $INTEM_{TM}$ enables the user to set all currently implemented constraints for both discovery of episodes and interactions.

The results of the mining run (frequent episodes) and the discovered interactions are then able to be viewed in both text format and as a directed graph. The directed graph not only allows the user to view the entire episode, but also shows the points at which interactions take place. This feature is most useful when the same sub-episode occurs at different points within the discovered frequent episode.

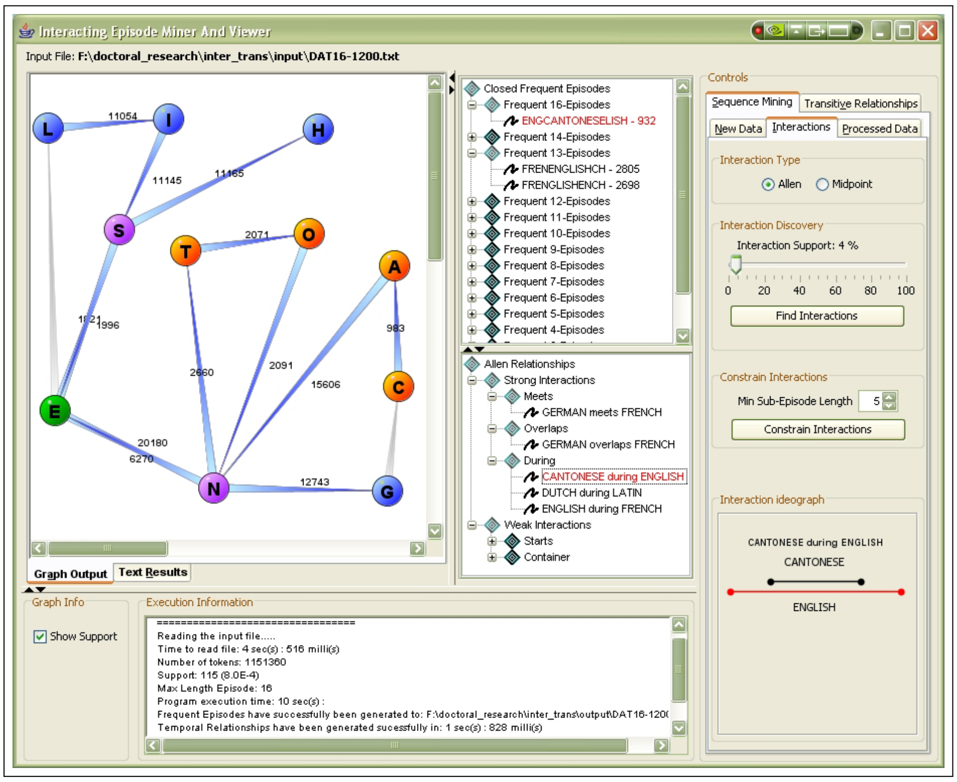
The interface of the $INTEM_{TM}$ application is comprised of four main areas (refer to Figure B.1):

1. The left panel (Viewing Pane), contains a tab pane that enables the user to switch between viewing the results in a text format or as a directed graph (see Section B.1.2).

2. The centre panel (Tree Pane) houses two tree structures the purpose of which are to enable selection of either a frequent episode or an interaction. The Viewing Pane will then display whichever selection that has been made (see Section B.1.2).

3. The right panel (Control Pane) has two main tabs (i.) Sequence Mining that has a further three options – New Data, Interactions and Processed Data, and (ii.) Transitive Relationships. These are discussed in Section B.1.3 and Section B.2.1 respectively.

4. The bottom panel (Execution Pane) contains an area for displaying program execution information and for toggling the support values displayed on the Viewing Pane (see Section B.1.4).



Input File: F:\doctoral_research\inter_trans\input\DAT16-1200.txt

Show Support [✓]

```
Closed Frequent Episodes
  Frequent 16-Episodes
    ENGCANTONESELISH - 932
  Frequent 14-Episodes
  Frequent 13-Episodes
    FRENENGLISHCH - 2805
    FRENGLISHENCH - 2698
  Frequent 12-Episodes
  Frequent 11-Episodes
  Frequent 10-Episodes
  Frequent 9-Episodes
  Frequent 8-Episodes
  Frequent 7-Episodes
  Frequent 6-Episodes
  Frequent 5-Episodes
  Frequent 4-Episodes
```

```
Allen Relationships
  Strong Interactions
    Meets
      GERMAN meets FRENCH
    Overlaps
      GERMAN overlaps FRENCH
    During
      CANTONESE during ENGLISH
      DUTCH during LATIN
      ENGLISH during FRENCH
  Weak Interactions
    Starts
    Container
```

Interaction Type: [✓ Allen] Midpoint

Interaction Discovery: Interaction Support: 4 %

Constrain Interactions: Min Sub-Episode Length 5

```
==================================
Reading the input file.....
Time to read file: 4 sec(s) : 516 milli(s)
Number of tokens: 1151360
Support: 115 (8.0E-4)
Max Length Episode: 16
Program execution time: 10 sec(s) :
Frequent Episodes have successfully been generated to: F:\doctoral_research\inter_trans\output\DAT16-120
Temporal Relationships have been generated sucessfully in: 1 sec(s) : 828 milli(s)
```

Graphic Output Legend

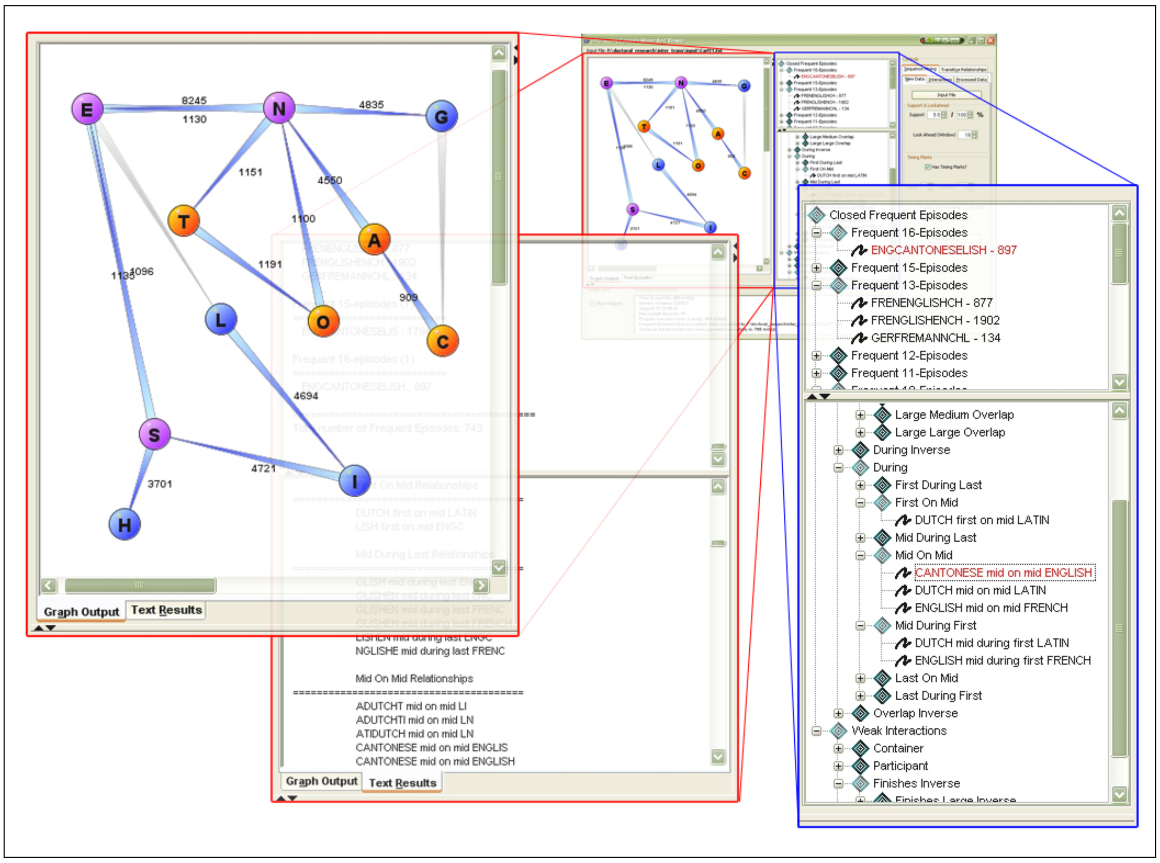
Node	**Description**
🟢	Root (green) Node for the interaction (E)
🔵	Enclosing (blue) sub-episode (G, L, I, H)
🟠	Enclosed (orange) sub-episode (C, A, T, O)
🟣	Shared (purple) node of both enclosing and enclosed sub-episodes (N, S)
Edge	
2091 ◀	Relationship (blue gradient) between two Nodes and the count. Direction is indicated by the shape of the wedge (possibly two-way (E $\leftrightarrow$ S)).
◁	The point(s) (gray gradient) at which the enclosed sub-episode begins/ends within the enclosing sub-episode

Figure B.1: Screenshot of the $INTEM_{TM}$ application. The *strong* interaction **CANTONESE** *during* **ENGLISH** – discovered in the DAT16-1200 file – is displayed.

B.1.2 Viewing the Output

Visualisation of data mining routines is not uncommon, but compared to the abundance of visualisations available for Association mining (Rogowitz and Treinish, 1993; Hofman, Siebes and Wilhelm, 2000; Rainsford and Roddick, 2000; Ong, Ong, Ng and Lim, 2002), (see also (Ceglar, Roddick, Mooney and Calder, 2003) for a survey of techniques across all areas), the area of sequence mining has very few, and those that are available are mainly in the areas of text mining (Wong, Cowley, Foote, Jurrus and Thomas, 2000), web mining (Cadez, Heckerman, Meek, Smyth and White, 2000; Demiriz and Zaki, 2002), or DNA (Hoffman, Grinstein, Marx, Grosse and Stanley, 1997).

The aim of the $INTEM_{TM}$ application is therefore to introduce a method whereby the results of sequence mining may be effectively displayed in an intuitive manner. As such the visualisation panes offer several alternative representations of the results from sequence mining runs. This includes not only the discovered frequent closed episodes but also any *strong* or *weak* interactions that those frequent closed episodes may contain. Figure B.2 shows the directed graph, text output and the trees from which selections for viewing can be made.



```
NGLISHE mid during last FRENC

Mid On Mid Relationships
==========================================
ADUTCHT mid on mid LI
ADUTCHTI mid on mid LN
ATIDUTCH mid on mid LN
CANTONESE mid on mid ENGLIS
CANTONESE mid on mid ENGLISH
```

```
Closed Frequent Episodes
  Frequent 16-Episodes
    ENGCANTONESELISH - 897
  Frequent 15-Episodes
  Frequent 13-Episodes
    FRENENGLISHCH - 877
    FRENGLISHENCH - 1902
    GERFREMANNCHL - 134
  Frequent 12-Episodes
  Frequent 11-Episodes
```

```
      Large Medium Overlap
      Large Large Overlap
    During Inverse
    During
      First During Last
      First On Mid
        DUTCH first on mid LATIN
      Mid During Last
      Mid On Mid
        CANTONESE mid on mid ENGLISH
        DUTCH mid on mid LATIN
        ENGLISH mid on mid FRENCH
      Mid During First
        DUTCH mid during first LATIN
        ENGLISH mid during first FRENCH
      Last On Mid
      Last During First
    Overlap Inverse
  Weak Interactions
    Container
    Participant
    Finishes Inverse
```

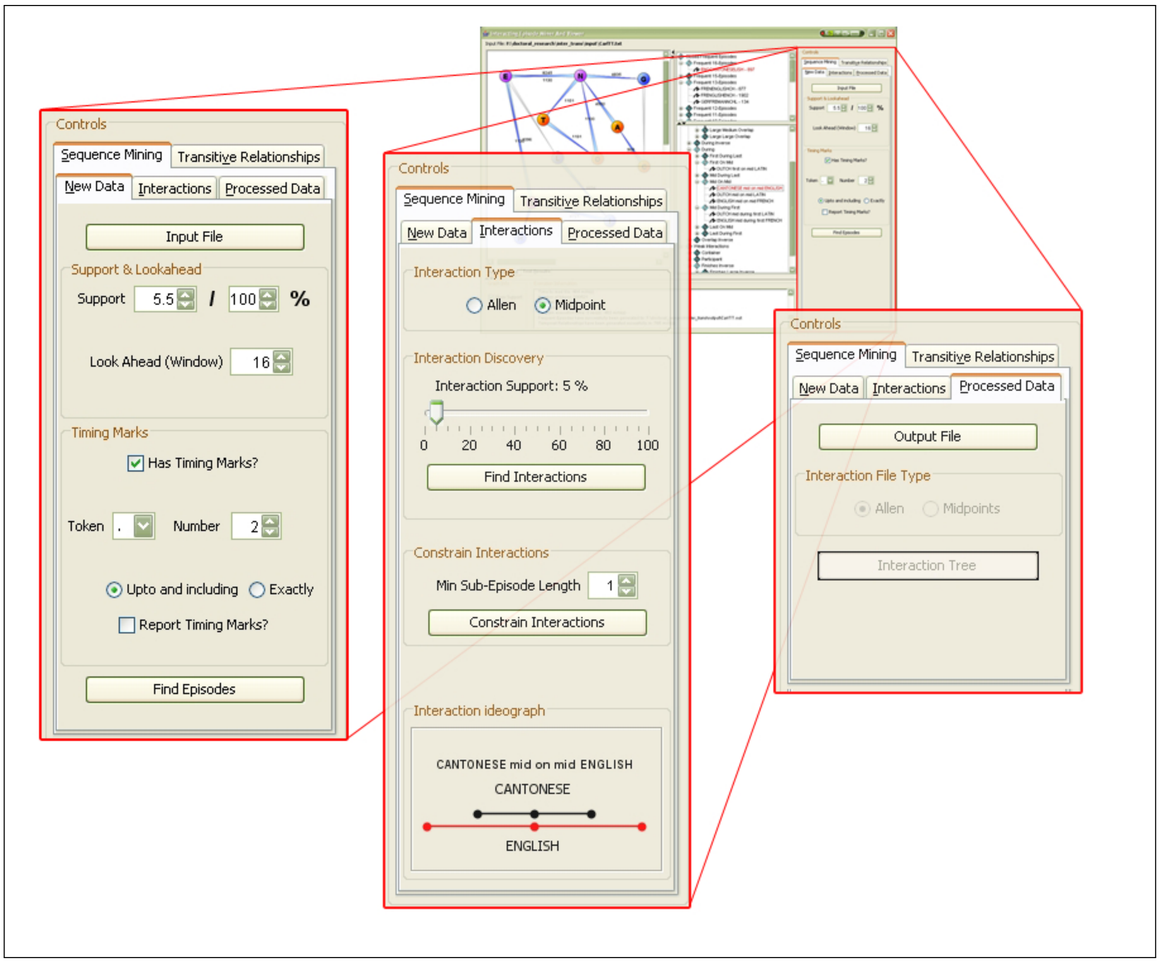
Figure B.2: Screenshot of the $INTEM_{TM}$ application showing the various components of the Viewing Pane and Tree Pane.

The upper tree contains the Closed Frequent Episodes organised by length and the lower tree the interactions. The lower tree containing the interactions is divided into two sections for each of *strong* and *weak* interactions and these are further divided into the temporal relationships from which they are comprised.

The directed graph has colour coded nodes (see Figure B.1 for the legend) and each node is able to be moved, using the mouse, so that a particular users needs can be accommodated. The support values can also be toggled on or off as required.

The text output tab of the Viewing Pane contains identical information to the tree structures and is included for those who may require a paper based copy of the results.

B.1.3 Controls for Mining



Support 5.5 / 100 %
Look Ahead (Window) 16
Timing Marks
[✓] Has Timing Marks?
Token . Number 2
[✓ Upto and including] Exactly
[ ] Report Timing Marks?
Allen [✓ Midpoint]
Interaction Support: 5 %
0 20 40 60 80 100
Min Sub-Episode Length 1

Figure B.3: Screenshot of the $INTEM_{TM}$ application showing the available options on the Sequence Mining tab of the Control Pane.

This section contains a description of the Control Panel Tabs that belong to the Sequence Mining task (refer to Figure B.3 for screenshots of the actual configuration):

- ***New Data Tab*** – Settings on this tab relate to new mining tasks.
  - **Input File** – A data source selected from a specific location on the file system.
  - **Support and Lookahead**
    - *Support* – This is used in conjunction with the maximum number of windows (*max_win*) to determine the minimum episode frequency (*min_freq*).
    - *Look Ahead (Window) lookahead* – This is the maximum length episode to be mined.

- **Timing Marks**
  - *Has Timing Marks* – Indicates whether the data to be mined has timing marks. If not then the following four items are not available.
  - *Timing Mark Token* – This list contains those unique tokens that are present in the data file. The list is compiled during the initial pass of the data when determining the single token episodes.
  - *Number of Tokens* – This number determines how many timing tokens can be present in any given episode (see Section 6.1, page 100 for a complete description).
  - *Inclusiveness* – A selection can be made whether to mine using exactly the number of timing marks, or up to and including that number.
  - *Reportable* – Indicates whether to include the timing marks in the output.
- **Find Episodes** – Perform the task of mining for frequent episodes using the selected constraints.

❒ ***Interactions Tab*** – Settings on this tab relate to interaction discovery.

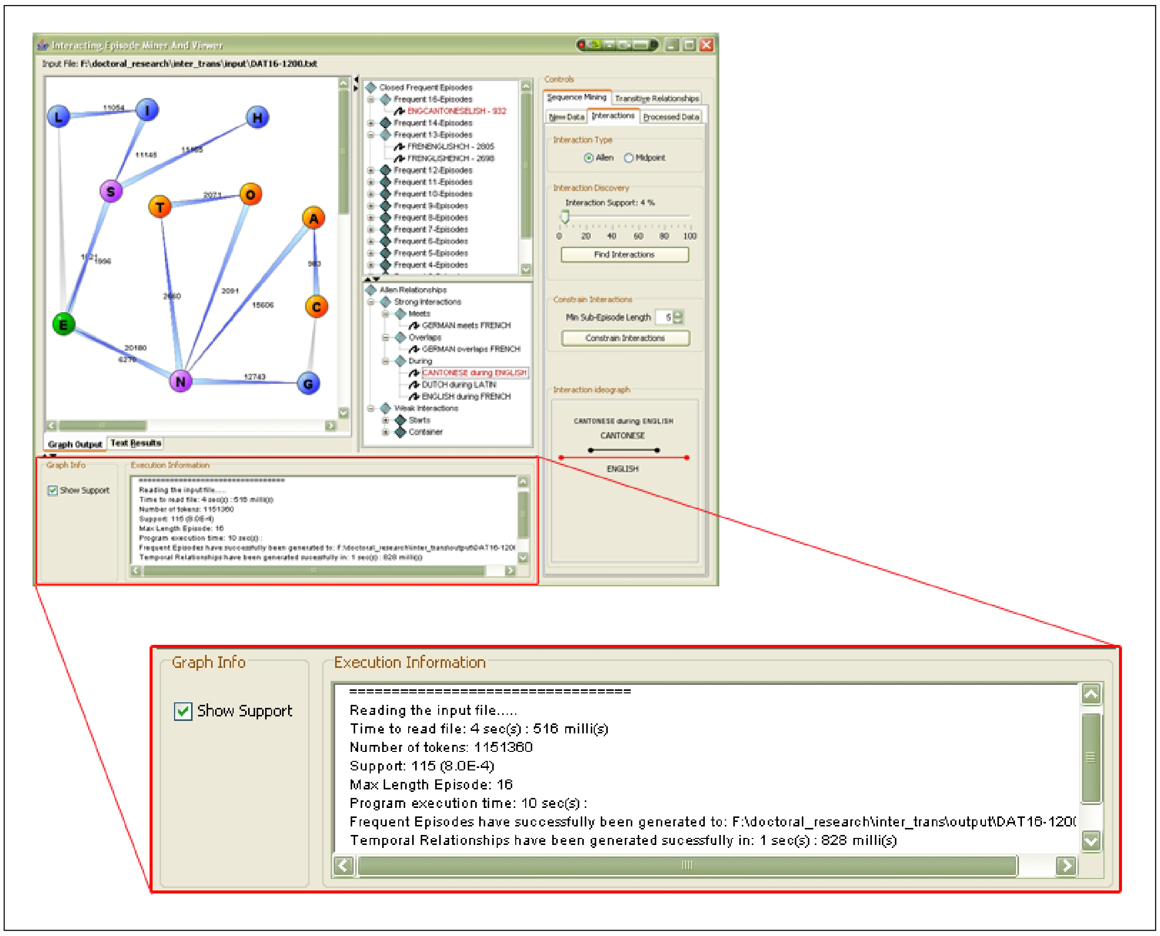
- **Interaction Discovery**
  - *Interaction Type* – Select the type of interactions to be discovered. Available selections are Allen or Midpoint.
  - *Interaction support* (*min_interaction_supp*) – This value is used in conjunction with the interaction count (*min_interaction_count*) to determine reportability (see Definition 5.14, page 86).
  - *Find Interactions* – Perform the task of interaction discovery using the selected constraints.
- **Constrain Interactions**
  - *Minimum Sub-episode length* (*min_interaction_length*) – The value selected here will constrain the output to interactions where both sub-episodes are of a length greater than or equal to the selected value. Only values up to and including the maximum length sub-episode are available for selection.
  - *Constrain Interactions* – Perform the operation of constraining the sub-episode length using the selected value.
- **Interaction Ideograph** – This graphic depicts the currently selected interaction.

❒ ***Processed Data Tab*** – Settings on this tab relate to data files that have been saved from previous mining runs. The flexibility offered by this option enables the visualisation to be decoupled from algorithms that produce the episodes and interactions, therefore output files generated from different mining algorithms may be used as long as they conform to the required input file specifications.

- **Output File** – An output data file (**.out*) that is selected for read only viewing.
- **Interaction File Type** – Indicates the type of output file that has been selected. The type will be determined by the settings that were in effect, either Allen or Midpoint interactions, when the file was saved.
- **Interaction Tree** – Generate the episode and interaction trees as well as the graphical and text outputs associated with the selected data file.

B.1.4 Execution Information

The Execution Pane (see Figure B.4) contains information relevant to the running of the algorithms for both episode discovery and interaction discovery. The output comprises information on the times for each process involved in both of these activities. The area also contains the mechanism for toggling the support values on the Viewing Pane.



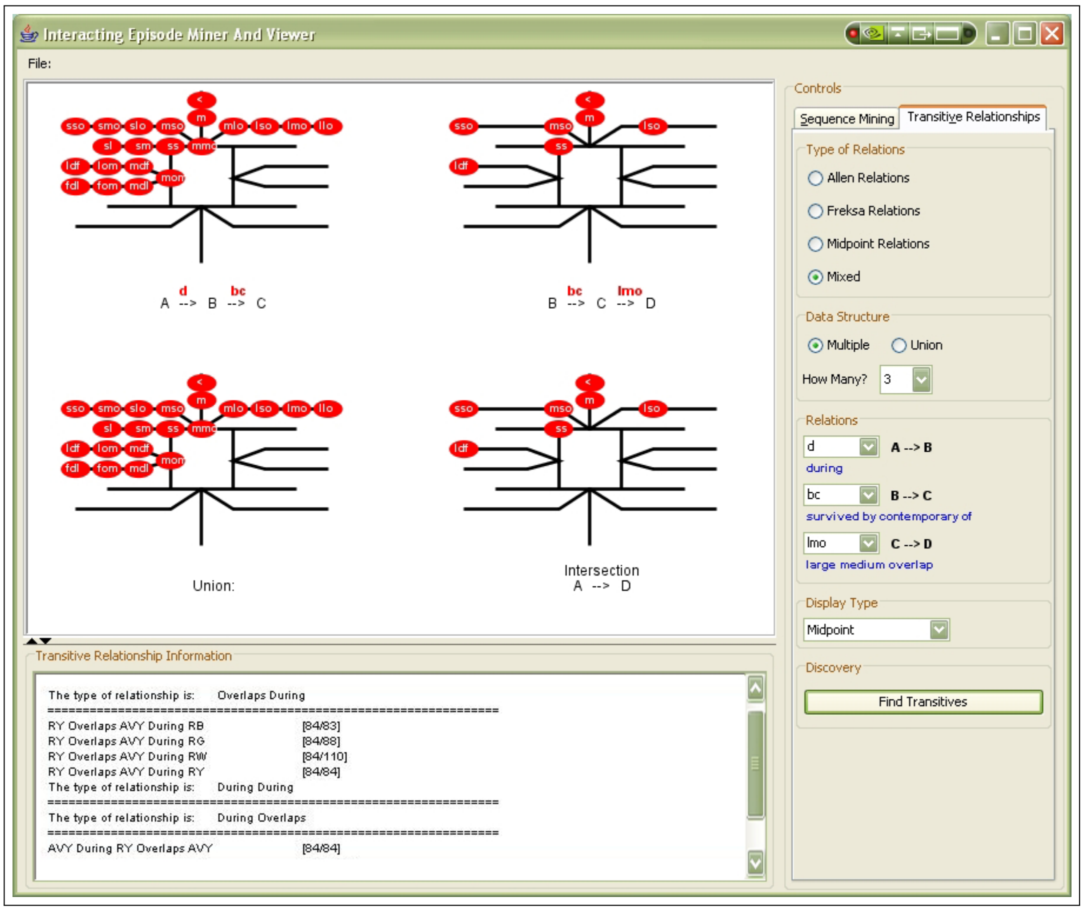
```
==================================
Reading the input file.....
Time to read file: 4 sec(s) : 516 milli(s)
Number of tokens: 1151360
Support: 115 (8.0E-4)
Max Length Episode: 16
Program execution time: 10 sec(s) :
Frequent Episodes have successfully been generated to: F:\doctoral_research\inter_trans\output\DAT16-120(
Temporal Relationships have been generated sucessfully in: 1 sec(s) : 828 milli(s)
```

Figure B.4: Screenshot of the $INTEM_{TM}$ application showing execution information for a mining run and interaction discovery.

B.2 Transitive Relationship Discovery

This module was developed to facilitate the use of the developed Variable-Length Midpoint Intervals (VLMI) (Chapter 4), and those of Allen (1983) and Freksa (1992), and the resultant transitivity tables (see Figure B.5). This tool is available as a both a standalone Java™ Applet and as a an integrated part of the $INTEM_{TM}$ application.

B.2.1 Transitive Relationships Interface



Type of Relations: Allen Relations / Freksa Relations / Midpoint Relations / [✓ Mixed]

Data Structure: [✓ Multiple] Union

How Many? 3

Relations:
d **A --> B** during
bc **B --> C** survived by contemporary of
lmo **C --> D** large medium overlap

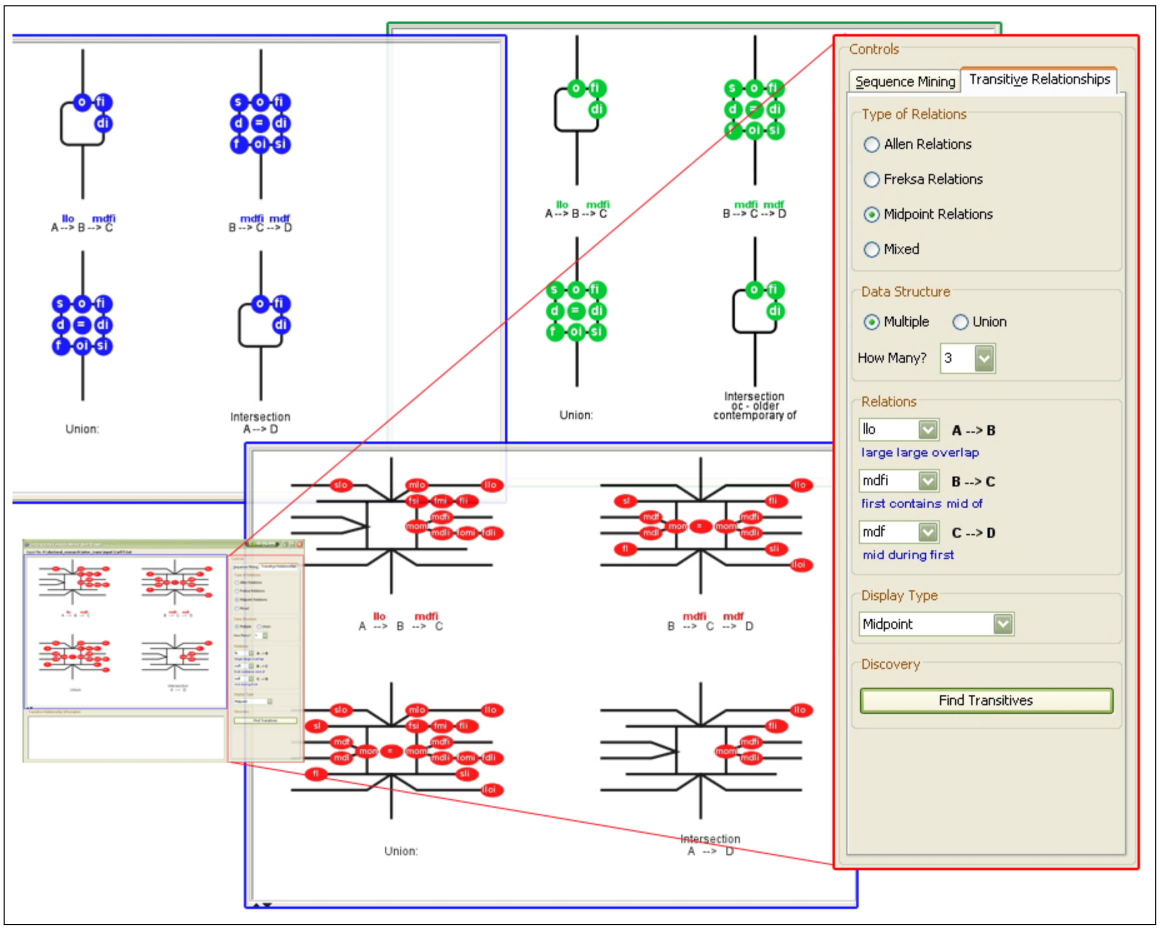
Display Type: Midpoint

```
The type of relationship is:     Overlaps During
=================================================================
RY Overlaps AVY During RB                      [84/83]
RY Overlaps AVY During RG                      [84/88]
RY Overlaps AVY During RW                      [84/110]
RY Overlaps AVY During RY                      [84/84]
The type of relationship is:     During During
=================================================================
The type of relationship is:     During Overlaps
=================================================================
AVY During RY Overlaps AVY                     [84/84]
```

Figure B.5: Screenshot of the transitive relationships interface showing the results when relationships from each of the available types of input are selected - Allen: *during* (**d**), Freksa: *survived by contemporary of* (**bc**), and Midpoint: *LargeMediumOverlap* (**lmo**).

The interface comprises three main areas (refer Figure B.5):

- ❒ The upper left panel graphical output display for the results.
- ❒ The lower left panel (not available in the stand-alone version) for the display of any transitive relationships that may have been discovered from the results of a sequence mining run.


Controls

Sequence Mining | Transitive Relationships

Type of Relations

( ) Allen Relations
( ) Freksa Relations
[✓ Midpoint Relations]
( ) Mixed

Data Structure

[✓ Multiple] ( ) Union

How Many? 3

Relations

llo A --> B
large large overlap

mdfi B --> C
first contains mid of

mdf C --> D
mid during first

Display Type

Midpoint

Discovery

Figure B.6: Screenshots of the transitive relationships interface showing all of the available output display types: top-left – Allen, top-centre – Freksa and bottom-centre – Midpoint. The Control Pane is shown on the right and the complete interface on the bottom-left.
All of the displays depict use of the *multiple* selection, with a value of three. This yields the configuration, for the screenshot: $A \xrightarrow{llo} B \xrightarrow{mdfi} C \xrightarrow{mdf} D$, which results in $A \xrightarrow{llo,fli,mdfi,momi,mdli} D$.

❒ The right panel (Control Pane) contains the controls for the selection of:

- **Type of relations** – a selection can be made from *Allen*, *Freksa*, *Midpoint*, or *Mixed*. The *Mixed* selection enables the relationships from all types to be used.
- **Data Structure** – how the configuration of the episodes is structured:
  - *Union* – This allows two paths to be explored. For example $A \rightarrow B \rightarrow D$ and $A \rightarrow C \rightarrow D$, which will yield the transitive relationship $A \rightarrow D$ resultant from the Union of the two paths.
  - *Multiple* – This is a 'chaining' of potentially related episodes. For example $A \rightarrow B \rightarrow C \rightarrow D$, which will yield the transitive relationship

$A \rightarrow D$ – the intersection of each resultant set.

- **Relations** – each available list will contain those relationships that coincided with the selected type. The number of selections will depend on the chosen Data Structure.
- **Display Type** – a selection can be made from Allen-style relationships, Freksa's conceptual neighbourhoods or more fine grained VLMI relationships. The available selections for output are dependent on which types of relations have been selected during the input and if two, or three, different types have been selected (one for each set of relations) the output options will defer to the more general in the following order: Midpoint → Allen → Freksa. For example if the relationships have been expressed using the Midpoint algebra then all possible outputs are available, if Allen relationships have been used then Allen and Freksa outputs are available, and so on. Figure B.6 shows all possible outputs from a given midpoint selection.
- **Discovery**
  - *Find Transitives* – Perform the task of discovering the transitive relationships using the selected constraints.

B.3 Experimental Results

This section outlines the experiments that have been run using the $INTEM_{TM}$ software with and without timing marks. All of the algorithms were implemented in the $Java^{TM}$ programming language and all experiments were conducted on a 2.6GHz AMD machine running $Microsoft^{®}Windows^{®}XP$ with 1Gb of RAM.

B.3.1 Mining without Timing Marks

This section outlines the experiments that have been run without timing marks under different support levels and a lookahead value of 60. Three of the four input files were synthetically produced ASCII text files ranging in size from 200Kb to 1.2Mb, the fourth was taken from the first 25,000 rows of the Human Genome. The set of tokens for the synthetic files was taken from the upper-case alphabetic characters, $T = \langle A \ldots Z, \#, / \rangle$, while the genome file had the five characters $T = \langle A, C, G, T, N \rangle$. Table B.1 summarises the nature of the files used.

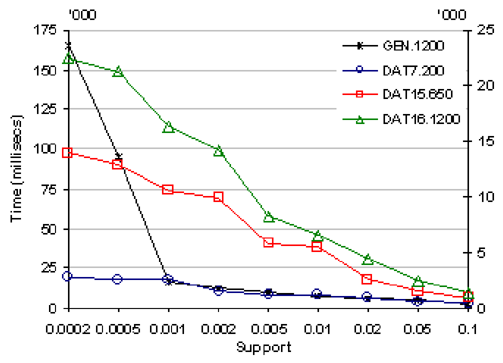
The smallest file (DAT7-200) was used for algorithmic development, since the composition, and therefore the expected results, was known. The remaining synthetic files and the Genome file (GEN-1200) were mined after algorithm completion. In common with many sequence mining applications and because of the differences between the disk and bus speeds of various platforms, the test algorithms were developed to be memory

Table B.1: Non-timing mark experimental file specifications.

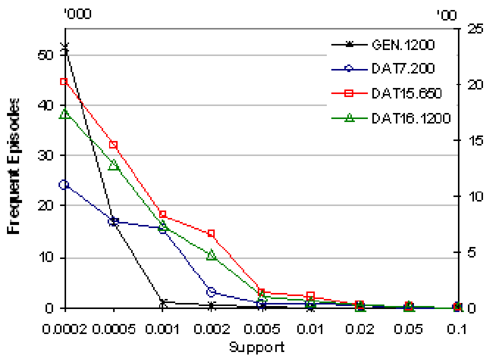
File Name	Size of Alphabet	Number of Tokens
DAT7-200	28	199,384
DAT15-650	28	650,918
DAT16-1200	28	1,151,360
GEN-1200	5	1,178,371

resident and thus the time provided can be more readily compared. An added time factor for reading the files should be included to obtain the total time[1]. Figure B.7(a) shows the actual processing times, excluding any I/O, for the mining of the episodes and the interactions concurrently. The times displayed in Figure B.7(a) are for the generation of the frequent episodes shown in Figure B.7(b).

The larger token set for the synthetic data files produced fewer frequent episodes, by a factor of 8 against the genome data (Figure B.7(b)), and as such the support metric that was used for reporting the frequent interactions (see §Section 5.3.2 for details) was more appropriate. Thus, in order to assess the algorithms using the genome data a support level was chosen where an excessive number of frequent episodes was not going to be a major contributing factor.



(a) Processing time as a function of support. GEN-1200 uses the primary y-axis (left) for its values.

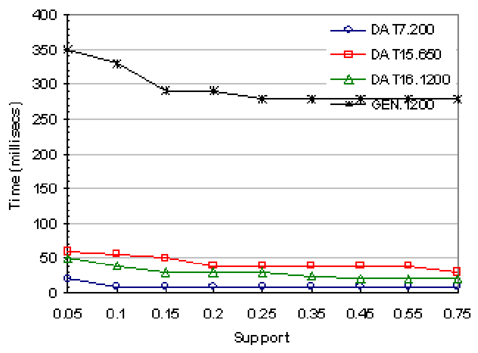


(b) Number of frequent episodes as a function of support.
GEN-1200 uses the primary y-axis (left) for its values.

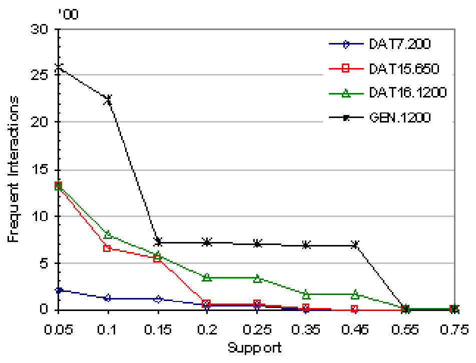
Figure B.7: Processing time and frequent episode production using a lookahead distance of 60 and varying levels of support.

Since the interactions are able to be mined independently of, as well as concurrently with, episode production actual processing times are able to be reported for the algorithms that have been developed. The times displayed in Figure B.8(a) are for the

[1]In the conducted experiments, for the Genome Data, this was approximately 4 seconds and proportionally quicker with the other datasets. Note that since the algorithms run in-memory, reading the input file only has to occur on the first run, after which different lookahead distances and support levels can be supplied without incurring this initial overhead.



(a) Processing time as a function of support.



(b) Number of frequent interactions as a function of support.

Figure B.8: Execution time and frequent interaction production using frequent episodes mined at a support of 0.0005 with varying levels of interaction support.

generation of the frequent interactions shown in Figure B.8(b). All of the tests were run using a minimum sub-episode length of one, which can be viewed as the worst case scenario (most frequent interactions produced), and although the results of constraining the minimum sub-episode length are not shown here, it is apparent that this would reduce both the processing time and the number of frequent interactions produced.

B.3.2 Mining with Timing Marks

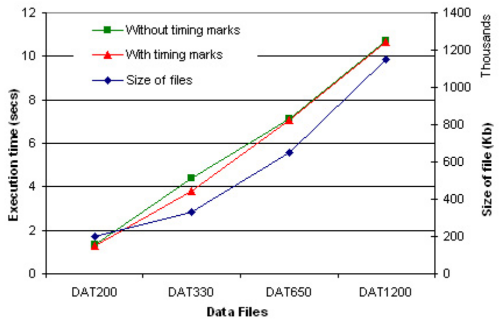
This section outlines the experiments that have been run with timing marks. All tests were conducted using a very small support level (0.005), a *lookahead* value of 20 and, when including timing marks, a *timing mark count* (tmc) of 2 with the reporting option set to exclude the marks from the output.

All of the test files comprised the alphabet $T = \langle A \ldots Z, \#, / \rangle$ and all had the added token $\langle . \rangle$ to enable the testing of the timing mark constraint. Table B.2 summarises the nature of the files used.

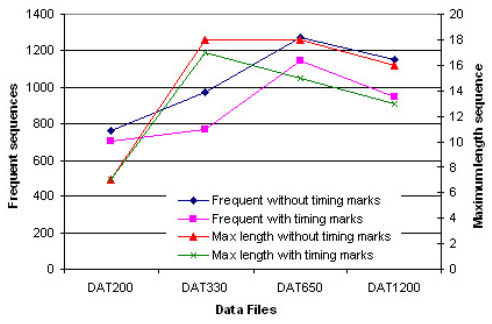
Table B.2: Timing mark experimental file specifications.

File Name	Size of Alphabet	Number of Tokens
DAT200	29	199,384
DAT330	29	330,522
DAT650	29	650,918
DAT1200	29	1,151,360

The results show that there is no overhead incurred when using the timing mark option, (see Figure B.9(a)). Indeed, since the constraint is implemented deeper in the process there is a slight speed increase when looking for sequences containing timing marks. The reason for the speed up can also seen, (see Figure B.9(b)), by the fact there are fewer sequences discovered with the timing mark option selected and in the majority of cases the maximum length of the discovered sequences is smaller.



(a) Execution times with and without timing marks for the test files.



(b) Number of frequent sequences and maximum length sequences with and without timing marks.

Figure B.9: Processing time and frequent episode production with and without timing marks.

Appendix C

Algorithms

The purpose of this appendix is to serve as a repository for the algorithms that have been referred to in the body of this thesis. The algorithms described herein appear in chapter order and are prefaced with a brief description. The description has been included for the purpose of providing context if the algorithms are read without the chapter to which they belong.

C.1 Interacting Episodes

The algorithms described in this section are supported by the text in Chapter 5 and is organised in a similar way. As such it contains two sections; the first (Section C.1.1) dealing with the discovery of frequent episodes and the second (Section C.1.2) with the discovery of any interactions that they (frequent episodes) may contain.

C.1.1 Frequent Episodes

This algorithm is the starting point for the process of not only episode discovery, but also interaction discovery. It is called after the data has been read and all constraints have been set. The major constraints at this point are the length of the episodes to be discovered and the support at which they will be deemed frequent. It terminates when no more frequent episodes are discovered, and hence no candidates can be generated, or when the maximum length episode is reached. At this point a simple pruning algorithm is applied, to ensure only closed frequent episodes remain, and the results from this are then able to be used in the next phase of the process. If the requirement that interactions be discovered in parallel has been chosen then this process is carried out, using Algorithm C.2, as indicated at line 17 in Algorithm C.1.

Algorithm C.1 Find Frequent Closed Episodes.

Require: a *sequence* **S**, of *tokens* **t**, a *lookahead* l and a *minimum support* σ

Ensure: the collection $\mathcal{F}(S,\ l,\ \sigma)$ of frequent closed episodes F_{ce} and an interaction list, I_l.

```
 1: procedure Find Frequent Closed Episodes($\mathbf{S}, l, \sigma$)
 2:     find $C_1 := \{\, \alpha \in T \mid |\alpha| = 1 \,\}$;
 3:     i := 0 ; found := true
 4:     while ($i^{++} <\ l$ and $found$) do
 5:         for ($j := 0$; $j < |S| - i + 1$; $j^{++}$) do
 6:             $\alpha := S_j, \ldots, S_{i+j}$
 7:             $\delta := (|S| - |\alpha| + 1) \times \sigma$;
 8:             if (i > 1) then
 9:                 if ($\alpha_k \in \mathcal{F}_k \mid \alpha_k = \langle t_1 \ldots t_{i-1} \rangle$) then
10:                     add $\alpha$ to $C_j$
11:                 end if
12:             else
13:                 add $\alpha$ to $C_j$
14:             end if
15:         end for
16:         found := $\mathcal{F}_i \neq \emptyset$ where $\mathcal{F}_i := \{\, \forall\, \alpha \in C_j \mid frequency\ (\alpha, S, l) \geq \delta \,\}$
17:         /* call to Algorithm C.2 using ($\mathcal{F}_{1..i}$) as the parameter
18:            if parallel discovery is required */
19:     end while
20:     $I_l$ := createInteractionList($F_e$) /* replica of $F_e$, without zero length trees */
21:     $F_{ce}$ := getClosedSetEpisodes($F_e$)
22: end procedure
```

C.1.2 Frequent Interactions

The following algorithm, Algorithm C.2, acts a wrapper to the main algorithm for the detection of relationships, and after the return from the task of relationship discovery, initiates the removal of any duplicate inverse relationships and prunes according to any user defined constraints. The constraints include a count, or a sub-episode length.

Algorithm C.2 Find Relationships.

Require: a collection of frequent closed episodes, F_{ce}, and an interaction list, I_l and a relationship type, *rel_type*.

Ensure: the collection of frequent interactions F_r.

```
 1: procedure findRelationships($F_{ce}$)
 2:     for ($i := 0$; $i < F_{ce}.size()$; $i^{++}$) do
 3:         $TreeMap < String, Integer > beta := F_{ce}.get(i)$
 4:         int size := (beta.firstKey()).length()
 5:         for ($j := 0$; $j < size - 1$; $j^{++}$) do
 6:             $TreeMap < String, Integer > alpha = I_l.get(j)$
 7:             for all (String episode : beta.keySet()) do
 8:                 for all (String sub_episode : alpha.keySet()) do
 9:                     findAnyRelationships(episode,sub_episode,beta.get(episode),rel_type)
                       /* Algorithm C.3 */
10:                 end for
11:             end for
12:         end for
13:     end for
14:     removeDuplicateInverseRelationships(rel_type)
15:     pruneCandidateInteractions()
16: end procedure
```

The next algorithm, Algorithm C.3, locates the endpoints and midpoints of the frequent sub-episode passed as a parameter and those of the remainder. A determination is then made regarding the class of interaction to which they belong and they are then added to the appropriate candidate interaction set.

Algorithm C.3 Finds any relationships that exist between two events.

Require: an episode e_1 and an episode e_2, $|e_2| < |e_1|$

Ensure: any temporal relationship $\theta_r(e_2, e_1 - e_2) \vee \psi^{w[p]}(e_i, e_j)$ that exists is added to the candidate interaction list, CI_r, or return if none.

```
1: procedure FINDANYRELATIONSHIPS(String $e_1$, String $e_2$, int count)
2:     if ($e_2 \nsubseteq e_1$) then
3:         return
4:     else
5:         int[ ] positions := $e1_{start}, e1_{mid}, e1_{end}, e2_{start}, e2_{mid}, e2_{end}$
6:     end if
7:     determine $\theta_r(e_2, e_1 - e_2) \vee \psi^{w[p]}(e_i, e_j)$ and add it to $CI_r$ with count
   /* This is achieved by comparisons between the start, mid and endpoints of the
   two sub_episodes and a lookup for the relevant set of constraints that is produced.
   These could be Allen, Table 3.1, or Midpoint, Table 4.3, relationships */
8: end procedure
```

Algorithm C.4 prunes the candidate interaction list using the user defined interaction support and produces a list of interactions that is frequent with respect to this metric, and any constraint with respect to the sub-episode lengths.

Algorithm C.4 Prune Candidate Interactions.

Require: a list of candidate relationships, CI_r and a *min_interaction_supp*, φ

Ensure: a list of frequent relationships F_r , where $\theta_r(e_i, e_j) \vee \psi^{w[p]}(e_i, e_j) \in \mathcal{R} \mid \xi \geq \varphi \wedge |e_i|, |e_j| \geq$ *min_sub-episode_len*

```
1: procedure PRUNE CANDIDATE INTERACTIONS
2:     for ($i := 0$; $i < CI_r.size()$; $i^{++}$) do
3:         $\theta_r := CI_r(i)$
4:         $\xi = \left[\frac{frequency\ \theta_{r_k}}{\sum_k frequency\ |\theta_{r_k}|}\right]$
5:         if ($\xi \geq \varphi$) then
6:             add $\theta_r$ to $F_r$
7:         end if
8:     end for
9:     return $F_r$
10: end procedure
```

C.2 Timing Marks

Algorithm C.5 is the main algorithm for using *timing marks*. The algorithm is called post episode discovery and therefore contains any timing tokens that may have been chosen – since the user has a choice of whether *timing marks* are used. The algorithm uses the *timing mark* heuristic for pruning the frequent episodes, which may include the options *exactly* or *up to and including* a certain number, and those that remain still contain the *timing mark*. This differs from algorithms that employ a 'don't care' type token , for example the work by Huang et al. (2004), since it may be the case that the interest is in both reporting the *timing mark* token, and its relevance to the reporting of the 'speed' of a frequent episode even if it (the *timing mark*) is not reported itself.

Algorithm C.5 Algorithm for using timing marks in episode discovery.

Require: A set of frequent sequences that are to be pruned in accordance with the *timing mark* heuristic.

Ensure: the collection of frequent sequences according to the *timing mark* constraints.

```
1: procedure pruneForTimingMarks(ArrayList aList)
2:     for ($i := 0$; $i < aList.size()$; $i^{++}$) do
3:         TreeMap tm := aList.get(i);
4:         TreeMap clTm := tm.clone();
5:         for all ($String\ cand : clTm.keySet()$) do
6:             int numMarks = countTimingMarks(cand);
7:             if ($exactly\_selected$) then
8:                 if ($numMarks \neq maxMarks$) then
9:                     tm.remove(cand);
10:                end if
11:            else
12:                if ($numMarks > maxMarks$) then
13:                    tm.remove(cand);
14:                end if
15:            end if
16:        end for
17:    end for
18: end procedure
```

Algorithm C.6 is the algorithm that removes the *timing marks* from the frequent episodes when they are not required to be reported. Since there is a need to visualise the frequent episodes in a tree structure (see Figure B.2) it is necessary once they have been removed that the episode be reassigned to the correct node.

Algorithm C.6 Removes the *timing marks* from the frequent episodes and reassigns them to the correct output containers.

Require: a list of frequent episodes that have been pruned for the required number of *timing marks*.

Ensure: the required frequent episodes without *timing marks*.

```
procedure REMOVEALLTIMINGMARKS(ArrayList aList)
    ArrayList modList := new ArrayList();
    for all (TreeMap tmap : aList) do
        TreeMap modTree := new TreeMap();
        for all (String cand : tmap.keySet()) do
            String newCand := removeTimingMarks(cand);
            if (!newCand.equals("")) then
                modTree.put(newCand, tmap.get(cand));
            end if
        end for
        modList.add(modTree);
    end for
    frequentList := reassignEpisodes(modList);
end procedure
```

Bibliography

Agrawal, R. C., Aggarwal, C. C. and Prasad, V. V. V. (1999), A tree projection algorithm for generation of frequent itemsets, *in* 'High Performance Data Mining Workshop', ACM Press, Puerto Rico.

Agrawal, R., Imielinski, T. and Swami, A. (1993), Mining association rules between sets of items in large databases, *in* P. Buneman and S. Jajodia, eds, 'Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data', ACM Press, Washington DC, USA, pp. 207–216.

Agrawal, R. and Srikant, R. (1994), Fast algorithms for mining association rules, *in* J. B. Bocca, M. Jarke and C. Zaniolo, eds, '20th International Conference on Very Large Data Bases, (VLDB)', Morgan Kaufmann, Santiago, Chile, pp. 487–499.

Agrawal, R. and Srikant, R. (1995), Mining sequential patterns, *in* P. S. Yu and A. S. P. Chen, eds, '11th International Conference on Data Engineering (ICDE'95)', IEEE Computer Society Press, Taipei, Taiwan, pp. 3–14.

Aho, A. (1990), Algorithms for finding patterns in strings, *in* J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. A: Algorithms and Complexity, Elsevier.

Ahonen, H., Heinonen, O., Klemettinen, M. and Verkamo, A. I. (1997), Applying data mining techniques in text analysis, Tech Report C-1997-23, University of Helsinki, Department of Computer Science.

Ahonen, H., Heinonen, O., Klemettinen, M. and Verkamo, A. I. (1998), Applying data mining techniques for descriptive phrase extraction in digital document collections, *in* 'Proceedings of the Advances in Digital Libraries Conference', IEEE Computer Society, p. 2.

Albert-Lorincz, H. and Boulicaut, J.-F. (2003*a*), A framework for frequent sequence mining under generalized regular expression constraints, *in* J.-F. Boulicaut and S. Dzeroski, eds, 'Proceedings of the Second International Workshop on Inductive Databases KDID', Rudjer Boskovic Institute, Zagreb, Croatia, Cavtat-Dubrovnik, Croatia, pp. 2–16.

Albert-Lorincz, H. and Boulicaut, J.-F. (2003*b*), Mining frequent sequential patterns under regular expressions: A highly adaptive strategy for pushing contraints, *in* D. Barbará and C. Kamath, eds, 'Proceedings of the Third SIAM International Conference on Data Mining', SIAM, San Francisco, CA.

Ale, J. M. and Rossi, G. H. (2000), An approach to discovering temporal association rules, *in* J. Carroll, E. Damiani, H. Haddad and D. Oppenheim, eds, '2000 ACM Symposium on Applied Computing', Vol. 1, ACM Press, Como, Italy, pp. 294–300.

Allen, J. F. (1981), An interval-based representation of temporal knowledge, *in* A. Drinan, ed., '7th International Joint Conference on Artificial Intelligence', Vancouver, pp. 221–226.

Allen, J. F. (1983), 'Maintaining knowledge about temporal intervals', *Communications of the ACM* **26**(11), 832–843.

Allen, J. F. (1984), 'Towards a general theory of action and time', *Artificial Intelligence* **23**(2), 123–154.

Allen, J. F. and Ferguson, G. (1994), 'Actions and events in interval temporal logic', *Journal of Logic and Computation* **4**(5), 531–579.

Allen, J. F. and Hayes, P. J. (1985), A common-sense theory of time, *in* '9th International Joint Conference on Artificial Intelligence (IJCAI-85)', Los Angeles, CA, pp. 528–531.

Allen, J. F. and Hayes, P. J. (1989), 'Moments and points in an interval-based temporal logic', *Computational Intelligence* **5**(4), 225–238.

Amir, A., Lewenstein, M. and Porat, E. (2000), Faster algorithms for string matching with k mismatches, *in* '11th Annual ACM-SIAM Symposium on Discrete Algorithms', Society for Industrial and Applied Mathematics, San Francisco, CA, USA, pp. 794–803.

Antunes, C. and Oliveira, A. L. (2003), 'Sequential pattern mining with approximated constraints'.

Arslan, A. N. and Egecioglu, O. (1999), An efficient uniform-cost normalized edit distance algorithm, *in* '6th Symposium on String Processing and Information Retrieval (SPIRE'99)', IEEE Comp. Soc, pp. 8–15.

Arslan, A. N. and Egecioglu, O. (2000), 'Efficient algorithms for normalized edit distance', *Journal of Discrete Algorithms* **1**(1), 3–20.

Artale, A. and Franconi, E. (2000), 'A survey of temporal extensions of description logics', *Annals of Mathematics and Artificial Intelligence* **30**(1-4), 171–210.

Ayres, J., Flannick, J., Gehrke, J. and Yiu, T. (2002), Sequential pattern mining using a bitmap representation, *in* '8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', ACM Press, Edmonton, Alberta, Canada, pp. 429–435.

Bacchus, F. and Kabanza, F. (2000), 'Using temporal logics to express search control knowledge for planning', *Artificial Intelligence* **116**(1-2), 123–191.

Badaloni, S., Falda, M. and Giacomin, M. (2004), 'Integrating quantitative and qualitative fuzzy temporal constraints', *AI Communications* **17**(4), 187–200.

Badaloni, S. and Giacomin, M. (1999), A fuzzy extension of Allens interval algebra, *in* E. Lamma and P. Mello, eds, 'Advances in Artificial Intelligence: 6th Congress of the Italian Association for Artificial Intelligence', Vol. 1792, Springer-Verlag Heidelberg, Bologna, Italy, pp. 155–165.

Badaloni, S. and Giacomin, M. (2002), Fuzzy extension of interval-based temporal sub-algebras, *in* 'Proceedings of IPMU 2002', Annecy, France, pp. 1119–1126.

Badaloni, S. and Giacomin, M. (2006), 'The algebra ia^{fuz}: a framework for qualitative fuzzy temporal reasoning', *Aritificial Intelligence* **17**(4), 872–908.

Batu, T., Ergün, F., Kilian, J., Magen, A., Raskhodnikova, S., Rubinfeld, R. and Sami, R. (2003), A sublinear algorithm for weakly approximating edit distance, *in* '35th ACM Symposium on Theory of Computing', ACM Press, San Diego, CA, USA, pp. 316–324.

Bayardo, R. J. and Agrawal, R. (1999), Mining the most interesting rules, *in* S. Chaudhuri and D. Madigan, eds, '5th International Conference on Knowledge Discovery and Data Mining', ACM Press, San Diego, CA, USA, pp. 145–154.

Bentley, J. L. and Sedgewick, R. (1997), Fast algorithms for sorting and searching strings, *in* '8th Annual ACM/SIAM Symposium on Discrete Algorithms', Society for Industrial and Applied Mathematics, New Orleans, Louisiana, USA, pp. 360–369.

Bettini, C., Wang, X. S. and Jajodia, S. (1996), Testing complex temporal relationships involving multiple granularities and its application to data mining, *in* '15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'96)', ACM Press, Montreal, Canada, pp. 68–78.

Bettini, C., Wang, X. S. and Jajodia, S. (1998), 'Mining temporal relationships with multiple granularities in time sequences', *Data Engineering Bulletin* **21**(1), 32–38.

Bettini, C., Wang, X. S., Jajodia, S. and Lin, J.-L. (1998), 'Discovering frequent event patterns with multiple granularities in time sequences', *IEEE Transactions on Knowledge and Data Engineering* **10**(2), 222–237.

Breslauer, D. and Gąsieniec, L. (1995), Efficient string matching on coded texts, *in* Z. Galil and E. Ukkonen, eds, '6th Annual Symposium on Combinatorial Pattern Matching', Springer, Berlin, Espoo, Finland, pp. 27–40.

Bunke, H. and Csirik, J. (1992), Edit distance of run-length coded strings, *in* '1992 ACM/SIGAPP Symposium on Applied Computing', ACM Press, Kansas City, Missouri, USA, pp. 137–143.

Cadez, I. V., Heckerman, D., Meek, C., Smyth, P. and White, S. (2000), Visualization of navigation patterns on a web site using model-based clustering, *in* '6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', Boston, Massachusetts, pp. 280–284.

Cai, Y. D., Clutter, D., Pape, G., Han, J., Welge, M. and Auvil, L. (2004), Maids: mining alarming incidents from data streams, *in* 'SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data', ACM Press, Paris, France, pp. 919–920.

Ceglar, A. and Roddick, J. F. (2006), 'Association mining', *ACM Computing Surveys* **38**(2).

Ceglar, A., Roddick, J. F. and Calder, P. (2003), Guiding knowledge discovery through interactive data mining, *in* P. C. Pendharkar, ed., 'Managing Data Mining Technologies in Organisations: Techniques and Applications', Idea Group Pub., Hershey, PA, pp. 45–87. Ch. 4.

Ceglar, A., Roddick, J. F., Mooney, C. H. and Calder, P. (2003), From rule visualisation to guided knowledge discovery, *in* S. J. Simoff, G. J. Williams and M. Hegland, eds, '2nd Australasian Data Mining Workshop (AusDM'03)', UTS, Canberra, Australia, pp. 59–94.

Chakrabarti, S., Sarawagi, S. and Dom, B. (1998), Mining surprising patterns using temporal description length, *in* A. Gupta, O. Shmueli and J. Widom, eds, '24th International Conference on Very Large Data Bases, (VLDB'98)', Morgan Kaufmann, New York, NY, pp. 606–617.

Chan, S., Kao, B., Yip, C. L. and Tang, M. (2002), Mining emerging substrings, Tech Report TR-2002-11, HKU CSIS. chan02mining.pdf.

Chen, X. and Petrounias, I. (2000), Discovering temporal association rules: Algorithms, language and system, *in* 'Proceedings of the 16th International Conference on Data Engineering', IEEE Computer Society, San Diego, California, USA, p. 306.

Chen, X., Petrounias, I. and Heathfield, H. (1998), Discovering temporal association rules in temporal databases, *in* M. T. Özsu, A. Dogac and O. Ulusoy, eds, 'International Workshop on Issues and Applications of Database Technology (IADT'98)', Berlin, Germany, pp. 312–319.

Cheng, H., Yan, X. and Han, J. (2004), Incspan: incremental mining of sequential patterns in large database, *in* '10th ACM SIGKDD International conference on Knowledge Discovery and Data Mining, KDD '04', ACM Press, New York, NY, USA, Seattle, WA, USA, pp. 527–532.

Chiu, D.-Y., Wu, Y.-H. and Chen, A. L. P. (2004), An efficient algorithm for mining frequent sequences by a new strategy without support counting, *in* 'Proceedings of the 20th International Conference on Data Engineering, ICDE 2004', IEEE Computer Society, Boston, MA, USA, pp. 375–386.

Cole, R. and Hariharan, R. (1998), Approximate string matching: a simpler faster algorithm, *in* '9th Annual ACM-SIAM Symposium on Discrete Algorithms', Society for Industrial and Applied Mathematics, San Francisco, CA, USA, pp. 463–472.

Cong, S., Han, J. and Padua, D. A. (2005), Parallel mining of closed sequential patterns, *in* R. Grossman, R. Bayardo and K. P. Bennett, eds, 'Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', ACM, Chicago, Illinois, pp. 562–567.

Cormode, G. and Muthukrishnan, S. (2002), The string edit distance matching problem with moves, *in* '13th Annual ACM-SIAM Symposium on Discrete Algorithms', Society for Industrial and Applied Mathematics, San Francisco, CA, USA, pp. 667–676.

Cotofrei, P. and Stoffel, K. (2002), Classification rules + time = temporal rules, *in* 'ICCS '02: Proceedings of the International Conference on Computational Science-Part I', Springer-Verlag, London, UK, pp. 572–581.

Dai, H. K. and Wang, G. (2005), Mining a class of complex episodes in event sequences, *in* N. Megiddo, Y. Xu and B. Zhu, eds, 'Algorithmic Applications in Management, First International Conference, AAIM 2005', Vol. 3521 of *Lecture Notes in Computer Science*, Springer, Xian, China, pp. 460–471.

Das, G., Lin, K.-I., Mannila, H., Renganathan, G. and Smyth, P. (1998), Rule discovery from time series, *in* '4th International Conference on Knowledge Discovery and Data Mining (KDD-98)', Knowledge Discovery and Data Mining, AAAI Press, pp. 16–22.

de Amo, S., Giacometti, A. and Santana, M. S. (2005), Milprit : Mining interval logic patterns with regular expression constraints, *in* 'I Workshop on Algorithms of Data Mining', Uberlândia, MG, Brazil.

Demiriz, A. and Zaki, M. J. (2002), 'webSPADE: A parallel sequence mining algorithm to analyze the web log data'.

Drakengren, T. and Jonsson, P. (1997*a*), 'Eight maximal tractable subclasses of allen's algebra with metric time', *Artificial Intelligence Research* **7**, 25–45.

Drakengren, T. and Jonsson, P. (1997*b*), Towards a complete classification of tractability in Allen's algebra, *in* 'International Joint Conference on Artificial Intelligence IJCAI'97', pp. 1466–1475.

Drakengren, T. and Jonsson, P. (1997*c*), 'Twenty–one large tractable subclasses of Allen's algebra', *Artificial Intelligence* **93**, 297–319.

Dubois, D. and Prade, H. (1989), 'Processing fuzzy temporal knowledge', *IEEE Transactions on Systems, Man, and Cybernetics* **19**(4).

El-Sayed, M., Ruiz, C. and Rundensteiner, E. A. (2004), Fs-miner: efficient and incremental mining of frequent sequence patterns in web logs, *in* A. H. F. Laender, D. Lee and M. Ronthaler, eds, 'Sixth ACM International Workshop on Web Information and Data Management (WIDM 2004)', Vol. ACM RENEWAL REQUIRED, ACM, Washington, DC, USA, pp. 128–135.

Fagin, R. and Halpern, J. Y. (1988), Reasoning about knowledge and probability, *in* M. Y. Vardi, ed., '2nd Conference on Theoretical Aspects of Reasoning about Knowledge', Morgan Kaufmann, San Francisco, pp. 277–293.

Freksa, C. (1992), 'Temporal reasoning based on semi-intervals', *Artificial Intelligence* **54**(1–2), 199–227.

Fu, Y. and Han, J. (1995), Meta-rule-guided mining of association rules in relational databases, *in* '1st International Workshop on Integration of Knowledge Discovery with Deductive and Object-Oriented Databases (KDOOD'95)', Singapore, pp. 39–46,.

Gaber, M. M., Zaslavsky, A. and Krishnaswamy, S. (2005), 'Mining data streams: a review', *SIGMOD Record* **34**(2), 18–26.

Galton, A. (2003), Temporal logic, *in* E. N. Zalta, ed., 'The Stanford Encyclopedia of Philosophy', winter 2003 edn, Stanford University.

Garofalakis, M. N., Rastogi, R. and Shim, K. (1999), SPIRIT: Sequential pattern mining with regular expression constraints, *in* '25th International Conference on Very Large Databases, VLDB'99', Edinburgh, Scotland, pp. 223–234.

Gennari, R. (1998), 'Temporal reasoning and constraint programming - a survey', *CWI Quarterly* **11**(2/3), 163–214.

Giannella, C., Han, J., Pei, J., Yan, X. and Yu, P. S. (2003), Mining frequent patterns in data streams at multiple time granularities, *in* H. Kargupta, A. Joshi, K. Sivakumar and Y. Yesha, eds, 'Next Generation Data Mining', pp. 191–212.

Godo, L. and Vila, L. (1995), Possibilistic temporal reasoning based on fuzzy temporal constraints, *in* C. Mellish, ed., 'IJCAI'95: Proceedings International Joint Conference on Artificial Intelligence', Montreal.

Guralnik, V., Garg, N. and Karypis, G. (2001), Parallel tree projection algorithm for sequence mining, *in* R. Sakellariou, J. Keane, J. R. Gurd and L. Freeman, eds, 'Euro-Par 2001: Parallel Processing, Proceedings of the 7th International Euro-Par Conference', Vol. 2150 of *Lecture Notes in Computer Science*, Springer, Manchester, UK, pp. 310–320.

Guralnik, V. and Karypis, G. (2004), 'Parallel tree-projection-based sequence mining algorithms', *Parallel Computing* **30**(4), 443–472.

Guralnik, V. and Srivastava, J. (1999), Event detection from time series data, *in* S. Chaudhuri and D. Madigan, eds, '5th International Conference on Knowledge Discovery and Data Mining', ACM Press, San Diego, CA, USA, pp. 33–42.

Guralnik, V., Wijesekera, D. and Srivastava, J. (1998), Pattern directed mining of sequence data, *in* R. Agrawal, P. E. Stolorz and G. Piatetsky-Shapiro, eds, 'Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)', AAAI Press, New York City, New York, pp. 51–57.

Guzmán, I. P. d. and Rossi, C. (1995), 'Lnint: A temporal logic that combines points and intervals and the absolute and relative approaches', *Journal of the IGPL* **3**(5), 745–764.

Hall, P. A. V. and Dowling, G. R. (1980), 'Approximate string matching', *ACM Computing Surveys* **12**(4), 381–402.

Halpern, J. Y. and Shoham, Y. (1991), 'A propositional modal logic of time intervals', *Journal of the ACM (JACM)* **38**(4), 935–962.

Han, J. and Kamber, M. (2001), *Data mining : Concepts and Techniques*, The Morgan Kaufmann series in data management systems, Morgan Kaufmann Publishers, San Francisco.

Han, J. and Pei, J. (2000), 'Mining frequent patterns by pattern growth: Methodology and implications', *SIGKDD Explorations Newsletter* **2**(2), 14–20.

Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U. and Hsu, M.-C. (2000), Freespan: frequent pattern-projected sequential pattern mining, *in* '6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', ACM Press, Boston, MA, USA, pp. 355–359.

Harada, L. (2004), 'Detection of complex temporal patterns over data streams', *Information Systems* **29**(6), 439–459.

Hay, B., Wets, B. and Vanhoof, K. (2002), Web usage mining by means of multidimensional sequence alignment methods, *in* O. R. Zaïane, J. Srivastava, M. Spiliopoulou and B. M. Masand, eds, 'WEBKDD 2002 - MiningWeb Data for Discovering Usage Patterns and Profiles, 4th International Workshop', Vol. 2703 of *Lecture Notes in Computer Science*, Springer, Edmonton, Canada, pp. 50–65.

Hilderman, R. J. and Hamilton, H. J. (2001), Evaluation of interestingness measures for ranking discovered knowledge, *in* D. W.-L. Cheung, G. J. Williams and Q. Li, eds, '5th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD 2001', Vol. 2035 of *LNCS*, Springer, Hong Kong, China, pp. 247–259.

Hingston, P. (2002), Using finite state automata for sequence mining, *in* '25th Australasian Conference on Computer Science (ACSC2002)', Australian Computer Society, Inc., Melbourne, Victoria, Australia, pp. 105–110.

Hirsch, R. (1996), 'Relation algebras of intervals', *Artificial Intelligence* **83**(2), 267–295.

Hoffman, P., Grinstein, G., Marx, K., Grosse, I. and Stanley, E. (1997), Dna visual and analytic data mining, *in* 'Conference on Visualization '97', IEEE Computer Society Press, Phoenix, AZ, USA, pp. 437–ff.

Hofman, H., Siebes, A. P. and Wilhelm, A. F. (2000), Visualizing association rules with interactive mosaic plots, *in* '6th SCM SIGKDD International Conference on Knowledge Discovery and Data Mining', ACM, Boston, MA, USA, pp. 227–235.

Höppner, F. (2001*a*), 'Discovery of temporal patterns – learning rules about the qualitative behaviour of time series'.

Höppner, F. (2001*b*), Learning temporal rules from state sequences, *in* 'IJCAI'01 Workshop on Learning from Temporal and Spatial Data', Seattle, USA, pp. 25–31.

Höppner, F. (2002), Discovery of core episodes from sequences – using generalization for defragmentation of rule sets, *in* 'Pattern Detection and Discovery in Data Mining', Vol. 2447 of *LNAI*, Springer, London, England, pp. 199–213.

Höppner, F. and Klawonn, F. (2001), 'Finding informative rules in interval sequences', *Lecture Notes in Computer Science* **2189**, 125+.

Huang, K.-Y., Chang, C.-H. and Lin, K.-Z. (2004), Prowl: An efficient frequent continuity mining algorithm on event sequences, *in* Y. Kambayashi, M. K. M. and and W. Wöß, eds, 'Data Warehousing and Knowledge Discovery, 6th International Conference, DaWaK 2004', Vol. 3181 of *Lecture Notes in Computer Science*, Springer, Zaragoza, Spain, pp. 351–360.

Huang, K.-Y., Chang, C.-H. and Lin, K.-Z. (2005), Closedprowl: Efficient mining of closed frequent continuities by projected window list technology, *in* 'SIAM International Conference on Data Mining', Newport Beach, Ca.

Hyyrö, H. (2003), 'A bit-vector algorithm for computing levenshtein and damerau edit distances', *Nordic Journal of Computing* **10**(1), 29–39.

Imberman, S. P., Domanski, B. and Thompson, H. (2002), 'Using dependancy/association rules to find indications for computerised tomography in a head trauma dataset', *Artificial Intelligence in Medicine* **26**(1-2).

Joshi, M. V., Karypis, G. and Kumar, V. (1999), Universal formulation of sequential patterns, Technical Report Under Preparation #99-21, Department of Computer Science, University of Minnesota.

Kam, P.-S. and Fu, A. W.-C. (2000), Discovering temporal patterns for interval-based events, *in* Y. Kambayashi, M. K. Mohania and A. M. Tjoa, eds, '2nd International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2000)', Vol. 1874 of *LNCS*, Springer, London, UK, pp. 317–326.

Keogh, E., Chu, S., Hart, D. and Pazzani, M. (1993), Segmenting time series: A survey and novel approach, *in* 'Data Mining in Time Series Databases', World Scientific Publishing Company.

Krokhin, A., Jeavons, P. and Jonsson, P. (2001), Reasoning about temporal constraints: Classifying the complexity in Allen's algebra by using an alebraic technique, Technical Report PRG-RR-01-02, Computing Laboratory, Oxford University.

Krokhin, A., Jeavons, P. and Jonsson, P. (2003), 'Reasoning about temporal relations: The tractable subalgebras of allen's interval algebra', *Journal of the ACM (JACM)* **50**(5), 591–640.

Kum, H., Pei, J. and Wang, W. (2002), ApproxMAP: Approximate mining of consensus sequential patterns, Technical report tr02-031, UNC-CH.

Ladkin, P. B. (1987), Models of axioms for time intervals, *in* 'AAAI-87', Morgan Kaufmann, pp. 234–239.

Landau, G. M., Myers, E. W. and Schmidt, J. P. (1998), 'Incremental string comparison', *SIAM Journal on Computing* **27**(2), 557–582.

Laur, P.-A., Symphor, J.-E., Nock, R. and Poncelet, P. (2005), Mining sequential patterns on data streams: A near-optimal statistical approach, *in* 'Second International Workshop on Knowledge Discovery from Data Streams', Porto, Portugal.

Li, R. and Carmo, J. (1995), 'On completeness of a positional interval logic with equality, overlap and subinterval relations', *Journal of the IGPL* **3**(5), 765–790.

Li, Y., Zhu, S., Wang, X. and Jajodia, S. (2002), 'Looking into the seeds of time: Discovering temporal patterns in large transaction sets', *Information Sciences* .

Lin, J., Keogh, E., Lonardi, S. and Chiu, B. (2003), A symbolic representation of time series, with implications for streaming algorithms, *in* ‘DMKD ’03: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery’, ACM Press, San Diego, California, pp. 2–11.

Lin, M.-Y. and Lee, S.-Y. (2004), ‘Interactive sequence discovery by incremental mining’, *Information Sciences* **165**(3-4), 187–205.

Luo, C. and Chung, S. M. (2004), A scalable algorithm for mining maximal frequent sequences using sampling, *in* ‘16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)’, IEEE Computer Society, Boca Raton, FL, USA, pp. 156–165.

Ma, C. and Li, Q. (2005), Parallel algorithm for mining frequent closed sequences, *in* V. Gorodetsky, J. Liu and V. A. Skormin, eds, ‘Autonomous Intelligent Systems: Agents and Data Mining, International Workshop, AIS-ADM 2005’, Lecture Notes in Computer Science, Springer, St. Petersburg, Russia.

Malik, J. and Binford, T. (1983), Reasoning in time and space, *in* ‘8th International Joint Conference on Artificial Intelligence’, Karlsruhe, West Germany, pp. 343–345.

Mannila, H. and Toivonen, H. (1996), Discovering generalized episodes using minimal occurrences, *in* ‘2nd International Conference on Knowledge Discovery and Data Mining (KDD’96)’, AAAI Press, Portland, Oregon, pp. 146–151.

Mannila, H., Toivonen, H. and Verkamo, A. I. (1995), Discovering frequent episodes in sequences, *in* U. M. Fayyad and R. Uthurusamy, eds, ‘1st International Conference on Knowledge Discovery and Data Mining (KDD-95)’, AAAI Press, Menlo Park, CA, USA, Montreal, Canada, pp. 210–215.

Mannila, H., Toivonen, H. and Verkamo, A. I. (1997), ‘Discovery of frequent episodes in event sequences’, *Data Mining and Knowledge Discovery* **1**(3), 259–289.

Marascu, A. and Masseglia, F. (2005), Mining sequential patterns from temporal streaming data, *in* ‘Proceedings of the first ECML/PKDD Workshop on Mining Spatio-Temporal Data (MSTD’05), held in conjunction with the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD’05)’, Porto, Portugal.

Marín, R., Cárdenas, M., Balsa, M. and Sánchez, J. (1997), ‘Obtaining solutions in fuzzy constraint networks’, *International Journal of Approximated Reasoning* **16**(3–4), 261– 288.

Masseglia, F., Cathala, F. and Poncelet, P. (1998), The PSP approach for mining sequential patterns, *in* ‘2nd European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD'98)', Vol. 1510 of *LNAI*, Springer Verlag, Nantes, France, pp. 176–184.

Masseglia, F., Poncelet, P. and Teisseire, M. (2000), Incremental mining of sequential patterns in large databases, Technical report, LIRMM.

Masseglia, F., Teisseire, M. and Poncelet, P. (2005), Sequential pattern mining: A survey on issues and approaches, *in* 'Encyclopedia of Data Warehousing and Mining', Information Science Publishing.

Meiri, I. (1991), Combining qualitative and quantitative constraints in temporal reasoning, *in* T. Dean and K. McKeown, eds, '9th National Conference on Artificial Intelligence', AAAI Press, Menlo Park, California, pp. 260–267.

Meiri, I. (1996), 'Combining qualitative and quantitative constraints in temporal reasoning', *Artificial Intelligence* **87**(1-2), 343–385.

Miller, D. P. (1986), Temporal reasoning, *in* '18th Conference on Winter Simulation', ACM Press, Washington, D.C., USA, pp. 437–439.

Miller, T. W. (2005), *Data and Text Mining A Business Applications Approach*, Pearson Education Inc., Upper Saddle River, New Jersey.

Mooney, C. H. (2005), Temporal mid-point demonstration applet, Technical Artefact SIE-05-004, School of Informatics and Engineering, Flinders University. http://www.infoeng.flinders.edu.au/research/techreps/SIE05004.zip.

Mooney, C. H., de Vries, D. and Roddick, J. F. (2004), A multi-level framework for the analysis of sequential data, *in* S. J. Simoff and G. J. Williams, eds, 'Australasian Data Mining Conference (AusDM'04)', Lecture Notes in Computer Science, Springer, Cairns, Qld, Australia, pp. 199–213.

Mooney, C. H. and Roddick, J. F. (2004), Mining relationships between interacting episodes, *in* M. W. Berry, U. Dayal, C. Kamath and D. Skillicorn, eds, '4th SIAM International Conference on Data Mining', SIAM, Lake Buena Vista, Florida.

Mooney, C. H. and Roddick, J. F. (2006), Marking time in sequence mining, *in* P. Christen, P. Kennedy, J. Li, S. Simoff and G. Williams, eds, 'Australasian Data Mining Conference (AusDM 2006)', Vol. 61, Conferences in Research and Practice in Information Technology (CRPIT), Sydney, Australia.

Mörchen, F. (2006), A better tool than Allen's relations for expressing temporal knowledge in interval data, *in* T. Eliassi-Rad, L. Ungar, M. Craven and D. Gunopulos, eds, 'Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06)', ACM Press, Philadelphia, Pennsylvania, pp. 668–673.

Navarro, G. (2001), 'A guided tour to approximate string matching', *ACM Computing Surveys* **33**(1), 31–88.

Nebel, B. and Bürckert, H.-J. (1995), 'Reasoning about temporal relations: a maximal tractable subset of Allen's interval algebra', *Journal of the Association for Computing Machinery* **42**(1), 43–66.

Nguyen, S. N., Sun, X. and Orlowska, M. E. (2005), Improvements of incspan: Incremental mining of sequential patterns in large database, *in* T. B. Ho, D. Cheung and H. Liu, eds, 'Proceedings of the 9th Pacific-Asia Conference, PAKDD 2005', Vol. 3518, Springer, Hanoi, Vietnam, pp. 442–451.

Oates, T., Schmill, M. D., Jensen, D. and Cohen, P. R. (1997), A family of algorithms for finding temporal structure in data, *in* '6th International Workshop on AI and Statistics'.

Ohlbach, H. J. (2004*a*), Fuzzy time intervals and relations - the full tire library, Technical report, Institut für Informatik, Universität München.

Ohlbach, H. J. (2004*b*), Relations between fuzzy time intervals, *in* '11th International Symposium on Temporal Representation and Reasoning (TIME'04)', Tatihou, Normandie, France, pp. 44–51.

Ong, K. H., Ong, K. L., Ng, W. K. and Lim, E. P. (2002), Crystalclear: Active visualization of association rules, *in* 'International Workshop on Active Mining (AM-2002) in Conjunction with the IEEE International Conference on Data Mining (ICDN'02)', IEEE Press, Maebashi City, Japan.

Oommen, B. J. and Loke, R. K. S. (1995), Pattern recognition of strings with substitutions, insertions, deletions and generalized transpositions, *in* 'IEEE International Conference on Systems, Man and Cybernetics', Vol. 2, pp. 1154–1159.

Oommen, B. J. and Zhang, K. (1996), 'The normalized string editing problem revisited', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **18**(6), 669–672.

Orlando, S., Perego, R. and Silvestri, C. (2004), A new algorithm for gap constrained sequence mining, *in* 'SAC Proceedings of the 2004 ACM Symposium on Applied Computing (SAC)', ACM Press, Nicosia, Cyprus, pp. 540–547.

Padmanabhan, B. and Tuzhilin, A. (1996), Pattern discovery in temporal databases: a temporal logic approach, *in* E. Simoudis, J. Han and U. Fayyad, eds, '2nd International Conference on Knowledge Discovery and Data Mining', AAAI Press, Portland, Oregon, pp. 351–354.

Pan, F., Cong, G., Tung, A. K. H., Yang, J. and Zaki, M. J. (2003), Carpenter: finding closed patterns in long biological datasets, *in* '9th ACM SIGKDD International

Conference on Knowledge Discovery and Data Mining (KDD '03)', ACM Press New York, NY, USA, Washington, D.C, pp. 637–642.

Parthasarathy, S., Zaki, M. J., Ogihara, M. and Dwarkadas, S. (1999), Incremental and interactive sequence mining, *in* 'Proceedings of the 1999 ACM CIKM International Conference on Information and Knowledge Management', ACM, Kansas City, Missouri, USA, pp. 251–258.

Pei, J. and Han, J. (2002), 'Constrained frequent pattern mining: a pattern-growth view', *SIGKDD Explorations* **4**(1), 31–39.

Pei, J., Han, J. and Mao, R. (2000), Closet: An efficient algorithm for mining frequent closed itemsets, *in* 'ACM SIGMOD International Workshop on Data Mining', ACM Press, Dallas, Texas, pp. 21–30.

Pei, 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* '2001 International Conference of Data Engineering (ICDE'01)', Heidelberg, Germany, pp. 215–226.

Pei, J., Han, J. and Wang, W. (2002), Mining sequential patterns with constraints in large databases, *in* '11th International Conference on Information and Knowledge Management', ACM Press, McLean, Virginia, USA, pp. 18–25.

Pinto, H., Han, J., Pei, J., Wang, K., Chen, Q. and Dayal, U. (2001), Multi-dimensional sequential pattern mining, *in* '10th International Conference on Information and Knowledge Management', ACM Press, Atlanta, Georgia, USA, pp. 81–88.

Povinelli, R. J. (2000), Identifying temporal patterns for characterization and prediction of financial time series events, *in* J. F. Roddick and K. Hornsby, eds, 'International Workshop on Temporal, Spatial and Spatio-Temporal Data Mining, TSDM2000', Vol. 2007 of *LNAI*, Springer, Lyon, France, pp. 46–61.

Prior, A. N. (1957), Time and modality, Oxford University Press.

Pujari, A. K. (1997), Neighbourhood logic and interval algebra, Technical Report 116 T, United Nations University International Institute for Software Technology UNU/IIST.

Rainsford, C. P. and Roddick, J. F. (1999), Adding temporal semantics to association rules, *in* J. M. Zytkow and J. Rauch, eds, 'Principles of Data Mining and Knowledge Discovery, Third European Conference, PKDD '99', Vol. 1704 of *Lecture Notes in Computer Science*, Springer, Prague, Czech Republic, pp. 504–509.

Rainsford, C. P. and Roddick, J. F. (2000), Visualisation of temporal interval association rules, *in* '2nd International Conference on Intelligent Data Engineering and

Automated Learning, (IDEAL 2000)', Vol. 1983 of *LNCS*, Springer, Shatin, N.T., Hong Kong, pp. 91–96.

Rajman, M. and Besançon, R. (1998), Text mining — knowledge extraction from unstructured textual data, *in* '6th Conference of International Federation of Classification Societies (IFCS-98)', Rome.

Roddick, J. F., Hornsby, K. and De Vries, D. (2003), A unifying semantic distance model for determining the similarity of attribute values, *in* M. Oudshoorn, ed., '26th Australasian Computer Science Conference (ACSC2003)', Vol. 16, ACS, Adelaide, Australia, pp. 111–118.

Roddick, J. F. and Mooney, C. H. (2005), 'Linear temporal sequences and their interpretation using midpoint relationships', *IEEE Transactions on Knowledge and Data Engineering* **17**(1), 133–135.

Rogowitz, B. E. and Treinish, L. A. (1993), An architecture for rule-based visualization, *in* G. M. Nielson and D. Bergeron, eds, 'IEEE Visualization '93', IEEE Computer Society Press, Los Alamitos, California, pp. 236–243.

Sahar, S. (1999), Interestingness via what is not interesting, *in* S. C. Madigan and D., eds, '5th International Conference on Knowledge Discovery and Data Mining', ACM Press, San Diego, CA, USA, pp. 332–336.

Sankoff, D. and Kruskal, J. B. (1999), *Time warps, string edits, and macromolecules / The theory and practice of sequence comparison*, David Hume series, reissue ed. edn, Center for the Study of Language and Information, Stanford, Calif.

Savary, L. and Zeitouni, K. (2005), Indexed bit map (ibm) for mining frequent sequences, *in* A. Jorge, L. Torgo, P. Brazdil, R. Camacho and J. a. Gama, eds, 'PKDD Knowledge Discovery in Databases: PKDD 2005, 9th European Conference on Principles and Practice of Knowledge Discovery in Databases', Vol. 3721 of *Lecture Notes in Computer Science*, Springer, Porto, Portugal, pp. 659–666.

Schwalb, E. and Vila, L. (1998), 'Temporal constraints: A survey', *Constraints* **3**(2–3), 129–149.

Seno, M. and Karypis, G. (2001), Lpminer: An algorithm for finding frequent itemsets using length-decreasing support constraint., *in* '1st IEEE Conference on Data Mining'.

Seno, M. and Karypis, G. (2002), Slpminer: An algorithm for finding frequent sequential patterns using length-decreasing support, Technical Report #02-023, University of Minnesota.

Shillabeer, A. and Roddick, J. F. (2005), Reconceptualising interestingness metrics for medical data mining, *in* J. Warren, ed., 'Australian Workshop on Health Data Mining', Mawson Lakes, SA.

Spiliopoulou, M. (1999), Managing interesting rules in sequence mining, *in* 'Principles of Data Mining and Knowledge Discovery', pp. 554–560.

Spiliopoulou, M. and Faulstich, L. C. (1998), WUM: a Web Utilization Miner, *in* 'Workshop on the Web and Data Bases (WebDB'98)', Valencia, Spain, pp. 109–115.

Srikant, R. and Agrawal, R. (1996), Mining sequential patterns: Generalizations and performance improvements, *in* P. M. G. Apers, M. Bouzeghoub and G. Gardarin, eds, '5th International Conference on Extending Database Technology, (EDBT'96)', Vol. 1057 of *LNCS*, Springer, Avignon, France, pp. 3–17.

Sun, X., Orlowska, M. E. and Li, X. (2003), Introducing uncertainty into pattern discovery in temporal event sequences, *in* 'Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003)', IEEE Computer Society, Melbourne, Florida, USA, pp. 299–306.

Sun, X., Orlowska, M. E. and Li, X. (2005), Finding temporal features of event-oriented patterns, *in* T. B. Ho, D. Cheung and H. Liu, eds, 'Proceedings of the 9th Pacific-Asia Conference, PAKDD 2005', Vol. 3518 of *Lecture Notes in Computer Science*, Springer, Hanoi, Vietnam,, pp. 778–784.

Sun, X., Orlowska, M. E. and Zhou, X. (2003), Finding event-oriented patterns in long temporal sequences, *in* K.-Y. Whang, J. J. and, K. S. and and J. Srivastava, eds, '7th Pacific-Asia Conference, PAKDD 2003', Vol. 2637 of *LNCS*, Springer, Seoul, Korea, pp. 15–26.

Tan, P., Kumar, V. and Srivastava, J. (2002), Selecting the right interestingness measure for association patterns, *in* '8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD02)', Edmonton, Canada, pp. 32–41.

Teng, W.-G., Chen, M.-S. and Yu, P. S. (2003), A regression-based temporal pattern mining scheme for data streams, *in* J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger and A. Heuer, eds, 'VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases', Morgan Kaufmann, Berlin, Germany, pp. 93–104.

Tichy, W. F. (1984), 'The string-to-string correction problem with block moves', *ACM Transactions on Computer Systems (TOCS)* **2**(4), 309–321.

Toivonen, H. (1996), Sampling large databases for association rules, *in* T. M. Vijayaraman, A. P. Buchmann, C. Mohan and N. L. Sarda, eds, 'VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases', Morgan Kaufmann, Mumbai (Bombay), India, pp. 134–145.

Tumasonis, R. and Dzemyda, G. (2004), The probabilistic algorithm for mining frequent sequences, *in* 'ADBIS (Local Proceedings)'.

van Beek, P. (1989), Approximation algorithms for temporal reasoning, *in* '10th International Joint Conference on Artificial Intelligence', Detroit, Michigan.

van Beek, P. (1990), Reasoning about qualitative temporal information, *in* 'National Conference on Artificial Intelligence', Boston, MA, pp. 728–734.

van Beek, P. and Manchak, D. W. (1996), 'The design and experimental analysis of algorithms for temporal reasoning', *Journal of Artificial Intelligence Research* **4**, 1–18.

Vautier, A., Cordier, M.-O. and Quiniou, R. (2005), An inductive database for mining temporal patterns in event sequences, *in* L. P. Kaelbling and A. Saffiotti, eds, 'IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence', Professional Book Center, Edinburgh, Scotland, UK,, pp. 1640–1641.

Vila, L. (1994), 'A survey on temporal reasoning in artificial intelligence', *AI Communications* **7**(1), 4–28.

Vila, L. and Godo, L. (1994), 'On fuzzy temporal constraint networks', *Mathware and Soft computing* **3**, 315–334.

Vilain, M. (1982), A system for reasoning about time, *in* 'National Conference on Artificial Intelligence', Pittsburg, pp. 197–201.

Vilain, M. B. and Kautz, H. A. (1986), Constraint propagation algorithms for temporal reasoning, *in* '5th National Conference of the American Association for Artificial Intelligence (AAAI-86)', Philadelphia, PA, pp. 377–382.

Vilain, M. B., Kautz, H. A. and van Beek, P. (1989), Constraint propagation algorithms for temporal reasoning: A revised report, *in* D. Weld and J. de Kleer, eds, 'Readings in Qualitative Reasoning about Physical Systems', Morgan Kaufmann, pp. 373 – 381.

Wagner, R. A. and Fischer, M. J. (1974), 'The string-to-string correction problem', *Journal of the ACM (JACM)* **21**(1), 168–173.

Wang, J. and Han, J. (2004), Bide: Efficient mining of frequent closed sequences, *in* 'Proceedings of the International Conference on Data Engineering (ICDE'04)', Boston, MA.

Wang, J., Han, J. and Pei, J. (2003), CLOSET+: searching for the best strategies for mining frequent closed itemsets, *in* L. Getoor, T. E. Senator, P. Domingos and

C. Faloutsos, eds, '9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', ACM Press, Washington, DC, USA, pp. 236–245.

Wang, K. (1997), 'Discovering patterns from large and dynamic sequential data', *Journal of Intelligent Information Systems* **9**(1), 33–56.

Wang, K. and Tan, J. (1996), Incremental discovery of sequential patterns, *in* 'ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery', Montreal, Canada.

Weida, R. (1995), Knowledge representation for plan recognition, *in* 'The Next Generation of Plan Recognition Systems: Challenges for and Insight from Related Areas of AI', Montreal, Canada, pp. 119–123.

Weiss, G. M. and Hirsh, H. (1998), Learning to predict rare events in event sequences, *in* R. Agrawal, P. Stolorz and G. Piatetsky-Shapiro, eds, '4th. International Conference on Knowledge Discovery and Data Mining (KDD'98)', AAAI Press, Menlo Park, CA, New York, NY, pp. 359–363.

Winarko, E. and Roddick, J. F. (2006), 'ARMADA - an algorithm for discovering richer relative temporal association rules from interval-based data', *Data and Knowledge Engineering* p. to appear.

Wong, P. C., Cowley, W., Foote, H., Jurrus, E. and Thomas, J. (2000), Visualizing sequential patterns for text mining, *in* 'IEEE Symposium on Information Visualization 2000 (INFOVIS)', IEEE Press, Salt Lake City, Utah, pp. 105–114.

Yan, X., Han, J. and Afshar, R. (2003), Clospan: Mining closed sequential patterns in large datasets, *in* '2003 SIAM International Conference on Data Mining (SDM'03)', San Fransisco, CA.

Yang, J., Wang, W. and Yu, P. S. (2001), Infominer: mining surprising periodic patterns, *in* '7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', ACM Press, San Francisco, California, pp. 395–400.

Yang, J., Wang, W., Yu, P. S. and Han, J. (2002), Mining long sequential patterns in a noisy environment, *in* 'ACM SIGMOD International Conference on Management of Data (SIGMOD'02)'.

Yang, Z. and Kitsuregawa, M. (2005), Lapin-spam: An improved algorithm for mining sequential pattern, *in* 'ICDEW '05: Proceedings of the 21st International Conference on Data Engineering Workshops', IEEE Computer Society, Tokyo, Japan, p. 1222.

Yang, Z., Wang, Y. and Kitsuregawa, M. (2005), Lapin: Effective sequential pattern mining algorithms by last position induction, *in* 'The 21st International Conference on Data Engineering (ICDE 2005)', Tokyo, Japan.

Yu, C.-C. and Chen, Y.-L. (2005), 'Mining sequential patterns from multidimensional sequence data', *IEEE Transactions on Knowledge and Data Engineering* **17**(1), 136–140.

Zaki, M. J. (1998), Efficient enumeration of frequent sequences, *in* '7th International Conference on Information and Knowledge Management', ACM Press, Bethesda, Maryland, United States, pp. 68–75.

Zaki, M. J. (2000), Sequence mining in categorical domains: Incorporating constraints, *in* A. Agah, J. Callan and E. Rundensteiner, eds, '9th International Conference on Information and Knowledge Management (CIKM2000)', ACM Press, McLean, VA, USA, pp. 422–429.

Zaki, M. J. (2001*a*), 'Parallel sequence mining on shared-memory machines', *Journal of Parallel and Distributed Computing* **61**(3), 401–426.

Zaki, M. J. (2001*b*), 'SPADE: An efficient algorithm for mining frequent sequences', *Machine Learning* **42**(1/2), 31–60.

Zaki, M. J. and Hsiao, C.-J. (2002), CHARM: An efficient algorithm for closed itemset mining, *in* R. L. Grossman, J. Han, V. Kumar, H. Mannila and R. Motwani, eds, '2nd SIAM International Conference on Data Mining (SDM'02)', SIAM, Arlington, VA, USA, pp. 457–473.

Zhang, M., Kao, B., Cheung, D. W.-L. and Yip, C. L. (2002), Efficient algorithms for incremental update of frequent sequences, *in* 'Pacific-Asia Conference on Knowledge Discovery and Data Mining', pp. 186–197.

Zhang, M., Kao, B., Yip, C. and Cheung, D. (2001), A gsp-based efficient algorithm for mining frequent sequences, *in* 'In Proceedings of IC-AI'001', Las Vegas, Nevada, USA.

Zheng, Q., Xu, K., Ma, S. and Lv, W. (2002), The algorithms of updating sequential patterns, *in* '5th International Workshop on High Performance Data Mining, in conjunction with the 2nd SIAM Conference on Data Mining'.