Mining Sequences of Temporal Intervals

Steffen Kempe and Jochen Hipp

DaimlerChrysler AG, Group Research, 89081 Ulm, Germany
{Steffen.Kempe, Jochen.Hipp}@daimlerchrysler.com

Abstract. Recently a new type of data source came into the focus of knowledge discovery from temporal data: interval sequences. In contrast to event sequences, interval sequences contain labeled events with a temporal extension. However, existing algorithms for mining patterns from interval sequences proved to be far from satisfying our needs. In brief, we missed an approach that at the same time: defines support as the number of pattern instances, allows input data that consists of more than one sequence, implements time constraints on a pattern instance, and counts multiple instances of a pattern within one interval sequence. In this paper we propose a new support definition which incorporates these properties. We also describe an algorithm that employs the new support definition and demonstrate its performance on field data from the automotive business.

1 Introduction

Mining sequences from temporal data is a well known data mining task which gained a lot of attention in the past, e.g. [2, 5]. In all these approaches, the temporal data is considered to consist of events. Each event has a label and a timestamp. In the following, we want to focus on temporal data where an event has a temporal extension. These temporally extended events are called temporal intervals. Each temporal interval can be described by a triplet (b, e, l) where b and e denote the beginning and the end of the interval and l its label. For example a sequence of temporal intervals may describe the medical history of a patient in a hospital or the data collected by a flight recorder.

At DaimlerChrysler we are interested in mining interval sequences in order to further extend the knowledge about our products. Thus, in our domain one interval sequence may describe the history of one vehicle. The configuration of a vehicle, e.g. whether it is an estate car or a limousine, can be described by temporal intervals. The build date is the begin and the current day is the end of such a temporal interval. Other temporal intervals may describe stopovers in a garage or the installation of additional equipment. Mining these interval sequences helps us in tasks like quality monitoring or improving customer satisfaction.

In the following section we give formal definitions of the mining task. In Section 3 we introduce a new support definition which is motivated by our experiences in the automotive industry. Next, we propose a novel algorithm for finding frequent patterns which implements the new support definition. The algorithm is tested on real world data from our domain in Section 5.

2 Foundations

As previously mentioned, we represent a temporal interval as a triplet (b, e, l).

Definition 1. *(Temporal Interval) Given a set of labels $l \in L$, we say the triplet $(b, e, l) \in \mathbb{R} \times \mathbb{R} \times L$ is a temporal interval, if $b \leq e$. The set of all temporal intervals over L is denoted by I.*

Definition 2. *(Interval Sequence) Given a sequence of temporal intervals, we say $(b_1, e_1, l_1), (b_2, e_2, l_2), \ldots, (b_n, e_n, l_n) \in I$ is an interval sequence, if*

$$\forall (b_i, e_i, l_i), (b_j, e_j, l_j) \in I, i \neq j : b_i \leq b_j \wedge e_i \geq b_j \rightarrow l_i \neq l_j \tag{1}$$

$$\begin{gathered} \forall (b_i, e_i, l_i), (b_j, e_j, l_j) \in I, i < j : \\ (b_i < b_j) \vee (b_i = b_j \wedge e_i < e_j) \vee (b_i = b_j \wedge e_i = e_j \wedge l_i < l_j) \end{gathered} \tag{2}$$

hold. A given set of interval sequences is denoted by $\mathbb{S}$.

Equation 1 above is referred to as the *maximality assumption* [4]. The maximality assumption guarantees that each temporal interval A is maximal, in the sense that there is no other temporal interval in the sequence sharing a time with A and carrying the same label. Equation 2 requires that an interval sequence has to be ordered by the beginning (primary), end (secondary) and label (tertiary, lexicographically) of its temporal intervals.

Without temporal extension there are only two possible relations. One event is before (or after as the inverse relation) the other or they coincide. In case of temporal intervals there are 7 possible relations (respectively 13 including inverse relations). These interval relations have been described by Allen in [3] and are depicted in Figure 1. Each relation of Figure 1 is a temporal pattern on its own that consists of two temporal intervals. Patterns with more than two temporal intervals are straightforward. One just needs to know which interval relation exists between each pair of labels. Using the set of Allen's interval relations $\mathbb{I}$, a temporal pattern is defined by:

Definition 3. *(Temporal Pattern) A pair $P = (s, R)$, where $s : 1, \ldots, n \rightarrow L$ and $R \in \mathbb{I}^{n \times n}$, $n \in \mathbb{N}$, is called a "temporal pattern of size n".*

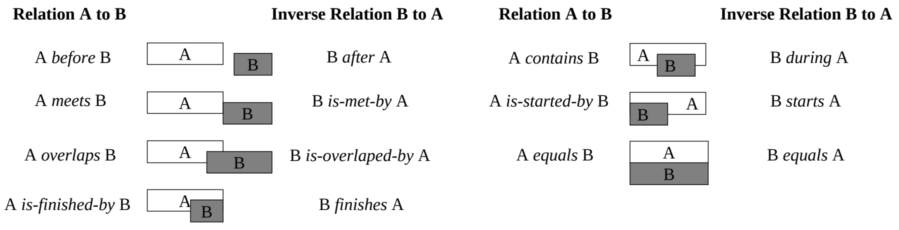
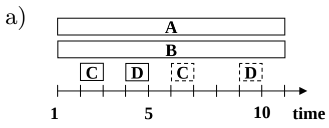


Fig. 1. Allen's Interval Relations



b)

	A	B	C	D
A	e	***e***	***c***	***c***
B	e	e	***c***	***c***
C	d	d	e	***b***
D	d	d	a	e

Fig. 2. a) Example of an Interval Sequence b) Example of a Temporal Pattern (e is the abbreviation for *equals*, c for *contains*, etc.)

If a temporal pattern holds true (is valid) for an interval sequence we consider this sequence as an instance of the pattern.

Definition 4. *(Instance) A temporal pattern $P = (s, R)$ holds true for an interval sequence $S = (b_i, e_i, l_i)_{1 \leq i \leq n}$, if $\forall i, j : s(i) = l_i \land s(j) = l_j \land R[i, j] = \mathrm{ir}([b_i, e_i], [b_j, e_j])$ with function* ir *returning the relation between two given intervals. We say that the interval sequence S is an instance of temporal pattern P. We say that an interval sequence S'* contains *an instance of P if $S \subseteq S'$, i.e. S is a subsequence of S'.*

Obviously a temporal pattern can only be valid if its labels have the same order as their corresponding temporal intervals have in an instance of the pattern.

Figure 2a) shows an example of an interval sequence which contains an instance of the temporal pattern given in Figure 2b).

The mining task is to find all temporal patterns in a given set of interval sequences which satisfy a user specified minimum support threshold. Note that this mining task is closely related to frequent itemset mining [1].

3 A New Support Definition

Previous investigations on discovering patterns from sequences of temporal intervals include the work of [4] and Papapetrou et al. [6]. Both approach the problem very differently from each other.

Höppner mines patterns from one sequence of temporal intervals only. The problem of multiple sequences is not addressed. He also requires that a pattern occurs within a certain time window. In contrast, Papapetrou et al. analyze multiple sequences but they do not have any time constraints on pattern instances.

The main difference between both approaches is the definition of support for a pattern. Höppner defines the *temporal support* of a pattern. This definition is closely related to Mannila's *frequency* in [5]. The temporal support can be interpreted as the probability to see an instance of the pattern within the time window if the time window is randomly placed on the temporal interval sequence. On the other hand, Papapetrou et al. count the number of instances for each pattern. The pattern counter is incremented once for each sequence that contains the pattern. If a temporal interval sequence contains multiple instances of a pattern then these additional instances will not further increment the counter.

Consider the pattern C *before* D in the example of Figure 2a). As the interval sequence contains an instance of the pattern the support of Papapetrou et al. is 1. Using a time window of 3 Höppner will calculate a support of 3 (a duration of 2 for the first occurrence and 1 for the second).

Yet, for our needs and comparable applications the accurate number of pattern instances is indispensable for generating knowledge in our domain. Thus we developed a new support definition which: 1. counts the number of pattern instances, 2. handles multiple instances of a pattern within one interval sequence, and 3. allows time constraints on a pattern instance.

In [5], Mannila et al. introduced *minimal occurrences* as a support definition for patterns in a single sequence of events. We extend the approach of minimal occurrences to the circumstances of temporal intervals.

Given a temporal pattern P and an interval sequence S, a time window $[t_b, t_e]$ of S is called *minimal occurrence* of P, if there is an instance of P in the time window but not in one of its proper subwindows.

A special case is if the temporal pattern P is of size 1, i.e. it contains only one label. Then every temporal interval (b, e, l) with $b < e$ leads to an infinite number of minimal occurrences. Therefore the minimal occurrences of P are $[b, e]$ for each temporal interval (b, e, l) and l is the label of P.

Minimal occurrences also provide an easy way to introduce time constraints on a pattern instance. Suppose a pattern instance is only valid if it occurs within a certain period of time, then we just need to count those minimal occurrences whose lengths do not exceed the time limit.

After we introduced minimal occurrences in our application, we realized that the support definition is still not sufficient. There is a subset of temporal patterns whose supports are calculated differently than we expected. Figure 2a) gives an example of such a temporal pattern (solid lines). The minimal occurrence is $[1, 11]$. In any smaller time window, the relation *equals* between the temporal intervals A and B is not visible. Thus if an interval sequence contains a second C *before* D during the temporal intervals A and B (dashed lines in Figure 2a)), it will not be counted as it produces the same minimal occurrence. This example is important to us because the temporal intervals C and D might describe garage stopovers for a car with A and B specifying its vehicle configuration. So we want to count multiple instances of temporal patterns (here C *before* D) for different combinations of configurations. Hence, in the example above, we have to count the minimal occurrences of the subpattern C *before* D given that the subpattern is contained in temporal intervals A and B.

As a result, we have to distinguish those temporal patterns from all other patterns where a subpattern is decisive for the support calculation. In the latter patterns there exists a subpattern which is contained in all other temporal intervals of the temporal pattern. We can decide whether a given temporal pattern P contains such a subpattern by transforming the problem into graph theory based on the upper triangular matrix of its relation table. We start by creating an empty graph. For each label of the temporal pattern we insert a vertex into the graph. Next we create an edge for each relation in the upper triangular matrix of the relation table which is not *contains* between the corresponding vertices in the graph. If the resulting graph is unconnected then there is a subpattern which is contained in all other temporal intervals of the pattern. This subpattern corresponds to one of the connected subgraphs.

We call a temporal pattern connected if its graph is *connected*. Otherwise we call the temporal pattern *unconnected*. In contrast to [5] we define the support of a connected pattern P as its total number of minimal occurrences in all sequences of $\mathbb{S}$. If P is unconnected the support is given by the total number of minimal occurrences of its subpattern.

4 New Algorithm

As in existing approaches, the main idea is to generate all frequent temporal patterns by applying the Apriori scheme of candidate generation and support evaluation. These two steps are alternately repeated until no more candidates are generated. Apriori starts with the frequent 1-patterns and then successively derives all k-candidates from the set of all frequent (k-1)-patterns. This approach requires that each subpattern of a frequent pattern is frequent. Unfortunately the extension of minimal occurrences to temporal intervals destroys this downward closure property. The problem arises if a temporal interval is used as the same part of a pattern for multiple instances. Consider, e.g., the interval sequence (1,11,A), (2,3,C), (6,7,C) (see Figure 2a)). There are two minimal occurrences of the pattern A *contains* C $[2, 3]$ and $[6, 7]$ but there is only one minimal occurrence of A $[1, 11]$. Here A is used twice as the same part of the pattern. Hence, the downward closure property is not guaranteed for minimal occurrences.

A closer investigation shows that *contains* is the only relation for which the property does not hold. Consider the pattern A *overlaps* B. The downward closure property can only fail if A overlaps two or more B's. This is impossible as these B's would have to share a time interval which is prohibited by the maximality assumption (Equation 1). The same argument holds for *meets*, *is-finished-by*, *is-started-by*, *equals* and their inverse relations. In case of A *before* B there can be several B's after the A but the definition of minimal occurrences allows the first B to be counted.

Obviously the downward closure property only fails for unconnected temporal patterns. Our approach to solve this problem is by treating connected and unconnected temporal patterns differently.

When generating candidates in the Apriori-way, the candidate pattern A *equals* B, A *and* B *contain* C would be generated based on the two subpatterns A *equals* B and A *contains* C. However, A *equals* B needs not necessarily be frequent even if the resulting candidate A *equals* B, A *and* B *contain* C is frequent.

In the example above, the subpatterns A *contains* C and B *contains* C are necessarily frequent if the candidate itself is frequent. Generally, in a valid unconnected (k+1)-pattern, $k \geq 2$, there exists a $j, 1 \leq j \leq k$, such that the first j labels always describe those temporal intervals which contain all other temporal intervals of the pattern (labels $j+1, \dots, k+1$). *Contains* implies "starts before and ends after", so the ordering of sequences guarantees exactly this property. Hence, in the opposite way, the last $k+1-j$ labels of the pattern are always responsible for the frequency of the pattern. If we remove the first or second label from a frequent (k+1)-pattern the resulting k-patterns must still be frequent. In other words, generating (k+1)-candidates by joining the two subpatterns that share the last k–1 labels ensures that the set of generated (k+1)-candidates is a superset of the frequent (k+1)-patterns. We only need to guarantee that the initial set of frequent 2-patterns contains all frequent unconnected patterns.

Thus, by modifying the candidate generation step, we can transfer the completeness property of the Apriori approach, to unconnected temporal patterns. In detail, our approach of candidate generation is as follows: The candidate patterns of size 1 are generated by using all available labels in the dataset. In the next step, we use all 1-candidates to create the candidate patterns of size 2. This guarantees that we will find all frequent 2-patterns, albeit they are connected or unconnected. For the actual candidate generation and test approach the frequent patterns of size k ($k \geq 2$) are used to generate the candidate patterns of size k+1. This is achieved by joining every pair of temporal patterns P and Q which are identical w.r.t. the last k-1 rows and columns of their relation table, i.e. they share common (k–1)-pattern on the last k-1 labels. Each temporal pattern describes the desired (k+1)-pattern except for one label. If we join P and Q to the temporal pattern R, as it is illustrated in Figure 3, then there is only one interval relation missing in R.

The missing interval relation describes the relation between the first labels of P and Q (l_1^p and l_1^q). Now we can extend R to a set of candidates by applying Allen's interval relations. The missing value in R is substituted by each of Allen's interval relations. Hence, the extension of R leads to 7 (k+1)-candidate patterns.

The second step is the support evaluation of the candidate patterns. As already mentioned, the labels of a valid temporal pattern have the same order as their counterparts in an instance would have. Therefore we can find an instance of a temporal pattern by using finite state machines which subsequently take the temporal intervals of an ordered temporal sequence as input.

It is straightforward to derive a finite state machine from a temporal pattern. For each label in the pattern a state is generated. The finite state machine starts in an initial state. The next state is reached if we input a temporal interval that

$$\begin{array}{c||c|c} \text{Pattern } P & l_1^p & l_2 \dots l_k \\ \hline\hline l_1^p & \text{e} & \text{C} \\ \hline l_2 & & \\ \vdots & \text{B} & \text{A} \\ l_k & & \end{array} \;\cup\; \begin{array}{c||c|c} \text{Pattern } Q & l_1^q & l_2 \dots l_k \\ \hline\hline l_1^q & \text{e} & \text{E} \\ \hline l_2 & & \\ \vdots & \text{D} & \text{A} \\ l_k & & \end{array} \;\Longrightarrow\; \begin{array}{c||c|c|c} \text{Pattern } R & l_1^p & l_1^q & l_2 \dots l_k \\ \hline\hline l_1^p & \text{e} & ? & \text{C} \\ \hline l_1^q & ? & \text{e} & \text{E} \\ \hline l_2 & & & \\ \vdots & \text{B} & \text{D} & \text{A} \\ l_k & & & \end{array}$$

Fig. 3. Temporal Patterns P, Q share a k-1 subpattern, joining P and Q yields R

contains the same label as the first label of the temporal pattern. From now on the next states can only be reached if the shown temporal interval carries the same label as the state and its interval relation to all previously accepted temporal intervals is the same as specified in the temporal pattern. If the finite state machine reaches its last state it also reaches its final accepting state.

We can derive the minimal time window in which this particular pattern instance is visible from the set of temporal intervals which has been accepted by the state machine. We know that the time window contains an instance of the pattern but we do not know whether it is a minimal occurrence. Therefore we apply a two step approach. First we will find all occurrences of a pattern using state machines. Second we will filter out all occurrences which are not minimal.

To find all occurrences of a pattern in an interval sequence we are maintaining a set of finite state machines. At first, the set only contains the finite state machine that is derived from the candidate pattern. Subsequently, each temporal interval from the interval sequence is shown to every finite state machine in the set. If a finite state machine can accept the temporal interval a copy of the state machine is made and added to the set. The temporal interval is shown to only one of these two state machines . Hence, there will always be a copy of the initial state machine in the set trying to find an occurrence of the pattern.

5 Performance Evaluation and Conclusions

In order to evaluate the performance of the algorithm we employed a dataset from our domain. This dataset contains information about 101 250 vehicles. We performed 5 different experiments varying the minimum support threshold from 3 200 down to 200.

Figure 4a) shows the number of candidates that are generated in each iteration. Obviously the number of candidates grows rapidly as the minimum support threshold gets lower. This general behaviour is well known from frequent itemset mining. In contrast to frequent itemsets, Figure 4a) shows two distinct peaks. There is one peak for the candidate patterns of size 2 and one peak for patterns of sizes 6-7. Moreover, the first peak does not vary with different minimum support thresholds. This peak is a result of the special candidate generation in the

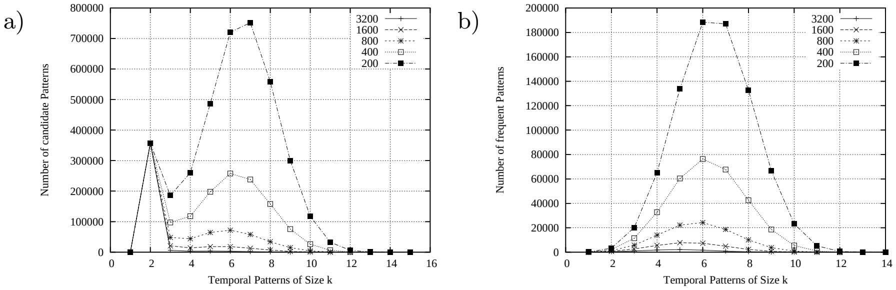


Fig. 4. a) Candidate Patterns generated b) Frequent Patterns found in each Iteration

first iteration of our algorithm. The candidates of size 2 are generated by using all the candidates of size 1 (the frequent patterns are only used in subsequent iterations). Consequently the number of 2-candidates is independent of the chosen minimum support threshold.

In Figure 4b) the number of frequent patterns is depicted for each iteration. For each minimum support threshold the maximum number of frequent patterns is found between the 5-th and 7-th iteration. Again the number of frequent patterns grows rapidly with decreasing minimum support thresholds.

The increasing number of frequent and candidate patterns also leads to longer runtimes of the algorithm for decreasing minimum support thresholds. While the first experiment (MinSupp=3 200) was finished within 36 minutes subsequent experiments took 98, 295, 1052 and 2822 minutes.

In this paper we presented a new algorithm for discovering frequent temporal patterns. The key advantages of this algorithm are the ability to mine data that consists of several separate interval sequences, a new support definition that allows counting multiple instances of a pattern per sequence, and finally the consideration of time constraints on pattern instances.

Whereas on its own, these features have been described before, e.g. [4, 6], an algorithm implementing them all together had not yet been available. For our and many other applications combining these features is essential. Thus, our approach opens a broad range of new applications for sequence mining.

Combining the sketched features is far from being straightforward. The main algorithmic challenge is that the downward closure property of support, also known as the Apriori-criteria is not met. In other words, we had to develop an algorithm that is efficient and still complete with respect to a minimal support threshold, although subpatterns of a frequent pattern may be infrequent. We finally tackled the issue by distinguishing so called connected and unconnected patterns based on the temporal relation *contains*.

Based on an analysis of warranty data from the automotive domain, we showed that the algorithm can be successfully applied to real world data. The results contained valuable knowledge far beyond current approaches and were produced within reasonable time.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int. Conf. on Very Large Databases (VLDB '94)*, pages 487–499, 1994.
2. R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of the 11th Int. Conf. on Data Engineering (ICDE '95)*, pages 3–14, 1995.
3. J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.
4. F. Höppner and F. Klawonn. Finding informative rules in interval sequences. *Intelligent Data Analysis*, 6(3):237–255, 2002.
5. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.
6. P. Papapetrou, G. Kollios, S. Sclaroff, and D. Gunopulos. Discovering frequent arrangements of temporal intervals. In *5th IEEE Int. Conf. on Data Mining*, 2005.