Mind the Time: Unleashing Temporal Aspects in Pattern Discovery

T. Lammarsch[a], W. Aigner[a], A. Bertone[b], S. Miksch[a], A. Rind[a]

[a]*Institute of Software Technology and Interactive Systems, Vienna University of Technology, Austria*
[b]*Institute for Cartography, Dresden University of Technology, Germany*

Abstract

Temporal Data Mining is a core concept of Knowledge Discovery in Databases handling time-oriented data. State-of-the-art methods are capable of preserving the temporal order of events as well as the temporal intervals in between. The temporal characteristics of the events themselves, however, can likely lead to numerous uninteresting patterns found by current approaches. We present a new definition of the temporal characteristics of events and enhance related work for pattern finding by utilizing temporal relations, like *meets*, *starts*, or *during*, instead of just intervals between events. These prerequisites result in MEMuRY, a new procedure for Temporal Data Mining that preserves and mines additional time-oriented information. Our procedure is supported by SAPPERLOT, an interactive visual interface for exploring the patterns. Furthermore, we illustrate the efficiency of our procedure presenting an benchmark of the procedure's run-time behavior. A usage scenario shows how the procedure can provide new insights.

Keywords: Visual Analytics, KDD, Temporal Data Mining, Data Mining, Time-oriented Data, Pattern Finding, Interactive Visualization

1. Introduction

Data Mining is a central part of Knowledge Discovery in Databases (KDD). A very important data type is time, and one of the most successful approaches for Temporal Data Mining (TDM) is the search for temporal patterns. Methods for temporal pattern finding include clustering, classification, and association rules. Their main goal is disclosing local structures of interest [1]. Several state-of-the-art methods adopt the concepts of events, which are tuples of a time interval (a temporal primitive with an extent [2]) and a set of conditions (e.g., 10 cars are passing a road between 2am and 3am). These methods consider patterns as combinations of events (e.g., after hours with 30 cars, there are often hours with 40 cars). Figure 1.a demonstrates patterns mined by the state-of-the-art approach MuTIny [3] (see below for more details about that approach). We formally define all necessary terms in Section 3.

The task of finding interesting patterns is important in several domains, which is demonstrated by the range of applications covered in related work (Section 2). In this paper, we focus on examples from traffic data analysis, but data from medicine, retail, and other domains are all conceivable. Wong et al. [4] show that similar tasks arise in different domains and how the same Visual Analytics (VA) methods can be used to deal with them.

Pattern finding usually requires complex parameterization. Many approaches require the conditions to be defined in advance (called Apriori algorithms). These algorithms often produce large numbers of patterns that need to be explored. VA can support parameterization, exploration, and iterative re-parameterization by intertwining the approach with interactive visual interfaces [5]. Starting from early work in pattern finding [6], research has increasingly focused on the temporal aspect of patterns. Current methods, like the MuTIny approach [3], are capable of preserving the temporal order of events as well as the intervals in between.[1]

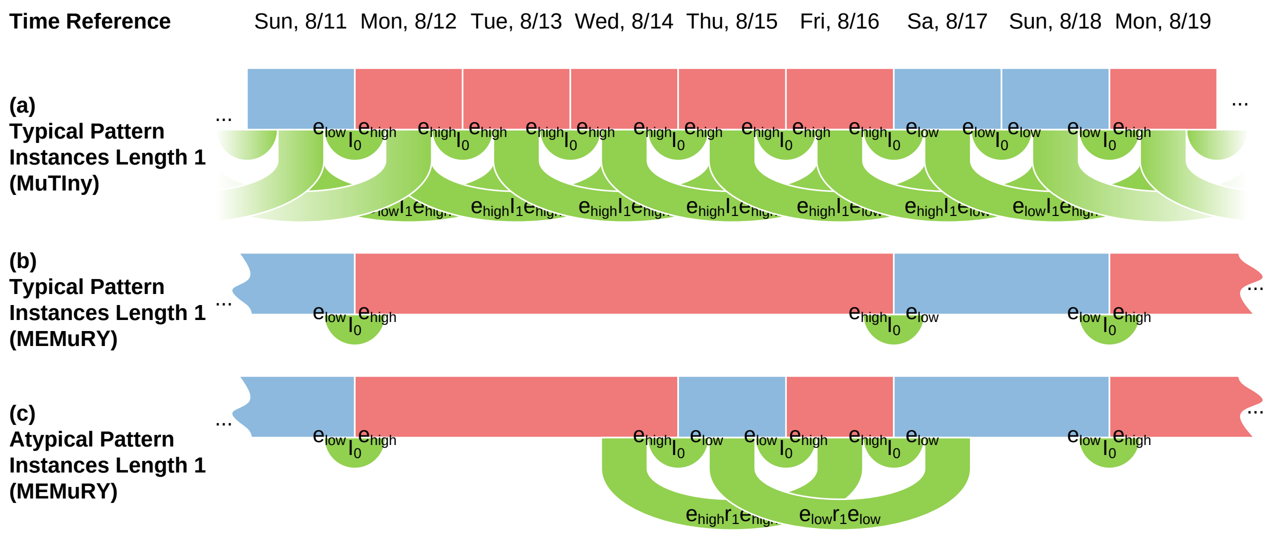
A weakness of current methods is that they only consider events of a fixed length which is usually predetermined by the raster interval[2] of the source data. The most advanced ones can deal with flexible time intervals between events, but they have to be regular multiples of the raster, and the event lengths are still fixed. If the conditions are met for a longer interval than the raster size, two consecutive events of the same type are found.

Consider the following example: a road is used more frequently on weekdays than on weekends. Traffic might be significantly lower on Saturday and Sunday (Figure 1.a). If the traffic dataset has one value for each day, then each event will also have a length of one day. Based on events of day-length, approaches tend to find patterns based on the business week:

Email addresses: `lammarsch@ifs.tuwien.ac.at` (T. Lammarsch), `aigner@ifs.tuwien.ac.at` (W. Aigner), `alessio.bertone@tu-dresden.de` (A. Bertone), `miksch@ifs.tuwien.ac.at` (S. Miksch), `rind@ifs.tuwien.ac.at` (A. Rind)

[1]According to Aigner et al. [2], what is often called "interval" in related work from TDM should be distinguished between "indeterminate interval" after pattern finding and "indeterminate span" during parameterization. In this paper, we will just use the term "interval" for both cases.

[2]A raster is "a fragmentation of time without gaps consisting of raster intervals (usually with same lengths). A raster interval is a unit of time that constitutes a raster: 'hour', 'day', 'week', or '30' " [7].



Time Reference
Sun, 8/11
Mon, 8/12
Tue, 8/13
Wed, 8/14
Thu, 8/15
Fri, 8/16
Sa, 8/17
Sun, 8/18
Mon, 8/19
(a) Typical Pattern Instances Length 1 (MuTIny)
(b) Typical Pattern Instances Length 1 (MEMuRY)
(c) Atypical Pattern Instances Length 1 (MEMuRY)

Figure 1: Examples for patterns generated with (a) the MuTIny approach [3] and (b,c) with the MEMuRY procedure from this paper. (a) Even for few event and interval types, MuTIny [3] finds a vast number of patterns. (b) There are considerably less patterns found by MEMuRY, when looking for events that start one day after other events end, no patterns are found in this example at all. (c) If something unusual happens, like a low value on a Thursday, there are more patterns than for regular weeks.

For five consecutive days, there is high traffic, and for two consecutive days there is low traffic. These sequences alternate. The main pattern found in this example is high traffic for one day (as the approaches only support the raster interval) followed by high traffic for one day. The next frequent pattern is low traffic for one day followed by low traffic for one day (slightly more frequent than high–low and low–high because of “long weekends”), and so on. The important information gained is strongly cluttered by unimportant information. When more intervals are looked for, e.g., one day in between, the number of patterns further increases. A similar problem can happen when the same dataset is looked at using an hour raster: during day hours, there often is much more fluctuation. At night, the traffic is continuously low. As a result, existing approaches hide patterns among the daily fluctuations behind a dominant pattern of low traffic follows low traffic. Such patterns are interesting, but by talking to domain experts, we found out that they are already well-known. An interview with a domain expert working in analysis of time-oriented data and scheduling tasks like shift-planning (Subsection 6.4) confirmed our own assessment: It is too difficult to apply state-of-the-art methods in practice if they generate too many patterns that stem from well-known effects of social time. The reason is that those patterns clutter the results when searching for new and unknown patterns: They make the list of results too large, so that it is hard to compute and even harder to visualize and understand.

Related work mostly relies on filtering out the less important patterns using the well-known concept of support (Section 2). In this paper, we give values for support as the fraction of *number of times a certain pattern exists/number of total patterns*. We also call this the frequency of a pattern, and it is unit-less. Only patterns with a certain frequency (often a high one) are considered. As we explained above, this approach is hard to parameterize if frequent patterns are not necessarily interesting, most likely because they are well-known to domain experts. Bertone et al. [5] propose including time information (like, “weekend”) in the event definition, so that it is easier to prune weekend events followed by weekend events. However, this approach still does not provide events that cover a whole weekend and users end up with many more different event and pattern types.

Another solution proposed for this problem stems from data simplification. These methods transform raw data, which usually is given in the form of numerical data values to events and, thus, help to keep the complexity low [8, 9, 10, 11]. According to the temporal aspects, these methods can simplify time by rasterization (converting the data to a given raster). If there are too many uninteresting patterns from several consecutive events of the same type, for example, during night hours or weekends, rastering the data to days can solve that problem, but possibly interesting patterns on finer granularities that exist during other times become hidden. Moreover, if a day raster is used to solve problems with night hours, the weekend problems arise more prominently. A fine raster size is needed to prevent information loss. Aggregating according to domain knowledge is better, so aggregating to blocks of eight hour length can even out the night/day issues while still giving some information. However, some information might still be hidden, and in real-world data, we also found that sometimes, the same effect just started one hour earlier or later. These hours would have been aggregated into the wrong block. As a result, we use the part of data simplification that simplifies the data dimensions into univariate events. For these events, only properties of nominal data are required. Aggregation over time is not an integrated part of our procedure, but we analyze its applicability in preprocessing in Section 6.

To deal with the shortcomings of existing pattern finding methods without resorting to aggregation over time, we presented a new procedure called Multi Event-length Multi Relation discoverY (MEMuRY) at the EuroVA 2013 workshop in Leipzig, Germany [12]. This paper is an extended version of our workshop paper. MEMuRY's basic idea is to reduce the number of unimportant patterns while retaining the important ones. This is done by reducing the number of events in a way that omits repetitive patterns from effects like weekends and night hours, which are usually the problematic cases. This is done by adapting the event length dynamically. This event length is preserved in the resulting pattern data (Figure 1.b), and can later be investigated visually, like in the arc view (Section 6, Figure 5).

In Figure 1.b, the weekend is still found, but represented by considerably less patterns. Therefore, these patterns do not clutter the possible case that something unusual happens, for example a Thursday with low traffic, as shown in Figure 1.c. We will further discuss the effect over the course of this publication, after our procedure has been explained in detail.

To gain the maximal benefit from this new procedure, it has to be integrated in an interactive visual interface, as the limitations of related work can be greatly reduced, but not fully solved by automated methods. We explain details about this in Section 4. Visualizations of output from our new procedure provide more insights than visualizations of output from related work (Section 6), as they are less cluttered.

Contributions. Our main contribution is a new procedure for TDM (Section 3) that has the following advantages over related work:

- It combines more information in fewer patterns and, thus, helps users to find important patterns.
- The procedure also provides more freedom in pattern definition by the means of free relation definition between events.

Our procedure is evaluated by several means:

- We implemented an example implementation that was benchmarked (Section 5).
- We followed a usage scenario showing how our procedure can be applied and describing findings (Section 6).
- We obtained an assessment of the applicability from the point of view of a domain expert, which is provided by listing the qualitative feedback from an interview (Subsection 6.4).

As a motivation for future work, we also provide an analysis how users can interact with the pattern finding procedure (Section 4).

2. Related Work

Related work relevant to our procedure can be broken down into three categories: pattern finding, data simplification, and interactive visual interfaces.

2.1. Pattern Finding

In their overview of TDM, Laxman and Sastry [1] present many techniques with key references, including global models as well as local methods, like pattern finding. Most work in finding patterns as a Data Mining task goes back to Agrawal et al. [6] who introduced the Apriori algorithm, but only considered patterns of events happening together in a set (they use the example of an event being the purchase of a product). They also consider time, but only as a method of separating different pattern candidates. However, they already perform planning towards further steps with more complex patterns that can span across multiple time steps, published by Srikant and Agrawal [13]. Agrawal et al. [6], Srikant and Agrawal [13], and others, also introduced the concept of "support". The support of a pattern is the frequency of its appearance. It is the main method to determine which patterns are important for many of the algorithms we present below. Support is a good method to find important patterns based on quantities, but as described in Section 1, it is also possible that frequent patterns are not interesting because they are well-known.

Mannila et al. [14] provide a concept of sequences of events, which is similar to the patterns of other approaches. For them, the events and the time steps in between are important, but they count the exact time steps between events, instead of using variable intervals. Magnusson [15] is among the first to explicitly mention time intervals between events. His T-patterns are tree-shaped and, therefore, differ from the patterns in most other approaches which are linear sequences of events. Chen et al. [16] introduce the I-Apriori algorithm which extends the Apriori algorithm [6] for pattern finding by the consideration of intervals between events (differences explained by Laxman et al. [1]). Hu et al. [17] provide a similar approach where the focus is on patterns with events that do not need to be consecutive, as long as the time intervals are kept.

Bertone et al. [3] provide a similar approach, called MUlti-Time INterval pattern discoverY (MuTIny). Several classes of time intervals can exist between events in this approach. For example, events that are 0–1 days apart form a kind of pattern, but if they are 1–7 days apart they form a different kind of pattern. The exact time intervals used can be configured by users, who can also consider calendar aspects, giving the intervals in hours, days, or weeks, and so on. Bertone et al. [5] also deal with interactive visualizations that allow users to engage with the process. Furthermore, they advance the concept of support by allowing user manipulation (see below). Still, frequent but known patterns, like the weekends mentioned in Section 1, can appear and have to be removed by users. In this approach, events are defined by users. They represent the occurrence of a certain configuration of values of one or more variables. Furthermore, users have to define intervals. These intervals can have any length, but users can apply different calendar granularities, like months or days, in their definition. After defining the events and intervals, an extended I-Apriori algorithm is applied in multiple iterations. In each iteration, each existing pattern (starting with patterns of length zero that are identical to events) is combined with each interval and each possible follow-up event that falls into that interval. Therefore, with each iteration, the patterns grow by one interval and one event. Between those iterations, the less frequent patterns are filtered out. This is done by users providing a threshold for the support.

When dealing with approaches based on the Apriori algorithm, two different views on the results are possible [18]:

1. Showing the occurrence of events and patterns over time. To distinguish that view from showing pattern classes, events or patterns are also called “event instances” or “pattern instances” in this context. Each time an event or pattern is found by an approach, an instance is created and usually kept as a tuple of one or more time references and one or more identifiers.
2. Counting the number of times each event or pattern, specified by their identifiers, exists, ignoring the time references, and showing the different types or classes of patterns that appear and their multiplicity. This is done by Bertone et al. in the 2C visualization [5]. The structure of the pattern classes is distributed over several concentric rings, and the start of the patterns is prominently in the center.

2.2. Data Simplification by Discretization

Simplifying the temporal aspects of the events themselves currently can only be done by changing the raster with methods like Piecewise Aggregate Approximation (PAA) by Keogh et al. [8, 9]. They were compared to methods, like Singular Value Decomposition, Discrete Fourier Transformation, and Discrete Wavelet Transformation [11]. The outcome of this comparison by Lin et al. [11] is that they have useful properties, but due to being real valued, have limited applicability.

PAA approximation is also part of SAX [10, 11]. Furthermore, SAX provides a kind of event generation that automatically distributes events in an equiprobable number of classes, but only based on one variable. As explained in Section 1, we perceive rasterization over time as difficult, as it can highly affect the results of pattern finding. Therefore, we keep it separate from classifying data values into events. The statistical event classification that is part of SAX, however, is also applied by us in Section 6.

For simplifying multivariate data into univariate events, we have described an interactive visual interface in an earlier publication [19] which we will describe further in the next subsection. In future work, we intend to consolidate that interface with our work presented here.

Another approach that classifies multivariate data from the medical domain is presented by Simonic et al. [20]. They show that by means of interactive visual interfaces, users can handle a complexity of data that would otherwise be overwhelming, or to the least much more time-consuming. The application of interactive visualization leads to the last part of our related work.

2.3. Interactive Visual Interfaces

Bertone et al. [5] not only propose a VA system as a solution of the complexity and multiple configuration levels of the MuTIny approach, but also demonstrate a prototype that allows for changing parametrizations at any stage, recalculating the steps done so far. However, the parameterization is mostly done by setting numeric values through traditional user interface widgets which is more difficult than direct manipulation would have been. Several other authors like Keim et al. [21] or Holzinger [22] also emphasize the importance of interactive visual interfaces to help users dealing with KDD methods.

Tominski [23] gives an overview how interactive visual interfaces are already used to find and display events according to the requirements and domain knowledge of users. He also provides an overview and formal model how events can be found with an interactive visual interface.

Our visual interface for data simplification [19] allows users to deal with multi-variate time-oriented data with methods from statistics, manual configuration, direct manipulation, or a combination of those methods. The result is a categorical value for each point in time. This value is a possible event classification that can be used by KDD approaches like MuTIny [3] or the MEMuRY procedure we present in this paper.

The VISITORS system [24, 25] has an interactive visual interface to explore and query patient data over time. It supports multivariate data and combines both actual values and events. The events are determined by a knowledge-based temporal abstraction system. Similarly, LifeLines2 [26] is a system to explore events in time-oriented patient data. LifeLines2 provides specific interaction techniques such as alignment and temporal summaries. It expects data to be either of a categorical scale or simplified in advance. EventFlow [27] extends Lifelines2 with an overview display, events with duration, and more advanced queries. Lan et al. [28] describe an approach that can also be used for the simplification of time-oriented data. They iteratively search event sequences and replace instances by one event (comparable to a pattern in the approaches described in Subsection 2.1). Activitree [29] provides a powerful interactive visual interface that users can employ to choose which patterns are important while performing algorithms, like those based on the Apriori algorithm [1]. This interface is an important alternative to the concept of support.

Furthermore, we investigated different methods for analysis of results. Bertone et al. focus on visualizing classes of patterns [5]. For exploring individual pattern instances (see above) over time, we introduce methods to pattern visualization that were proposed for different tasks before: ArcDiagram [30], Timelines [2], or ThemeRiver [31]. We will describe the work of Bertone et al. more closely in Section 4, and the other techniques in Section 6.

3. Our Novel Temporal Pattern Finding Procedure

The Multi Event-length Multi Relation discoverY procedure (MEMuRY) is primarily related to the MuTIny approach. [3]. However, we introduce a different definition of events and also extend the intervals to event relations. We do not focus on a certain data domain, but we assume that time is the single reference in the data and each point or interval in time refers to a set of values. We call one such combination of a time reference and data values a data element.

3.1. Defining Events

Our event definition allows for events of any length and is applicable for rastered data (equal time steps with one data element for each one) as well as unrastered data. We also allow temporal conditions for event definition.

There are many different ways to define events. We have abstracted a definition from the related work discussed above, which can also be seen as a simplified variant of the philosophical approach by Kim [32]:

Definition 1. *An **event e** is an interval $[t_{begin}, t_{end}]$ during which given conditions $\{\chi_0, \ldots, \chi_n\}$ are fulfilled.*

Example 1. *If the condition given is "low traffic" and it is fulfilled for a Saturday and a Sunday, the interval over those two days is the appropriate event.*

The conditions that are fulfilled can also be temporal, e.g., an interval is during another interval, e.g., a certain day of week, or an interval has a certain length.

Our goal is computability, so we have to further define these conditions. Helpful definitions for comparing and finding certain values are given in the task framework by Andrienko and Andrienko [33]. For elementary tasks that consider a finite number of data elements, they define lookup tasks, comparison tasks, and relation-seeking tasks. Without loss of generality for those definitions, in our procedure we always use time as reference domain and everything else as data values. We also assume that time is being discretized in collecting the data and that the data elements are sorted by the start of each time reference.

Based on these definitions and assumptions, we can define a condition as follows:

Definition 2. *A **condition χ_i** is fulfilled for a data element d if d is a possible result for an elementary task.*

Example 2. *If a task is "find traffic values of less than 30 cars per hour", then all data elements where the value of cars per hour is less than 30 fulfill the condition.*

As elementary tasks may look for the reference (time), it is possible to define conditions based on time as well as on data values. According to the definition above we can modify our event definition and define event classes:

Definition 3. *An **event e** is a set of contiguous data elements $\{d_0, \ldots, d_m\}$ sorted by the time references t_d that all fulfill the same conditions $\{\chi_0, \ldots, \chi_n\}$.*

Example 3. *If a Saturday and a Sunday (which are a set of contiguous data elements sorted by their time references) both fulfill the condition of having traffic of less than 30 cars per hour, they can form an event, but they only do if an event of this kind is defined by the user.*

Definition 4. *An **event class E** is a collection of events $\{e_0, \ldots, e_h\}$ that fulfill the same conditions $\{\chi_0, \ldots, \chi_n\}$.*

Example 4. *If the cars per hour are less than 30 for all Saturdays and Sundays, but not for other days, and if an appropriate event is defined by the user, then all weekends are events of the same class.*

3.2. Event Relations and Patterns

To combine events to patterns, the original I-Apriori algorithm searches for possible follow-up events for existing patterns that fall into given time intervals. We enhance this by replacing the intervals with temporal relations, based on the ones presented by Allen et al. [34]: *before*, *equal*, *meets*, *overlaps*, *during*, *starts*, *finishes*. Intervals between events are an indeterminate variant of *before*.

Definition 5. *A **pattern p^k** of length k is a tuple (p^{k-1}, r, e), where p^{k-1} is a pattern of length $k-1$, r is a time relation, and e is an event.*

Definition 6. *A **pattern p^0** of length 0 is an event e.*

Definition 7. *A **pattern class P^k** of length k is a collection of patterns $\{p_0^k, \ldots, p_j^k\}$ that consist of events of the same class in the same order and the same relations in the same order.*

We give examples for pattern of length 0, 1, and 2:

Example 5. *The event in Example 3 is also a pattern of length 0.*

Example 6. *If there is an event of less than 30 cars per hour on a weekend (which is also a pattern of length 0), and another kind of event of 30–39 cars per hour on the next Monday, and "meets" is a time relation defined by the user, then "less than 30 cars per hour meets 30–39 cars per hour" is a pattern that exists over those two intervals and the given relation. This pattern is of length 1.*

Example 7. *If there is a pattern as described in Example 6 from a Saturday to a Monday, and there is another kind of event of 40–49 cars per hour on the next Tuesday ("meets" has already been given as a time relation in Example 6), the pattern, the relation, and the event form a pattern of length 2. In practice, the pattern is usually given as a tuple of all the single events and their relations in between.*

3.3. The Procedure

The procedure consists of two steps (event finding and pattern finding). First, the events have to be found. Conditions that, e.g., limit the minimum length of an event can lead to the situation that a number of data elements do not fulfill the conditions, but with another data element, they do (Examples 8 and 9). Therefore, we need to make two subsidiary definitions:

Definition 8. ***Loose conditions** are a set of conditions where one or more elements have been removed to gain intermediate results.*

Example 8. *Let us assume a user is not interested in events that have less than 30 cars per hour and end on a Sunday because that is "normal" and thus already a well-analyzed artifact. Instead, events that have less than 30 cars per hour and end on a Monday are important. Without using loose conditions, our procedure would not find an event of less than 30 cars per hour from Saturday to Monday, because for Saturday and Sunday, the "ends on a Monday" condition would not be fulfilled. By temporarily removing that condition, the event can be found.*

Definition 9. *An **event candidate** is a set of data elements that is sorted by the time reference together with a set of loose conditions it fulfills as well as that loose conditions' source conditions.*

Example 9. *As long as only the loose conditions are fulfilled for an event, it is not clear whether it really exists. Thus, it is an event candidate. For Example 8, various event candidates of consecutive days that have less than 30 cars per hour can come up. In the end, they have to be checked if they end on a Monday, thus fulfilling the full set of conditions. Those event candidates become events.*

Event Finding. The event finding is explained in Procedure 1. The basic idea is to go through the data elements and see if they fulfill the conditions for forming new events, or if they can be added to lists of open events that are kept at the same time. If a data element cannot be added to an open event, the event is "closed" and no further elements are added.

Differences to Related Work. Other approaches, like MuTIny, consider each data element a single event, even if that means that several consecutive events of the same type are formed.

Pattern Finding. The resulting events are also considered patterns of length 0. For longer patterns, the pattern finding in Procedure 2 can be performed iteratively k times to find patterns of length k. The basic idea for pattern finding is to go through all combinations of patterns from the last step with relations and events, and add them to the list of new patterns if the combination fulfills the relation.

Differences to Related Work. Other approaches consider less possible temporal relations between events. Some approaches only find patterns if events happen simultaneously, some consider only the order of events and ignore the actual time, some require an exact, user-defined, number of time steps between events for them to belong to a pattern. MuTIny can have several user-defined time intervals between events.

Before proceeding to the implementation, we discuss how to parameterize the procedure in the next section.

4. Interactive Visual Analysis

Like other pattern finding methods, MEMuRY relies on a number of parameterization issues: event configuration, pattern configuration, choosing important patterns, and exploration of results. However, there is no linear optimal order to perform these tasks as insights might immediately open up new questions for users.

4.1. The VA Process

As a better alternative to a linear approach, and based on the system proposed by Bertone et al. [5], we propose to arrange the MEMuRY procedure and its interactive visual interfaces according to one of the general definitions of the VA process, e.g., a more specialized variant for time-oriented data by Lammarsch et al. [35] (Figure 2):

Procedure 1: Event Finding: A number of conditions have to be given for each event class the user wants to define. Those are subsets of a collective set of conditions.

```
input  : dataSet, conditionsSet
output: events
eventCandidates, closedEventCandidates,
looseConditionsSet,
  events ← empty key,value-tables;
foreach conditions in conditionsSet do
    looseConditions ← conditions without those that would
    need events to end later or be longer than they actually are;
    looseConditionsSet: add (looseConditions,conditions);
end
foreach dataElement in dataSet do
    foreach (eventCandidate,looseConditions) in
    eventCandidates do
        combined ← eventCandidate ⋂ dataElement;
        if combined fulfills looseConditions then
            eventCandidate ← combined;
        else
            closedEventCandidates: add
              (eventCandidate,looseConditions);
            eventCandidates: remove
              (eventCandidate,looseConditions);
        end
    end
    foreach (looseConditions,conditions) in
    looseConditionsSet do
        if dataElement fulfills looseConditions then
            eventCandidates: add
              (dataElement,looseConditions);
        end
    end
end

foreach (eventCandidate,looseConditions) in
closedEventCandidates do
    if eventCandidate fulfills conditions of looseConditions
    from looseConditionsSet then
        events: add (eventCandidate,conditions);
    end
end
```

Procedure 2: Pattern Finding: Source patterns of length $k-1$ are transformed to patterns of length k (each consisting of $k+1$ events and k relations).

```
input  : sourcePatterns, events, relations, threshold
output: patterns
filteredPatterns ← sourcePatterns without patterns from
classes that are less frequent than threshold;
foreach filteredPattern in filteredPatterns do
    foreach relation in relations do
        foreach event in events do
            if relation (filteredPattern,event) is fulfilled then
                patterns: add (filteredPattern,relation,event);
            end
        end
    end
end
```

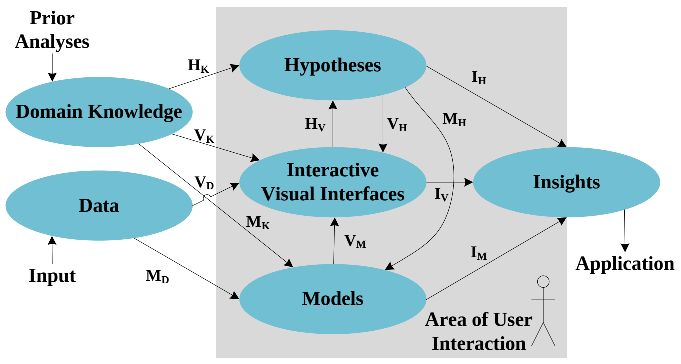


Figure 2: The VA process for time-oriented data by Lammarsch et al. [35], based on work by Keim et al. [36] and Bertini and Lalanne [37]: H_K: Take hypotheses from domain knowledge. V_K: Visualize domain knowledge directly. M_K: Take models from domain knowledge. V_D: Visualize data. M_D: Generate models from data. H_V: Build hypotheses based on visualizations. V_M: Visualize models. V_H: Visualize hypotheses. M_H: Validate hypotheses to form models. I_H: Gain insights from hypotheses. I_M: Gain insights from models. I_V: Gain insights from visualizations.

1. Configuring events can be seen as a way for users to formulate hypotheses. They apply their domain knowledge to do this and use interactive visual interfaces.
2. The procedure calculates event instances which can be seen as models. They are validated in the form that they actually exist in the data as defined. The event classes and instances can again be analyzed by interactive visualizations, and first insights can be drawn.
3. Configuring relations that are used to build patterns from the events are again hypotheses. Again, interactive visual interfaces are used.
4. Finding the patterns based on these relations is once again a calculation of models. These patterns can also be explored with interactive visualization and insights can be drawn from them.
5. Choosing important patterns for analysis with further iterations can be done by filtering based on support (models) or forming hypotheses about which patterns are important.

As a result, the VA system can iterate over the center part of the process, taking the data as input as well as an increasing amount of domain knowledge, and generating an increasing number of insights.

4.2. *Tasks and Requirements*

Based on the process, we can formulate the tasks in detail:

1. Parameterizing a method for classifying data elements, which might also be multivariate, into event classes, i.e., categorical data. These parameters are used by Procedure 1. As shown by Bertone et al. [5], temporal aspects can be included in the event definitions, and for those, like for other important data types like space, special interfaces are possible.

2.a Parameterizing the conditions for events forming a pattern. These parameters are used by Procedure 2. Interfaces for this task need to respect the various possibilities as presented by Allen [34].
The interview we performed with a domain expert (Subsection 6.4) showed that temporal relations are very important when looking for the interesting patterns. For this, relations that are not known should be used. For example, low traffic follows high traffic might be known to domain experts, but, using their domain knowledge, they might wonder what happens between events that are one day apart.

2.b Choosing the most important patterns between iterations in order to perform the next iteration with a filtered set. This is done as part of Procedure 2.
The basic idea for building longer patterns from shorter ones is that a pattern is only interesting if it is based on a shorter interesting pattern. Thus, looking only at an end result of patterns that has only been filtered automatically in between can occlude interesting pattern. Ideally, users find shorter interesting patterns first, and continue with patterns that start with the found patterns [5].

3. Showing pattern classes (as done by Bertone et al. [5]) as well as pattern instances over time.
For a complete analysis, we expect that users will need both. This can be achieved using linked views.
4. Re-parameterization needs to be supported through direct manipulation [38]: interactive, based on patterns already found by previous iterations, and immediate. So for example, if a user detects an important pattern in the results, he/she might want to directly interact with it, slightly change the parameterization of one of its events, and immediately see how this affects the appearance of the pattern at the iteration step he/she is currently exploring.

4.3. *Proposed VA System*

We propose a VA system based on the same paradigms as the one by Bertone et al. [5]. That system has strong features regarding re-parameterization and the iterative process which we intend to re-implement. However, more direct manipulation is needed. We have described this in detail for event configuration [19] (Section 2). For parameterizing relation, we have to devise an interactive visual interface yet.

For exploration of results, we will now propose a visual interface in detail that we also implemented and tested (Section 6): As explained above, it is possible to visually compare pattern classes (e.g., whether e_0 and e_1 are more often related by r_0 or by r_1), or patterns over time (e.g., $e_0 r_0 e_1$ appears frequently during this interval, but there is also one occurrence of $e_0 r_1 e_1$ that seems important). The visualization of different types of patterns has already been focused on by Bertone et al. [5]. Another possibility is to analyze the instances of patterns. To spark research on visualizations that have seen less attention in related work, we focused on showing pattern instances over time. We developed a combined visualization that can show the same patterns as a river view similar to ThemeRiver [39] (Figure 3), a number of timelines [2] (Figure 4), or an arc view similar to the ArcDiagram [30] (Figure 5). The views can be selected manually, but for novice users, we implemented an automated change of view as the user zooms in/out on the time axis. This feature was inspired by a similar one in the Midgaard system called SemanticTimeZoom [40, 41]. We consider our current prototype for exploring results a start for future work, but not a main contribution of this publication.

The interface enables zooming and panning for all visualizations and keeps the time range when switching views. We also provide detail on demand, showing pattern labels of patterns the mouse cursor hovers. The arcs of the arc view are transparent by default, but a hovered arc becomes opaque, so it is easier to distinguish. We named our new combined visualization Semantic Alternating Possibilities for Pattern and Event Research Looking Over Time (SAPPERLOT[3]).

Our prototype supports no direct re-parameterization yet, and the iterations have to be done by manually running the procedure, so supporting the full process in one system is our main task for future work (Section 7).

5. Implementation and Runtime Verification

The MEMuRY procedure was implemented based on TimeBench [42], a software library for implementing VA applications dealing with time-oriented data. TimeBench is based on prefuse [43] and uses similar software design patterns, but it provides more support for analytical methods as well as a data structure that is well-suited for the procedure.

The relations are implemented as temporal predicates [42] that are capable of comparing two intervals – one is part of the events found by the procedure, the other one is part of the possible relations provided by users. In addition to predicates directly described by Allen [34], there is a predicate that shifts intervals in time in order to allow enhanced relations as described by Lammarsch et al. [44].

To test the runtime behavior, we have made a benchmark for a number of parameters. In addition, we have benchmarked a reimplementation of the MuTIny approach [3] that shares most of its code base with our new implementation, making the approaches comparable. The parameters we varied were:

- The length of the dataset (64–512, raising exponentially)
- The number of different data values existing which imposes the maximum number of events possible (1–4)
- The number of different events actually parameterized, so some data values would result in no events existing at that point in time (1–3)
- The number of relations parameterized (1–3)
- The number of pattern finding steps performed after initial event finding (1–4), resulting in patterns that have those lengths

For each combination of parameters, 8 tests were run.

The number of patterns is the most important factor for the runtime behavior. We used visualizations and linear regression and found that there are linear relations with different slopes depending on the dataset length, but for a constant dataset length, the runtime is linear based on the number of patterns (Figure 6 and Table 1). Doubling the dataset length roughly doubles the

[3]In German, “sapperlot” is an exclamation sometimes used when realizing an unexpected fact.

Table 1: Slope of the linear regression for runtime (number of patterns) based on several dataset lengths

	MuTIny	MEMuRY
Elements	Time (ms/pattern)	Time (ms/pattern)
64	0.6373	0.6049
128	1.267	1.151
256	2.556	2.355
512	5.049	4.544

runtime duration per pattern found. These results show that the dataset length is the dominating factor in T.

The behavior of MuTIny was only slightly worse regarding time per pattern, but we noticed that there were much more patterns in total.

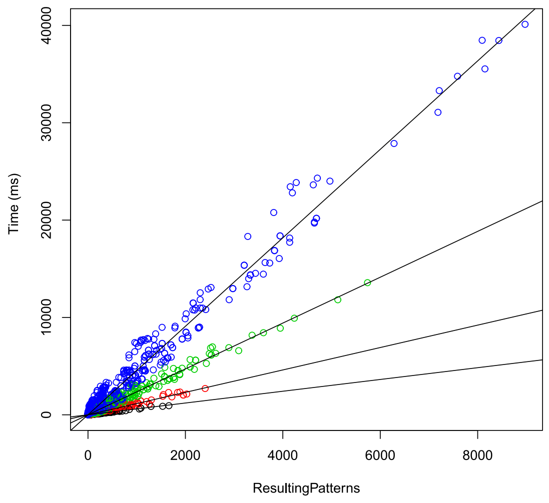


Figure 6: The runtime behavior of our MEMuRY implementation as a result of the number of patterns found by the procedure. The values are colored black for 64 data elements, red for 128 data elements, green for 256 data elements, and blue for 512 data elements. For each class, a linear regression model is plotted.

Second, we wanted to know what affects the number of patterns found. In some cases, few patterns were found, in other cases, many patterns were found. We explored several scatterplots of the number of events searched, the number of relations searched, and the number of steps performed vs. the number of potential patterns. Increasing those parameters increased the highest number of potential patterns slightly, but only as spreadings. Adding them (Figure 7) provides a more distinct spreading, but for each parameterization, a low number of resulting patterns still seems to be possible. Figure 7 compares the increase in spreading for MEMuRY to the one for MuTIny. While for MEMuRY, the random order of values can lead to varying numbers of events and thus to more blur in the number of patterns, there are parameterizations that lead to exactly one number of patterns in MuTIny, so there are less different points in the visualization. The values used to build the x-axis were put together that way based on our visual analysis in order to provide summary visualizations for both approaches. In total, the number of patterns resulting from MuTIny is almost five times as high as for MEMuRY. We will examine this further based on a real-world dataset in Section 6 to see what information is in

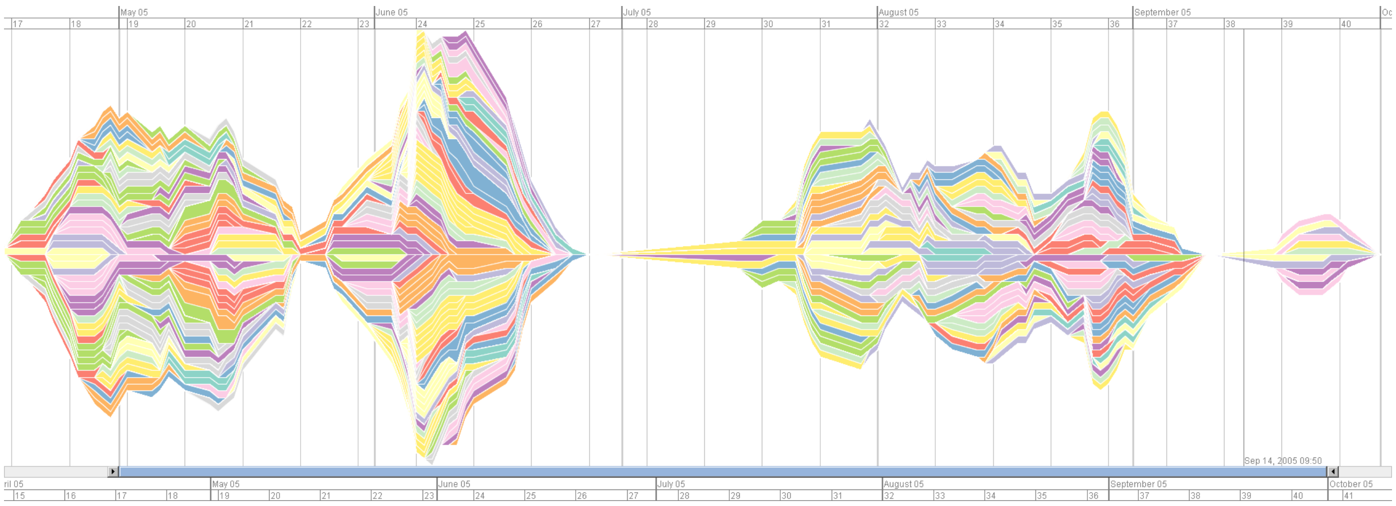


Figure 3: A river view based on ThemeRiver[39]: time is on the horizontal axis, each band stands for a certain class of patterns (the composition of a class is given as detail on demand). The width of a band shows the number of instances simultaneously existing at a given time. If there are no more than 12 classes, each class has a separate color. As there are too many classes in this visualization, similar patterns are given the same color. This visualization shows patterns based on a raster of days; there is no support threshold. We could get some information from this visualization, but needed to impose a threshold for further analysis.

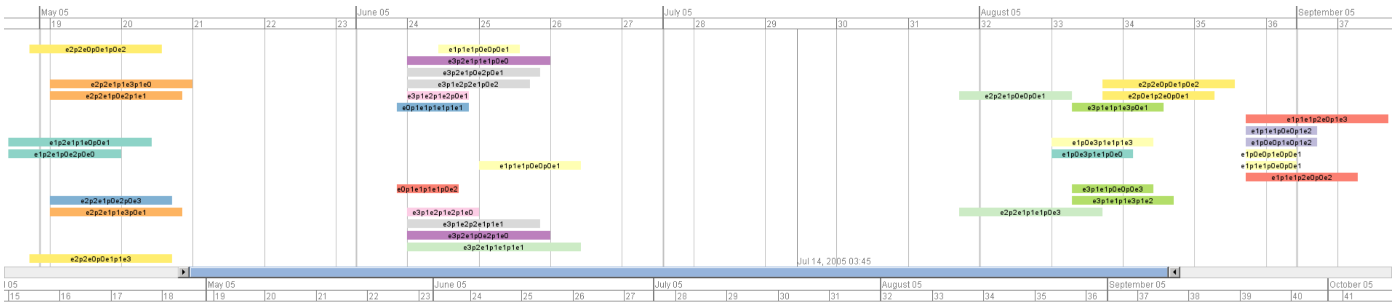


Figure 4: Timeline view based on Timelines [2] showing pattern instances. Time is on the horizontal axis, the vertical position is used to prevent overlap between bands. The patterns are based on a raster of days. If there are more pattern classes then colors available, patterns starting the same and then diverging are given the same color. As only patterns more frequent than 0.005 of the total number are shown, the view is zoomed in a bit so that only the time with remaining patterns is covered.

the most prominent patterns for each approach.

Test Conditions and Limitations. Benchmarking pattern finding approaches based on the Apriori algorithm is difficult, because their behavior heavily depends on the dataset and the parameterization. The parameters are usually fitted to the dataset, so iterating over fixed parameterizations does not make sense for a real-world dataset. Thus, we had to use a synthetic random dataset. For real world data, information in the form of time-related structures in a dataset result in some patterns being more frequent than others and allow filtering for a given support threshold. In MEMuRY, artifacts in the data can also reduce the number of patterns by having fewer events. For completely stochastic data, there is no sensible way to reduce the number of patterns between the calculation of one pattern length and the next pattern length, so we expect a worst case scenario. We omitted the filtering between steps, because depending on support, we would get either no patterns, all patterns, or a totally random number of patterns (in case of k patterns and threshold near $1/k$).

In addition, modern computer systems complicate benchmarks. Newer processors distribute code freely to their calculation units as optimally as they can. Operating systems preemptively schedule processing power and memory to processes. In our case, the Java Virtual Machine imposes another layer of virtualization, whereas the foremost problem is that the garbage collector cannot be controlled directly. To minimize these effects, the tests were run on a Core i7-2600K processor running at 3.4 GHz. The Java Virtual Machine and the benchmarking code together took 13% of processing power distributed over 4 virtual cores and did not seem to be able to take more, so other code running on the system should not be likely to have affected it. We took measures to prevent the processor from applying its TurboBoost feature which would have changed the processor frequency.

6. Visual Exploration of Results and Usage Scenario

To evaluate MEMuRY and SAPPERLOT, we followed a usage scenario,[4] looking for patterns in the Dodgers Loop Sensor

[4] A usage scenario is a form of evaluation where real data is analyzed by the developers of an analysis tool [45].

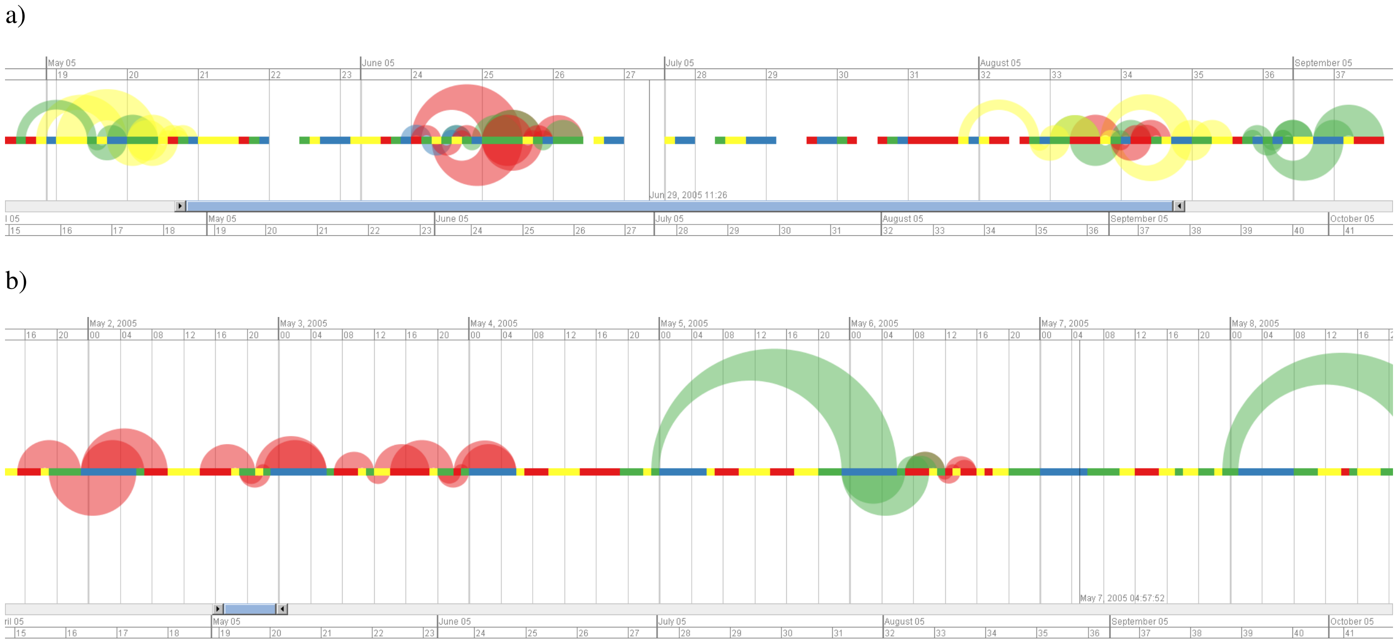


Figure 5: Arc views based on the ArcDiagram [30] showing pattern instances. The horizontal axis represents time, the vertical axis is used to spread arcs. The patterns visualized here are based on non-overlapping events that fall into one of four classes (blue/very low, green/low, yellow/high, and red/very high). The opaque bars in the middle show the temporal sequence of events. Patterns are of length 3. They are represented by transparent arcs that are colored according to their first event. Upper arcs represent the temporal relation between the first and second event and between the third and forth event, lower arcs represent the temporal relation between the second and third event. a) The patterns are based on a raster of days with a support threshold of 0.005. The view is zoomed in so that all patterns found are shown. The information taken from this view is explained in Section 6.2.1. b) The patterns are based on a raster of hours with a support threshold of 0.005. The view is zoomed in so that an interesting area can be seen. Two types of patterns both in red are shown. They seem to stem from baseball games. The green pattern might be an effect resulting from Saturday night life. These findings are explained in more detail in Section 6.2.2.

dataset [46] which is publicly available. This dataset shows traffic on a highway ramp near a baseball stadium. The data is expected to contain artifacts from calendar effects, like weekends and rush-hours, but also from baseball games held. Thus, a ground truth is available.

The dataset shows the number of cars in five minute intervals. The total length is about half a year. Based on these factors, we decided to use limited rasterization as pre-processing. To find different kinds of patterns, we decided to analyze at the three aggregation levels of (1) five minute intervals, (2) hours, and (3) days. The dataset contains missing values; we decided to count full hours or days as missing values if one of its source values is missing. The analyses were performed using MEMuRY, but also with MuTIny as baseline for comparison.

6.1. Parameterization

For event parameterization, we only had to deal with one data column, the number of cars. Therefore, we could simply distribute the number of cars into four quantiles of equal probability. This choice was motivated by research by Lin et al. [11]. Looking at a histogram, it also appeared sensible to us. Domain experts might find better parameterizations using the histograms, though. For dealing with more complex datasets, using the simplification interface we described in an earlier publication would be a possible way [19]. The resulting classes are given in Table 2.

For each raster level, we chose three different kinds of relations as shown in Table 3. The relations are all describing fixed

Table 2: Event parameterization: the classifications of the number of cars on the from the Dodgers Loop Sensor dataset [46] that were used by us. Values are not always starting with zeros as our value range mirrors the actual minimum in the dataset.

Raster	Very Low (e_0)	Low (e_1)	High (e_2)	Very High (e_2)
5-Minute	0–9	10–22	23–31	32–90
1-Hour	0–105	106–277	278–364	365–662
1-Day	3518–5450	5451–6076	6077–6566	6567–7661

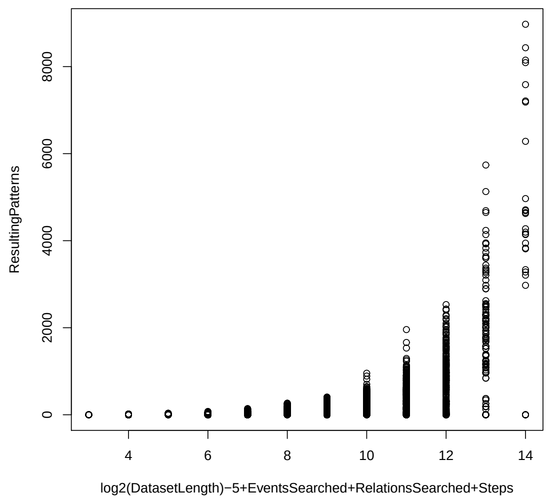
Table 3: The various relations we used for our usage scenario dependent on the raster of the data. The relations are given between a pattern and a possible new event that could be added.

Raster	Very Short (p_0)	Short (p_1)	Long (p_2)
5-Minute	meets	ends 5 minutes before	starts 1 hour before
1-Hour	meets	ends 1 hour before	starts 1 day before
1-Day	meets	ends 1 day before	starts 1 week before

intervals, no interval ranges, but we have variable intervals regarding the begin or end of events as they are of variable length themselves.

We performed 3 steps resulting in patterns of length 3 with 4 events and 3 relations in between. That is a conservative increase from the patterns of length 2 presented by Bertone et al. [5].

We started with the day raster and a threshold of 0 for support. While the calculation of patterns was still possible, there were simply too many for visual exploration over time. The river view (Figure 3) shows them in sum, but more detail is



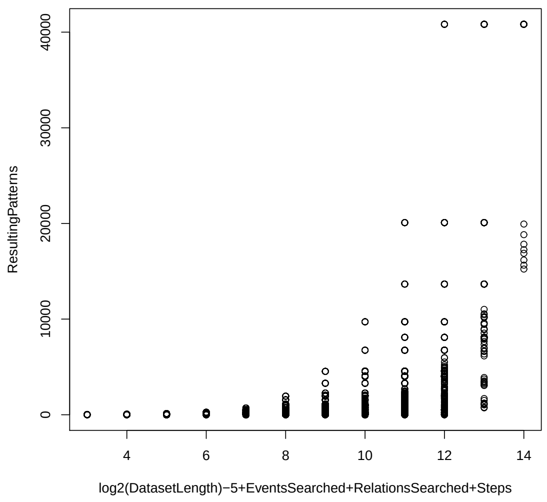


Figure 7: The influence of several parameters on the number of patterns found by MEMuRY (left) compared to the influence of the same parameters on numbers found by MuTIny (right). The vertical scales are different, a disadvantage of this figure we decided to accept in order to show more details in the right part of the left visualization.

not possible. Thus, we increased the threshold only slightly to 0.005, resulting in much fewer patterns. This seemed to happen for even such a low threshold because many patterns occurred only once.

6.2. Exploration of Results

Using MEMuRY on a day raster, we found 4 intervals that are about 4 days long and contain more frequent patterns. Each of those intervals has 1–2 days that are the basis for most of the patterns and might be important – further analysis would be necessary. With the same parameters, MuTIny also found patterns, but the information gained from them, like the business week, was mostly composed of patterns that appear in most datasets based on social effects. These patterns are already known according to experts for time-oriented data we talked to, for example in our expert interview (Subsection 6.4).

Using MEMuRY on an hour raster, we found various patterns. Many of those belong to 2 slightly different classes which we could identify as indicating baseball games at 2 different times of day that can also be found in the ground truth. We were aware that there could be patterns based on those games, but we did not know in advance were to search and what they would look like. However, they visually stood out. Other patterns belong to a third class that we do not fully understand yet, but suspect to be related to “Saturday night life”.

Using MEMuRY on the 5-minute-raster, we found many patterns that seem to indicate times of high traffic. However, at this level it was hard to gain more information.

6.2.1. Day Raster in Detail

In the day raster analysis, we found that the patterns remaining after filtering are roughly where the “bulks” of the first river view were. The timelines (Figure 4) show in their labels that several patterns start with the same event, but other patterns starting at different events are also temporally close. There seem to be 3–4 clusters (groups that are close to each other in time) of patterns.

The arc view (Figure 5.a) reveals that a reason for the gaps between the pattern clusters could be the missing values. What is hard to see in the timelines view, but prominent in the arc view, is that for the first cluster, patterns start with medium to high traffic (green and yellow arcs), while for the second cluster, many patterns are sparked by a single Monday with very high traffic (several red arcs that all are connected to one small and one large red arc starting on the same day). There indeed seem to be two more clusters, one with rather high traffic (mostly yellow arcs), and another one based on two days with medium traffic (green arcs to the right). Few patterns start with low traffic, but events with low traffic are often more than one day long, are part of several patterns.

Most events of all clusters start with a one-day-event, but many follow-up events are longer. An effect of weekends in the data can be seen, but they do not dominate the patterns. For more in-depth analysis and interpretation of these results, more domain knowledge would be necessary.

Comparison to MuTIny. We compared those results to MuTIny results with exactly the same parameterization, also visualized using SAPPERLOT. The first and last cluster found by MuTIny are based on the same patterns. In the middle, MuTIny found two small clusters different from the one found by MEMuRY. Even in the areas where both approaches found patterns, they are built differently. Almost all patterns found by MuTIny are related to weekends and are of limited interest, as the importance of weekends is well known. Perhaps the only interesting information is that weekends with low traffic after weekends with high traffic seem rather frequent. This is gained only from the first part of patterns. The rest of those patterns show that for all kinds of weekends, typical traffic for Mondays and Tuesdays follows. This information is more or less trivial. The new clusters we found show two weekends and seem to indicate that weekends often have low traffic, and that the traffic on Monday is more related to the traffic on Friday than on Saturday and Sunday. This also is no new information. In sum, MuTIny found more patterns altogether, but when filtering for support, less patterns remained. We estimate that each pattern contains considerably less information. The whole information is still present in the patterns from MuTIny, but it is very difficult and time-consuming to gain this information as it is strongly cluttered.

6.2.2. Hour Raster in Detail

In the hour raster, MEMuRY found many more patterns, which might be due to the longer dataset, but they were also distributed more evenly over time. All patterns start with low or very high traffic, so patterns based on very low or high traffic seemed to be filtered out early in the process.

To see more details, we can zoom and pan (Figure 5.b). A very frequent pattern that also had a characteristic shape is very high traffic starting between 12pm and 4pm, followed by a gradual decline to very low traffic from around 11pm to around 6am. The time seems odd for rush hour, and actually we found out that this pattern usually happens on days when Dodgers games are in the afternoon. Interestingly, MEMuRY found this pattern and could classify them into the same class even if the actual times vary slightly. What resulted in a different, but also frequent pattern that we found afterwards are slight fluctuations in the decline, with the traffic sometimes going back up at around 10pm. This pattern seems to appear when Dodgers games are in the evening. Thus, we can conclude that MEMuRY can detect Dodgers games. While the first kind of pattern usually appears alone when there are actually games in the afternoon, the second kind of pattern often appears during the same time as an instance of the first pattern and two other patterns that are different combinations of the participating events. A better parameterization might increase the performance in order to get only single patterns, if this is the goal of the user.

Another pattern that appeared frequently in the hour raster was that one week the traffic was low at 11pm, and the next day it was already very low at the same time, followed by traffic quickly increasing to very high in the morning. This pattern appeared usually when there were no Dodgers games and often on Sundays, but also on other days of week. Perhaps on Saturdays, more people (low traffic) are on the road at night than on Sundays (very low traffic), and that effect is so regular that this pattern has high support.

Comparison to MuTIny. For using MuTIny on the hour raster, we had to drastically reduce the support threshold, as there would have been more patterns than our test system could handle in reasonable time. The result was mostly the same pattern cluster repeated over and over again, which was shown nicely by the river view. The river view also showed that there were some deviations, but they were hard to interpret, as they were cluttered by the large number of total patterns in the other views. The timelines showed that the most prominent feature of the typical pattern was very low traffic followed by more very low traffic. The arc view finally showed that this was because of repeating night hours (also well-known information) which were the most prominent pattern. After some zooming and panning, most of the deviating patterns seemed to be nights were the traffic was already low instead of very low early in the morning and other slight deviations from the main pattern. However, we could not find out more details of the distribution of those nights. We could not find the Dodgers games in the MuTIny results.

6.3. Lessons Learned

The river view is well suited to see how many patterns of certain types exist at certain times. Further, it is useful exactly when the more detailed visualizations are not useful. Thus, it should be used over the course of an iterative analysis, and ideally allow for manual selection of support or improving selection further using dynamic queries. In addition, only performing the procedure for certain time ranges and dynamically extending them helped much as we did it manually, so integrating this concept into a full system should greatly increase the user experience for larger datasets.

It was difficult to compare pattern classes in the timelines. We provided 12 distinguishable colors from ColorBrewer [47], but they often were not enough for the pattern classes. What could help is an advanced distance measure for pattern classes and a dynamic color scheme that makes use of this measure. Alternatively, the colors from the event bar in the arc view could be used for multi-colored patterns. These ideas could, for example, have improved the time needed to understand that the recurring patterns in the hour raster are indeed Dodgers games.

The arc view can be very effective when analyzed thoroughly, but it takes much time to interpret and starts with a puzzling view. Zooming and panning helps a lot, but further filtering methods and new ideas regarding color of the arcs are needed to make it more usable.

Altogether, the data raster seems to be important. We expected MEMuRY to strongly reduce this problem by automatically combining data elements to longer events, but in reality, there are limits. For example, short-time but high-value outliers, like they can appear in traffic data, can disrupt events. Using MEMuRY on the 5-minute-raster showed that for most days, there are short patterns in the morning or at noon which might stem from rush hours. At the time of Dodgers games, there are often patterns, but they are irregular and therefore hard to match. Checking slightly larger time rasters helped a great deal to circumvent that problem – using a raster of hours, MEMuRY already performed much better than MuTIny. The interactive visual interface needs to be extended to quickly skim various rasters.

6.4. Qualitative Feedback from a Domain Expert

To get qualitative feedback, we performed an interview with a domain expert. She is working with time-oriented data, together with a company that mostly deals in shift-plan management. In the past, other domain experts from that company have pointed out the shortcomings of the MuTIny approach [3].

The domain expert confirmed that fewer patterns are good because they are easier to handle. She also agreed that no information is lost because the lengths of events are saved.

The domain expert stressed that the weekend pattern was still there in our examples, even though there were fewer different patterns describing it. As a result, she explained that no new information is gained if the relations between events describe known behavior: A relation of “follows directly” would still find “weekend follows business week”, “business week follows weekend”; “starts one week later” would still find “weekend starts one week after another weekend starts”; and so on. However, by trying out new relations, like “one day in between”, patterns would come up that are new. Those patterns are much easier to find with MEMuRY, because they are not cluttered between patterns from, e.g., two weekdays separated by a third one, like Monday, Wednesday. By this assessment, we are assured in our future work goal to fully integrate the procedure in a full VA system instead of just exploring the results.

A last idea brought up by the domain expert was to define events by deduction of the cyclic components of the values. To do this, we would have to consolidate our proposal for event definition [19] with a statistic approach on VA for time-series analyses that we are pursuing in parallel to this work [48].

7. Conclusion and Future Work

As our main contribution, we presented a novel pattern finding procedure for time-oriented data that follows and improves the I-Apriori [16] and MuTIny [3] approaches. We described and prototypically implemented a procedure that finds patterns as well as an interactive visualization for the results. A usage scenario showed that the procedure can indeed find valuable patterns that could not be found by a similar state-of-the-art-approach. We also discussed interactive, visual, iterative parameterization and exploration of the results using VA. Based on our prototype, we made benchmarks on the worst case scenario of random data.

Our definition of events results in different patterns than those from other definitions that base events on rastered datasets and do not allow for variable event lengths. Considering our goal to simplify temporal structures, this is intended. In Figure 1.a and 1.b, for example, our procedure does model business weeks and weekends (which is no new information), but instead of using four classes of patterns of length 1 per interval, for a total of 8 classes, it needs only 2 which have much less instances and contain the same information. Moreover, if something unexpected happens, like in Figure 1.c, different patterns appear, while for events that are locked to the raster length, only the frequency of the same event classes would vary.

The benchmark of stochastic data (Section 5) shows that our procedure indeed produces much less patterns then MuTIny, with comparable performance per pattern. The usage scenario in Section 6 showed that the procedure can find very interesting patterns that stem from social time, but the actual processes do not need to be known in advance (like the baseball games we later verified our results against). Moreover, the procedure provides more compact information based on known structures, like the weekends. Thus, we consider our procedure and visual interface to be successful in advancing TDM.

To help users in better applying the procedure to their practical problems, we need to develop an interactive visual system for iteratively performing knowledge discovery based on MEMuRY. In Section 4, we described the features that such a tool should have. In the end of Section 6, we summarized our insights from our existing system for result exploration, and how those should be integrated in the next step, which we then also intend to test with a larger population of users.

We feel confident that MEMuRY and its possible advancements will be a valuable addition to classical KDD approaches. Furthermore, we consider the VA approach to be the most effective one for interactively and visually parameterizing the procedure and exploring the results. Thus, developing a prototype VA system for the procedure is the key part of our future work.

Acknowledgments. This work was supported by the Austrian Science Fund as part of the HypoVis project (Project Number #P22883). We would like to thank the editors and the reviewers for their detailed recommendations and suggestions. We also would like to thank Theresia Gschwandtner for providing her expertise in time-series analysis from a business-oriented point of view.

References

[1] Laxman S, Sastry P. A survey of temporal data mining. Sadhana 2006;31(2):173–98. doi:10.1007/BF02719780.

[2] Aigner W, Miksch S, Schumann H, Tominski C. Visualization of Time-Oriented Data. Springer; 2011.

[3] Bertone A, Lammarsch T, Turic T, Aigner W, Miksch S, Gaertner J. MuTIny: a multi-time interval pattern discovery approach to preserve the temporal information in between. In: Proceedings of the IADIS European Conference on Data Mining, ECDM. 2010, p. 101–6.

[4] Wong BW, Xu K, Holzinger A. Interactive visualization for information analysis in medical diagnosis. In: Information Quality in e-Health. LNCS 7058; Springer; 2011, p. 109–20.

[5] Bertone A, Lammarsch T, Turic T, Aigner W, Miksch S. Does Jason Bourne need Visual Analytics to catch the Jackal? In: Kohlhammer J, Keim D, editors. Proceedings of the First International Symposium on Visual Analytics Science and Technology held in Europe, EuroVAST. Eurographics; 2010, p. 61–7. doi:10.2312/PE/EuroVAST/EuroVAST10/061-067.

[6] Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases. ACM SIGMOD Record 1993;22(2):207–16. doi:10.1145/170036.170072.

[7] Gschwandtner T, Gärtner J, Aigner W, Miksch S. A taxonomy of dirty time-oriented data. In: Quirchmayr G, Basl J, You I, Xu L, Weippl E, editors. Multidisciplinary Research and Practice for Information Systems, Proceedings CD-ARES 2012. LNCS 7465; Heidelberg: Springer; 2012, p. 58–72. doi:10.1007/978-3-642-32498-7_5.

[8] Keogh E, Chakrabarti K, Pazzani M, Mehrotra S. Locally adaptive dimensionality reduction for indexing large time series databases. ACM SIGMOD Record 2001;30(2):151–62. doi:10.1145/376284.375680.

[9] Keogh E, Chakrabarti K, Pazzani M, Mehrotra S. Dimensionality reduction for fast similarity search in large time series databases. Knowledge and Information Systems 2001;3(3):263–86. doi:10.1007/PL00011669.

[10] Lin J, Keogh E, Lonardi S, Chiu B. A symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery. 2003, p. 2–11. doi:10.1145/882082.882086.

[11] Lin J, Keogh E, Wei L, Lonardi S. Experiencing SAX: a novel symbolic representation of time series. Data Mining and Knowledge Discovery 2007;15(2):107–44. doi:10.1007/s10618-007-0064-z.

[12] Lammarsch T, Aigner W, Bertone A, Miksch S, Rind A. Mind the time: Unleashing the temporal aspects in pattern discovery. In: Pohl

M, Schuman H, editors. Fourth International Eurovis Workshop on Visual Analytics held in Europe, EuroVA. 2013, p. 31–5. doi:10.2312/PE.EuroVAST.EuroVA13.031-035.

[13] Srikant R, Agrawal R. Mining sequential patterns: Generalizations and performance improvements. In: Apers P, Bouzeghoub M, Gardarin G, editors. Advances in Database Technology – Proceedings of the 5th International Conference on Extending Database Technology, EDBT. LNCS 1057; Springer; 1996, p. 1–17. doi:10.1007/BFb0014140.

[14] Mannila H, Toivonen H, Inkeri Verkamo A. Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery 1997;1(3):259–89. doi:10.1023/A:1009748302351.

[15] Magnusson M. Discovering hidden time patterns in behavior: T-patterns and their detection. Behavior Research Methods 2000;32(1):93–110. doi:10.3758/BF03200792.

[16] Chen Y, Chiang M, Ko M. Discovering time-interval sequential patterns in sequence databases. Expert Systems with Applications 2003;25(3):343–54. doi:10.1016/S0957-4174(03)00075-7.

[17] Hu Y, Huang T, Yang H, Chen Y. On mining multi-time-interval sequential patterns. Data & Knowledge Engineering 2009;68(10):1112–27. doi:10.1016/j.datak.2009.05.003.

[18] Lammarsch T. Facets of Time—Making the Most of Time's Structure in Interactive Visualization. Ph.D. thesis; Vienna University of Technology; 2010. URL: http://publik.tuwien.ac.at/files/PubDat_217966.pdf; supervisors: Silvia Miksch (Vienna University of Technology), Daniel Keim (University of Konstanz).

[19] Lammarsch T, Aigner W, Bertone A, Bögl M, Gschwandtner T, Miksch S, et al. Interactive visual transformation for symbolic representation of time-oriented data. In: Holzinger A, Ziefle M, Glavinić V, editors. Human-Computer Interaction and Knowledge Discovery in Complex, Unstructured, Big Data. LNCS 7947; Springer Berlin Heidelberg; 2013, p. 400–19. doi:10.1007/978-3-642-39146-0_37.

[20] Simonic KM, Holzinger A, Bloice M, Hermann J. Optimizing long-term treatment of rheumatoid arthritis with systematic documentation. In: Proceedings of the 5th International Conference on Pervasive Computing Technologies for Healthcare, PervasiveHealth. IEEE; 2011, p. 550–4.

[21] Keim DA, Kohlhammer J, Ellis G, Mansmann F, editors. Mastering The Information Age – Solving Problems with Visual Analytics. Eurographics Association, Goslar; 2010.

[22] Holzinger A. On knowledge discovery and interactive intelligent visualization of biomedical data – challenges in human-computer interaction & biomedical informatics. In: Proceedings of the International Conference on Data Technologies and Application (DATA 2012). SciTec Press; 2012, p. 5–16.

[23] Tominski C. Event-based concepts for user-driven visualization. Information Visualization 2011;10(1):65–81. doi:10.1057/ivs.2009.32.

[24] Klimov D, Shahar Y, Taieb-Maimon M. Intelligent selection and retrieval of multiple time-oriented records. Journal of Intelligent Information Systems 2010;35(2):261–300. doi:10.1007/s10844-009-0100-0.

[25] Klimov D, Shahar Y, Taieb-Maimon M. Intelligent visualization and exploration of time-oriented data of multiple patients. Artificial Intelligence in Medicine 2010;49(1):11–31. doi:10.1016/j.artmed.2010.02.001.

[26] Wang T, Plaisant C, Shneiderman B, Spring N, Roseman D, Marchand G, et al. Temporal summaries: Supporting temporal categorical searching, aggregation and comparison. IEEE Transactions on Visualization and Computer Graphics 2009;15(6):1049–56. doi:10.1109/TVCG.2009.187.

[27] Monroe M, Lan R, Morales del Olmo J, Shneiderman B, Plaisant C, Millstein J. The challenges of specifying intervals and absences in temporal queries: A graphical language approach. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '13; New York, NY, USA: ACM; 2013, p. 2349–58. doi:10.1145/2470654.2481325.

[28] Monroe M, Lan R, Lee H, Plaisant C, Shneiderman B. Temporal event sequence simplification. IEEE Transactions on Visualization and Computer Graphics 2013;19(12). Forthcoming.

[29] Vrotsou K, Johansson J, Cooper M. Activitree: interactive visual exploration of sequences in event-based data using graph similarity. IEEE Transactions on Visualization and Computer Graphics 2009;15:945–52. doi:10.1109/TVCG.2009.117.

[30] Wattenberg M. Arc diagrams: Visualizing structure in strings. In: Proceedings of the IEEE Symposium on Information Visualization, InfoVis. 2002, p. 110–6. doi:10.1109/INFVIS.2002.1173155.

[31] Havre S, Hetzler E, Whitney P, Nowell L. ThemeRiver: Visualizing thematic changes in large document collections. IEEE Transactions on Visualization and Computer Graphics 2002;8(1):9–20. doi:10.1109/2945.981848.

[32] Kim J. Events as property exemplifications. In: Brand M, Walton D, editors. Action Theory, Proceedings of the Winnipeg Conference on Human Action. D. Reidel Publishing; 1976, p. 159–77.

[33] Andrienko N, Andrienko G. Exploratory analysis of spatial and temporal data: a systematic approach. Springer Verlag; 2006.

[34] Allen J. Maintaining knowledge about temporal intervals. Communications of the ACM 1983;26(11):832–43. doi:10.1145/182.358434.

[35] Lammarsch T, Aigner W, Bertone A, Miksch S, Rind A. Towards a concept how the structure of time can support the visual analytics process. In: Miksch S, Santucci G, editors. Proceedings of the Second International Workshop on Visual Analytics held in Europe, EuroVA. Eurographics Publications; 2011, p. 9–12. doi:10.2312/PE/EuroVAST/EuroVA11/009-012.

[36] Keim DA, Mansmann F, Schneidewind J, Thomas J, Ziegler H. Visual analytics: Scope and challenges. In: Simoff SJ, Böhlen MH, Mazeika A, editors. Visual Data Mining. LNCS 4404; Berlin: Springer; 2008, p. 76–90. doi:10.1007/978-3-540-71080-6_6.

[37] Bertini E, Lalanne D. Surveying the complementary role of automatic data analysis and visualization in knowledge discovery. In: Proceedings of the ACM SIGKDD Workshop on Visual Analytics and Knowledge Discovery: integrating Automated Analysis with interactive Exploration, VAKD. ACM; 2009, p. 12–20.

[38] Hutchins EL, Hollan JD, Norman DA. Direct manipulation interfaces. Human-Computer Interaction 1985;1(4):311–38. doi:10.1207/s15327051hci0104_2.

[39] Havre S, Hetzler B, Nowell L. Themeriver: Visualizing theme changes over time. In: Proceedings of the IEEE Symposium on Information Visualization, InfoVis. IEEE; 2000, p. 115–23.

[40] Bade R, Schlechtweg S, Miksch S. Connecting time-oriented data and information to a coherent interactive visualization. In: Dykstra-Erickson E, Tscheligi M, editors. Proceedings of the SIGCHI conference on Human factors in computing systems. Vienna, Austria: ACM; 2004, p. 105–12. doi:10.1145/985692.985706.

[41] Aigner W, Rind A, Hoffmann S. Comparative evaluation of an interactive time-series visualization that combines quantitative data with qualitative abstractions. Computer Graphics Forum 2012;31(3):995–1004. doi:10.1111/j.1467-8659.2012.03092.x.

[42] Rind A, Lammarsch T, Aigner W, Alsallakh B, Miksch S. TimeBench: A data model and software library for visual analytics of time-oriented data. IEEE Transactions on Visualization and Computer Graphics 2013;19(12). Forthcoming.

[43] Heer J, Card S, Landay J. Prefuse: a toolkit for interactive information visualization. In: Proceedings of the SIGCHI conference on Human factors in computing systems. ACM; 2005, p. 421–30. doi:10.1145/1054972.1055031.

[44] Lammarsch T, Rind A, Aigner W, Miksch S. Developing an extended task framework for exploratory data analysis along the structure of time. In: Matkovic K, Santucci G, editors. Proceedings of the EuroVis Workshop on Visual Analytics in Vienna, Austria, EuroVA. Eurographics Publications; 2012, p. 31–5. doi:10.2312/PE/EuroVAST/EuroVA12/031-035.

[45] Sedlmair M, Meyer M, Munzner T. Design study methodology: Reflections from the trenches and the stacks. IEEE Transactions on Visualization and Computer Graphics 2012;18(12):2431–40. doi:10.1109/TVCG.2012.213.

[46] Hutchins J. Dodgers loop sensor data set. UCI Machine Learning Repository; 2006. URL: http://archive.ics.uci.edu/ml/datasets/Dodgers+Loop+Sensor; accessed on March 2nd, 2013.

[47] Harrower M, Brewer CA. ColorBrewer.org: An online tool for selecting colour schemes for maps. The Cartographic Journal 2003;40(1):27–37. doi:10.1179/000870403235002042.

[48] Bögl M, Aigner W, Filzmoser P, Lammarsch T, Miksch S, Rind A. Visual analytics for model selection in time series analysis. IEEE Transactions on Visualization and Computer Graphics 2013;19(12). Forthcoming.