Martin, T. P., & Azvine, B. (2016). An Incremental Fuzzy Approach to Finding Event Sequences. In J. P. Carvalho, M-J. Lesot, U. Kaymak, S. Vieira, B. Bouchon-Meunier, & R. R. Yager (Eds.), Information Processing and Management of Uncertainty in Knowledge-Based Systems: 16th International Conference, IPMU 2016, Eindhoven, The Netherlands, June 20-24, 2016, Proceedings, Part I. (pp. 525-536). (An Incremental Fuzzy Approach to Finding Event Sequences; Vol. 610). Springer International Publishing. DOI: 10.1007/978-3-319-40596-4_44

An Incremental Fuzzy Approach to Finding 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. Recent years have seen increasing volumes of data generated by online systems, such as internet logs, physical access logs, transaction records, email and phone records. These contain multiple overlapping sequences of events related to different individuals and 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 and possible security breaches. Statistical machine learning approaches have had some success but do not allow human insight to be included easily. We have recently presented a framework for representing sequences of related events, with scope for assistance from human experts. This paper describes the framework and presents a new algorithm which (i) allows the addition of new event sequences as they are identified from data or postulated by a human analyst, and (ii) allows subtraction / removal of sequences that are no longer relevant. Examination of the sequences can be used to further refine and modify general patterns of events.

Keywords: Event Sequences, Incremental Algorithm, Fuzzy, X-mu

1 Introduction

Collaborative intelligence aims to combine the power of machines with the interpretive skills, insight and lateral thinking provided by human analysts. The role of the computer in this partnership is to gather data, transform it algorithmically, and provide visualisation. The role of the human is to provide creativity and insight in analysing and understanding the data, and to extract "knowledge" - which may be in the form of predictive rules, normal and unusual patterns in the data, further insight into underlying mechanisms, etc.

In order to implement a successful collaborative intelligent system, it is necessary to exchange knowledge between the components - in particular between humans and machines, although in a multi-agent system we may also need to consider human-human and machine-machine exchange. Typically, machine processing is centred on well-defined entities and relations which may range from the flat table structures of database systems through graph-based representations and up to ontological approaches involving formal logics. Human language and communication, on the other hand, is based on a degree of vagueness and ambiguity enabling efficient transmission of information between humans without the

need for precise definition of every term used. There is a fundamental mis-match between the representations used by machine and human components in a collaborative intelligent system. Even quantities that can be measured precisely (such as height of a person or building, volume of a sound, amount of rainfall, colour of an object, etc.) are usually described in human language using non-precise terms such as *tall, loud, quite heavy, dark green*, etc. More abstract properties such as *beautiful landscape, delicious food, pleasant weather, clear documentation, corporate social responsibility,* are fundamentally ill-defined, whether they are based on a holistic assessment or reduced to a combination of lower-level, measurable quantities. However, we are generally able to judge the degree to which a particular instance possesses such a property. Zadeh's initial formulation of fuzzy sets [1] was inspired primarily by the flexibility of definitions in natural language.

Large volumes of data are generated by monitoring and recording systems, such as internet logs, phone records, GPS monitors and physical access logs (e.g. to buildings), financial transactions, etc. Linking records together into sequences (whether within a single data source or across multiple sources) is a complex task which is ideally suited to the notion of collaborative intelligence. Specific problems in extracting sequences of related events include determination of what makes events "related', how to find groups of "similar" sequences, identification of typical sequences, and detection of sequences that deviate from expected patterns, where the notion of "expected" can either be derived from previous observations or from human analysis.

The ability to incorporate human knowledge and expertise is an area which distinguishes collaborative intelligence from (widely used) statistical machine learning approaches. In cases where insufficient data is available, or where the data lead to incorrect conclusions, machine learning is not reliable and human insight is required. For example, in the emergence of a previously unseen malware threat, a human analyst could use knowledge (e.g. of a zero-day exploit) to identify the likely behaviour before a statistically significant body of data has been gathered to train a machine learning system. An example of incorrect conclusions could come from records of card-based entry / exit barriers where "tailgating" occurs, i.e. an individual follows someone else through the barrier without swiping their card. Such data will give misleading behaviour patterns.

The issues involved in discerning event sequences are strongly linked to the concept of information granulation introduced by Zadeh [2] 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 a degree) equivalent. 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 properties (event C is *very similar* to event D because both involve *similar entities/activities*).
However, at the same time it is necessary to recognise that most machine-based algorithms require crisp, well-defined boundaries when processing data. In this work, we use the $X - \mu$ method to translate consistently between the (generally

fuzzy) human knowledge representation and the (generally crisp) data required by machines. We describe a compact and expandable *sequence pattern* representation, which allows the addition of new event sequences as they are identified, and subtraction of sequences that are no longer relevant. The main contribution of the paper is the presentation of the incremental algorithms to add and remove sequences. Although the algorithm for sequence addition was presented briefly in [3], this paper contains a more detailed explanation. The algorithm for sequence subtraction has not been previously published (other than patent [4]).

2 Background

2.1 X-Mu Approach - Conversion between Fuzzy and Crisp

Human intelligence includes the ability to identify a group of related entities (e.g. physical objects, events, abstract ideas) and to subdivide them into smaller subgroups at an appropriate level of granularity for the task at hand. Such groups are rarely specified by "necessary and sufficient" conditions, but are better modelled by membership functions, where we can compare different entities and judge whether one belongs more strongly to the set than another. 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. Within a universal set, the absolute value of the membership function for an element is generally less important than the relative value, compared to other elements. Whilst the end points 0 and 1 obviously correspond to classical non-membership and full membership in the set, other values are most useful in comparing the strength of membership (e.g. Bill Gates belongs more strongly than Larry Ellison to the set of *rich people*). In this interpretation of fuzzy sets, there is an underlying assumption that membership values are *commensurable*, i.e. that membership of (say) 0.8 in the set of *rich people* can be interpreted in the same way as membership of 0.8 in the set of *tall people* or membership 0.8 for a temperature value in the set of temperatures *near-freezing*. Such commensurability is routinely assumed in fuzzy control applications (for example, an inverted pendulum where a membership in a set of cart velocities might be combined with membership in a set of angular accelerations). We adopt the commensurability assumption in this paper. The interested reader is referred to [5], [6].

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 fuzzy *interval*. For example, calculating the average age of four employees known to be 20, 30, 50 and 63 is inherently simpler than when the ages are given as *young, quite young, middle-aged* and *approaching retirement*. In the latter case, we must handle interval arithmetic AND membership grades. In a similar fashion, querying a database to find employees who are aged over 60 is simpler than finding employees *approaching retirement age*.

The $X - \mu$ method [6] recasts the fuzzy approach as a mapping from membership to universe, allowing us to represent a set, interval or single value that

varies with membership, e.g. the mid-point of an interval. This natural idea is difficult to represent in standard fuzzy theory, even though it arises frequently e.g. the cardinality of a discrete fuzzy set or the number of answers returned in response to a fuzzy query.

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 approach described in the next section assumes we have crisp equivalence classes. We allow fuzziness in the definition of sets used by human experts, and use the $X - \mu$ method to ensure we have crisp sets, by working at a specific membership level. The $X - \mu$ method also allows us to work with *intensional* definitions of equivalence classes (parameterised by membership), but we do not cover this aspect here.

3 Directed Acyclic Sequence Graphs (DASG) : Graph Representation of Event Sequences

For any sequence of events, we create a directed graph representation 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 equivalent 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 DAWG (directed acyclic word graph) [7]. 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 (viewed as a variant of dictionary storage). Most methods (e.g. [8, 9]) assume that all words (letter sequences) are available and can be presented to the algorithm in a specific order. Sgarbas [7] 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 data is presented in a standard object-attribute-value table format (e.g. CSV), with additional attributes calculated as necessary. We assume that the 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

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	14:30	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	12:30	12	c	out
18	jan-3	14:30	12	c	in
19	jan-3	15:00	12	c	out

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.

An example, used throughout the rest of the paper, is shown in Table 1. This is a small subset of benchmark data taken from mini-challenge 1 of the VAST 2009 dataset[1], which 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.

The DASG representation assumes that we can subdivide the attributes into the following categories:

- Event identifier - a key value which uniquely identifies a row of the table. In the example, *eventID* takes this role.
- Event Sequencer - one or more attributes with an associated total order, used to determine whether one event precedes or succeeds another. In the example, *Date* or *Time* or both could take this role.
- Event Linkage - one or more attributes with an equivalence relation that determines whether two events are linked (part of the same sequence). For example, in Table 1 we define a sequence of events involving the same employee, where there is no more than 8 hours between contiguous events.
- Event Categorisation - one or more attributes with an equivalence relation that determines whether two events (in different sequences) can be considered as examples of the same event category. For example, we might group

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

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	

together events that happen at approximately the same time, and/or involving the same swipecard actions (building+in, classified+in or classified+out). The definition of *approximately the same time* can be fuzzy, but must be made crisp (via $X - \mu$) to define the equivalence relation.
- Recorded Data - for subsequent analysis, we can record one or more of the attributes associated with an event. This may be as simple as counting the number of instances, or may involve more sophisticated processing such as association rules between events.

There is no restriction on the number of attributes. 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. 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 first define the linkage relations, to detect candidate sequences. Here, for a candidate sequence of n events:

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

we define the following computed quantities :

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

and restrictions (for $j > 1$) :

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

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

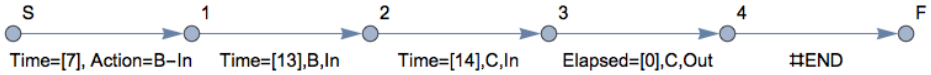

Fig. 1. DASG representation of the first sequence (events 1-2-3-4) from Table 1. The labels show the categorisation attributes, namely the equivalence class of the event time and the entrance and direction.

– successive events in a single sequence conform to allowed transitions between locations and are on the same day, within a 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 linkage 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,

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

where I is the identity relation and the notation [7] represents the set of start times from 7:00-7:59. As mentioned in Section 2.1, fuzzy equivalence classes are converted to crisp sets at a specific membership or to intensional definitions, parameterised by membership. We represent each identified sequence as a path labelled by its event categorisations (Fig 1) The algorithms presented in the next section allow us to incrementally add unseen sequences into a minimal DASG which represents exactly the set of sequences seen so far (Fig 2). 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, or 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
$StartOfSimilarEdge(edge, endnode)$

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 algorithm proceeds in three distinct phases (corresponding to the three *while* loops in Figure 2. In the first and second parts, we move step-by-step through the new event sequence and the graph, beginning at the start node S. If an event categorisation matches an outgoing edge, we follow that edge to the next node and move on to the next event in the sequence. If the new node has more than one incoming edge, we must copy[2] 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 at some point, 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. Note that 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) omitted. Finally, in the third stage, we search for sequences with one or more common ending events. Where possible, the paths are merged.

The advantage of this algorithm is that it allows *incremental* modification, so that new sequences can be added at any time. Although the example shows the sequence patterns derived from data, it is also possible for a human expert to specify and add a sequence pattern without it having been seen in the data. Hence (for example) a previously unseen cyber-attack sequence could be added to the DASG and the matching event sequence would be detected as soon as it occurred. In contrast, a purely data-driven method would first flag the new sequence as unknown (not matching any pattern in the graph), and would only recognise subsequent occurrences after graph updating.

The representation also allows straightforward removal of sequences. For example, if it is known that a sequence will never be seen again because it is screened out by a different process, or because external changes make it impossible, then processing and storage efficiency are improved by removing the sequence pattern from the graph. However, we must take care not to remove any edge which is part of another pattern. Figure 4 shows the algorithm. The process is straightforward - we consider nodes where the indegree ≥ 1 , and the outdegree ≥ 1 (four possibilities).

[2] The copy and merge operations are also used when removing sequences

```
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. Algorithm to extend a minimal graph by incremental addition of a sequence of edges

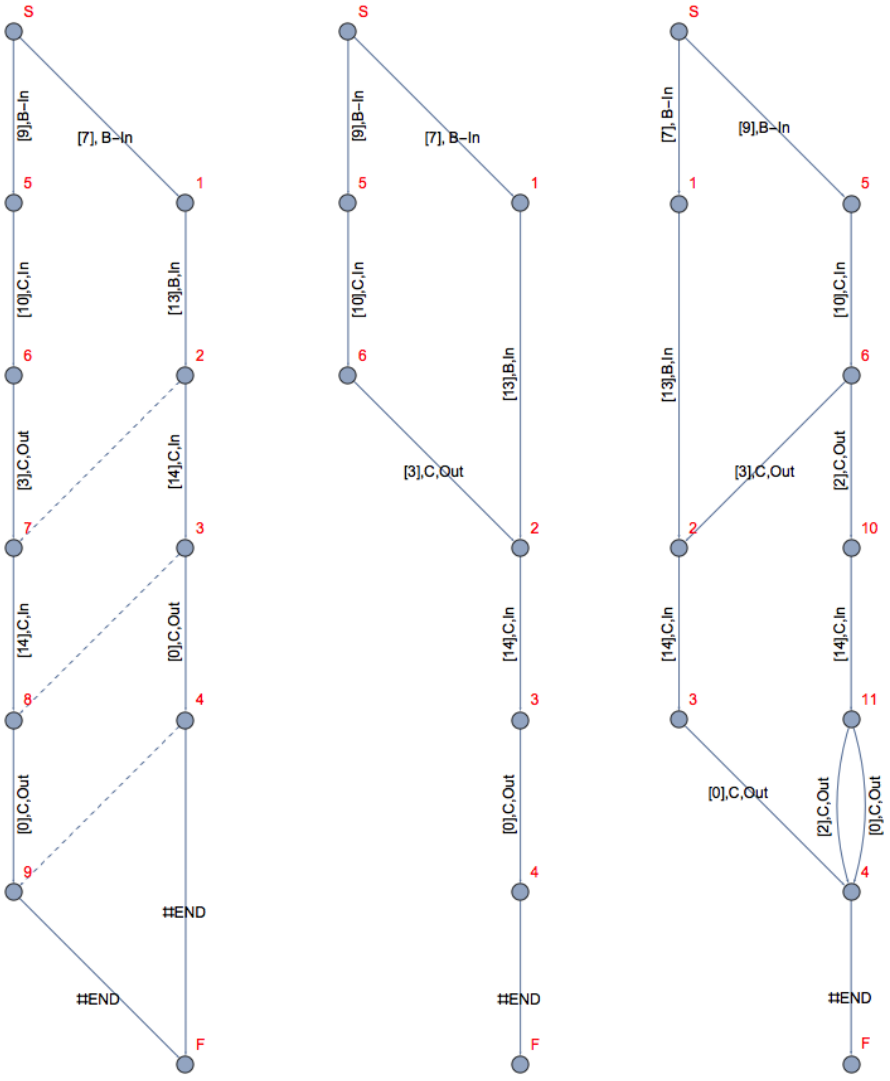

Fig. 3. Adding further sequences : (left) graph after phase 1 of adding the second sequence, dotted lines indicate nodes which are identical in phase 3; (centre) final graph (after identical nodes have been merged) representing sequences 1 and 2; (right) graph representing all four sequences

```
Algorithm ReduceGraph
Input :Graph G, start node S, end node F, the current DASG (minimal)
       Sequence C[0 - N_Q]  representing the sequence of event categories
       to be removed. Each element is an event categorisation i.e. a
       start-node and C[i] uniquely specifies an edge.
       NB the sequence must be present in the graph and there must be at
       least one sequence in the graph after removal.

Output : updated minimal graph, excluding the removed sequence

Local variables :  Node startNode, endNode, potentialDuplicates (LIST)
       Edge currentEdge
       integer seqCounter;

startNode = S
endNode =  F
seqCounter = 0
currentEdge = (startNode, C[0])

WHILE  endNode > startNode   AND  seqCounter < N_Q

    IF  OutDegree(currentEdge.END) > 1 AND InDegree(currentEdge.END)==1
    THEN startNode =  currentEdge.END
    ELSE
        IF OutDegree(currentEdge.END)==1 AND InDegree(currentEdge.END) > 1
        THEN endNode =  currentEdge.END
    ELSE
        IF OutDegree(currentEdge.END) >1 AND InDegree(currentEdge.END) >1
    THEN
        ADD currentEdge.END to potentialDuplicates
    ENDIF
    currentEdge = (currentEdge.END, C[seqCounter])
    seqCounter++
ENDWHILE

IF (endNode > startNode)
THEN
    Duplicate each node in potentialDuplicates
    Delete each edge in Sequence[] from startNode to endNode
    Apply 3rd phase of ExtendGraph (merge final edges)
ENDIF
```

Fig. 4. Algorithm to reduce a minimal graph by incremental removal of an existing sequence of edges

- If a node has indegree one and outdegree greater than one, then the path up to and including this node must be retained, because it contributes to paths other than the path to be deleted. In this case, any potential duplicates up to this point can be discarded as the nodes are not altered.
- If a node has indegree greater than one and outdegree equal to one, then the path from this node to the end must be retained, because it contributes to paths other than the path to be deleted.
- If a node has indegree one and outdegree one, it (and its incoming and outgoing edges) can potentially be removed, depending on the path.
- Finally a node with both in- and out- degree greater than one might require modification and is marked as a potential duplicate.

4 Summary

The DASG representation allows us to store event sequence patterns in a compact directed graph format, with efficient incremental algorithms to add a previously unseen pattern to the graph, and to remove a pattern from the graph. Sequence patterns can be generated from data or by a human expert. The DASG representation allows fuzzy specification of categories and equivalence relations, which are converted to crisp relations using the $X - \mu$ approach. An efficient implementation of the DASG is possible by compiling the graph into a set of instructions for a virtual machine (described in [4], [10]).

References

1. L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.
2. 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.
3. T. P. Martin and B. Azvine. Representation and identification of approximately similar event sequences. *Flexible Query Answering Systems, FQAS2015*, pages 87–99, 2015.
4. T P Martin and B. Azvine. Patent pct/gb2014/000378 : Sequence identification, 2014.
5. Didier Dubois and Henri Prade. Gradualness, uncertainty and bipolarity: Making sense of fuzzy sets. *Fuzzy Sets and Systems*, 192:3–24, 2012.
6. T. P. Martin. The x-mu representation of fuzzy sets. *Soft Computing*, 19:1497 – 1509, 2015.
7. 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.
8. D. Revuz. Minimization of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92:181 – 189, 1992.
9. J. E. Hopcroft and J. D Ullman. *Introduction to Automata Theory, Languages, and Computation,*. Addison-Wesley, Reading, MA, 1979.
10. T. P. Martin and B. Azvine. A virtual machine for event sequence identification using fuzzy tolerance. In *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, page to appear. IEEE Press.