

Mining Relationships Between Interacting Episodes

Carl H. Mooney and John F. Roddick
School of Informatics and Engineering
Flinders University of South Australia,
PO Box 2100, Adelaide, South Australia 5001,
Email: {carl.mooney, roddick}@infoeng.flinders.edu.au

Abstract

The detection of recurrent episodes in long strings of tokens has attracted some interest and a variety of useful methods have been developed. The temporal relationship between discovered episodes may also provide useful knowledge of the phenomenon but as yet has received little investigation. This paper discusses an approach for finding such relationships through the proposal of a robust and efficient search strategy and effective user interface both of which are validated through experiment.

Keywords: Temporal Sequence Mining.

1 Introduction and Related Work

While the mining of frequent episodes is an important capability, the manner in which such episodes interact can provide further useful knowledge in the search for a description of the behaviour of a phenomenon. For example, discovering a relationship between input data which was hitherto thought to be independent might lead to the discovery of physical correlations between the environments within which the sensors generating the input data operate. Moreover, any temporal relationships might be used to predict future values in the sensor providing the later input.

General association mining algorithms are used to generate frequent itemsets from which *intra-transaction* association rules are generated. The *sequence* mining task is the discovery of *inter-transaction* associations – sequential patterns – across the same, or similar data. This problem was first addressed by Agrawal and Srikant [2] for mining transactional databases. The solution was based on extending the Apriori algorithm [1] and introduced the AprioriAll, AprioriSome, and DynamicSome algorithms. In order to address identified shortcomings, notably the time between associated sequences, single transaction constraints and to a lesser extent user-defined taxonomies, the GSP (Generalised Sequential Patterns) algorithm [23] was developed. GSP

incorporated time constraints (minimum and maximum gap between episodes), and sliding windows, and proved to be more efficient than its predecessors.

Typically improvements in performance have come about by employing a depth-first approach to the mining, as opposed to the more traditional breadth-first approach, and it has been recognised by Yang *et al.* [25] that these methods generally perform better when the data is memory-resident and when the patterns are long. As a result, algorithms based on a depth-first traversal of the search space were introduced and there was an increased focus on incorporating constraints into the mining process. Among these algorithms are SPADE (Sequential PAttern Discovery using Equivalence classes) [27] and its variant cSPADE (constrained SPADE) [26] which relies on combinatorial properties and lattice based search techniques and allow constraints to be placed on the mined sequences, and the SPIRIT (Sequential Pattern mIning with Regular expressIon constraints) algorithm [11] using regular expression constraints. All of these algorithms use a candidate generation and prune method requiring multiple passes over the data that has inherent problems with large datasets and long sequences, and as a result pattern-growth algorithms have appeared. PrefixSpan [21] being one representative of this type of algorithm.

The ever increasing amount of data being collected has also introduced the problems of how to handle the addition of new data within an existing ruleset and the possible non-relevance of older data, and in association rule mining methods have been proposed to deal with this [8]. With respect to these problems sequence mining is no different and similar techniques have also been developed [18, 20, 28].

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 (episodes) that occur frequently together. Solving this problem requires a different approach, and several types of algorithm have been proposed for different domains. Manilla *et al.* [17] developed the WINEPI algorithm and evaluated it on alarm detection data; regular expressions have been used to develop string matching algorithms [3, 7] and in web mining [22]. In addition, Yang *et al.* [25] developed a method that uses a *compatibility matrix*¹ in conjunction with a *match* metric (“aggregated amount of occurrences”) to discover long sequential patterns within (primarily) gene sequence data.

The purpose of generating frequent sequences, irrespective of the method, is to be able to infer some *rules*, and thus potential *knowledge* about behaviour. For example, given the sequence $A \dots B \dots C$ occurring in a string multiple times, rules such as: token (or event) $A \xrightarrow{\text{before}} B \xrightarrow{\text{before}} C$, can be expressed. It has been argued by Padmanabhan *et al.* [19] that these types of inference, and hence the temporal patterns, have limited expressive power. For this reason mining for sequences and the generation of rules based on first-order temporal logic (FOTL) [19], has extended the previous work by Manilla *et al.* [16] 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. One disadvantage to this approach is that no intermediate results are obtained and thus any mining for a different temporal logic pattern must be conducted on the complete database, incurring significant overhead.

Höppner *et al.* [13, 14] use a modified rule semantic, 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 [4] to describe rules that exist within these temporal patterns. Kam *et al.* [15] 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.

This type of temporal inference is aligned with the type of interactions we are mining, although in this paper we are limiting the scope of our search to a subset of Allen’s temporal relationships, and we propose a method that is based on *point relationships* that may exist within the frequent episodes, not between the frequent episodes. We also take the approach of mining

¹a conditional probability matrix that specifies the likelihood of symbol substitution

the frequent episodes and incrementally mining the results, in situ, to obtain the interactions. By proceeding in this manner we are able to save the intermediary results (frequent episodes) and the interactions in order that both may be mined at a future time, using different constraints, with a significantly reduced overhead.

This paper investigates the mining of temporal relationships within frequent episodes in a potentially very long string of tokens. Sections 2 and 3 presents the problem formally and presents a methodology for solving it. Since visualisation techniques have not been well examined, Section 4 discusses the user interface constructed to view discovered interactions, and Section 5 discusses our experiments using real-world data. Section 6 concludes with some discussion of future work.

2 Frequent Episode Discovery

The problem of discovering interacting episodes is one that consists of two distinct parts. First (phase 1), the mining of the frequent episodes and second (phase 2), using the discovered frequent episodes as input for the discovery of the interactions. However, since the frequent episodes are available after each pass of the input sequence, phase 2 can be performed in parallel. Note that the two phases are sufficiently different, as is the terminology used, and as such in this section we define the sequence mining problem and the following section defines the interacting episode mining problem. Each section begins by defining the notation that will be used and is followed by detailing the method and the algorithms designed for the tasks.

Let the set of available input tokens (the alphabet), denoted T , be defined as $T = \langle t_1, \dots, t_k \rangle \mid t_i \neq t_j, i \neq j, 1 \leq i, j \leq k$. A sequence S is then defined as a time ordered ($<$) sequence of input tokens and is denoted $S = \langle s_1, s_2, \dots, s_m \rangle \mid s_i \in T, 1 \leq i \leq m$. An *episode*, denoted E , is a sequence of tokens, $\langle s_n, s_{n+1}, \dots, s_{n+k} \rangle$, where $E \subseteq S$. The time at which an event occurs can either be inherent in the input data (as would be the case with alarm detection data), or be implied from the ordering of the input sequence. In the context of our current work this ordering must be strictly *less than* ($<$), that is the relationships being discovered do not require the consideration of events that occur simultaneously. Future work will address the case where this constraint is relaxed.

The user defined *lookahead*, l (similar to Mannila *et al.*’s window concept[17]), defines the maximum length episode to be mined, where $|E| \leq l \leq |S|$. A window, denoted w , is defined as the length of E , where $|E| \leq l$, at any point during the mining process. Therefore the maximum number of windows, *max_win* is given by $|S| - w + 1$ and the *frequency* of E in S is

defined as the number of windows in which E appears. The minimum frequency required for an episode to be reported, min_freq denoted δ , is calculated using a support, σ (user defined), multiplied by max_win at any given point in the mining run. Calculating the minimum frequency in this manner allows for potentially more interesting longer episodes to be reported at a lower threshold since there are fewer windows for longer episodes.

The values for both the *lookahead* l and support σ can be varied on each successive mining run, refer to Section 4 for details. The *tuning* of support for different datasets poses quite a challenge and often requires a detailed knowledge of the domain so that the mining does not produce either too many or too few results.

Thus the problem for this phase becomes: find all episodes

$$E_i \text{ on } \{S \mid E_i \leq l, freq(E_i) \geq \delta, \delta = (|S| - w + 1) \times \sigma\}$$

The algorithm we use for finding the frequent episodes, see Algorithm 2.1, 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 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 [1], and finally frequency derivation through a scan the dataset. The algorithm used is a modified version of the WINEPI algorithm [17] for finding serial episodes, which uses a combination of this principle (downward closure) and a prefix lookup. In our case a similar technique is used, but since only episodes that consist of contiguous tokens are presently of interest, the window width can be increased by one and then a check can be made to see if the first k tokens of the $(k+1)$ episode occur in the current window. In order to minimize the size of the candidate sets on subsequent passes of the algorithm we take advantage of this and maintain a set of the k -prefixes of frequent episodes which can then be used to improve valid candidate generation. Thus on each subsequent pass of the algorithm the window width is increased by one and the first k tokens of the generated $(k+1)$ candidates are checked against the k -prefixes and retained if there is a match. This candidate pool is then pruned and those candidates that meet the min_freq requirements are stored in \mathcal{F}_{k+1} . The algorithm terminates when either the *lookahead*, l , is reached or $\mathcal{F}_{k+1} = \emptyset$.

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

Algorithm 2.1 Main algorithm for finding frequent episodes and frequent interactions

Input: a sequence S , of tokens $t \in T$, a *lookahead* l and a *support* σ

Output: the collection $\mathcal{F}(S, l, \sigma)$ of frequent episodes E and the collection F_r of frequent interactions.

```

1: find  $C_1 := \{\alpha \in T \mid |\alpha| = 1\}$ ;
2:  $i := 0$ ; found := true
3: while  $i^{++} < l$  and found do
4:   for  $j := 0$ ;  $j < |S| - i + 1$ ;  $j^{++}$  do
5:      $\alpha := S_j, \dots, S_{i+j}$ 
6:      $\delta := (|S| - |\alpha| + 1) \times \sigma$ ;
7:     if  $i > 1$  then
8:       if  $\alpha_k \in \mathcal{F}_k \mid \alpha_k = \langle t_1 \dots t_{i-1} \rangle$  then
9:         add  $\alpha$  to  $C_j$ 
10:      end if
11:     else
12:       add  $\alpha$  to  $C_j$ 
13:     end if
14:   end for
15:   found :=  $\mathcal{F}_i \neq \emptyset$  where
      $\mathcal{F}_i := \{\forall \alpha \in C_j \mid frequency(\alpha, S, l) \geq \delta\}$ 
     /* prune */
16:   if  $i > 2$  and found then
17:     findCurrentRelationships( $\mathcal{F}_{1..i}, i$ )
     /* Algorithm 3.1 & 3.2 */
18:   end if
19: end while
20:  $F_r :=$  pruneCandidateInteractions( $I_r$ )
     /* Algorithm 3.3 */
21: return  $\mathcal{F}(S, l, \sigma)$ ,  $F_r$ 

```

3 Discovering Interacting Episodes

During the discovery of the frequent episodes (Algorithm 2.1, line 16), the second phase, finding the interactions that exist within them begins. Let an interaction θ be a temporal relationship between sub-episodes $(e_i, e_j) \mid |e_i| + |e_j| \leq l$, denoted $\theta_r(e_i, e_j) \in \mathcal{R}$, where \mathcal{R} is the set of temporal relationships as described by Allen [4]. To allow for varying length interactions to be discovered, or simply to minimise the interaction length, a *min_interaction_length*, γ , can be supplied by the user, and to allow for varying levels of support a user-defined *min_interaction_supp*, φ , can also be supplied. The exact nature of φ will be discussed later. Thus the problem for this phase is: find all

$$\theta_r(e_i, e_j) \text{ on } \{E_i \mid e_i, e_j \geq \gamma, \theta_r(e_i, e_j) \geq \varphi\}$$

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



Figure 1: Section of an input string showing varying window widths.

EXAMPLE 1.

Given the frequent episodes E_1, E_2, E_3 where:

$$E_1 = \langle B, R, I, J, A, V, E \rangle,$$

$$E_2 = \langle B, I, R, A, J, V, E \rangle, \text{ and}$$

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

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 the temporal relationship *IJ during BRAVE* exists.

A more complex example can be shown using the frequent episodes from Figure 1 as a source for the discovery of the interactions.

EXAMPLE 2.

Given the following frequent episodes:

$$A_1 = \langle G, L, A, T, I, N, R, E, E, K \rangle$$

$$B_1 = \langle E, N, G, C, A, N, T, O, N, E, S, E, L, I, S, H \rangle$$

$$C_1 = \langle G, E, F, R, E, R, M, A, N, N, C, H \rangle$$

$$C_2 = \langle G, E, R, M, A, N, F, R, E, N, C, H \rangle$$

$$D_1 = \langle L, D, U, T, C, H, A, T, I, N \rangle$$

$$D_2 = \langle L, A, D, U, T, C, H, T, I, N \rangle$$

- Episodes $A_1, B_1, D_1,$ and D_2 are all examples of the **during** relation, denoted $\theta_d(e_1, e_2)$ – *LATIN during GREEK, CANTONESE during ENGLISH and DUTCH during LATIN* respectively,
- Episode C_1 is an example of an **overlap** relation, denoted $\theta_o(e_1, e_2)$ – *GERMAN overlaps FRENCH,* and
- Episode C_2 is an example of a **meets** relation, denoted $\theta_m(e_1, e_2)$ – *GERMAN meets FRENCH.*

From our experience, while in simple examples, one can easily detect the relationship, as the episodes get longer or more numerous, this task becomes increasingly difficult quite quickly. A further feature in the discovery is the point at which embedded noise becomes part of the dominant relationship. This depends on a number of aspects:

- Whether the sub-episode is frequent in its own right or whether it is only frequent with its noise. For example, given the frequent episodes $\alpha_1\beta\alpha_2$ and β and the non-frequent episode $\alpha_1\alpha_2$ we need to decide whether $\alpha_1\alpha_2$ is reportable and/or whether $\alpha_1\beta\alpha_2$ is a separate reportable episode from β .
- The decision of how to deal with common tokens within both a dominant and an embedded sub-episode, as in *DUTCH during LATIN* in Example 2.
- The decision of how to handle noise that interrupts an episode at different locations. Given frequent episodes $\alpha_1\alpha_2\alpha_3, \alpha_1\beta\alpha_2\alpha_3$ and $\alpha_1\alpha_2\beta\alpha_3,$ and infrequent episodes $\alpha_1\alpha_2\alpha_4\beta\alpha_3$ and $\alpha_1\beta\alpha_2\alpha_4\alpha_3,$ how can it be recognised (simply) that α_4 is noise and that β is during $\alpha_1\alpha_2\alpha_3$?

The rest of this section demonstrates how we discover the interactions and deal with these problems.

3.1 Algorithms for Interaction Discovery

Time can be viewed as both discrete and linear in nature and, with the exception of Allen [5], a logic of intervals can be constructed using points rather than from intervals themselves [12]. We also take this view in our approach to discovering interactions and as such the algorithm for discovering candidate interactions within the discovered frequent episodes is based on the set of point relationships, Figure 2, between two episodes **a** and **b**.

- | | |
|------|-----------------------------------|
| i. | a.start and b.start |
| ii. | a.start and b.end |
| iii. | a.end and b.start |
| iv. | a.end and b.end |

Figure 2: Point relationships

These relationships can be used to express the complete set of Allen’s [4] temporal relations, those of which are immediately relevant *during, overlaps, meets* and their inverses, are summarised in Table 1. To handle the remaining seven of Allen’s relationships (*starts, finishes, before* and their inverses, and *equal*),

Relation	Sym	Example	Conditions	Endpoint Constraints
α meets β β is met by α	m mi	$\alpha\alpha\alpha\beta\beta\beta$	$\alpha.start < \beta.start$ and $\alpha.end < \beta.start$	$< < < <$
α during β β contains α	d di	$\alpha\alpha\alpha$ $\beta\beta\beta\beta\beta\beta$	$\alpha.start > \beta.start$ and $\alpha.end < \beta.end$	$> < > <$
α overlaps β β is overlapped by α	o oi	$\alpha\alpha\alpha$ $\beta\beta\beta$	$\alpha.start < \beta.start$ and $\alpha.end < \beta.end$	$< < > <$

Table 1: Temporal Relationship Summary

and the extensions of Freksa [10] who deals with semi-intervals, requires a level of *relaxation* with respect to the position one or more of the tokens. This may be appropriate, for example, if data are from n polled sensors and the presented position of a token can be up to $n - 1$ places out of position with respect to actual events.

Interaction candidate generation is computed by searching for each $E_k \in \mathcal{F}_k$ in all $\{\mathcal{F}_{k+n} \mid n = 1, \dots, max_len_episode - 1\}$. This method will yield e_i , the frequent $E_k \in \mathcal{F}_k$, and e_j , the substring which remains after E_k has been removed from the frequent $E_{k+n} \in \mathcal{F}_{k+n}$. A determination can then be made, using the four point relationships (Figure 2) as to which temporal relationship they satisfy, see Algorithms 3.1 and 3.2 below.

Algorithm 3.1 findCurrentRelationships

Input: a collection $\mathcal{F}(S, l, \sigma)$ of frequent episodes E and the current iteration level of the main algorithm, $curr$.

Output: the collection of candidate interactions I_r

```

1: for  $i := 0; i < curr - 1; i^{++}$  do
2:    $e_i := \mathcal{F}_i$ 
3:   for  $j := 0; j < \mathcal{F}_{curr.size}; j^{++}$  do
4:      $e_j := \mathcal{F}_j$ 
5:      $I_r := \text{findAnyRelationships}(e_j, e_i)$ 
        /* Algorithm 3.2 */
6:   end for
7: end for
8: return  $I_r$ 

```

The volume of candidates generated in this manner can be high and therefore a suitable reporting threshold needs to be applied. This threshold could be based on a combination of a number of factors including:

- The combined episode length,
- The maximum number of possible combinations that a sub-episode can occur within another sub-episode,

Algorithm 3.2 findAnyRelationships

Input: an episode \mathbf{f} and an episode \mathbf{p} , $|p| < |f|$

Output: the temporal relationship $\theta_r(p, f - p)$ that exists between \mathbf{p} and \mathbf{f} or, **null** if no relationship exists.

```

1: if  $(\mathbf{f}-\mathbf{p}) = \emptyset$  then
2:   return null
3: else
4:   find  $\mathbf{p}.start, \mathbf{p}.end, (\mathbf{f}-\mathbf{p}).start$  and  $(\mathbf{f}-\mathbf{p}).end$ 
5:   determine  $\theta_r$  and add it to  $I_r$ 
6: end if
7: return  $I_r$ 

```

- Whether the frequency (count) of an interaction expressed as a percentage of the total frequency of the frequent episodes of length $|e_i| + |e_j|$ is greater than a user defined support.

While the determination of appropriate reporting thresholds is an area of ongoing research, for the purposes of this investigation we have found that the last criterion yielded satisfactory results, for both the synthetic and the genome data used (see §5). Formally, to be included in the frequent interactions,

$$F_{r_i} \leftarrow \left[\frac{\text{frequency } r}{\sum_{k=0}^n \text{frequency } |r_k|} \geq \varphi \right]$$

where φ is the user defined *min_interaction_supp*. It should be noted that this metric is highly dependent on the number of frequent episodes that were discovered and is therefore linked to the *min_support*, δ , the setting of which has already been discussed. The method for performing this task is shown in Algorithm 3.3. Moreover, we report only maximal interactions where, given that an interaction is comprised of an antecedent and a consequent, all subsets of the longest antecedent and consequent are pruned.

As well as the relationships outlined in Table 1,

Enclosing		Enclosed		Combined		Relationship Type
<i>FR</i>	<i>NF</i>	<i>FR</i>	<i>NF</i>	<i>FR</i>	<i>NF</i>	
✓		✓		✓		During, Overlap, Meets
✓		✓			✓	<i>not possible</i>
✓			✓	✓		Participant
	✓	✓			✓	<i>not possible</i>
	✓	✓		✓		Container
	✓		✓		✓	<i>never reported</i>

Table 2: Possible configurations of sub-episodes within a frequent episode. *FR*: frequent, *NF*: non-frequent

Algorithm 3.3 pruneCandidateInteractions

Input: a list of candidate relationships, I_r and a $min_interaction_supp, \varphi$

Output: a list of frequent relationships F_r , where

$$\{\theta_r(e_i, e_j) \in \mathcal{R} \mid \theta_r(e_i, e_j) \geq \varphi\}$$

1: **for** $i := 0; i < I_r.size; i^{++}$ **do**

2: $r := I_r(i)$

3: $r.support := \frac{frequency\ r}{\sum_{k=0}^n frequency\ |r_k|}$

4: **if** $r.support \geq \varphi$ **then**

5: $\text{add } r \text{ to } F_r$

6: **end if**

7: **end for**

8: **return** F_r

the questions raised in the previous section can also be answered using this point based approach. The source of common tokens within both a dominant and an embedded sub-episode is handled directly by the point based approach since the only interest 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. Sub-episodes interrupting at different locations satisfy one of the three classes of relationship, θ_d, θ_o , or θ_m , and therefore yield the same two sub-episodes, e_i and e_j , and as such 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. In addition, since this method finds all relationships, the problem of whether the sub-episode is frequent in its own right or whether it is only frequent with its noise can be handled by reporting in such a way as to discriminate between those that are fully temporal, that is both sub-episodes e_i and e_j are frequent, and those in which either e_i or e_j are not frequent in their own right. For the latter case we have defined relationships that are either, *container* relationships, where the enclosing

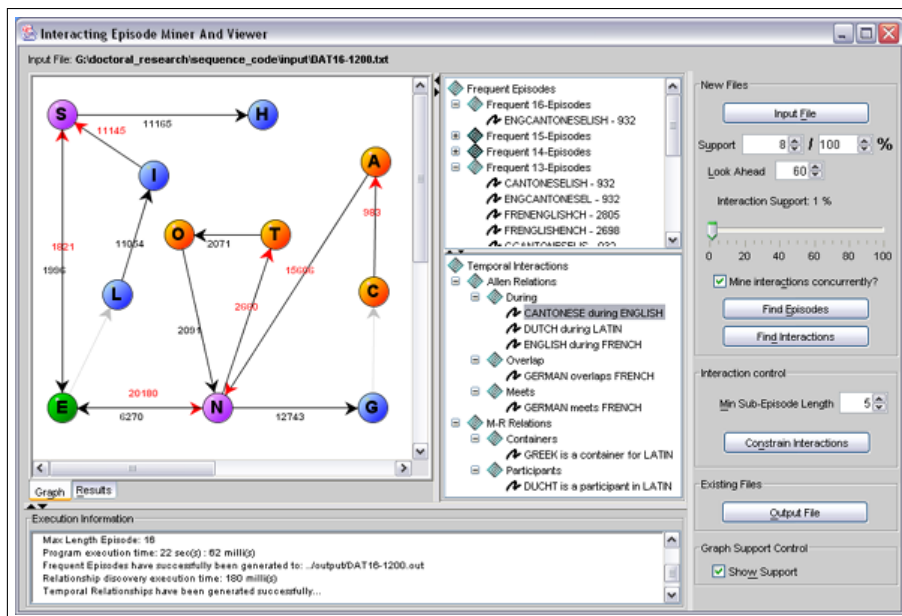
sub-episode is not frequent, or *participant* relationships, where the enclosed sub-episode is not frequent. This leads to the possibilities outlined in Table 2.

4 User Interface

In order 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, we have developed a graphical user interface. A common problem identified with data mining routines is that the amount of results produced can be large and difficult to interpret, hence methods for constraining the output have been implemented. We have also adopted the position that allowing the user to minimise the resultant output is beneficial and therefore the user interface for the **INTERacting Episode Miner and viewer (INTEM)**, Figure 4, enables the setting of all currently implemented constraints (*lookahead, episode support, interaction support* and *minimum interaction length*). The results of the mining run (frequent episodes) and the discovered frequent 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 sequence, 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. A brief description of the interface and the way in which it can be used is given below.

The interface is comprised of four main areas:

- The left panel, which contains a tab pane enables the user to switch between viewing the results in a text format or as a directed graph.
- The right panel houses the controls for; the selection of input and output files, setting of the lookahead, support (episode and interaction), concurrent mining of interactions and the minimum sub-episode length. There is also a check box



Legend	
Node Colour	Description
Green	Root Node for the interaction (E)
Blue	Enclosing sub-episode (G, L, I, H)
Orange	Enclosed sub-episode (C, A, T, O)
Purple	Shared node of both enclosing and enclosed sub-episodes (N, S)
Edge Colour	
Black	Relationship between two Nodes and their supports
Gray	The point(s) at which the enclosed sub-episode begins/ends within the enclosing sub-episode

Figure 3: Screen shot of the **INTEM** application showing the interaction *CANTONESE during ENGLISH* which was discovered in the DAT16-1200 file

enabling the support to be displayed/not displayed on the graph.

- The centre panel houses two tree structures the purpose of which is to enable selection of either a frequent episode or an interaction and have the graphical display reflect that selection.
- The bottom panel contains an area for displaying program execution information.

Currently the application can be used in two modes:

Mode 1: For selection of an input file to mine. This results in saved output files for the frequent episodes, frequent interactions and a file that is the backing for the graphical display.

Mode 2: For selection of previously generated output. In this mode files from a previous mining session

are read in to the application so that interactions may be viewed.

The flexibility offered by **Mode 2** 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. The **INTEM** software, while functional, is still being developed and therefore further research, dependent on the application area, will be undertaken in order to facilitate a greater range of user defined parameters and interaction. It is intended that this process will be undertaken in parallel with algorithmic developments in this area.

5 Experimental Results

This section outlines the experiments we have run using the **INTEM** software under different support levels and a lookahead value of 60. All of the algorithms and the user interface (see Section 4) were developed using Java™ (J2SDK 1.4.2) and the experiments were conducted on a 1.2GHz Athlon PC with 512Mb of main memory. 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, $T = \langle A \dots Z, \#, / \rangle$, while the genome file had the five characters $T = \langle A, C, G, T, N \rangle$. Table 3 summarises the nature of the files used.

File Name	Size of Token Set	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

Table 3: Experimental file specifications

The smallest file (DAT7-200) was used for algorithmic development, since we knew the composition and what we expected to find. 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 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 time³ Figure 4(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 4(a) are for the generation of the frequent episodes shown in Figure 4(b).

The larger token set for the synthetic data files produced fewer frequent episodes, by a factor of 8 against the genome data (Figure 4(b)), and as such the support metric that was used for reporting the frequent interactions (see §3.1 for details) was more appropriate. Thus, in order to assess the algorithms using the genome data a support level was chosen

³In our 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.

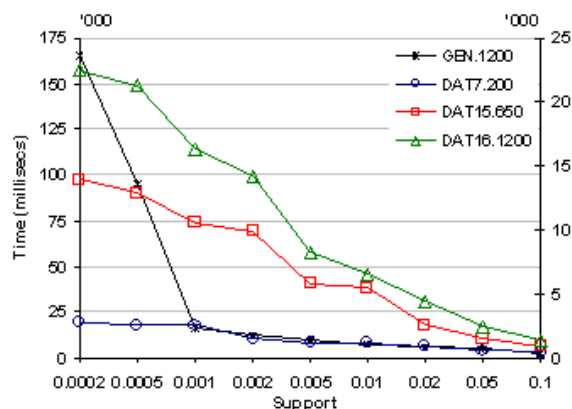
where an excessive number of frequent episodes was not going to be a major contributing factor. Since the interactions are able to be mined independently of, as well as concurrently with, episode production actual processing times are able to be reported, Figure 5(a), for the algorithms that have been developed. The times displayed in Figure 5(a) are for the generation of the frequent interactions shown in Figure 5(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.

6 Future Work

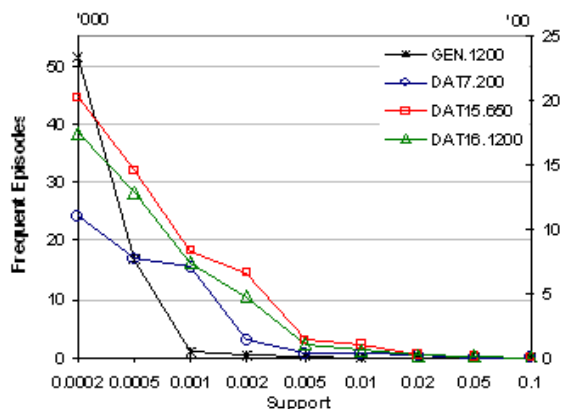
This paper discusses the development of a method for mining relationships between interacting episodes based on a subset of the temporal logic expressed by Allen [4] and shows that the method is robust. However, related research questions remain, including; the allowance of noise in one or both of the sub-episodes, in order to accommodate the full set of temporal relationships and those that are based on the semi-intervals of Freksa [10], the meaning and implications of the two newly defined relationships (*container* and *participant*), and the relevance of the methodology in different domains. In answering these questions it may be necessary to incorporate methods available in constrained pattern mining research, or perhaps to extend those that are applicable to the current environment.

The reporting of episodes and interactions 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 §3.1), while adequate for some domain data, has proved to be lacking in datasets that generate numerous frequent episodes, and hence a different heuristic is required. Methods that are not based solely on frequency metrics, for example, the concept of *information gain* as discussed in [24], and the use of string edit distance techniques [6, 9] applied to the candidate episodes and interactions, both independently and in concert may assist in overcoming the problems associated with numerous frequent episodes, and allow us to report interactions that are both frequent and of interest.

Finally, the fact that our algorithms are memory resident imposes a limitation on the size of the files that we are able to process and while this may not be a problem in some domains, a different strategy for

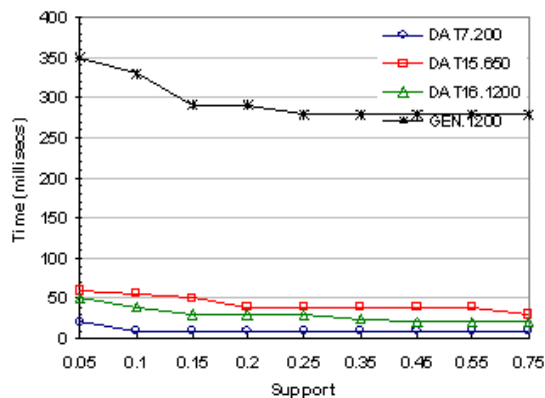


(a) Processing time as a function of support. GEN-1200 uses the primary y-axis (left) for its values

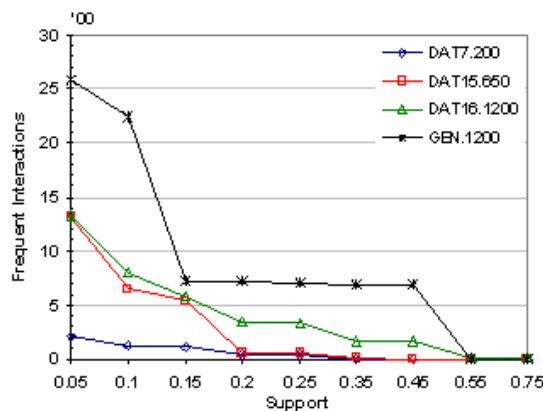


(b) Number of frequent episodes as a function of support. GEN-1200 uses the primary y-axis (left) for its values

Figure 4: Processing time and frequent episode production using a lookahead distance of 60 and varying levels of support



(a) Processing time as a function of support



(b) Number of frequent interactions as a function of support

Figure 5: Execution time and frequent interaction production using frequent episodes mined at a support of 0.0005 with varying levels of interaction support

dealing with the majority of real-world data sets will be necessary. It is our belief that solving this problem will also allow us to make progress toward processing real-time data, thus expanding the application areas to which our process is applicable.

References

- [1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *20th Int. Conf. Very Large Data Bases, (VLDB)*, pages 487–499. Morgan Kaufmann, 1994.
- [2] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee S. P.

Chen, editors, *Eleventh International Conference on Data Engineering*, pages 3–14. IEEE Computer Society Press, Taipei, Taiwan, 1995.

- [3] A.V. Aho. Algorithms for finding patterns in strings. In J van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity. Elsevier, 1990.
- [4] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
- [5] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.
- [6] Abdullah N. Arslan and Omer Egecioglu. Efficient algorithms for normalized edit distance. *Journal of Discrete Algorithms*, 1(1):3–20, 2000.
- [7] Jon L. Bentley and Robert Sedgewick. Fast algorithms for sorting and searching strings. In *Proceedings of*

- the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 360–369. Society for Industrial and Applied Mathematics, New Orleans, Louisiana, United States, 1997.
- [8] David Wai-Lok Cheung, Sau Dan Lee, and Benjamin Kao. A general incremental technique for maintaining discovered association rules. In *Fifth International Conference On Database Systems For Advanced Applications*, pages 185–194, Melbourne, Australia, 1997.
- [9] Graham Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 667–676, San Francisco, California, 2002. Society for Industrial and Applied Mathematics.
- [10] Christian Freksa. Temporal reasoning based on semi-intervals. *Artificial Intelligence*, 54:199–227, 1992.
- [11] Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. SPIRIT: Sequential pattern mining with regular expression constraints. In *The VLDB Journal*, pages 223–234. 1999.
- [12] Joseph Y. Halpern and Yoav Shoham. A propositional modal logic of time intervals. *Journal of the ACM (JACM)*, 38(4):935–962, 1991.
- [13] Frank Höppner. Discovery of temporal patterns – learning rules about the qualitative behaviour of time series, 2001.
- [14] Frank Höppner and Frank Klawonn. Finding informative rules in interval sequences. *Lecture Notes in Computer Science*, 2189:125+, 2001.
- [15] Po-Shan Kam and Ada Wai-Chee Fu. Discovering temporal patterns for interval-based events. In Yahiko Kambayashi, Mukesh K. Mohania, and A. Min Tjoa, editors, *Second International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2000)*, volume 1874, pages 317–326, London, UK, 2000. Springer.
- [16] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovering frequent episodes in sequences. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining KDD-95*, Montreal, Canada, 1995. AAAI Press.
- [17] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.
- [18] F Masseglia, P Poncelet, and M Teisseire. Incremental mining of sequential patterns in large databases. Technical report, LIRMM, 2000.
- [19] Balaji Padmanabhan and Alexander Tuzhilin. Pattern discovery in temporal databases: a temporal logic approach. In Evangelos Simoudis, Jiawei Han, and Usama Fayyad, editors, *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 351–354, Portland, Oregon, 1996. AAAI Press.
- [20] Srinivasan Parthasarathy, Mohammed Javeed Zaki, Mitsunori Ogihara, and Sandhya Dwarkadas. Incremental and interactive sequence mining. In *CIKM*, pages 251–258. TR715, CS Dept., University of Rochester, 1999.
- [21] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. PrefixSpan mining sequential patterns efficiently by prefix projected pattern growth. In *Proc. 2001 Int. Conf. Data Engineering (ICDE’01)*, pages 215–226, Heidelberg, Germany, 2001.
- [22] Myra Spiliopoulou and Lukas C. Faulstich. WUM: a Web Utilization Miner. In *Workshop on the Web and Data Bases (WebDB98)*, pages 109–115. Valencia, Spain, 1998.
- [23] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin, editors, *Proc. 5th Int. Conf. Extending Database Technology, (EDBT)*, volume 1057, pages 3–17. Springer-Verlag, Avignon, France, 1996.
- [24] Jiong Yang, Wei Wang, and Philip S. Yu. Infominer: mining surprising periodic patterns. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 395–400, San Francisco, California, 2001. ACM Press.
- [25] Jiong Yang, Wei Wang, Philip S. Yu, and Jiawei Han. Mining long sequential patterns in a noisy environment. In *SIGMOD Conference*, 2002.
- [26] Mohammed Javeed Zaki. Sequence mining in categorical domains: Incorporating constraints. In Arvin Agah, Jamie Callan, and Elke Rundensteiner, editors, *CIKM*, pages 422–429. 2000.
- [27] Mohammed Javeed Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.
- [28] Qingguo Zheng, Ke Xu, Shilong Ma, and Weifeng Lv. The algorithms of updating sequential patterns. In *Proc. of 5th International Workshop on High Performance Data Mining, in conjunction with 2nd SIAM Conference on Data Mining*, 2002.