Pattern Graphs: A Knowledge-Based Tool for Multivariate Temporal Pattern Retrieval

Sebastian Peter*, Frank Höppner† and Michael R. Berthold*

*Nycomed-Chair for Bioinformatics and Information Mining, University of Konstanz, Box 712, 78457 Konstanz, Germany.
†Department of Computer Science, Ostfalia University of Applied Sciences, 38302 Wolfenbüttel, Germany.

***Abstract*—We introduce a new, powerful query formulation formalism for complex, multivariate sequence data. The new query language, termed *pattern graphs*, is capable of reflecting more aspects of temporal patterns than earlier proposals. The underlying graph structure of the pattern graph makes the query intuitive to use and therefore understandable not only for the data analyst. We present algorithms to match patterns against data and demonstrate its usefulness on real data from the automobile industry.**

I. Introduction

In this paper we propose a query language that enables users to specify complex patterns over multivariate sequential data and retrieve matches of such patterns on real data. Although data mining literature already offers some approaches to discover patterns in time series automatically, our experience is that these approaches are often difficult to communicate to the domain expert and are too limited to express the rich expert knowledge that already exists. Ignoring the existing knowledge may initiate excessive data mining, involving time-consuming manual scanning of results and long discussions regarding patterns that were already known to the expert. Therefore we seek a pattern language that is easily comprehensible and expressive at the same time. We focus on multivariate sequential data, because experts quickly start to connect different aspects in their argumentation and the pattern should be able to reflect such connections appropriately.

In the following section we provide an overview of the related work. In section III we present the notion of a *pattern graph* as the query model. Section IV discusses the algorithm that finds matches of the pattern in multivariate sequential data. In section V we show the application of our approach on a real-life problem. We conclude the paper in section VI.

II. Related work

Many tools exist that deal with univariate time series, for example [1] presents an approach where the user places rectangles in the time-value space to filter those series that do not pass through these rectangles. Although easy to understand, the expressiveness of this approach is limited, because the expert must restrict the time series in the (absolute) value *and* (absolute) time *simultaneously*. Multivariate features (not necessarily time series only) are often plotted row by row: in Fig. 1 we can see an example with four properties, where the black rectangles indicate the periods in time at which some

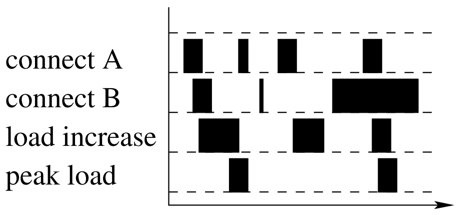

Fig. 1. Representation of multivariate, sequential data: the black rectangles denote the intervals when the predicate (labels to the left) holds.

property holds. The expert may find interesting patterns by recognising the *temporal relationship between feature occurrences*, rather than expressing their occurrences in terms of absolute time points. Such a representation has turned out to be useful, e.g. in the medical domain [2]. Various ways to define

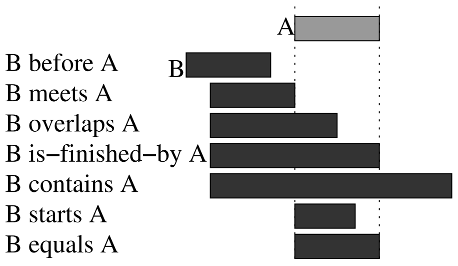

Fig. 2. Thirteen possible relationships between two intervals. The inverse relationships (after $\leftrightarrow$ before) have been omitted.

patterns in this notion have been proposed in literature, many of them relying on Allen’s interval relationships [3] (cf. Fig. 2) or variants thereof.Some approaches (e.g. [4]) define a pattern by specifying the exact relationship for every pair of intervals. In [5] it is argued that a full pattern specification via Allen’s relationships is overly strict and a partially ordered sequence of simultaneous (sub-) intervals is proposed. See also [6] for an overview.

While these different representations have their individual strengths, they also have their weaknesses: Thinking of predicting a certain state of some network server (breakdown, overload, attack, malfunction, etc.) on the history of, say, the last 24 hours, a situation as simple as “there was only one connection to server A” (during the last n hours) is hard to express for the above-mentioned pattern languages. A situation like “there was a connection to B while the connection to A was lost" is impossible to represent for approaches that rely on an explicitly given interval relationship, as the exact position of B relative to A is not known [4]. Temporal constraints "the connection to A was lost for at least 4 hours" or "... at most 4 hours" are usually ignored completely or introduced in a post-processing step. In our earlier work [7] we introduced temporal constraints to address this problem, but we were still not able to formulate overlapping temporal constraints such as "the connection A was lost for at least 4 hours and during that time connection B was lost for at least 30 minutes". Such expressions are, however, frequently used by experts when arguing about a course of events. The formalism introduced in the next section addresses these shortcomings using a flexible graph structure which allows to model parallel dependencies, partial order of events and provides better support for temporal constraints.

III. Pattern Graphs

In this section, we formalize the notion of a *pattern graph*, which allows to express constraints on the behaviour of multivariate, sequential data. The pattern graph will be used to query the given sequential data. We assume that m attributes (rational, binary or categorical) are given and each of them has a value range denoted by D_j. By $\vec{s} \in D$ with $D = (D_1 \times \cdots \times D_m)$ we denote all m measurements at one point in time.

Definition 1 (sequence). *A sequence S consists of an arbitrary number of data vectors $(\vec{s}_1, \ldots, \vec{s}_n) \in \mathcal{S}$ with $\mathcal{S} = \bigcup_{i=1}^{\infty} D^i$. Thus, $\mathcal{S}$ defines the set of all sequences. Let $|S| = n$ denote the length of the sequence S. The index i of an element $\vec{s}_i$ serves as an (integer-valued) time point.*

For example a sequence S with attributes A, B and C may look like this:

$$\begin{array}{c} A \\ B \\ C \end{array} \begin{pmatrix} true & true & false & false \\ 3.4 & 8 & 19 & 20 \\ high & middle & low & low \end{pmatrix}$$

A has a boolean value range, B a nominal and C a categorical value range. This series has four data vectors ($|S| = 4$) which correspond to the columns of the table.

Definition 2 (Subsequence). *A subsequence from index a to b from the sequence S is defined as $S|_{[a,b]}$.*

So the subsequence $S|_{[1,2]}$ consists of the first two columns of S:

$$\begin{array}{c} A \\ B \\ C \end{array} \begin{pmatrix} true & true \\ 3.4 & 8 \\ high & middle \end{pmatrix}$$

A *pattern graph* is an acyclic, directed graph with one source ($\top$) and one sink ($\bot$). The nodes of the graph carry the constraints for the sequence. A sequence matches (fits) the pattern graph if it is possible to assign subsequences from S to all nodes of the graph (called mapping), so that the subsequences satisfy the constraints stated by the corresponding nodes. The assignment has to be complete in the sense, that (1) each node is given a subsequence, (2) the subsequences assigned to connected nodes are contiguous, (3) the (empty) prefix of S is assigned to the source, the (empty) suffix of S to the sink node.

Semantic. The pattern graph divides the sequence into several parts and each part has to satisfy the respective constraints. These parts are represented by the nodes, where the constraints are given by the nodes value and temporal constraints. The temporal constraints restrict the length and the value constraints the behaviour of the respective subsequence. The edges between the nodes represent the order of the parts. If a node has an outgoing edge it means that directly after the associated subsequence there has to be another part that fulfils the constraints of the successor node. If a node has two or more outgoing edges, all parts belonging to the following nodes have to begin at the same time. On the other hand, if a node has two or more incoming edges all parts belonging to the preceding nodes have to end at the same time and the part of the node has to begin directly afterwards.

Graphical representation. Before we can show an example graph, we have to specify the graphic representation of a pattern graph. In Fig. 3 we can see the example pattern graph with the following meanings:

1) The temporal constraint of a node is represented above the node. We only consider temporal constraints on the duration in this paper, therefore we show the interval of valid node durations. A star represents an unlimited duration.
2) The value constraint(s) of a node are shown inside the node.
3) If the node has the '*don't care*'-constraint ($\equiv$ no constraints) the node is labelled '?'.

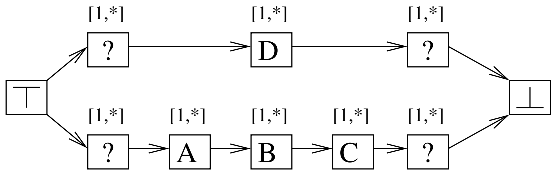

Fig. 3. Example pattern graph with two parallel paths from $\top$ to $\bot$.

Example. Fig. 4 shows two sequences where the vertical axis denotes some binary properties A-D that hold over certain periods of time (black bars, time on horizontal axis). We now want to show if these sequences can be validly mapped to the pattern graph in Fig. 3.

The graph shown in Fig. 3 can be decomposed into two different paths: For the lower path the sequence has to be divided in five contiguous parts, so that the first part satisfies the '*don't care*' constraint, during the second part the property A has to hold, B in the third, etc. The last part is again a '*don't care*'-part. All of these five parts require a duration of at least one time unit (but have no upper bound on the duration). In parallel to the lower path, the upper path requires '*don't care*', 'D' and '*don't care*' again with durations between 1 and ∞ time units.

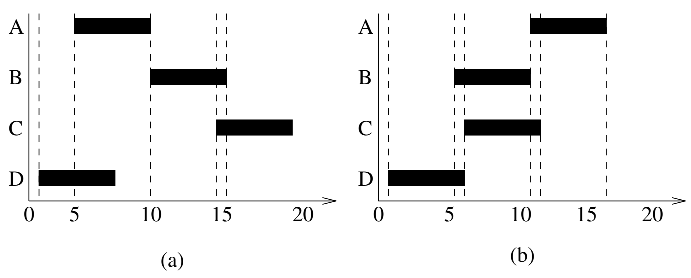

Fig. 4. Two example sequences with four binary properties A-D.

The sequence shown in 4(a) has valid mappings on the graph, because we can clearly see the A before B before C relation. And D is present during the sequence as well. Due to the fact that B and C are overlapping, it is possible to assign different subsequences to the B and C node. This means that the pattern graph has more than one valid mapping. On the other hand we cannot find a valid mapping for the sequence shown in Fig. 4(b), because we cannot find the relation A before B. If A were true within $[6, 9]$ (rather than $[10, 15]$), we would have another valid mapping.

Possible graph constructs. In Sec. II we mentioned that the pattern graph is able to express partial order of events and can deal with overlapping temporal constraints. In this paragraph we show the graph constructs which enable these situations. In order to allow partial ordering the graph shown in Fig. 5 is needed. The graph requires A before C and B before C but the

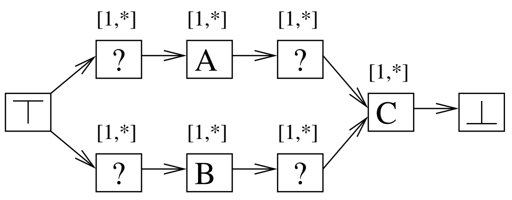

Fig. 5. Example pattern graph with the partial order construct: A and B before C and no explicit relation between A and B.

relationship between B and C is not defined, thus the relation could be any one of the 13 allen's relations.

To model the situation "the connection A was lost for at least 4 hours and during that time connection B was lost for at least 30 minutes" the pattern graph in Fig. 6 could be used.

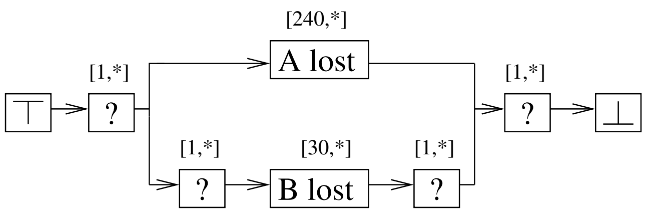

Fig. 6. Example pattern graph showing two overlapping temporal constraints

Due to the two parallel paths between the two '*don't care*' nodes, the lost connection of A and B have to occur together at some point of time. The two additional '*don't care*'constraints in the "B lost" path allow an occurrence of "B lost" somewhere during "A lost". Finally the length of the connection losses are modeled with the temporal constraints $[240, *]$ and $[30, *]$.

Formal definition. The following four definitions provide a more detailed and formal definition of the pattern graph.

Definition 3 (constraint)**.** *Let $\mathcal{C} = \{C \mid C : \mathcal{S} \to \mathbb{B}\}$ denote the set of all possible constraints for (sub)sequences. We distinguish between value constraints and temporal constraints. A value constraint restricts the values of a subsequence, whereas a temporal constraint narrows the acceptable length of the subsequence.*

Examples for value constraints on a (sub)sequence $S = (\vec{s}_1, \ldots, \vec{s}_n) \in \mathcal{S}$ are:

- $C(S) = \text{true}$ — This constraint is always satisfied and will be denoted by '?' or '*don't care*'.
- $C(S) = (\forall i, j : 1 \le i \le n, 1 \le j \le m : (\vec{s}_i)_j \in D'_j)$ for some $D'_j \subseteq D_j$, $1 \le j \le m$. This constraint holds if all values remain in the specified ranges D'_j.

An example of a temporal constraint is:

- $C(S) = (a \le |S| \le b)$ for $1 \le a \le b$. This constraint holds, if the duration of the sequence lies within a valid range $[a, b]$ of durations. We say that the temporal constraint C is induced by the interval of valid durations $[a, b]$. This will be the only temporal constraint we consider in this paper.

In our example sequence, the value constraint "*A has to be true*" holds for $S|_{[1,2]}$ but not for the whole sequence S, because at time point three and four the attribute A takes the value 'false'.

Definition 4 (pattern graph)**.** *A pattern graph M is a tuple $(V, E, \mathcal{C}_{val}, \mathcal{C}_{temp})$, where (V, E) represents an acyclic directed graph with a finite node set $V \subseteq \mathbb{N} \cup \{\top, \bot\}$ and the edge set $E \subseteq (V \times V)$. Furthermore the graph has the following properties:*

- $\forall (v, w) \in E : w \ne \top$
  (Node $\top$ has only outgoing edges.)
- $\forall (v, w) \in E : v \ne \bot$
  (Node $\bot$ has only incoming edges.)
- $\forall v \in V \backslash \{\top, \bot\} : (\exists w \in V : (v, w) \in E) \wedge (\exists w \in V : (w, v) \in E)$ *(All nodes $v \in V \backslash \{\top, \bot\}$ have at least one incoming and outgoing edge.)*

$\mathcal{C}_{val}$ and $\mathcal{C}_{temp}$ are maps that assign a value constraint and a temporal constraint, resp., to each node $v \in V \backslash \{\top, \bot\}$. For simplification, let $C^v_{val} = \mathcal{C}_{val}(v)$ and $C^v_{temp} = \mathcal{C}_{temp}(v)$.

Definition 5 (mapping)**.** *Let a sequence S and a pattern graph $M = (V, E, \mathcal{C}_{val}, \mathcal{C}_{temp})$ be given. By $\mathcal{I}$ we denote the set of all intervals lying within $[1, |S|]$. A mapping $B : V \to \mathcal{I}$ assigns to each node $v \in V \backslash \{\top, \bot\}$ a contiguous subsequence of S. Thus $B(v) := [a, b]$ denotes the start and end index of the associated subsequence that is mapped to node v (v is mapped to $S|_{B(v)}$). The fictitious subsequence $S|_{[0,0]}$ is mapped to $\top$ and $S|_{[|S|+1,|S|+1]}$ to $\bot$.*

Definition 6 (valid mapping). *A valid mapping of a pattern graph $M = (V, E, \mathcal{C}_{val}, \mathcal{C}_{temp})$ and a sequence $S = (\vec{s}_1, ..., \vec{s}_n)$ is a mapping B with the following additional properties ($V' = V \backslash \{\top, \bot\}$):*

1) $\forall (v, w) \in E, B(v) = [a, b], B(w) = [c, d] : b + 1 = c$ *(no gaps)*
2) $\forall i : 1 \leq i \leq |S| : \exists v \in V' : i \in B(v)$ *(each index is assigned to at least one node)*
3) $\forall v \in V' : C^v_{val}(S|_{B(v)}) = true$ *(all value constraints satisfied)*
4) $\forall v \in V' : C^v_{temp}(S|_{B(v)}) = true$ *(all temporal constraints satisfied)*

IV. Matching

In this section we explain how to find valid mappings (matches) of a sequence to a given graph. For the remainder of this section, we assume a pattern graph $M = (V, E, \mathcal{C}_{\text{val}}, \mathcal{C}_{\text{temp}})$ and a sequence S is given. Furthermore, for the sake of an efficient matching algorithm, we restrict ourselves to *closed value constraints*, which additionally satisfy the following condition: $C(S|_{[a,d]}) \Rightarrow \forall a \leq b \leq c \leq d : C(S|_{[b,c]})$ (if C holds on $S|_{[a,d]}$, it does also hold on all subsequences $S|_{[b,c]}$).

A. Preliminaries

Definition 7 (valid node locations). *By $T_N(v)$ for a node $v \in V$ we address the set of all time indices that satisfy the value constraint C^v_{val} associated with v. For the special nodes $\bot$ and $\top$ we declare $T_N(\top) = \{ 0 \}$ and $T_N(\bot) = \{ |S|+1 \}$. Such constraints typically hold over a period of time indices, they may therefore be written as a set of intervals.*

Up to now, we know that for any valid mapping B we have $B(v) \subseteq T_N(v)$. A time period $[a, b] \subseteq T_N(v)$, however, may be unsuitable for $B(v)$, if v is connected to another node w (by an edge $(v, w) \in E$), but $b + 1 \notin T_N(w)$. For a mapping to be valid we have to ensure that all constraints hold at node crossings.

Definition 8 (valid edge locations). *Let $(v, w) \in E$, let $[a, b]$ induce C^v_{temp} and $[c, d]$ induce C^w_{temp}. The set of all valid edge positions $T_E(v, w)$ for edge (v, w) is defined as the set of all time indices t, where C^v_{val} holds for at least a time units up to time t and C^w_{val} holds for c time units after time t at least. More formally:*

- $T_E(\top, w) = \{0\}$ *if* $C^w_{val}(S|_{[1,c]})$*, otherwise* $\emptyset$
- $T_E(v, \bot) = \{|S|\}$ *if* $C^v_{val}(S|_{[|S|-a+1,|S|]})$*, otherwise* $\emptyset$
- $t \in T_E(v, w) \Leftrightarrow C^v_{val}(S|_{[t+1-a,t]}) \wedge C^w_{val}(S|_{[t+1,t+c]})$

Now, for any valid mapping B, we have further restricted the possible outcome of $B(v)$ to some $[a, b] \subseteq T_N(v) \cap T_E(v, w)$ for $(v, w) \in E$. In fact, if there is more than one outgoing edge, say (v, w) and (v, w'), the possible edge positions have to fulfil even more constraints. If w' itself has a connection (v', w') this also influences possible values for the end time b of $B(v) = [a, b]$. We collect all edges that influence the valid position of an edge in a group:

Definition 9 (edge group). *The set $G(v, w)$ for $(v, w) \in E$ is implicitly defined by:*

1) $(v, w) \in G(v, w)$
2) $(x, y) \in G(v, w) \Leftrightarrow (x, y) \in E \wedge \exists (u, z) \in G(v, w) : u = x \vee z = y$

By $\mathcal{G} = \{G(v, w) \mid (v, w) \in E\}$ we denote the set of all edge groups of the pattern graph. This means that an edge group of an edge consists of the edges that could be reached by an alternating path of forward and backward edges.

The definition of an edge group is based on the '*no gap*' constraint of the pattern graph: all edges that are reachable via a path of alternating forward and backward edges, beginning by an arbitrary edge of the edge group, influence each other wrt. valid node locations.

This leads us to definition 10.

Definition 10 (valid edge group location). *Given $E' \in \mathcal{G}$, the set of all valid edge group locations $T_G(E')$ is defined by those points in time that fulfil all constraints on all individual edges of the group: $T_G(E') = \bigcap_{(v,w)\in E'} T_E(v, w)$.*

B. Matching-algorithm

With the preliminary definitions we are now able to introduce the graph matching algorithm (Alg. 1). As it comes to implementation, we have to settle on the data structures for the various sets of time points (such as T_N, T_E, T_G). As mentioned earlier, chances are high that the time points in these sets are not disconnected but lumped together (just consider for example the value constraints $x > 3$, which usually hold over a period of time). We therefore use sequences of intervals as data structures for the above mentioned sets. Note, that an interval $[3, 5]$ actually refers to the set $\{3, 4, 5\}$ because our time dimension is discrete, and that a single time point 3 will be encoded as an interval $[3, 3]$. A set $T = \{1, 2, 3, 4, 7, 8, 9\}$ is thus represented by a minimal set of intervals $R = \{[1, 4], [7, 9]\}$. The set is minimal in the sense that we have no duplicate entries (whenever two intervals have a nonempty intersection, we replace it by its union).

Algorithm 1 graph matching

```
Require: Pattern Graph $M = (V, E, \mathcal{C}_{\text{val}}, \mathcal{C}_{\text{temp}})$, sequence $S$,
    nodes $v \in V$ in topological order.
Ensure: valid mappings
1: calculate valid node locations $T_N(v)$, $v \in V$
2: calculate valid edge locations $T_E(e)$, $e \in E$
3: determine edge groups $\mathcal{G}$
4: calculate valid edge group locations $T_G(E')$, $E' \in \mathcal{G}$
5: combine values from edge group locations to mappings
6: for all mappings do
7:     check mapping
8: end for
9: return valid mappings where $\neg\exists E' \in \mathcal{G} : T_G(E') = \emptyset$
    holds
```

The algorithm basically computes the sets of locations as they were defined before. The valid node locations are easily obtained by scanning through the sequence once. The edge locations can be derived (cf. Alg. 2) from the node locations by applying a special shrinking operation to the interval sets:

$$\text{shrink}_{l,r}(T) := \{[a+l, b-r] \mid [a,b] \in T\}$$

For instance, $\text{shrink}_{1,0}(\{[1,4],[7,9]\}) = \{[2,4],[8,9]\}$ shrinks all intervals by one time unit at the beginning, or $\text{shrink}_{2,1}(\{[1,4],[7,9]\}) = \{[3,3]\}$ shrinks at both ends. In the second example, one of the intervals vanished completely.

Algorithm 2 calculate valid edge locations

```
1: for all $(v,w) \in E$ do
2:     let $C^v_{\text{temp}}$ be induced by $[a,b]$
3:     let $C^w_{\text{temp}}$ be induced by $[c,d]$
4:     $T_E(v,w) \leftarrow \text{shrink}_{a-1,0}(T_N(v)) \cap \text{shrink}_{-1,c}(T_N(w))$
5: end for
```

A position p to switch from one node v to node w (according to an edge $(v,w) \in E$) is only valid if the value constraints C^v_{val} hold for a sufficiently long period before p and C^w_{val} for sufficiently period long after p. The sets $T_N(v)$ and $T_N(w)$ contain the locations that fulfil the value constraints of the node, so we just have to ensure that the subsequence assigned to v/w will be long enough (addition or subtraction of the minimal temporal constraints). By shrinking the sets of valid node locations by the minimal duration (given by the temporal constraint) we assure that all remaining time points may serve as an edge location.

Algorithm 3 calculate valid edge group locations

```
1: for all $E' \in \mathcal{G}$ do
2:     $T_G(E') \leftarrow \bigcap_{e \in E'} T_E(e)$
3: end for
```

The next step is shown in Alg. 3, where we compute the valid edge group locations as expected. This step is relatively simple because we have already computed the possible positions for each edge.

At this point we have one or more continuous intervals with valid edge locations for each edge group. In principle, we may select an edge location for each edge group, which gives us the subdivision of the original sequence S in the desired parts. However, there are some aspects that have not yet been covered: (1) the upper bound of the temporal constraints and (2) some edge locations (although valid) may prevent us from mapping the pattern completely to the sequence (e.g. because there is nothing left to match the remainder of the pattern). These two aspects are covered in algorithm 4 by applying a forward and backward sweep, similar to critical path planning [8]. The main idea is that we go through the nodes of the graph in a topological order and propagate the reachable positions (within the temporal constraint and satisfying the value constraint) from the incoming edge positions to the outgoing edge positions.

Algorithm 4 check mapping

```
1: repeat
2:     for all $v \in V$ (topological order) do
3:         for all $(v,w) \in E$ do
4:             let $C^v_{\text{temp}}$ be induced by $[a,b]$
5:             $T'_E \leftarrow \{[t_1+a, t_2+b] \mid [t_1,t_2] \in T_G(G(\cdot,v))\}$
6:             $T'_G \leftarrow T_G(G(v,w)) \cap T'_E$
7:             $T_G(G(v,w)) \leftarrow \{\ p \in T'_G \mid \exists t \in T_G(G(\cdot,v)) \wedge$
               $\exists t' \in T_N(v) : t \cap t' \cap p \neq \emptyset\ \}$
8:         end for
9:     end for
10:    for $v \in V$ (reverse topological order) do
11:        for all $(w,v) \in E$ do
12:            let $C^v_{\text{temp}}$ be induced by $[a,b]$
13:            $T'_E \leftarrow \{[t_1-b, t_2-s] \mid [t_1,t_2] \in T_G(G(v,\cdot)\ \}$
14:            $T'_G \leftarrow T_G(G(w,v)) \cap T'_E$
15:            $T_G(G(w,v)) \leftarrow \{\ p \in T'_G \mid \exists t \in T_G(G(v,\cdot)) \wedge$
               $\exists t' \in T_N(v) : t \cap t' \cap p \neq \emptyset\ \}$
16:        end for
17:    end for
18:    if $\exists E' \in \mathcal{G} : T_G(E') = \emptyset$ then
19:        return
20:    end if
21: until $T_G(E')$, $E' \in \mathcal{G}$ do no longer change
```

For a concrete mapping we have to pick a position for an arbitrary edge and then iteratively pick the position for the next edges, but we also have to keep the temporal constraint satisfied.

A detailed proof of correctness is beyond the scope of this paper. But we want to explain the main idea behind the proof. We start with all possible mappings and in each step we remove only invalid mappings. The first five steps of algorithm 1 prepare the final check, which is done by algorithm 4. During the forward sweep, positions (for an outgoing edge) may be discarded because of two reasons: First we may not find a suitable position in the incoming edge position to satisfy the temporal constraint. And secondly because the subsequences created by two consecutive edge positions do not satisfy the value constraint. In both cases the removed positions cannot be a part of a valid mapping. Note that during the forward and backward sweep the value constraint of a node is checked by testing to see if both edge positions intersect with the same interval of the valid node locations of the enclosed node. This is only possible because of the restriction to *closed value constraints*. Another important aspect in the forward and backward sweep is topological ordering. This ensures that we only further propagate from an edge group, once all of its predecessors have already been dealt with. Where in the crucial path analysis only one forward and backward sweep suffices, we have to repeat the sweep multiple times. (During the backward sweep, positions are removed but they could be necessary to reach other edges during the forward sweep).

V. Application

We applied our pattern graph to real world data from a German car manufacturer. Several cars were equipped with recording devices that captured various measurements, such as current speed, gear, pedal state and angles, etc. One (intermediate) goal is to identify driving cycles with a specific duration in the data, which appears pretty simple at first glance. (Once these cycles have been extracted, they will form the basis of subsequent research.) In a test-bed situation, a driving cycle may be defined as a sequence of acceleration, constant speed and deceleration. However, if we define 'cycle' by such a pattern (cf. Fig. 7), it matches far more situations than the experts actually had in mind.

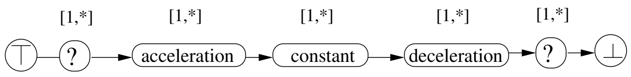

Fig. 7. First pattern graph to query driving cycles.

In figure 8 we see some of the matches marked by a black rectangle, many of them not qualifying as a driving cycle, e.g. the small rectangle at high speed in the middle. One problem is, of course, the choice of a threshold that distinguishes *constant speed* from ac-/deceleration, another aspect may be duration of the acceleration.

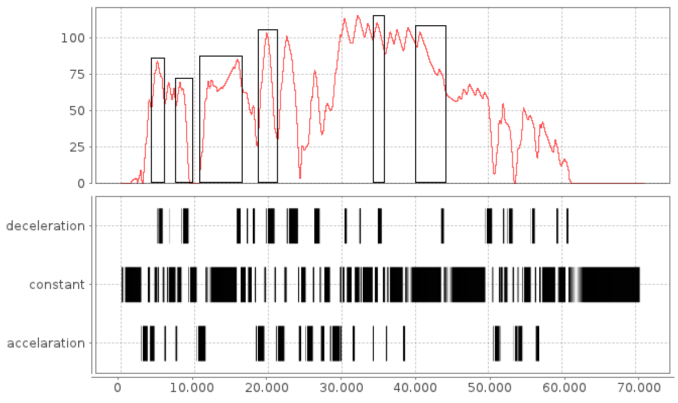

Fig. 8. Some (not all) cycles found with the pattern graph shown in 7. The red line displays the current speed and the intervals show the current behaviour of the car.

Luckily we have multivariate data, so rather than sticking to the speed profile alone, the expert may focus on other variables as well. Our (common sense) background knowledge tells us, that an acceleration involves gear shifting, however, the driver may also shift gears back and forth because the traffic density requires it. However, shifting gears up twice may be a good indicator for acceleration at the beginning of a driving cycle. To further constrain the start of a *new* cycle, we additionally require that engine revolutions are rather low before we shift gears up twice, because otherwise we may be *within* a cycle that started earlier. Visual inspection of the matches found by this intermediate pattern draws our attention to situations in which the two up-shifts are disrupted by a down-shift, so we simply exclude such occurrences from the pattern. In this knowledge-based, explorative manner, we arrive at the pattern graph shown in Fig. 9.

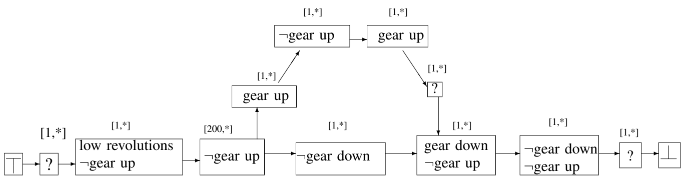

Fig. 9. Pattern graph to query driving cycles.

As we can see it is still a rather simple pattern graph with one parallel branch and 13 nodes. Please note that nodes near the start and end are '*don't care*' nodes, which enable us to find the cycles anywhere in the middle of the sequence. We do not try to capture *constant speed* any longer, but use a *don't care* node instead (still without down-shift). In this manner we elegantly include freewheeling situations that definitely belong to the driving cycle but would not fit the concept of *constant speed*. Actually, even an emergency break at high speed would match that part of the patterns, providing no down shift is involved, as this would mark the end of the cycle.

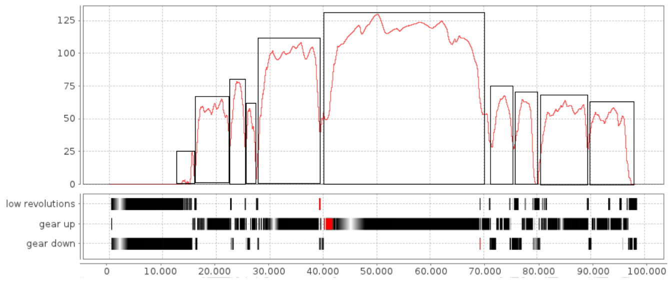

Fig. 10. All found cycles (marked with rectangles) with the enhanced graph. The red line shows the speed of the car, and the intervals below show the relevant parts of the extracted interval sequence.

In Fig. 10 we can see all of the matches found by querying the sequence with the pattern graph above. The figure shows the speed profile of the reference trip in the first place, while the black rectangles contain the retrieved cycles. In the second plot we can see the extracted labelled interval sequence from the original data, where red colours indicate the intervals responsible for the longest retrieved cycle. As one can see we were able to retrieve all cycles.

The pattern graph in Fig. 9 has been constructed interactively by setting up the graph and inspecting the matches on the reference trip shown in Fig. 10. We have compared the retrieved matches from nine further trips with the cycles that were identified manually beforehand. Figure 11 shows one example: The pattern graph performs well on this unseen data, most of the cycles are retrieved as expected. Only the cycle at the end of trip was not found. The reason lies in the requirement of having two *gears up*: in this cycle we have only one gear up before the *gear down* appears. Please note that *gear*

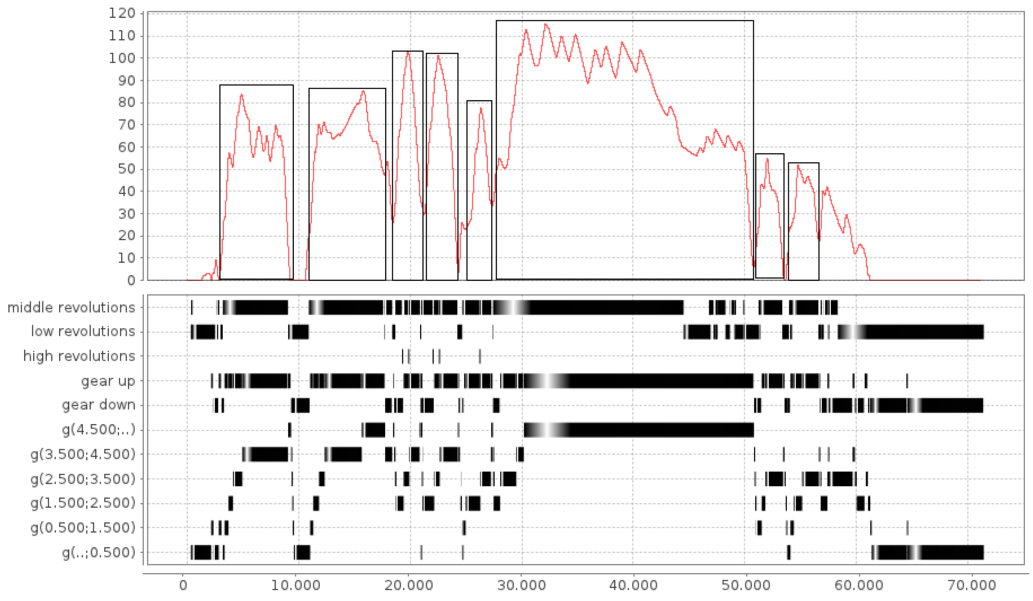

Fig. 11. All found cycles marked in the evaluation sequence. The line-plot shows the speed and the lower plot shows the corresponding interval sequence.

up or *gear down* interval stays active as long as no further gear change occurs. The labels *g(..,0.500)*, *g(0.500;1500)*, ... and *g(4.500;...)* correspond to the current gear (idle, first, ... fourth or higher).

The table shows the performance over the nine trips:

trips	cycles	correctly found	undetected
9	107	96	11

As we can see most of the cycles were retrieved correctly, but we have missed a few. What most of the undetected cycles have in common is, that the driver only shifted up one gear, because he was already driving in a high gear. Only a small modification is needed to retrieve all cycles of a specific duration: We introduce a new node with a '*don't care*' value constraint and a temporal constraint with the desired durations (e.g. [3000,*]) and connect it to the *gear up* and *gear down*, ¬*gear up* node as shown in (Fig. 12). The different groups are retrieved by changing the temporal constraint.

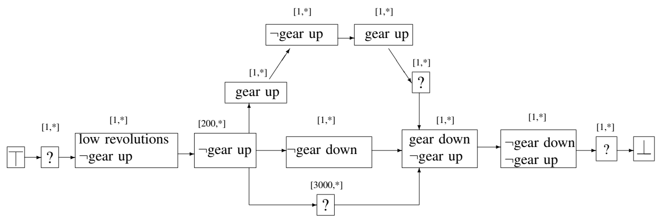

Fig. 12. Pattern graph with temporal constraint on the cycle duration

VI. Conclusions

We have introduced a new, powerful formalism to describe patterns in multivariate sequences and have demonstrated its applicability on real world data. We have shown that patterns may appear easy to grasp at first glance, but a straightforward implementation may leave a number of (exceptional) cases uncovered (as is often the case with real data). Offering an intuitive and expressive means of combining constraints on multiple variables, close to the human perception of the situation, allows us to make full use of the expert's domain knowledge. We strongly believe that the pattern graph concept supports the interaction of the domain expert (being able to express his/her current level of knowledge) and data mining techniques to further improve a manually constructed pattern, which will be part of our future work.

Acknowledgements

We would like to thank Dr. Werther from Volkswagen AG for kindly providing the data.

References

[1] H. Hochheiser and B. Shneiderman, "Dynamic query tools for time series data sets: Timebox widgets for interactive explorations," *Information Visualization*, vol. 3, pp. 1–18, 2004.

[2] Y. Shahar and M. A. Musen, "A temporal-abstraction system for patient monitoring." *Proceedings of the Annual Symposium on Computer Application in Medical Care*, pp. 121–127, 1992.

[3] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, pp. 832–843, November 1983.

[4] F. Höppner and F. Klawonn, "Finding informative rules in interval sequences," in *Proceedings of the 4th International Conference on Advances in Intelligent Data Analysis*, ser. IDA '01. London, UK, UK: Springer-Verlag, 2001, pp. 125–134.

[5] F. Mörchen, "A better tool than allen's relations for expressing temporal knowledge in interval data," in *Theory and Practice of Temporal Data Mining (TPTDM 2006) – Workshop of the 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2006, pp. 25–34.

[6] ——, "Unsupervised pattern mining from symbolic temporal data," *ACM SIGKDD Explorations Newsletter*, vol. 9, no. 1, pp. 41–55, 2007.

[7] S. Peter and F. Höppner, "Finding temporal patterns using constraints on (partial) absence, presence and duration," in *International Conference on Knowledge-Based and Intelligent Information & Engineering Systems*, 2010, pp. 442–451.

[8] J. Kelley, "Critical path planning and scheduling: Mathematical basis," *Operations Research*, vol. 9, no. 3, 1961.