Representation and Identification of Approximately Similar Event Sequences

T P Martin[a,b] and B Azvine[b]

[a]Machine Intelligence & Uncertainty Group, University of Bristol, BS8 1UB, UK
[b]BT TSO Security Futures, Adastral Park, Ipswich, IP5 3RE, UK

Abstract. The MARS (Modelling Autonomous Reasoning System) project aims to develop a collaborative intelligent system combining the processing powers and visualisation provided by machines with the interpretive skills, insight and lateral thinking provided by human analysts. There is an increasing volume of data generated by online systems, such as internet logs, transaction records, communication records, transport network monitors, sensor networks, etc. Typically, these logs contain multiple overlapping sequences of events related to different entities. Information that can be mined from these event sequences is an important resource in understanding current behaviour, predicting future behaviour and identifying non-standard patterns. In this paper, we describe a novel approach to identifying and storing sequences of related events, with scope for approximate matching. The event sequences are represented in a compact and expandable *sequence pattern* format, which allows the addition of new event sequences as they are identified, and subtraction of sequences that are no longer relevant. We present an algorithm enabling efficient addition of a new sequence pattern. Examination of the sequences by human experts could further refine and modify general patterns of events.

1 Introduction

With the increasing volume of data generated by online systems (internet logs, transaction records, communication records, transport networks, sensor networks, etc.) there is a clear requirement for machine assistance in filtering, fusing and finding causal relations in data flows. Current AI approaches rely heavily on statistical and machine learning algorithms which require large amounts of data and rest on an assumption that the previous behaviour of a system is a good guide to future behaviour.

The aim of the MARS[1] (Modeling Autonomous Reasoning System) project is to develop a collaborative intelligent system able to process real-time (or near real-time) unstructured data feeds by combining the processing powers and visualisation provided by machines with the interpretive skills, insight and lateral thinking provided by human analysts. The overall product of this research is a cognitive software system that exhibits highly flexible and autonomous behaviour, and that can provide insights into the functioning of a broad range of

[1] UK Technology Strategy Board Reference: 1110_CRD_TI_ICT_120182

complex social-technical systems monitored by data flows. An early version of MARS was successful in helping to combat cable theft[2], leading to significant savings for UK telecoms and rail networks.

A key aspect in the analysis of lower level data such as event logs is the identification of sequences of linked events. For example, internet logs, physical access logs, transaction records, email and phone records contain multiple overlapping sequences of events related to different users of a system. Information that can be mined from these event sequences is an important resource in understanding current behaviour, predicting future behaviour and identifying non-standard patterns and possible security breaches.

An event sequence is a time-stamped series of observations or measurements of a fixed set of attributes. An example could be a sequence of records containing: *access card no., date/time, building and entrance identifier, direction of access (in/out), result (access granted / refused)*
or
source IP address/port, destination IP address/port, date/time, amount of data transferred.

Specific problems in extracting sequences of related events include determination of what makes events "related", finding groups of "similar" sequences, identification of typical sequences, and detection of sequences that deviate from previous patterns. In this paper we describe a novel approach to identifying sequences of related events, with scope for assistance from human experts. The event sequences are represented in a compact and expandable sequence pattern format, which allows the addition of new event sequences as they are identified, and subtraction of sequences that are no longer relevant. Examination of sequences can be used to refine and modify general patterns of events. The specific focus of this short paper is the algorithm to incrementally add a new sequence pattern.

2 Background

Humans are adept at acquiring knowledge of sequential patterns, even when the knowledge is acquired and used in an implicit manner, i.e. without conscious awareness [1]. In an effort to reproduce this learning behaviour in machines, investigators have focused on statistical methods such as time-series analysis and machine learning approaches. These are not ideally suited to the task, because they generally require a large sample of data, usually represented as numerical features, and because they typically seek to fit data to known distributions. There is evidence that human behaviour sequences can differ significantly from such distributions - for example, in sequences of asynchronous events such as sending emails or exchanging messages. Barabasi [2] showed that many activities do not obey Poisson statistics, and consist instead of short periods of intense activity followed by longer periods in which there is no activity.

[2] www.newscientist.com/article/dn21989-ai-system-helps-spot-signs-of-copper-cable-theft.html

A related problem is that statistical machine learning methods generally require a significant number of examples to form meaningful models, able to make predictions. Where we are looking for a new behaviour pattern (for example, in network intrusion) it may be important to detect it quickly, before a statistically significant number of incidents have been seen. A malicious agent may even change the pattern before it can be detected. Finally, it is highly advantageous to allow human experts to specify patterns. This is difficult or impossible with data-driven methods.

2.1 Fuzzy Indiscernibility of Events

A fundamental aspect of human information processing and intelligence is the ability to identify a group of entities e.g. physical objects, events, abstract ideas) and subdivide them into smaller subgroups at an appropriate level of granularity for the task at hand. Early humans needed to determine whether an animal was dangerous or not; for the non-dangerous case, a second level of analysis might distinguish between edible and non-edible (or uncatchable). From the perspective of decision-making (stay or run, hunt or ignore), the precise identification of different species is not relevant - there is no need for discernibility within the broad sub-categories.

The concept of information granulation was introduced by Zadeh [3] to formalise the process of dividing a group of objects into sub-groups (granules) based on “indistinguishability, similarity, proximity or functionality”. In this view, a granule is a fuzzy set whose members are (to some degree) equivalent to each other. In a similar manner, humans are good at dividing events into related groups, both from the temporal perspective (event A occurred a few minutes before event B but involves the same entities) and from the perspective of non-temporal event properties (event C is very similar to event D because both involve entities which are similar).

2.2 The X-Mu Approach to Fuzzy

Standard fuzzy approaches typically require modification of crisp algorithms to allow set-valued variables. This is most apparent in fuzzifications of arithmetic, where a single value is replaced by a (so-called) fuzzy number, which actually represents an *interval.* For example, calculating the average age of four employees known to be 20, 30, 50 and 63 is inherently simpler than the same calculation when the ages are given as *young, quite young, middle-aged* and *approaching retirement.* In the latter case, we must handle interval arithmetic as well allowing for membership grades. In a similar fashion, querying a database to find employees who are aged over 60 is simpler than finding employees who are *approaching retirement age.*

In the classical fuzzy approach, for any predicate on a universe U, we introduce a membership function

$\mu : U \rightarrow [0,\ 1]$

representing the degree to which each value in U satisfies the predicate; since there is generally a set of values which satisfy the predicate to some degree, we must modify algorithms to handle sets of values rather than single values. These sets represent equivalent values - that is, values which cannot be distinguished from each other. In this work, we are dealing with events that are equivalent because their attributes are indiscernible - however, these sets of events may vary according to membership, interpreted as the degree to which elements can be distinguished from each other.

The $X - \mu$ method [4] recasts the fuzzy approach as a mapping from membership to universe, allowing us to represent a set, interval or a single value that varies with membership. An example of a single value is the mid-point of an interval. This is a natural idea which is difficult to represent in standard fuzzy theory, even though it arises in common aspects such as the cardinality of a discrete fuzzy set or the number of answers returned in response to a fuzzy query.

3 Algorithm for Converting a Stream of Events into a Minimal Directed Graph

3.1 Directed Graph Representation of Event Sequences

For any sequence of events, we create a directed graph representation of the events in which each edge represents a set of indiscernible events. Clearly for reasons of storage and searching efficiency it is desirable to combine event sequences with common sub-sequences, as far as possible, whilst only storing event sequences that have been observed. This problem is completely analogous to dictionary storage, where we are dealing with single letters rather than sets of events, and we can utilise efficient solutions that have been developed to store dictionaries. In particular, we adopt the notion of a directed acyclic word graph, or DAWG [5]. Words with common letters (or events) at the start and/or end are identified and the common paths are merged to give a minimal graph, in the sense that it has the smallest number of nodes for a DAWG representing the set of words (event sequences). Several algorithms for creating minimal DAWGs have been proposed. In the main, these have been applied to creation of dictionaries and word checking, efficient storage structure for lookup of key-value pairs and in DNA sequencing (which can be viewed as a variant of dictionary storage). Most methods (e.g. [6, 7]) operate under the assumption that all words (letter sequences) are available and can be presented to the algorithm in a specific order. Sgarbas [5] developed an incremental algorithm which allowed additional data to be added to a DAWG structure, preserving the minimality criterion (i.e. assuming the initial DAWG represented the data in the most compact way, then the extended DAWG is also in the most compact form).

We assume that the data is in a time-stamped tabular format (for example, as comma separated values with one or more specified fields storing date and time information) and that data arrives in a sequential manner, either row by row or in larger groups which can be processed row-by-row. Each row represents an event; there may be several unrelated event sequences within the data stream but we assume events in a single sequence arrive in time order. It is not necessary to store the data once it has been processed, unless required for later analysis.

Each column i in the table has a domain D_i and a corresponding attribute name A_i . There is a special domain O whose elements act as unique identifiers (e.g. row number or event id).

Formally, the data can be represented by a function

$$f : O \to D_1 \times D_2 \times \cdots \times D_n \text{ or a relation } R \subset O \times D_1 \times D_2 \times \cdots \times D_n$$

where any given identifier $o_i \in O$ appears at most once. We use the notation $A_k(o_i)$ to denote the value of the k^{th} attribute for object o_i. Our aim is to find ordered sequences of events (and subsequently, groups of similar sequences, where the grouping is based on indiscernibility of attributes).

3.2 Sequence-extending relations.

Event sequences obey

- each event is in at most one sequence
- events in a sequence are ordered by date and time
- an event and its successor are linked by relations between their attributes, such as equivalence, tolerance, and other relations.

These are referred to as *sequence-extending relations*. Note that it is possible to have different sequence-extending relations for different sequences; also it is possible to change the sequence-extending relations dynamically. In the graph structure described below, the sequence-extending relations are associated with nodes in the graph. It is also possible to identify sequences when an event is missing, although this is not covered here. Any event that is not part of an existing sequence is the start of a new sequence. For any attribute A_i we can define a tolerance relation R_i where

$$R_i : D_i \times D_i \to [0,\ 1]$$

is a reflexive and symmetric fuzzy relation and $\forall j :\ R_i\left(o_j\,,\ o_j\right) = 1$ Then the tolerance class of objects linked through attribute A_i is

$$T\left(A_i\,,\ o_m\right) = \left\{o_j/\chi_{mj}\ \middle|R_i\left(A_i\left(o_m\right),\ A_i\left(o_j\right)\right) = \chi_{mj}\right\}$$

Note that this set includes (with membership 1) all objects with the attribute value $A_i(o_m)$. The tolerance class can be expressed equivalently as a set of pairs.

Finally we requiref a total order relation P_T, defined on a distinguished attribute (or small subset of attributes) representing a timestamp. We can then define sequences (in the obvious way) and projected sequences

$$\begin{aligned}
&\forall i :\ P_T\left(A_T\left(o_i\right),\ A_T\left(o_i\right)\right)\\
&\forall i \neq j :\ P_T\left(A_T\left(o_i\right),\ A_T\left(o_j\right)\right) > 0 \ \ \to\ \ P_T\left(A_T\left(o_j\right),\ A_T\left(o_i\right)\right) = 0\\
&Q\left(o_i\right) = \left\{o_i/\chi_{ti}\ \middle|P_T\left(o_t,\ o_i\right) = \chi_{ti}\right\}
\end{aligned}$$

where A_T is the timestamp attribute (or attributes) and the ordering of events models the obvious temporal ordering. The time attribute t_i obeys $t_i \leq t_{i+1}$ for all i. It is treated as a single attribute although it could be stored as more than one (such as date, time of day). We assume that a number of sequence-extending relations $R_1 \dots R_n$ are defined on appropriate domains.

Two events o_i and o_j are potentially linked in the same sequence if

$$\min\left(Q_T\left(o_i,\ o_j\right),\ \min_m R_m\left(o_i,\ o_j\right)\right) \geq \mu$$

i.e. all required attributes satisfy the specified sequence-extending relations to a degree greater than some threshold μ. We write

$$potential-link\left(o_i\ ,\ o_j\ ,\mu\right) \leftrightarrow \min\left(Q_T\left(o_i,\ o_j\right),\ \min_m R_m\left(o_i,\ o_j\right)\right) \geq \mu$$

and

$$\begin{gathered} linked\left(o_i\ ,\ o_j\ ,\mu\right) \leftrightarrow potential-link\left(o_i\ ,\ o_j\ ,\mu\right) \\ AND \\ \nexists o_k : potential-link\left(o_i\ ,\ o_k\ ,\mu\right)\ AND\ potential-link\left(o_{ki}\ ,\ o_j\ ,\mu\right) \end{gathered}$$

i.e. two events are linked if they satisfy the specified tolerance and equivalence relations to a degree μ or greater, and there is no intermediate event.

3.3 Event Categorisation Relations

We also define equivalence classes on some of the domains, which are used to compare and categorise events from different sequences. An equivalence class on one or more domains is represented by a value from each domain - for times denoted by *day* and *hour* values, we might define equivalence for week-day rush hour (*day=Mon-Fri, hour=8,9,17,18*), other-weekday (*day=Mon-Fri, hour≠8,9,17,18*) and weekend (*day=Sat,Sun*).

These can easily be extended to fuzzy equivalence classes. The important feature is that the equivalence classes partition the objects - i.e. each object belongs to exactly one equivalence class for each domain considered. In the $X-\mu$ fuzzy case, the equivalence classes can vary with membership but always partition the objects. Where necessary operations can be incrementally extended to cover different memberships.

Formally, for a specified attribute A_i we define

$$S\left(A_i, o_m\right) = \left\{o_j\ \middle|A_i\left(o_j\right) = A_i\left(o_m\right)\right\}$$

and the set of associated equivalence classes (also called elementary concepts) is

$$C_i = \left\{S\left(A_i, o_m\right)\ \middle|o_m \in O\right\}$$

(as an example consider time or elapsed time below). In the propositional case C_i contains just one set, whose elements are the objects for which attribute i

is true. In the fuzzy case, elements are equivalent to some degree. Specifying a membership threshold gives a nested set of equivalence relations, so that once a membership threshold is known, we can proceed as in the crisp case. The operation can be extended to multiple attributes.

We use the selected attributes to find the *EventCategorisation.* This is an ordered set of equivalence classes from one or more attributes (or $n-$tuples of attributes):

$$B_k \in \{A_1,\ \dots\ ,\ A_n\}$$
$$EventCategorisation\,(o_i) = [B_k\,(o_i)\ |k = 1,\ \dots\ ,\ m]$$

i.e. each B_k is an attribute, and the event categorisation of some object o_i is given by the equivalence classes corresponding to its attribute values. The result is not dependent on the order in which the attributes are processed. This order can be optimised to give fastest performance when deciding which edge to follow from given node.

For any set of sequences, we create a minimal representation using a DASG (directed acyclic sequence graph) as shown in Fig. 1. The graph is a deterministic finite automaton, with no loops and a unique end node. Each event is represented by a labelled edge. The edge label shows the equivalence classes applicable to the event, referred to as the event categorisation. The source node S is a single starting point for all sequences; to ensure we have a unique end node, F, we append a dummy "end of sequence" ($\#END$) event to all sequences.

3.4 Worked Example Based on the VAST Challenge 2009 Dataset

We illustrate the algorithm using a small subset (Table 1) of benchmark data taken from mini-challenge 1 of the VAST 2009 dataset[3]. This gives swipecard data showing employee movement into a building and in and out of a classified area within the building. No data is provided on exiting the building. In this data,

Emp = set of employee ids = {10, 11, 12}
Date,Time = date / time of event
Entry points = {B - building, C - classified section}
Access direction = {in, out}

We have selected three employees for illustration purposes; rows in the initial table were ordered by date/time, but have been additionally sorted by employee here to make the sequences obvious. We first define the sequence-extending relations, to detect candidate sequences. Here, for a candidate sequence of n events:

$$S_1 = (o_{11}, o_{12}, o_{13},\ \dots\ , o_{1n})$$

we define the following computed quantities :

$$\begin{aligned} ElapsedTime \quad & \mathit{\Delta T}_{ij} = Time\,(o_{ij}) - Time\,(o_{ij-1}) \\ with \quad & \mathit{\Delta T}_{i1} = Time\,(o_{i1}) \end{aligned}$$

[3] http://hcil2.cs.umd.edu/newvarepository/benchmarks.php

Table 1. Sample Data from the VAST 2009 MC1 Dataset

eventID	Date	Time	Emp	Entrance	Direction
1	jan-2	7:30	10	b	in
2	jan-2	13:30	10	b	in
3	jan-2	14:10	10	c	in
4	jan-2	14:40	10	c	out
5	jan-2	9:30	11	b	in
6	jan-2	10:20	11	c	in
7	jan-2	13:20	11	c	out
8	jan-2	14:10	11	c	in
9	jan-2	15:00	11	c	out
10	jan-3	9:20	10	b	in
11	jan-3	10:40	10	c	in
12	jan-3	14:00	10	c	out
13	jan-3	14:40	10	c	in
14	jan-3	16:50	10	c	out
15	jan-3	9:00	12	b	in
16	jan-3	10:20	12	c	in
17	jan-3	13:00	12	c	out
18	jan-3	14:30	12	c	in
19	jan-3	15:10	12	c	out

Table 2. Allowed Actions (row = first action, column = next action)

	b,in	c,in	c,out
b,in	x	x	
c.in			x
c,out	x	x	

and restrictions (for $j > 1$) :

$$\begin{aligned}
Date\left(o_{ij}\right) &= Date\left(o_{ij-1}\right) \\
0 < Time\left(o_{ij}\right) - Time\left(o_{ij-1}\right) &\leq T_{thresh} \\
Emp\left(o_{ij}\right) &= Emp\left(o_{ij-1}\right) \\
\left(Action\left(o_{ij-1}\right), Action\left(o_{ij}\right)\right) &\in AllowedActions \\
\text{where} \quad Action\left(o_{ij}\right) &= \left(Entrance\left(o_{ij}\right),\ Direction\left(o_{ij}\right)\right)
\end{aligned}$$

where the relation *AllowedActions* is specified in Table 2.

These constraints can be summarised as

- events in a single sequence refer to the same employee
- successive events in a single sequence conform to allowed transitions between locations and are on the same day, within a specified time (T_{thresh}) of each other

We choose a suitable threshold e.g. $T_{thresh} = 8$, ensuring anything more than 8 hours after the last event is a new sequence. We identify candidate sequences

by applying the sequence-extending relations. Any sequence has either been seen before or is a new sequence. In Table 1, candidate sequences are made up of the events:
$1 - 2 - 3 - 4$,
$5 - 6 - 7 - 8 - 9$,
$10 - 11 - 12 - 13 - 14$,
$15 - 16 - 17 - 18 - 19$

We also define the *EventCategorisation* equivalence classes used to compare events in different sequences. Here,

$$\begin{aligned} &EquivalentAction = I_{Action} \\ \text{For direction } In, \quad &EquivalentEventTime = \{[7], [8], \ldots\} \\ \text{For direction } Out, \quad &EquivalentElapsedTime = \{[0], [1], [2], \ldots\} \end{aligned}$$

where I is the identity relation and the notation [7] represents the set of start times from 7:00-7:59. With this definition, events 5 and 10 are equivalent since both have *Entrance=b, Direction=In* and *Time* in 7:00-7:59. Formally,

$$EventCategorisation\,(o_5) = ([b, in], [7])$$
$$EventCategorisation\,(o_{10}) = ([b, in], [7])$$

Similarly, events 7 and 12 are equivalent, as both have *Entrance =c, Direction = Out* and *ElapsedTime* in 3:00-3:59.
We represent each identified sequence as a path labelled by its event categorisations (Fig 1 left) and combine multiple sequences into a minimal DASG representing all sequences seen so far (Fig 1 right). Nodes are labelled by unique numbering; since the graph is deterministic, each outgoing edge is unique. An edge can be specified by its start node and event categorisation. Below, we also refer to an edge by its event categorisation if there is no ambiguity about its start node.

Standard definitions are used for *InDegree, OutDegree, IncomingEdges* and *OutgoingEdges* of a node, giving respectively the number of incoming and outgoing edges, the set of incoming edges and the set of outgoing edges. We also apply functions *Start* and *End* to an edge, to find or set its start and end nodes respectively and *EdgeCategorisation* to find its categorisation class.
Finally, let the function $ExistsSimilarEdge(edge, endnode)$ return true when:
edge has end node *endnode*, event categorisation L and start node *S1*
AND
a second, distinct, edge has the same end node and event categorisation L but a different start node $S2$
AND
$S1$ and $S2$ have $OutgoingEdges(S1) == OutgoingEdges(S2)$

If such an edge exists, its start node is returned by the function
$StartOfSimilarEdge(edge, endnode)$

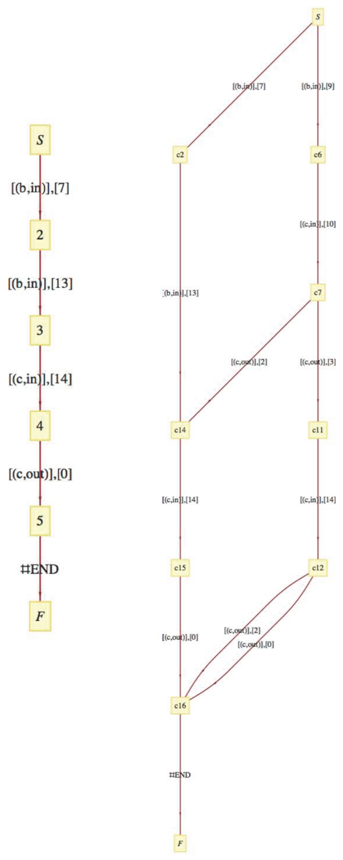

Fig. 1. (left) Event sequence 1-2-3-4 from Table 1. The label [(b,in)],[7] represents all events whose entrance and direction is (b, in) and time is equivalent to 7 (at a given membership). (right) DASG representing the 4 distinct sequences in Table 1

The function $MergeNodes(Node1, Node2)$ deletes $Node2$ and merges its incoming and outgoing edges with those of $Node1$.

The function $CreateNewNode(Incoming, Outgoing)$ creates a new node with the specified sets of incoming and outgoing edges.

The graph can be used to identify sequences of events that have already been seen. If a new sequence is observed (i.e. a sequence which differs from each sequence in the graph by at least one event categorisation), it can be added to the graph using the algorithm below.

The algorithm proceeds in three stages. In the first and second parts, we move step-by-step through the event sequence and graph, beginning at the start

```
Algorithm ExtendGraph
Input :Graph G (minimal current sequence), start node S, end node F
CandidateSequence Q[0 - NQ] representing the candidate sequence;
         each element is an event identifier. The sequence is not
          already present in the graph and is terminated by #END
Output : updated minimal graph, incorporating the new sequence
Local variables :  Node startNode, newNode, endNode, matchNode
         Edge currentEdge, matchEdge
         Categorisation currentCategorisation
         integer seqCounter;

startNode = S
seqCounter = 0
WHILE  EventCategorisation(S[seqCounter]) ∈ OutgoingEdges(StartNode)
     currentEdge = (startNode, EventCategorisation(Q[seqCounter] )
     endNode = End (currentEdge)
     IF InDegree (endNode) > 1
     THEN
          newNode =CreateNewNode({currentEdge}, OutgoingEdges(endNode))
          IncomingEdges(endNode) = IncomingEdges (endNode) -
     currentEdge
          startNode = newNode
     ELSE
          startNode = endNode
     seqCounter++
ENDWHILE

WHILE seqCounter < NQ                          // create new path
     currentEdge = (startNode, EventCategorisation (S[seqCounter]) )
     startNode = CreateNewNode({currentEdge}, { })
     seqCounter++
ENDWHILE

currentCategorisation = #END
currentEdge = (startNode, #END )      // last edge, labelled by #END

IncomingEdges(F) = IncomingEdges (F) + currentEdge
endNode = F
nextEdgeSet = {currentEdge}

WHILE  nextEdgeSet contains exactly one element (i.e currentEdge)
                 AND ExistsSimilarEdge(currentEdge, endNode)

     matchNode = StartOfSimilarEdge(currentEdge, endNode)
     startNode = Start (currentEdge)
     IncomingEdges(endNode) = IncomingEdges (endNode) - {currentEdge}
     nextEdgeSet = IncomingEdges (startNode)
     IncomingEdges (matchNode)= nextEdgeSet∪IncomingEdges (matchNode)
     endNode = matchNode
     currentEdge ∈ edgeSet        // choose any element,
END WHILE                         //  "while" loop terminates if >1
```

Fig. 2. Extending a minimal graph by incremental addition of a sequence of edges

node S. If an event categorisation matches an outgoing edge, we follow that edge to the next node and move to the next event in the sequence. If the new node has more than one incoming edge, we copy it; the copy takes the incoming edge that was just followed, and the original node retains all other incoming edges. Both copies have the same set of output edges. This part of the algorithm finds other sequences with one or more common starting events.

If we reach a node where there is no outgoing edge matching the next event's categorisation, we create new edges and nodes for the remainder of the sequence, eventually connecting to the end node F. As the sequence is new, we must reach a point at which no outgoing edge matches the next event's categorisation; if this happens at the start node S then the first stage is (effectively) missed.

Finally, in the third stage, we search for sequences with one or more common ending events. Where possible, the paths are merged. As shown in Sgarbas [5], adding a new sequence using the incremental algorithm takes time roughly in the order of $|S|$, the number of unique sequences stored.

4 Summary

The paper outlines a way of storing event sequences in a compact directed graph format, and gives an efficient incremental algorithm to update the graph with an unseen sequence. A human expert can easily add sequence patterns, even if these have not been seen in the data yet. An algorithm to remove sequence patterns has also been developed and will be presented in a future paper, together with experimental results showing the efficiency and effectiveness of the approach.

References

1. A Cleeremans and James L. McClelland. Learning the structure of event sequences. *J Experimental Psychology*, 120:235 – 253, 1991.
2. A. L. Barabasi. The origin of bursts and heavy tails in human dynamics. *Nature*, pages 207–211, 2005.
3. L. A. Zadeh. Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. *Fuzzy Sets and Systems*, 90:111–127, 1997.
4. T. P. Martin. The x-mu representation of fuzzy sets. *Soft Computing*, 19:1497 – 1509, 2015.
5. K. N. Sgarbas, N. D. Fakotakis, and G. K. Kokkinakis. Optimal insertion in deterministic dawgs. *Theoretical Computer Science*, 301, Numb 1-3:103–117, 2003.
6. D. Revuz. Minimization of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92:181 – 189, 1992.
7. J. E. Hopcroft and J. D Ullman. *Introduction to Automata Theory, Languages, and Computation,*. Addison-Wesley, Reading, MA, 1979.