Pattern Graphs: Combining Multivariate Time Series and Labelled Interval Sequences for Classification

Sebastian Peter, Frank Höppner and Michael R. Berthold

Abstract Classifying multivariate time series is often dealt with by transforming the numeric series into labelled intervals, because many pattern representations exist to deal with labelled intervals. Finding the right preprocessing is not only time consuming but also critical for the success of the learning algorithms. In this paper we show how pattern graphs, a powerful pattern language for temporal classification rules, can be extended in order to handle labelled intervals in combination with the raw time series. We thereby reduce dependence on the quality of the preprocessing and at the same time increase performance. These benefits are demonstrated experimentally on 10 different data sets.

1 Introduction

In recent years the development of cheaper sensors and bigger storage capacities has led to an increase in the amount of data gathered periodically. Companies are now able to use (mobile and/or wireless) sensor networks more efficiently in many different domains (e.g. health care, climate, traffic, business processes to name a few) to collect data often of varying dimensions. By analysing temporal data, companies try to gather more insight into their processes and are thereby able to draw conclusions, enabling them for example, to predict the market for the next week or optimise the output by improving the production process.

One important aspect during the analysis step is often finding *typical* or *characteristic* situations. Various notions of *multivariate temporal patterns* are described

Sebastian Peter, Michael R. Berthold
University of Konstanz, Nycomed-Chair for Bioinformatics and Information Mining, Box 712, D-78457 Konstanz, Germany email: {sebastian.peter, michael.berthold}@uni-konstanz.de

Frank Höppner
Ostfalia University of Applied Sciences, Department of Computer Science, D-38302 Wolfenbüttel, Germany email: f.hoeppner@ostfalia.de

in literature as a means of grasping or capturing these situations. Example applications for multivariate temporal patterns include the discovery of dependencies in wireless sensor networks [1], the exploration of typical (business) work flows [3] or the classification of electronic health records [2]. Temporal patterns are often applied to labelled interval data, as the resulting patterns are easy to understand for the experts and also allow us to deal with multivariate data. To incorporate numerical time series in the patterns, they are discretized and their behaviour is described by a linguistic term ('low revolutions', 'slowly accelerating'), which holds over a given period of time, hence the term 'labelled (temporal) interval'. The effectiveness of such patterns depends strongly on this discretisation step. In this paper we extend the powerful concept of pattern graphs (see Fig. 1 as an example) enabling us to deal directly with time series data and overcome the sensitivity of the preprocessing phase.

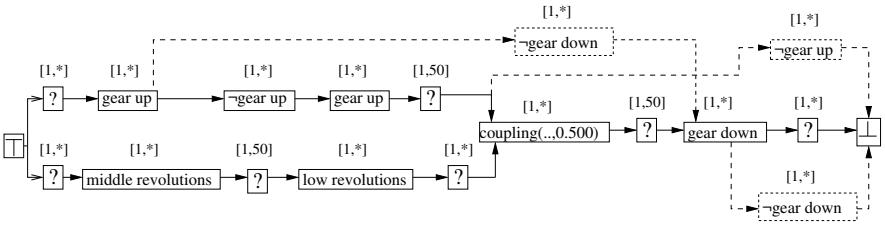
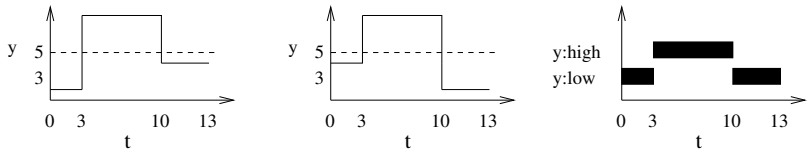


Fig. 1 Example of a pattern graph describing a driving cycle (learned from data, see [10]).

The paper is outlined in the following manner. The next section reviews related work and further motivation for our work. We then provide an introduction to pattern graphs (Sec. 3) and the matching and learning algorithms (Sec. 4) [11, 10]. In Section 5, we contribute the changes necessary to incorporate numeric time series. Section 6 presents the experimental results, and we conclude the paper in Section 7.

2 Motivation and Related Work

In this paper we concentrate on multivariate temporal patterns to characterise the evolution of multiple variables over time. These patterns are used in the antecedents of classification rules. The data consists of labelled temporal intervals; the labels may address categorical (e.g. 'gear-shift' in Fig. 1) or numerical features (e.g. 'low revolutions' in Fig. 1). These labelled intervals and their relationships are combined to form *temporal patterns*, for example by specifying the relationships between all observed intervals such as '*A before B*', '*A overlaps C*' and '*C overlaps B*' [5, 2]. This notation is quite strict and somewhat ambiguous [7], because the qualitative relationship does not carry quantitative information about the degree of overlap or size of a gap. Other approaches contain such information [3], but consider only those events that do not include *durations* and thus offer no means to express concurrency



(a) positive class (b) negative class (c) interval sequence

Fig. 2 Two time series (a) and (b) discretised to the interval sequence (c) by using the threshold 5 (dotted line), thereby losing information to distinguish them from each other.

as in '*A and B must co-occur for 5-10 time units*'. To provide more robustness against noise some approaches allow only parts of the intervals to be addressed [4, 9]. The recently proposed pattern graphs [11] satisfy most of the shortcomings and will be used in this paper (and will be introduced below in more detail).

Regardless of the pattern language, when the recorded data is numeric in nature, this leads to the problem of having to convert the numeric data into labelled intervals. This is usually done by applying thresholds, clustering methods or methods dedicated to finding stable intervals (e.g. 'Persist' [8]). This step is time consuming: multiple iterations and manual inspections are needed for a suitable discretisation as a bad discretisation can render all efforts to retrieve good patterns useless. An example is shown in Fig. 2, where the values of two time series (a) and (b) are discretised using threshold 5, leading to the same sequence of labelled intervals (with labels [low: $y \leq 5$] and [high: $y > 5$]) in Fig. 2(c). In this case the sequences are not distinguishable anymore, which is undesirable if both series belong to different classes and we look are looking for a temporal pattern that distinguishes both classes from each other. Furthermore the one perfect discretisation may not exist in a situation, where for class *(a)* the threshold needs to be 5 whereas for class *(b)* 6 and for class *(c)* a threshold of 7 would be perfect. To overcome this problem, the selection of optimal thresholds may be postponed to the learning algorithm itself instead of leaving it as a preprocessing step.

3 Pattern Graphs

This section reviews pattern graphs, which were first introduced in [11]. We consider m (categorical or numeric) attributes with value range D_j, composed of multivariate observations $\mathbf{x} \in D$ with $D = (D_1 \times ... \times D_m)$.

Definition 1 ((sub)sequence). A sequence S consists of an arbitrary number of observations $(\mathbf{x}_1, ..., \mathbf{x}_n) \in \mathscr{S}$ with $\mathscr{S} = \bigcup_{i=1}^{\infty} D^i$. Let $|S| = n$ denote the length of the sequence S. A subsequence from index a to b of S is denoted by $S|_{[a,b]}$.

To describe those parts of the series that are relevant for the classification task, we apply (local) constraints to subsequences:

Definition 2 (set of constraints for (sub)sequences). Let $\mathscr{C} = \{C \mid C : \mathscr{S} \to \mathbb{B}\}$ denote the set of all (possible) constraints on (sub)sequences. We distinguish between value constraints, restricting the acceptable values of the (sub)sequence, and temporal constraints, which limit their duration. For a sequence $S = (\mathbf{s}_1, ..., \mathbf{s}_k)$, examples of value constraints are:

- $C(S) = true$ ("don't care": is always satisfied)
- $C(S) = true \Leftrightarrow \forall i : 1 \leq i \leq k : s_{i,j} \in C_j$ with $C_j \subseteq D_j$ for all $1 \leq j \leq m$. This constraint limits the range of accepted values for the sequence.

In this paper we consider only one type of temporal constraint:

- Given $t \in T$, $T = \{(a,b) | 1 \leq a \leq b\} \subseteq \mathbb{N}^2$, a temporal constraint is defined as $C(S) = true \Leftrightarrow a \leq |S| \leq b$. Therefore a temporal constraint is represented by an interval $[a,b]$ and restricts the duration of the (sub)sequence S to lie within these bounds. Here a is considered the minimal and b the maximal temporal constraint.

Up to now, pattern graphs have only been used for interval sequences, that is, a condition (described by the interval label) either holds or not ($D_j = \{0,1\}$). We thus have three different value constraints: $C_j \subseteq \{0\}$ (absent), $C_j \subseteq \{1\}$ (present) and $C_j \subseteq \{0,1\}$ (don't care). A pattern graph defines a partial order of constraints:

Definition 3 (pattern graph). A tuple $M = (V, E, \mathscr{C}_{val}, \mathscr{C}_{temp})$ is a pattern graph, iff (V,E) is an acyclic directed graph with exactly one source ($\top$), one sink ($\bot$), a finite node set $V \subseteq \mathbb{N} \cup \{\top, \bot\}$ and an edge set $E \subseteq (V \times V)$, for which the following properties hold with $V' = V \backslash \{\top, \bot\}$:

- $\forall (v_1, v_2) \in E : v_2 \neq \top$ (no incoming edge to $\top$)
- $\forall (v_1, v_2) \in E : v_1 \neq \bot$ (no outgoing edge from $\bot$)
- $\forall v \in V' : (\exists w \in V : (v,w) \in E) \wedge (\exists w \in V : (w,v) \in E)$
  (all nodes $v \in V'$ have at least one incoming and outgoing edge)

Finally, $\mathscr{C}_{val} : V' \to \mathscr{C}$ is a function, mapping each node $v \in V'$ to a value constraint $C \in \mathscr{C}$, whereas $\mathscr{C}_{temp} : V' \to \mathscr{C}$ maps each node $v \in V'$ to a temporal constraint $C \in \mathscr{C}$. By $\mathscr{C}^v_{val}$ we abbreviate $\mathscr{C}_{val}(v)$, i.e., the value constraint assigned to v. We define $\mathscr{C}^v_{temp} := \mathscr{C}_{temp}(v)$ analogously.

Definition 4 (mapping). A mapping B for a sequence S and a pattern graph $M = (V, E, \mathscr{C}_{val}, \mathscr{C}_{temp})$ assign each node $v \in V \backslash \{\top, \bot\}$ a continuous subsequence $S|_{[a,b]}$. $\top$ is assigned the fictitious subsequence $S|_{[0,0]}$ and $\bot$ the subsequence $S|_{[|S|+1,|S|+1]}$. $B(v) = [a,b]$ denotes the start and end index of the subsequence of S assigned to node v.

Definition 5 (match, valid mapping). A valid mapping B for pattern graph $M = (V, E, \mathscr{C}_{val}, \mathscr{C}_{temp})$ and a sequence S of length n is a mapping with the following additional properties: with $V' = V \backslash \{\top, \bot\}$

$$\forall (v_1, v_2) \in E : B(v_1) = [a,b] \wedge B(v_2) = [c,d] \Rightarrow b+1 = c \quad \text{(no gaps)} \tag{1}$$

$$\forall i : 1 \leq i \leq n : \exists v \in V' : i \in B(v) \text{ (each index is assigned at least once)} \tag{2}$$

$$\forall v \in V' : \mathscr{C}^v_{val}(S|_{B(v)}) = \text{true} \quad \text{(value constraint holds)} \tag{3}$$

$$\forall v \in V' : \mathscr{C}^v_{temp}(S|_{B(v)}) = \text{true} \quad \text{(temporal constraint holds)} \tag{4}$$

Having defined the pattern graph in detail we will now give an example to illustrate the semantics of the pattern graph. Fig. 3 (a) shows an example of a pattern graph with one path, which is read as follows: the temporal constraint of a node is depicted above the node. A star represents an unlimited duration. The value constraint of a node is shown inside the node. We have two kinds of value constraints for attribute A: A means that the attribute is active ($D_A = \{1\}$) and $\neg A$ requires its absence ($D_A = \{0\}$). A node labelled '?' (*don't care*) is unconstrained. Please note that if the node states A the behaviour of the other attributes is unconstrained.

Figure 3 (b) shows a sequence where the vertical axis reveals two attributes A and B, which hold over certain periods of time (black bars, time on horizontal axis). We now discuss whether these sequence match the pattern graph in Fig. 3 (a). As this is a simple graph, it contains only one path from source to sink. For this graph the sequence has to be divided into four contiguous parts. Such that the first part satisfies the '*don't care*' constraint, in the second part the property A must hold; in the third part property B must hold and in the last part both A and B must hold . All of these four parts require a duration of at least one time unit (but have no upper bound on the duration) with the exception of the A node with a minimum duration of 3. The sequence shown in Fig. 3(a) can be mapped to the graph, because we can clearly see that A is active until B begins and is active until the end, also that during the last part A becomes active again through to the end of the sequence. Actually the pattern graph has more than one valid mapping (discussed later in Sec. 4). A more complex and expressive pattern graph is found in Fig. 1, which describes a driving cycle derived from real data [10].

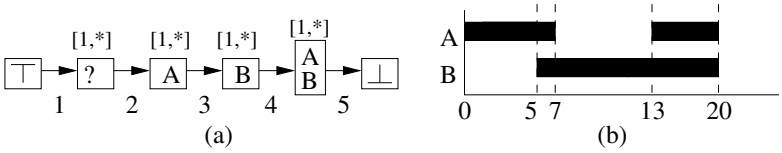


Fig. 3 Example for multiple mapping candidates on a given graph (a) and sequence (b)

4 Matching and Learning Pattern Graphs

Matching a pattern graph to a sequence is essentially a combinatorial problem; an efficient matching algorithm can be found in [11]. Often multiple matches are pos-

sible and for each edge $e = (u,v) \in E$ the algorithm provides a set of valid edge positions $p(e)$, i.e., a set of positions t that satisfy all value constraints of node u for $t' < t$ and all value constraints of node v for $t' \geq t$. For the graph in Fig. 3(a) and the sequence in Fig. 3(b) we have a set of valid locations $p(e) = \{0\}$ for the edge e from node $\top$ to $\bot$, but for the edge e' from B to AB we have $p(e') = \{13, 14, ..., 19\}$. These edge positions are organised in so-called *mapping candidates*, which map each edge of the graph to one contiguous interval of valid edge positions. So a *mapping candidate C* may be considered as a precursor of a *valid mapping B* in the following sense (by $(\cdot, v)$ we denote any edge leading to v):

$$\forall v \in V : B(v) = [s,e] \quad \Rightarrow \quad s \in C(\cdot, v) \wedge e \in C(v, \cdot)$$

Multiple mapping candidates exist, from which one or more valid mappings may be derived. Consider the graph in Fig. 3. The '?' node is valid during [0,20], the 'A' node is satisfied during [0,7] and [13,20], 'B' during [5,20] and finally 'AB' during [5,7] and [13,20]. Out of these sets of valid positions the matching algorithm derives two mapping candidates C_1 and C_2, assigning each edge its admissible positions. Three valid mappings B_1-B_3, obtained from C_1 and C_2, are shown below (many more are possible).

C_1 1:[0,0], 2:[1,4], 3:[5,7], 4:[13,19], 5:[20,20]	B_1 [0,1] - [1,4] - [4,13] - [13,20]
	B_2 [0,2] - [2,7] - [7,16] - [16,20]
C_2 1:[0,0], 2:[13,15], 3:[16,18] 4:[17,19] 5:[20,20]	B_3 [0,13] - [13,17] - [17,18] - [18,20]

While all edge positions from the mapping candidates ensure that the value constraints hold, the positions must also fulfil the temporal constraint of the node. For instance, if the temporal constraint for the node labelled 'A B' was [5,*], the mapping B_2 and B_3 would no longer be valid (because the sequence assigned to this node has only a length of ≤ 4).

A two-phased learning algorithm for pattern graphs has been introduced in [10], where a general pattern, matching all instances of a class, is learned in the first phase. The second phase implements a beam-search, where in each iteration, the k-best pattern graphs are processed further by special refinement operators, which add new nodes or edges to the graph, modify temporal constraints or add value constraints to nodes in order to improve some measure of interestingness (we apply the J-measure [12]).

5 Extending Pattern Graphs to Time Series

In order to enable pattern graphs to deal with a numeric range $D_j \subseteq \mathbb{R}$ (cf. Def. 1) we introduce new value constraints called *series constraints*, where $C_j = \{x | x \leq \sigma\} \subseteq D_j$ or $C_j = \{x | x \geq \sigma\} \subseteq D_j$ for some threshold σ (cf. Def. 2). With these additional constraints we enable pattern graphs to overcome the obstacle of finding the best discretisation to convert time series to labelled intervals, as every node may now use its own threshold σ instead of relying on the predefined intervals alone.

This enables us to use different thresholds for different classes and it also allows us to use different series constraints for the same series within the same class (local constraints in different nodes). For example we can create the pattern graphs shown in Fig. 4 (a) and (b) which are able to separate the sequences shown in Fig. 2 (a) and (b) nevertheless they have the same interval representation as shown in Fig. 2.

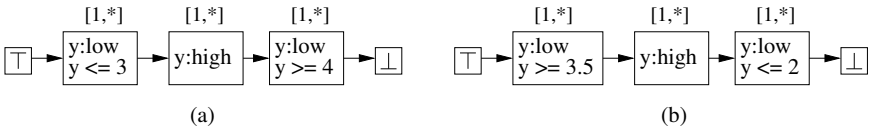


Fig. 4 Two pattern graphs with a new constraint to distinguish the positive between the negative sequences from Fig. 2.

To learn such constraints automatically from data, we have to extend the beam search operators. While it is quite easy to check whether a given assignment of subsequences to graph nodes represents a valid mapping, it is much more complicated to derive new conditions that are in some way 'optimal' for the set of all possible mappings. This is due to the fact that a graph node with the constraint '*y:low*' may match an interval $[s,e]$ with this label in many different ways: any sub-interval $[s',e'] \subseteq [s,e]$ satisfies the node constraints; such a constraint still leaves many possibilities for valid mappings. In order to refine (or introduce new) node constraints, we have to consider *all* of these potential mappings at the same time, in order to calculate what would be the best additional constraint to distinguish good from bad cases. Enumeration of all possible mappings is not feasible because of their large number. Thus we operate directly on the *mapping candidates* of the matching algorithm.

Without loss of generality we will consider only the $x \leq \sigma$ constraint in the following. The new operator is instantiated for each individual graph node $v \in V$. As with all the other operators, it receives all mapping candidates (that already reflect the value constraints of that node), the temporal constraint and the data sequence. The objective is to derive a threshold σ on one (numeric) variable x of the sequence, such that the additional node constraint $x \leq \sigma$ improves the discriminative power of the pattern. Expressed more formally: if P is a pattern graph, let $m_P(s) = 1$ denote that P has a valid mapping to $s \in S$ (0, else). Let G' denote the resulting pattern if P is extended by the constraint $x \leq \sigma$ in node v. Then a confusion matrix from $m_{G'}(S) \in \{0,1\}$ and a class $k/\neg k$ is created for G' to evaluate its utility.

The naive approach to find the best refinement is to extend P with every possible series constraint for x and then match all of the resulting graphs to all sequences. It is sufficient to examine only those thresholds σ that change the matching result of some $s \in S$, that is, for σ we obtain $m_{G'}(s) = 1$ but for $\sigma - \varepsilon$ we have $m_{G'}(s) = 0$, because it is only at these thresholds that the confusion matrices of the rule change. Thus, we need to determine only as many confusion matrices as we have sequences. How do we find the threshold σ for a given sequence s? If a subsequence, mapped to node v, shall satisfy the constraint, we have to choose σ as the maximum of all x. However we do not know this subsequence in advance, but have to consider all

possible subsequences that may be obtained from the mapping candidates. To let all subsequences satisfy the constraint, we have to pick the smallest of all maximum values of all possible subsequences. If this value were reduced only slightly (-ε), there would be at least one subsequence for node v that would not match anymore - with the result that no valid mapping exists anymore. Lemma 1 shows that it is sufficient to inspect only the shortest possible subsequences rather than all possible subsequences.

Lemma 1. *By* $\max S$ *we denote the maximum of the x-values in a (sub) sequence S. Let* $\mathcal{Q}$ *be the set of all subsequences (that may occur in a valid mapping to node v) and* $\mathcal{Q}'$ *the set of shortest subsequences*[1]. *Then* $\min\max_{S\in\mathcal{Q}} S = \min\max_{S\in\mathcal{Q}'} S$ *holds.*

Proof. Let $S \in \mathcal{Q}$. Without loss of generality let us assume that $S \notin \mathcal{Q}'$. Thus, S is not among the shortest subsequences and therefore we find a $T \in \mathcal{Q}' \subseteq \mathcal{Q}$ such that T is a subsequence of S. All values of T are contained in S, but S contains additional entries, therefore we have $s := \max S \geq \max T =: t$. Thus, we know that $\min\max_{S\in\mathcal{Q}'} S \leq t \leq s$. This means, that $s = \max S$ can be ignored safely in the calculation of $\min\max_{S\in\mathcal{Q}} S$.

Algorithm 1 findBestSeriesSmallerThanThresholdConstraint

Require: S: all sequences
Require: v: node to refine
Require: v_{min}: minimal length of the node
Ensure: best refinement

1: **for** $s \in S$ **do**
2: find mapping candidates C_M
3: $values \leftarrow \bigcup_{c\in C_m} getMinMaxValueForMappingCandidate(s, c, v_{min}, v)$
4: $\sigma \leftarrow \min_{values}$
5: collect all thresholds σ in set Σ
6: **end for**
7: sort all thresholds in Σ in ascending order
8: create and evaluate confusion matrices for all found thresholds.
9: add the threshold σ with highest measure to the node v as the series constraint
10: **return** refined pattern graph.

The outline of the refinement operator to find the best $x \leq \sigma$ is shown in algorithm 1. In the lines 1-6 the algorithm computes the threshold as defined by the Lemma 1. It utilises algorithm 2 to find the maximum value of all shortest subsequences for a given *mapping candidate*. v_{min} denotes either the minimal temporal constraint of node v, or is a greater value if the graph structure requires longer sequences in order to satisfy the temporal constraints of other nodes (for example due to parallel paths).

We find the best refinement by evaluating all possible confusion matrices and picking the one with the highest interestingness measure. In order to avoid overfitting, the series constraint with the best measure will be relaxed similar to the binary

[1] shortest in the following sense: $\forall s' \in Q' : \neg\exists s \in Q : s \subset s'$

split operator in decision tree learning: We search for the next greater value and use the mean of both. This does not change the prediction of the new pattern on the training set, but is less restrictive for new instances. The refinement is completed in line 9 by adding the series constraint with the computed value to the node.

From algorithm 2 we can see that the shortest subsequences for a single mapping candidate are always subsequences with the same length, shifted by one time unit. This allows us to use a priority queue, in order to extract the constraint value efficiently.

Overfitting. An important way of avoiding overfitting is to prevent nodes with the minimum temporal constraints 1 from being refined with a series constraint. This would allow the pattern graph to focus on one single time point and would thus stimulate overfitting. We therefore enforce a minimal length of a node to be refined with the new series constraint. If a node has a minimal duration of 1 during refinement, the minimal length will be set to this minimal length (lower bound of v_{min}). This has the consequence for step 1 that only subsequences with the minimal length, which are mappable to the node have do be analysed. Additionally we have added a likelihood ratio [6] test after every refinement and keep only those graphs with statistically significant improvements to avoid overfitting (which is a problem common to all rule learners).

6 Experimental Evaluation

The experimental evaluation is divided into two different experimental setups. In the first experiment we show that the series constraints help to overcome the prepro-

Algorithm 2 getMinMaxValueForMappingCandidate

Require: s: sequence
Require: c: mapping candidate
Require: v_{min}: minimal length of the node
Require: v: node to refine
Ensure: smallest maximum value of the subsequences contained in c

1: $pl \leftarrow$ latest start position $\in c((\cdot, v))$
2: $pe \leftarrow$ earliest end position $\in c((v, \cdot))$
3: $S_m \leftarrow \emptyset$
4: $begin \leftarrow pe$ - v_{min}.
5: **if** $begin > pl$ **then**
6: **return** maximum value contained in $s|_{[pl,pe]}$
7: **else**
8: **while** $begin \leq pl$ **do**
9: $S_m \leftarrow S_m \cup s|_{[begin,begin+v_{min}]}$
10: $begin \leftarrow begin + 1$
11: **end while**
12: **end if**
13: **return** smallest value out of the maximum values from the subsequences $\in S_m$

cessing problem discussed earlier. In the second experiment we show that the new approach is able to perform better, even if a good discretisation is applied beforehand.

6.1 Robustness Against Preprocessing Errors

To show that series constraints could help dealing with sub-optimal preprocessing of the data we took nine data sets from the UCR time series repository[2]. This repository already supplies training and tests partitions for each data set in a common format. All of these data sets consist of a raw univariate time series which requires some preprocessing: moving average smoothing was applied to the series and we also extracted an additional slope series. Thereby we artificially converted the data into a multivariate time series (original and slope time series). In the second step this preprocessed time series had to be converted into a labelled interval series by applying 3-quantile discretisation. To achieve different discretisation the quantile boundaries are selected randomly for each iteration. We are aware of the fact that for the given data sets there are algorithms that perform better, however most of these approaches were not able to deal with multivariate data. These approaches often utilise 1-nearest neighbour classification (1NN) with Euclidean distance or dynamic time warping, whereas the pattern graphs rely on simple elements only (like intervals with a value $\geq \sigma$) and thus highlight structural differences. These simple elements keep pattern graphs interpretable even in the case of complex multivariate data (see Sect. 6.2). Therefore 1NN-approaches are not the real competitors. To show the improvement of the learned graphs with series constraints and allow future comparison for follow up work we decided to use these data sets. Table 1 displays the results obtained by applying the beam with and without the series constraints for 30 iterations per class and data set. The parameter for the minimum sequence length was set to 10, but the results obtained by using additional operators with 5, 15 and 20 as minimal length led to nearly identical results. The first row names the data set, the second row displays the class (for which the pattern graphs were learned for). The left side represents the search without series constraints; the right side represents the search with series constraints. The third row indicates accuracy and the fourth contains standard deviation.

For most of the classes the learned pattern graphs with the series constraints are able to perform significantly better in terms of accuracy. We can also see that in most cases standard deviation has decreased, showing that suboptimal discretisation has been compensated. For four classes only small improvements in terms of accuracy and standard deviation occurred. For two classes the performance with series constraints deteriorates: for the class # 3 from the Synthetic Control data set, standard deviation increases while accuracy drops. In these cases the series constraint led the

[2] Keogh, E., Zhu, Q., Hu, B., Hao. Y., Xi, X., Wei, L. & Ratanamahatana, C. A. (2011). The UCR Time Series Classification/Clustering Homepage: `www.cs.ucr.edu/~eamonn/time_series_data/`

Data set	Gun Point				Waver				Yoga				Coffee			
Class	1		2		-1		1		1		2		1		2	
Accuracy	67,5	79	61,3	75,1	16,4	91	58,6	77,5	48,2	56,7	51,4	52,7	55,8	60,6	59,5	71
Std. dev.	12,3	5,7	14,2	4,8	21,3	2,7	10,6	0	4,1	2,7	4,5	3,2	13,1	1,1	10,3	1,8

Data set	Lightning 2				Synthetic Control											
Class	-1		1		1		2		3		4		5		6	
Accuracy	53,5	54,1	51,6	56,1	62,5	89,3	88	90,1	93,3	87,7	92,2	93,2	91,2	92,2	87,4	94,1
Std. dev.	8,1	0	8,2	5,3	34	2,1	19,9	5,6	3	5,6	2,9	3,6	3,1	3,3	15	1,4

Data set	Two Patterns								CBF					
Class	1		2		3		4		1		2		3	
Accuracy	34,8	96,8	27,79	91,1	27,4	93,2	29,8	97,5	76,3	95,9	76,1	95,8	67	77,2
Std. dev.	20,6	0	13,5	2,9	13,6	1,7	18,1	0,4	13,5	0	11,4	0,2	15,1	8,5

Data set	Fish													
Class	1		2		3		4		5		6		7	
Accuracy	33,9	85,7	37,1	89,7	42,5	91,8	33,7	24,9	71,4	91,4	28,2	86,3	64,7	90,4
Std. dev.	31,1	1,1	34,8	0,1	36,1	2,1	28,6	22,1	34,4	3,3	26,3	1,2	30,2	1,2

Table 1 Results on the data when the thresholds vary

beam search into a local maximum (the interestingness measure is also lower on the training set). This leaves room for further improvement of this approach, because without the limitations of the beam we should obtain at least the same performance as before.

6.2 Improved Accuracy on Data with Good Discretization

We obtained a data set from a German company, which produces power tools, amongst other things. This data set consists of 8 different classes, where each class describes a different screwing process: screwing in and out using different gears of the power tool and different screws. Each of these 564 instances are described by five time series, e.g. voltage/current at the battery or engine etc. In a first step we manually discretised each of this series to labelled intervals in an interactive manner until we were satisfied with the results. Therefore we may safely assume that discretisation is good and it would be hard to achieve better discretisation. We applied the beam search (minimal length: 10) to this data set 30 times with and without the series constraint. In each iteration we randomly partitioned the data set into 80% training and 20% test. As a result of the good discretisation we assume that the accuracy results would be similar, but may be improved by applying different thresholds to one class or in between classes.

Table 2 shows the result. If we sum up the average improvements for all of the individual classes, the new graphs performed 6.6% better. Most of the pattern graphs learned with series constraints perform better (up to 3%) and are only slightly worse

Power Tools																
Class	1		2		3		4		5		6		7		8	
Approach	a	b	a	b	a	b	a	b	a	b	a	b	a	b	a	b
Accuracy	98,7	98,9	98,1	99,5	98,8	98,9	98,7	98,2	97,4	99,1	96,8	97,3	98,4	98,4	95,9	99,0
Std. Dev.	0,7	0,9	0,9	0,6	0,9	0.9	0,9	1,3	1,4	1,1	1,4	1,4	1,1	1,6	2,3	2,7
Avg Imp.	0,3		1,4		0,01		-0,5		1,7		0,6		0		3,1	

Table 2 Results on the power tool data set

for one class #4 (-0.5%). However improvements 'per-class' are not significant. So far, each rule has predicted just one class. Next we combine the individual rules to a single, multi-class classifier: we only classify an instance if and only if one pattern graph has a valid mapping on the instance. In case no or more than one pattern graph matches we predict "unknown". The box plot in Fig. 5 and Table 3 shows the result of this classifier, where we use the same pattern graphs as in Table 2, thus the results origin from the same 30 runs with random training and test sets.

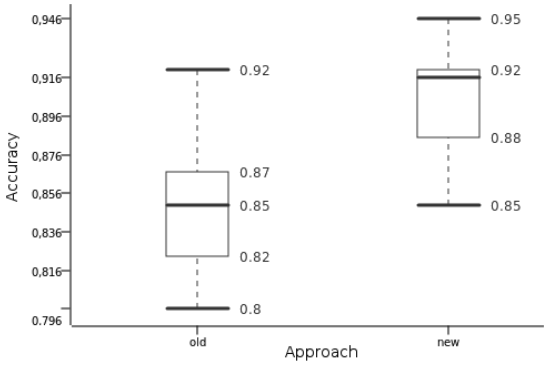


Fig. 5 Box plots showing the results of the complete classifier on the power tool data set

Approach	old	new
Accuracy	84,7%	90,5%
Std. dev.	3,2%	2,8%

Table 3 Mean accuracy and standard deviation of the complete classifier for the powertool dataset

We can see that by using series constraints the overall accuracy has improved by an average of 5.8%. It is also interesting to note that in all 30 iterations, the lowest accuracy of the new approach is at least as good as the average result without the series constraints. Additionally the mean accuracy using the new approach is nearly equal to the best result obtained without the series constraints (-1.5%). By inspecting the learned pattern graphs, we observed that one or two additional series constraints were derived per class. Depending on the class, the thresholds were slightly different, which explains the improved accuracy as the number of false positives was able to be reduced, without increasing the number of false negatives.

7 Conclusion

In this paper we have shown that the results of pattern-learning algorithms for labelled sequences rely heavily on the discretisation of the source-time series. The quality of the learned patterns varies considerably when discretisation changes. In order to overcome the problem of finding good discretisation, which is time consuming and not always possible, we introduced an algorithm capable of mining labelled intervals together with the corresponding time series. The first experiment has shown that in comparison to the approach without the series constraint, the quality of the patterns is higher, resulting in and allowing for a more robust approach compared to a priori discretisation. Furthermore we have shown that even in situations, where discretisation already performs well, the quality of the patterns can be increased, as different levels of discretisation for different classes and even different thresholds within one class can be utilised.

For future work the synergies of labelled intervals and numeric time series may be improved further as, so far, we have only used simple constraints ($\leq$, $\geq$). However it is possible to use more sophisticated constraints on mean values or standard deviation. This kind of constraint may provide further insight into the patterns.

Acknowledgements We would like to thank Stefan Mock from the Robert Bosch GmbH for kindly providing the data.

References

1. Basile, T.M.A., Mauro, N.D., Ferilli, S., Esposito, F.: Relational temporal data mining for wireless sensor networks (2009)
2. Batal, I., Valizadegan, H., Cooper, G.F., Hauskrecht, M.: A pattern mining approach for classifying multivariate temporal data. In: Bioinformatics and Biomedicine (BIBM), 2011 IEEE International Conference on, pp. 358–365. IEEE (2011)
3. Berlingerio, M., Pinelli, F., Nanni, M., Giannotti, F.: Temporal mining for interactive workflow data analysis. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '09, pp. 109–118. ACM, New York, NY, USA (2009)
4. Chen, Y.C., Jiang, J.C., Peng, W.C., Lee, S.Y.: An efficient algorithm for mining time interval-based patterns in large database. In: Proc. Int. Conf. Inf. Knowl. Mngmt., pp. 49–58. ACM (2010)
5. Höppner, F.: Discovery of temporal patterns – learning rules about the qualitative behaviour of time series. 2168, pp. 192–203. Freiburg, Germany (2001)
6. Kalbfleisch, J.G.: Probability and statistical inference: probability, vol. 2. Springer-Verlag (1985)
7. Mörchen, F.: Unsupervised pattern mining from symbolic temporal data. SIGKDD Explor. Newsl. **9**(1), 41–55 (2007)
8. Mörchen, F., Ultsch, A.: Optimizing time series discretization for knowledge discovery. In: Proc. Int. Conf. Knowl. Disc. and Data Mining, pp. 660–665. ACM (2005)
9. Mörchen, F., Ultsch, A.: Efficient mining of understandable patterns from multivariate interval time series. pp. 181–215. Springer (2007)

10. Peter, S., Höppner, F., Berthold, M.R.: Learning pattern graphs for multivariate temporal pattern retrieval. In: Proc Int Symp Intel. Data Analysis (2012)
11. Peter, S., Höppner, F., Berthold, M.R.: Pattern graphs: A knowledge-based tool for multivariate temporal pattern retrieval. In: 6th IEEE International Conference on Intelligent Systems (IS'12) (2012)
12. Smyth, P., Goodman, R.M.: An information theoretic approach to rule induction from databases. IEEE Trans. Knowledge Discovery and Engineering **4**(4), 301–316 (1992)