# Mining Temporal Patterns in an Automotive Environment

**Steffen Kempe**
Daimler AG
Group Research
Steffen.Kempe@daimler.com

**Rudolf Kruse**
University of Magdeburg
Dept. of Knowl. and Language Engineering
Kruse@iws.cs.uni-magdeburg.de

## Abstract

Mining frequent temporal patterns from interval-based data proved to be a valuable tool for generating knowledge in the automotive business. Many problems in our domain contain a temporal component and thus can be formulated by using interval sequences. In this paper we present three substantially different applications which can all be addressed by the same mining task: mining of frequent temporal patterns. We show that contemporary approaches for temporal pattern mining are not addressing this task sufficiently and present our algorithmic solution *FSMTree*. Further, we discuss the assessment of temporal rules which can be derived from the set of frequent patterns.

**Keywords:** frequent temporal patterns, industrial application.

## 1 Introduction

Mining sequences from temporal data is a well known data mining task which gained much attention in the past (see e.g. [2, 7]). In all these approaches, the temporal data is considered to consist of events. Each event has a label and a timestamp. Hence, two events can either happen one after the other or exactly at the same time. 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.

At Daimler we are interested in analyzing sequences of temporal intervals in order to further extend the knowledge about our products, to improve customer satisfaction, or to assist the development process. In the following, we will describe three application examples from different business areas which can all be represented by using temporal intervals.

Application one: Quality monitoring of a vehicle fleet. A major task for any vehicle manufacturer is to monitor the quality of the product in the field. Temporal data mining can support this task by identifying sequences of faults or combinations of faults and vehicle configurations which occur more often than others. In this case one interval sequence describes 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 beginning and the current day is the end of such a temporal interval. Other temporal intervals contain information about garage stopovers or the installation of additional equipment. The frequent patterns from a set of interval sequences (i.e. from a vehicle fleet) can be used by an engineer to introduce product changes if necessary.

Application two: Analytical customer relationship management. Many of our customers are rebuyers, i.e. they buy a vehicle, keep it for a certain period of time, sell it, and buy a new vehicle. These customers are especially valuable as they show a high brand loyalty. Mining the information that we have about these "good" customers might help us to determine where we have cross- and upselling potential for other customers. The information about the customers comes from two sources. The first source is the sale itself. Here the vehicle type (model line, special option codes, etc.) and sales type (whether leasing or full buy was preferred) are of particular interest. The second source are questionnaires which were sent to the customers. The questionnaires range from micro-economic questions (age, size of household, income, etc.) to customer satisfaction (repair shops, marketing, etc.). The patterns found are useful to guide marketing or commercial actions.

Application three: Mining CAN-Bus data. The popularity of electronic systems (navigation, mobile phones, antiblock system, etc.) has led to a steady increase in the number of electronic control units (ECU) within a vehicle. Most of the ECUs are communicating with each other over a CAN-Bus System (Controller Area Network). During the development of new ECUs or a new model line the network traffic of the CAN-Bus is particularly analyzed as it contains the status and all status changes of the connected ECUs. There is also information about the general driving conditions available (e.g. speed, gear, steering angle, etc.). All these information can be expressed by using interval sequences. Then one interval sequence belongs to one trip of one vehicle. Mining these sequences for frequent patterns helps to identify typical driving situations for the ECUs. In case of an ECU malfunction temporal patterns can also support electronic diagnostics by pointing out the conditions and their relations under which the malfunction occurred.

Despite their different application areas all three examples share a common problem setting. There are instances (vehicles, customers, trips) which are described by a sequence of temporal intervals. The mining task is to find all frequent temporal patterns within these interval sequences. The rest of this paper is organized as follows. In the next Section we introduce related work in the field of mining frequent temporal patterns from interval-based data. We argue that the commonly used definitions for the support of temporal patterns are not feasible for our application examples. Therefore, we formally define our understanding of the mining task. Afterwards in Section 3 we briefly present our algorithm for mining frequent temporal patterns — *FSMTree*. In Section 4 we describe open issues regarding the assessment of temporal rules which can be derived from the set of frequent patterns.

## 2 Related Work and Foundations

Previous investigations on discovering patterns from interval sequences include the work of Höppner [4], Kam and Fu [5], Papapetrou et al. [8], and Winarko and Roddick [10]. These approaches can be divided into two different groups.

The main difference between both groups is the definition of support. Höppner defines the *temporal support of a pattern*. This definition is closely related to the *frequency* in [7]. The temporal support can be interpreted as the probability to see an instance of the pattern within a time window if the time window is randomly placed on the interval sequence. All other approaches count the number of instances for each pattern. The pattern counter is incremented once for each sequence that contains the pattern. If an interval sequence contains multiple instances of a pattern then these additional instances will not further increment the counter. This way of counting instances of a pattern was introduced in [2].

For our applications neither of the support definitions turned out to be satisfying. Höppner's temporal support of a pattern is hard to interpret in our domain, as it is generally not related to the number of instances of this pattern in the data. Also neglecting multiple instances within one interval sequence is infeasible in our applications. Thus, we extended the approach of minimal occurrences in [7] to the demands of temporal intervals (see below). In contrast to previous approaches, this support definition allows at the same time: 1. to count the number of pattern instances, 2. to handle multiple instances of a pattern within one interval sequence, and 3. to apply time constraints.

Counting the pattern instances as many times as they occur is a demand that is shared in all our applications. The reason is that the patterns have to be interpreted by a domain expert before further actions are started. For a domain expert the number of pattern instances is important information which is at the same time easily understandable. Applying time constraints is a demand that arises in the quality monitoring application. Here a sequence of faults that extends over a long time is less likely to bear a causal connection than a sequence that extends over a short time. Hence, a desired time constraint is to count only pattern instances which do not extend longer than a user defined threshold.

As mentioned above we represent a temporal interval as a triplet. Based on the temporal intervals we define interval sequences.

**Definition 2.1** *(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 2.2** *(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 (1) $\forall (b_i, e_i, l_i), (b_j, e_j, l_j) \in I, i \neq j :$ $b_i \leq b_j \wedge e_i \geq b_j \Rightarrow l_i \neq l_j$ and (2) $\forall (b_i, e_i, l_i), (b_j, e_j, l_j) \in I, i < j :$ $(b_i < b_j) \vee (b_i = b_j \wedge e_i < e_j) \vee (b_i = b_j \wedge e_i = e_j \wedge l_i < l_j)$ 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. Due to the temporal extension of temporal intervals the possible relations between two intervals become more complex. 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 2.3** *(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" or "n-pattern".*

Figure 2 shows an example of an interval sequence. The corresponding temporal pattern is given in Figure 2.b.

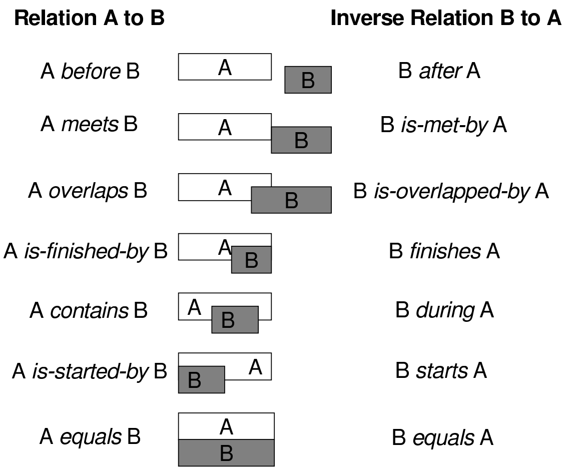
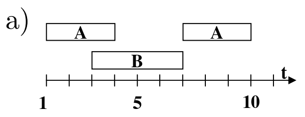


Figure 1: Allen's Interval Relations

a) 

b)

| | A | B | A |
|---|---|---|---|
| A | e | o | b |
| B | io | e | m |
| A | a | im | e |

Figure 2: a) Example of an Interval Sequence b) Example of a Temporal Pattern (e stands for *equals*, o for *overlaps*, b for *before*, m for *meets*, io for *is-overlapped-by*, etc.)

Note that a temporal pattern needs not necessarily be valid in the sense that it must be possible to construct an interval sequence for which the pattern holds true. On the other hand, if a temporal pattern holds true for an interval sequence we consider this sequence as an instance of the pattern.

**Definition 2.4** *(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 $\forall i, j : s(i) = l_i \wedge s(j) = l_j \wedge R[i, j] = \mathrm{ir}([b_i, e_i], [b_j, e_j])$ 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. Next, we define the support of a temporal pattern.

**Definition 2.5** *(Minimal Occurrence) For a given interval sequence $S$ a time interval (time window) $[b, e]$ is called a minimal occurrence of the k-Pattern* P *($k \geq 2$), if 1. the time interval $[b, e]$ of $S$ contains an instance of* P*, and 2. there is no proper subinterval $[b', e']$ of $[b, e]$ which also contains an instance of* P*. For a given interval sequence $S$ a time interval $[b, e]$ is called a minimal occurrence of the* 1*-Pattern* P*, if 1. the temporal interval $(b, e, l)$ is contained in $S$, and 2. $l$ is the Label in* P.

**Definition 2.6** *(Support) The support of a temporal pattern $P$ for a given set of interval sequences $\mathbb{S}$ is given by the number of minimal occurrences of $P$ in $\mathbb{S}$: $Sup_{\mathbb{S}}(P) = |\{[b, e] : [b, e]$ is a minimal occurrence of $P$ in $S \wedge S \in \mathbb{S}\}|$.*

As an illustration consider the pattern *A before A* in the example of Figure 2.a. The time window $[1, 11]$ is not a minimal occurrence as the pattern is also visible e.g. in its subwindow $[2, 9]$. Also the time window $[5, 8]$ is not a minimal occurrence. It does not contain an instance of the pattern. The only minimal occurrence is $[4, 7]$ as the end of the first and the beginning of the second $A$ are just inside the time window.

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

## 3 Algorithmic Solution: FSMTree

The main idea is to generate all frequent temporal patterns by applying the Apriori scheme of candidate generation and support evaluation [1]. These two steps are alternately repeated until no more candidates are generated. The Apriori scheme starts with the frequent 1-patterns and then successively derives all $k$-candidates from the set of frequent ($k$-1)-patterns.

The candidate generation is achieved by joining any two frequent patterns of size $k$, which share a common ($k$-1)-pattern (see e.g. [4, 6]). However, in this paper we focus on the support evaluation of the candidate patterns, as it is the most time consuming part of the algorithm. *FSMTree* uses finite state machines which subsequently take the temporal intervals of an interval sequence as input to find all instances of a candidate pattern. This approach is also favored in [7].

It is straightforward to derive a state machine from a temporal pattern. For each label in the temporal pattern a state is generated. The state machine starts in an initial state. The next state is reached if we input a temporal interval with 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 state machine reaches its last state it also reaches its final accepting state. Consequently the temporal intervals that have been accepted by the state machine are an instance of the temporal pattern. The minimal time window in which this pattern instance is visible can be derived from these temporal intervals. Figure 4 depicts an example of four state machines that are combined in a single datastructure.

The main idea to find all pattern instances is to use a set of state machines. At first, the set only contains the state machines that are derived from the candidate patterns. Subsequently, each temporal interval from the interval sequence is shown to every state machine in the set. If a state machine can accept the temporal interval, a copy of the state machine is added to the set. The temporal interval is shown only to one of these two state machines. Hence, there will always be a copy of the initial state machine in the set trying to find a new instance of the pattern. In this way we can also handle situations in which single state machines do not suffice. Consider the pattern *A meets B* and the interval sequence (1, 2, A), (3, 4, A), (4, 5, B). Without using look ahead a single state machine would accept the first temporal interval (1, 2, A). This state machine is stuck as it cannot reach its final state because there is no temporal interval which *is-met-by* (1, 2, A). Hence the pattern instance (3, 4, A), (4, 5, B) could not be found by a single state machine. Here this is not a problem because there is a copy of the first state machine which will find the pattern instance.

Figures 3 and 4 give an example of *FSMTree*'s support evaluation. There are four candidate patterns (Figure 3.a – 3.d) for which the support has to be evaluated on the given interval sequence in Figure 3.e.

In an initialization step *FSMTree* efficiently organizes all state machines within a prefixtree-like datastructure. Figure 4 depicts the prefixtree for the candidate patterns of Figure 3. Each path of the tree is a state machine which belongs to one of the candidates. But here different state machines can share states if their candidate patterns share a common pattern prefix.

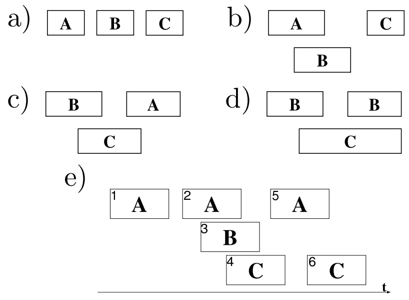


Figure 3: a) – d) four candidate patterns of size 3 e) an interval sequence

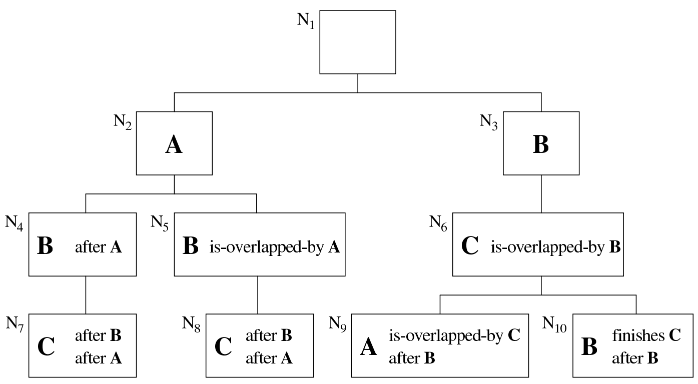
For the support evaluation *FSMTree* maintains a list of nodes from the prefix tree. In the first step the list only contains the root node of the tree. Afterwards all temporal intervals of the interval sequence are processed subsequently. Each time a node of the set can accept the current temporal interval its corresponding child node is added to the set. In the example *FSMTree* will need 11 nodes in order to find all three pattern instances.

## 4 Temporal Rules

The discovery of frequent temporal patterns is only the first part to support our applications. Like in frequent itemset mining we have to derive rules from the frequent patterns to gain additional information. A temporal rule consists of two parts: the antecedent and the consequence. Both parts are temporal patterns but the antecedent is a subpattern of the consequence.

Consider the example in Figure 2. If we remove the last label of the temporal pattern we get the subpattern *A overlaps B*. The combination of both patterns in a temporal rule is depicted in Figure 5. This temporal rule "forecasts" the existence of a third label *A* that *is-met-by B* and *after* the first *A*.

We derive such rules in the same way as asso-



$N_1$
$N_2$ A
$N_3$ B
$N_4$ B after A
$N_5$ B is-overlapped-by A
$N_6$ C is-overlapped-by B
$N_7$ C after B after A
$N_8$ C after B after A
$N_9$ A is-overlapped-by C after B
$N_{10}$ B finishes C after B

Figure 4: FSMTree: prefix tree of state machines based on the candidates of Figure 3

ciation rules are derived from frequent itemsets [1]. For each frequent temporal pattern we use the set of all its subpatterns to generate temporal rules. However, in our applications we can restrict ourselves to temporal rules which make a prediction in the future.

It is important to point out some differences between association rules and temporal rules. In an association rule $X \Rightarrow Y$, $X$ and $Y$ denote disjoint itemsets ($X \cap Y = \emptyset$). The intersection of $X$ and $Y$ is empty because the semantics of association rules allows only one possible relation between the items of $X$ and $Y$ — they are in the same transaction. Therefore the rule could be verbalized as "*If* a transaction contains the items of $X$ *then* it **also** contains the items of Y". In a temporal rule the relations between the labels of the antecedent and the predicted labels are more complex. If the consequence pattern of Figure 5 would only contain the predicted labels (i.e. the 1-pattern $A$) then we would not know whether e.g. $A$ *is-overlapped-by*, *is-finished-by*, or *is-met-by* $B$. Hence, in a temporal rule the antecedence pattern must be subpattern of the consequence pattern to completely explain the relations between the predicted labels and the labels of the antecedence.

$$\begin{array}{c|cc} & \text{A} & \text{B} \\ \hline \text{A} & \text{e} & \text{o} \\ \text{B} & \text{io} & \text{e} \end{array} \Longrightarrow \begin{array}{c|ccc} & \text{A} & \text{B} & \text{A} \\ \hline \text{A} & \text{e} & \text{o} & \text{b} \\ \text{B} & \text{io} & \text{e} & \text{m} \\ \text{A} & \text{a} & \text{im} & \text{e} \end{array}$$

Figure 5: Example of a Temporal Rule

For association rules there are many rule measures available which help to identify strong correlations between items, e.g. lift, gini-index, J-measure, conviction, etc. (see [9] for an overview). The most basic measures are the support and the confidence of a rule. Following the well known paths of association rules these measures can be directly transferred to temporal rules. Thus, for a temporal rule $X \Rightarrow Y$ the support simply states how often the rule is correctly applicable, $\text{Sup}(X \Rightarrow Y) = \text{Sup}(Y)$. The confidence gives the ratio of how often a rule is correctly applicable to the number of times it is applicable (only antecedence holds true), $\text{Conf}(X \Rightarrow Y) = \frac{\text{Sup}(Y)}{\text{Sup}(X)}$.

Unfortunately, it turns out that more sophisticated measures are often not transferable to temporal rules in our application setting. Figure 6.a shows the 2×2-contingency table for the association rule $X \Rightarrow Y$. The contingency table simply gives the supports of $X$, $Y$ and their opposites $\overline{X}$, $\overline{Y}$ alone and in any combination ($XY$, $X\overline{Y}$,$\overline{X}Y$ and $\overline{XY}$). In fact, this small table contains all information (like a priori and conditional probabilities of the antecedence or consequence) that is necessary to compute contemporary rule measures [9]. Figure 6.b shows the result if we transfer the idea of a contingency table to temporal rules. The columns denote the predicted pattern and its inverse. In contrast to association rules, where predicted and consequence pattern are the same, we have to extract the predicted pattern by removing the antecedence from the consequence pat-

| a) | $Y$ | $\overline{Y}$ | |
|---|---|---|---|
| $X$ | $\lvert XY\rvert$ | $\lvert X\overline{Y}\rvert$ | $\lvert X\rvert$ |
| $\overline{X}$ | $\lvert\overline{X}Y\rvert$ | $\lvert\overline{XY}\rvert$ | $\lvert\overline{X}\rvert$ |
| | $\lvert Y\rvert$ | $\lvert\overline{Y}\rvert$ | $\sum = N$ |

| b) | $Y\backslash X$ | $\overline{Y\backslash X}$ | |
|---|---|---|---|
| $X$ | $\lvert Y\rvert$ | $\lvert X\rvert - \lvert Y\rvert$ | $\lvert X\rvert$ |
| $\overline{X}$ | $\lvert Y\backslash X\rvert - \lvert Y\rvert$ | - | - |
| | $\lvert Y\backslash X\rvert$ | - | - |

Figure 6: Contingency tables for the a) association and b) temporal rule $X \Rightarrow Y$ ($|X|$ is used as abbreviation for Sup($X$))

tern (denoted by $Y\backslash X$). The last entry of the first row and column give the support of the antecedence and the predicted pattern, Sup($X$) and Sup($Y\backslash X$). The first entry within the table denotes how often antecedence and predicted pattern occur together. Since, $X$ is a subpattern of $Y$ this information is given by Sup($Y$). The second entry in the first row (column) denotes how often the antecedence (predicted pattern) occurred without being a part of the consequence pattern. This entry can be calculated by the difference between Sup($X$) and Sup($Y$) (and Sup($Y\backslash X$)−Sup($Y$) respectively). We failed to fill out the remaining entries due to semantical problems. For association rules e.g. Sup($\overline{X}$) denotes the number of transactions in the database which do not contain $X$. This value is given by Sup($\overline{X}$)$= N -$ Sup($X$) where $N$ is the number of transactions in the database. This way of calculating Sup($\overline{X}$) is possible as a transaction can contain an itemset at most once. For temporal rules $N$ is the number of interval sequences in the database. Unfortunately, the number of interval sequences is not the basic set upon which the remaining entries can be calculated. The reason is that we allow counting multiple instances of a pattern within one interval sequence. Consider e.g. a set of interval sequences where the pattern $X$ is contained several times in each sequence. Then follows Sup($X$)$> N$ and calculating Sup($\overline{X}$) the association rule way leads to Sup($\overline{X}$)$< 0$. Since the support is a nonnegative function we cannot use the number of interval sequences as a basic set. Hitherto we have not found a solution to complete the entries in the contingency table of a temporal rule which is at the same time consistent with the needs of our applications (i.e. counting a pattern multiple times within an interval sequence). We consider this problem an open issue and it will be a focus of our future work.

We cannot evaluate any rule measure that is based on one of the missing entries at the moment. E.g. the j-measure (which is favored in [4]) is not calculable as it needs the a priori probability of $\overline{X}$, $P(\overline{X}) = \frac{\text{Sup}(\overline{X})}{N}$. Therefore, our practical approach was to concentrate on a rule measure (besides support and confidence) that can be calculated upon the given contingency table. Our starting point was the confidence of a temporal rule $X \Rightarrow Y$. Usually, the confidence is interpreted as the probability that the rule makes a correct prediction. Another valid interpretation for temporal rules is that it gives the average number of times a given instance of $X$ can be completed to an instance of $Y$. A high confidence does not mean that there is a high correlation between the antecedence and the predicted pattern because the predicted pattern may simply occur very often in the data. Therefore we discount the confidence by the average number of times the predicted pattern occurs in an interval sequence Avg($Y\backslash X$)$= \frac{\text{Sup}(Y\backslash X)}{N}$. We call the resulting measure lift because the way calculating it corresponds to association rules (Lift$(X \Rightarrow Y) = \frac{\text{Sup}(XY)N}{\text{Sup}(X)\text{Sup}(Y)}$).

$$\begin{aligned} \text{Lift}(X \Rightarrow Y) &= \frac{\text{Conf}(X \Rightarrow Y)}{\text{Avg}(Y\backslash X)} \\ &= \frac{\text{Sup}(Y)N}{\text{Sup}(X)\text{Sup}(Y\backslash X)} \end{aligned}$$

Nevertheless, the interpretation of lift values is different from association rules. For association rules it describes how often the occurrence of the antecedence increases the occurrence of the consequence. This interpretation may be false for temporal rules depending on the dataset used. On the other hand rules with high lift values are still more likely to express causal correlations than low values. For this reason we gained an important benefit for our applications by using it as a ranking for large sets of temporal rules.

## 5 Conclusion

In this paper we presented three different applications of temporal data mining. Although the applications originate from different business areas of our automotive environment they are all subject of the same mining task — the discovery of frequent patterns from interval-based data. We showed that contemporary approaches are not addressing this task sufficiently from the viewpoint of our applications where we have to count the instances of a pattern as many times as they occur. Hence, we defined our understanding of the mining task which uses the number of minimal occurrences as support definition for temporal patterns. Next, we described *FSMTree* an algorithm based on the Apriori-approach for mining all frequent temporal patterns from a given set of interval sequences.

In the last Section we focused on temporal rules. Temporal rules are derived from the set of frequent patterns in the same way as association rules are derived from frequent itemsets. Despite this connection we pointed out important differences between both rule types. Especially the assessment of temporal rules with the well known rule measures remains an open issue in our settings. Briefly described, for a given temporal pattern $X$ it is easy to obtain $\mathrm{Sup}(X)$. But counting a pattern multiple times in an interval sequence makes $\mathrm{Sup}(\overline{X})$ hard to interpret as it cannot be calculated as $N - \mathrm{Sup}(X)$ anymore. Thus, many classical rule measures are not calculable. Nevertheless, we managed to introduce the lift measure for temporal rules. The lift is closely related to the classical lift and proved to be very valuable in our applications if used as a way to rank a set of temporal rules.

## References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int. Conf. on Very Large Databases*, pages 487–499, 1994.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of the 11th Int. Conf. on Data Engineering*, 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] P.-S. Kam and A. W.-C. Fu. Discovering temporal patterns for interval-based events. In *Data Warehousing and Knowledge Discovery, 2nd Int. Conf.*, pages 317–326, 2000.

[6] S. Kempe and J. Hipp. Mining sequences of temporal intervals. In *10th Europ. Conf. on Principles and Practice of Knowledge Discovery in Databases*, pages 569–576, 2006.

[7] 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.

[8] P. Papapetrou, G. Kollios, S. Sclaroff, and D. Gunopulos. Discovering frequent arrangements of temporal intervals. In *5th IEEE Int. Conf. on Data Mining*, 2005.

[9] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Int. Conf. on Knowledge Discovery and Data Mining*, pages 32–41, 2002.

[10] E. Winarko and J. F. Roddick. Discovering richer temporal association rules from interval-based data. In *Int. Conf. on Data Warehousing and Knowledge Discovery*, pages 315–325, 2005.