COMMIT/

The research reported in this thesis was supported by the Dutch national program COMMIT.

SIKS Dissertation Series No. 2017-07
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Cover created by Karien Verhappen, John Dohmen and Herco Dijk (Studio Kraft).

ISBN 978-90-393-6721-6

Insight in Information:
from Abstract to Anomaly

Inzicht in Informatie
van Samenvatting naar Afwijking
(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor aan de Universiteit Utrecht op gezag van de rector magnificus, prof. dr. G.J. van der Zwaan, ingevolge het besluit van het college voor promoties in het openbaar te verdedigen op woensdag 17 mei 2017 des middags te 2.30 uur

door

Roel Bertens
geboren op 25 september 1986 te Goirle

Promotor: Prof. dr. A.P.J.M. Siebes
Copromotor: Dr. J. Vreeken

Contents

CHAPTER 1

Introduction

As a result of cheap data storage, nowadays it is not the question if a company or institution collects data or not, but rather how much they collect. Transforming data into information and getting insight in this information is perhaps the most important problem in our data rich society. That is, only collecting data serves no goal, but data becomes valuable when insight can be gained from it.

Data mining is the subfield of computer science that concerns itself with transforming large amounts of data into information in the form of patterns. The idea is that the identified patterns result in new insights by exposing interesting structure or behaviour in the data. It may be obvious that defining what exactly is interesting is one of the key challenges.

One of the main applications of data mining on which we focus in this thesis is exploratory data analysis. In this analysis we make use of summaries and characterisations of a dataset to gain insight. That is, by inspecting and exploring the patterns that comprise these models we can extract important information from the data.

Additionally, these summaries can also be used for other data mining tasks, such as the identification of irregular or abnormal data points. All these deviations from what could be expected are called anomalies. We also focus on anomaly detection in this thesis, for which the goal is to gain more insight in the information we already have.

Before we more elaborately discuss data summarisation and anomaly detection we first discuss how to find patterns in sequence data, which is the mainly discussed type of data in this thesis. Since it are patterns that form the building blocks for many data mining tasks. Finally, we switch to multivariate sequential data to introduce our research problems. The main contributions of this thesis are the extension of summarisation and anomaly detection techniques to this more complex data type. In the following we also outline the remainder of this thesis.

Sequence Mining

Before we move to the actual contributions of this thesis we first discuss the type of data that we focus on, which is sequence data. The aforementioned enormous pile of collected data comprises a lot of sequences. Consider the following examples: web data where each user performs a series of clicks or visits a sequence of webpages, alarm systems that produce lists of events, transaction data where for each customer we record its transactions, text data that comprise strings of words, and sensors that measure a multitude of different data over time. We can gain insight in these data by studying the patterns that they contain. For example, in alarm system data a pattern can be a sequence of events that frequently occurs before a false alarm is triggered. This pattern, when presented to a domain expert, may lead to an explanation for the false alarms. For a sensor that measures the temperature of an engine, consider a pattern that describes a very quick fluctuation from cold to hot to cold again. Maybe this pattern results in the insight that the sensor is broken, or maybe there is another explanation. Either way, it captures information about the dataset. As a final example, consider patterns that describe series of visited webpages. These patterns may provide insight about how people navigate through your website and may expose why certain webpages are viewed more often than others.

In Chapter 2 we review the field of sequence mining and discuss the different kinds of patterns that are studied in sequence data. Unfortunately the well-known pattern explosion often buries more insight than it exposes. This is a result of the enormous number of patterns present in each dataset. As noted earlier, defining when a pattern is interesting is not easy. The most common approach is to mine all frequently occurring subsequences. That is, each subsequence that occurs more often than a predefined number of times, the threshold, is regarded as a pattern. The problem with this approach, however, is that setting this threshold too low results in too many patterns for the domain expert to consider. Alternatively, by setting it too high we might miss important information.

Consider a dataset containing series of mouse clicks for users of a website. When many people visit the website there will be many series of clicks that a lot of users perform. For example, ‘first click on topic A and then on topic B’ or ‘first on topic A and then on topic C’, and vice versa and so on. You can imagine that for a growing website the total number of such patterns will quickly become enormous. Moreover, most of these patterns will only supply very obvious information and no new insights. Again, extracting only a small set of interesting patterns from this enormous set is not an easy task.

Summarisation

Many attempts have been made to battle the pattern explosion, from which most focus on reducing the amount of redundancy in the set of all frequent patterns. That is, instead of returning all patterns that occur more often than a certain threshold, these methods only return a condensed or more constrained set. Consider again the earlier discussed mouse click dataset and the frequently occurring pattern ‘click topic A and then click topic B’. From this information we can derive that the smaller pattern ‘click topic A’ will also be frequent, so we

can choose not to report it. Such approaches, however, still return pattern sets that are far too large to be investigated manually. To arrive at more manageable sized pattern sets we rather look for a summary or abstract of the dataset than a condensed or constrained representation of the set of all patterns in the data. Such a summary comprises a small set of patterns that gives a concise but complete view on the dataset. As a result, instead of being buried by the enormous set of all frequent patterns, these summaries are small enough for domain experts to study and can thus lead to new insights.

In Chapter 3 we explain how the Minimum Description Length principle can be used to find such summaries by looking for the set of patterns that gives the best compression of the data. Moreover, we review existing work on the summarisation of transaction and univariate event sequence data which form the inspiration for the research problems discussed later in this chapter.

Anomaly Detection

Besides serving as summaries these pattern sets can also be used for other data mining tasks such as classification, clustering, data generation for privacy protection, and anomaly detection. The latter focusses on the identification of data that significantly differ from the rest of the dataset — so different that it gives rise to the suspicion that they were generated by a different mechanism. Such an anomaly may, e.g., occur because of an error, it may be an outlier, or it may be a highly unexpected data point. Whatever the reason, it provides useful information to the data miner.

As described before, we can use a summary to compress a dataset and we find the best summary by the set of patterns that gives the best compression. Now to identify anomalies we can use our summary again by compressing each (sub)sequence of data with the summary built on the entire dataset. Now, when there are sequences that compress much worse than all other sequences these are considered to be anomalous.

Recall the example dataset consisting of mouse clicks for users of a website. Assume that, given a computed summary, the data for one user compresses much worse than the data for all other users. Studying the clicks of this user that caused the bad compression may help to understand how this user behaved differently from other users. It might, for example, be a result of a very unlogical series of clicks or it might become clear that the bad compression is a result of an error during the data collection process.

Since this thesis follows the path described above by first focusing on mining good abstracts which we later use to identify anomalies, it is clear that the subtitle of this thesis reads ‘from Abstract to Anomaly’.

Multivariate Sequential Data

A lot of the earlier discussed sequence data are multivariate. Examples of such are collections of sensors in a network, multiple frequency bands in seismic or ECG data, annotated text, and multiple temporal features calculated over the same source. To gain insight in such multivariate data we want to be able to identify multivariate structure. That is, the used patterns should be

able to capture correlation between the aligned sequences. For example, consider a dataset that consists of the readings of two aligned sensors that measure for each time step the speed and altitude of somebody who is running. Then we want our patterns to be able to describe, for example, an increasing speed together with a decreasing altitude.

In this thesis we extend the state of the art in summarisation and anomaly detection techniques to this multivariate domain, since these research topics are highly under studied. As a result, the first research problem we focus on is as follows:

How to summarise multivariate sequential data in terms of easily understandable patterns that are able to capture multivariate structure.

That is, we aim to combine the works on sequence mining from Chapter 2 and summarisation from Chapter 3 to summarise multivariate datasets using small sets of, possibly multivariate, patterns.

More specifically, in Chapter 4 we characterise seismograms by employing the Minimum Description Length principle [13]. These characterisations can aid the identification of an event in a new seismogram (an earthquake, passing truck, or something else) by comparing it to seismograms that already have been identified. The patterns used to characterise a seismogram can span multiple frequency bands of the decomposed seismic signal, but may contain no gaps in either time or between adjacent frequency bands.

In Chapter 5 we generalise the ideas from Chapter 4 by considering a much richer pattern language [15]. That is, patterns may contain gaps in either direction and to better compress a dataset we allow patterns to interleave. Moreover, we propose a heuristic algorithm to summarise multivariate sequential data by a small but representative set of, possibly multivariate, patterns that together describe the dataset well. The algorithm that builds such summaries, DITTO, shows great performance both on synthetic and real world datasets.

As already discussed, besides the insight that the patterns in a summary provide, we can also use a summary for many other data mining tasks, such as anomaly detection. Our second research problem puts the focus on anomalies as follows:

How to identify and characterise unexpected behaviour in multivariate sequential datasets.

To solve this problem we briefly shift our attention from sequential to transaction datasets in Chapter 6 where we recall two classes and formally introduce a new class of anomalies and describe how to efficiently identify them. This new class of anomalies describes unexpected co-occurrences of patterns in the dataset. As an example, consider a dataset containing people's drinking habits where roughly half of the people drinks soft drink $\mathbb{C}$ and the other half drinks soft drink $\mathbb{P}$. Now each individual who drinks $\mathbb{C}$ or $\mathbb{P}$ is not surprising. However, someone drinking both $\mathbb{C}$ and $\mathbb{P}$ is an anomaly by definition, since drinking both is unexpected.

Subsequently, in Chapter 7 we merge the ideas of Chapter 5 and 6 and show how to identify, but also how to explain, different classes of anomalies in multivariate event sequences. We show that the proposed algorithms work well on synthetic data and lead to interesting insights on real world datasets. For example, in the domain of smart condition monitoring that focusses on predicting the ideal moment for maintenance of industrial equipment, using

sensor data we were able to identify a failure in a railway switch about 10 days ahead of time as a result of anomalies in the data.

Finally, in Chapter 8 we draw conclusions and summarise the main contributions of this thesis. We also point to several opportunities for further research.

CHAPTER 2

Sequence Mining

There are many places where data occurs in sequences. Examples range from web click data to alarm systems and from transaction to sensor data. Since sequence data is ubiquitous there is much interest to extract valuable information from it. To be able to summarise a sequence dataset or to identify its anomalies we must first be able to find patterns in this data. It are these patterns that form the building blocks for different summarisation and anomaly detection techniques reported on later in this thesis.

Many different approaches have been introduced in literature to gain insight by capturing regularity in the form of patterns. The amount and type of regularity that can be described by the patterns depends on the used pattern language. For example, when we discuss sequential pattern mining approaches in Section 2.1 we focus on subsequences of item(set)s that occur in many sequences. Slightly differently, when we discuss frequent episode mining in Section 2.2 we are interested in subsequences of events (also called episodes) that occur often in the data, i.e., possibly multiple occurrences per sequence. Finally, we only briefly discuss motif discovery in time series in Section 2.3 since working on continuous data directly is yet another field of research.

2.1 Sequential Pattern Mining

Here we focus on approaches that mine sequential patterns from sequence data. We first define the sequence datasets that we consider and we provide the notation we use to describe the sequential patterns we want to mine. Thereafter, we discuss different approaches using many examples to provide the reader with some intuition.

Notation

In this section we consider datasets D of tuples (sid, s), where sid is a unique number identifying each sequence s. A sequence is an ordered list of itemsets, denoted by $\langle s_1 s_2 ... s_n \rangle$,

where s_j is an itemset and n is the length of the sequence. An itemset s_j is a non-empty set of items, denoted by $(i_1 i_2 ... i_m)$, where i_j is an item of the ordered alphabet Ω and m is the size of the itemset also denoted by $|s_j|$. We always order the items within an itemset alphabetically increasing. The sum over the sizes of all itemsets, denoted by $||s||$, gives the size of the sequence, that is $||s|| = \sum_{j=1}^{n} |s_j|$. A sequence of length k is also called a k_l-sequence and a sequence of size k a k_s-sequence. For example, the top row in the dataset displayed in Table 2.1 is a 3_l-sequence, thus containing three itemsets. It is also a 5_s-sequence, because its itemsets contain two, one and two items, respectively. Note that we omit the brackets for itemsets comprising only one item.

A sequence $s_a = \langle a_1 a_2 ... a_n \rangle$ is considered a subsequence of another sequence $s_b = \langle b_1 b_2 ... b_m \rangle$ (denoted as $s_a \sqsubseteq s_b$) when there exist integers $i_1 < i_2 < ... < i_n$ such that $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$, ..., $a_n \subseteq b_{i_n}$. When sequence s_a is a subsequence of sequence s_b it also holds that s_b is a supersequence of s_a and we say that s_a occurs in s_b. For example, the sequence $\langle acc \rangle$ occurs in (is a subsequence of) sequence $\langle (ac)c(ac) \rangle$.

We call a sequence p a (frequent) sequential pattern when it occurs more often than a predefined minimum support threshold min_sup, i.e. for at least min_sup tuples (sid, s) in D it holds that p is a subsequence of s. More formally, the support of a sequence p is defined as follows.

$$support(p) = \sum_{s \in D} \begin{cases} 1 & \text{if } p \sqsubseteq s \\ 0 & \text{otherwise} \end{cases}$$

For a sequential pattern we also refer to each itemset as an element. For example, given a minimum support of four the dataset shown in Table 2.1 contains the sequential pattern $\langle (ac)c \rangle$ which has two elements, (ac) and c, and is highlighted in red.

Table 2.1: On the left we show an example sequence dataset with the sequential pattern $\langle (ac)c \rangle$ highlighted in red (given min_sup = 4). On the right we show the transformed dataset where each sequence is replaced by the set of frequent elements that it contains.

Original dataset	
sid	Sequence
1	$\langle (ac)c(ac) \rangle$
2	$\langle d(ab)a(ab) \rangle$
3	$\langle (ac)(de)bc \rangle$
4	$\langle (ac)ca \rangle$
5	$\langle abc(ac)c \rangle$

Transformed dataset	
sid	Frequent elements
1	$\{a, c, (ac)\}$
2	$\{a\}$
3	$\{a, c, (ac)\}$
4	$\{a, c, (ac)\}$
5	$\{a, c, (ac)\}$

First Approaches

The problem of mining sequential patterns was first introduced by Agrawal and Srikant [4]. They proposed an Apriori-style algorithm to mine the complete set of sequential patterns. Their

algorithm comprises five phases, the sort phase, the frequent element phase, the transformation phase, the sequence phase, and the maximal phase.

In the sort phase the data is fitted to the form of the example dataset on the left in Table 2.1. For example, consider a dataset of multiple transactions from different customers in which we want to find patterns in the form of sequences of item(set)s bought by many customers. Then, in the sort phase the transactions are first grouped per customer and then sorted on transaction time such that each row in the dataset forms a sequence of itemsets (transactions).

In the frequent element phase the set of all frequent itemsets is mined. That is, all sequences of length one with a support above the min_sup threshold, also called the frequent 1_l-sequences, are found.

In the transformation phase each sequence in the dataset is replaced by the set of frequent elements that it contains. For example, on the right in Table 2.1 we show the transformation of the original dataset on the left in Table 2.1.

The sequence phase mines the sequential patterns by iteratively generating larger candidate sequences from a seed set, starting with the frequent elements, and checking their support in the data to form the seed set for the next iteration. The Apriori property states that any supersequence of an infrequent sequence cannot be frequent. Using this property, in each iteration only a reduced set of candidates from all possible sequences of a specific length has to be considered. More specifically, given all frequent n_l-sequences the candidate sequences of length $n + 1$ are generated. Then the support for each candidate is computed, leaving only the frequent ones as the seed set for the next iteration. This process is repeated until no more sequential patterns are found. In Table 2.2 we see the frequent 1_l-sequences, the candidate 2_l-sequences and the frequent 2_l-sequences corresponding to the example dataset in Table 2.1 and a min_sup of four.

Finally, in the maximal phase we can select only the maximal patterns. A sequential pattern is maximal if it is not a subsequence of any other pattern in the set. We discuss maximal patterns in more depth at the end of this section.

Besides allowing for a richer set of patterns, the GSP (Generalised Sequential Patterns) algorithm, introduced by Srikant and Agrawal [82], improves the performance of their previous work. They generalised their pattern language as follows. Firstly, they allow for time constraints between adjacent elements in a pattern. Secondly, they allow for items in an element of a sequential pattern to come from different itemsets of a sequence, as long as these items occur within a predefined time window. Thirdly, they allow sequential patterns to include items across all levels of a predefined taxonomy (*is-a* hierarchy) on the items. Moreover, they changed their approach to combine patterns of the same size (k_s-sequences) instead of the same length (k_l-sequences) to generate new candidates. The pseudocode for the GSP algorithm is given in Algorithm 1. In short, in the first scan of the dataset the frequent items are computed (line 1), whereafter larger patterns are mined iteratively. The patterns found in the k^{th} iteration serve as a seed set to generate candidates of size $k + 1$ (line 5). In each iteration a complete scan of the dataset is made (line 6) to compute the support for all candidate sequences (line 7-9), whereafter only the frequent ones are selected (line 10).

Table 2.2: We show a single iteration of generating and counting sequential patterns. More specifically, the frequent 2_l-sequences (right column) are a result of counting the supports for the candidate 2_l-sequences (middle column) which in turn are created by combining frequent 1_l-sequences (left column) corresponding to the example dataset in Table 2.1 and min_sup = 4.

Frequent 1_l-sequences	Candidate 2_l-sequences	Frequent 2_l-sequences
$\langle a \rangle$	$\langle aa \rangle$	$\langle aa \rangle$
$\langle c \rangle$	$\langle ac \rangle$	$\langle ac \rangle$
$\langle (ac) \rangle$	$\langle a(ac) \rangle$	$\langle cc \rangle$
	$\langle ca \rangle$	$\langle (ac)c \rangle$
	$\langle cc \rangle$	
	$\langle c(ac) \rangle$	
	$\langle (ac)a \rangle$	
	$\langle (ac)c \rangle$	
	$\langle (ac)(ac) \rangle$	

Algorithm 1 The GSP algorithm

Input: A minimum support threshold min_sup and a dataset D
Output: The set of sequential patterns $\mathcal{F}$

```
1:  F_1 ← { frequent items or 1_s-sequences }
2:  k ← 1
3:  while F_k ≠ ∅ do
4:      k ← k + 1
5:      C_k ← generate the set of candidate k_s-sequences from F_{k−1}
6:      for all S ∈ D do
7:          for all α ∈ C_k do
8:              if α ⊑ S then
9:                  support(α) ← support(α) + 1
10:     F_k ← { α ∈ C_k | support(α) ≥ min_sup }
11: return F
```

Reducing the Number of Dataset Scans

For larger datasets, containing long and numerous sequences, more and more candidates need to be considered. That is, the possible number of candidate sequences is exponential to the size of the patterns to be found. Moreover, many scans of the dataset are needed to count the supports for all these candidates. Next, we discuss some improvements on the GSP algorithm that reduce both the number of complete scans of the dataset and the size of the candidate set by decomposing the original search space.

Vertical Id-lists

Zaki introduced the SPADE algorithm [108], greatly reducing runtime compared to GSP using efficient lattice search techniques and simple join operations. That is, where GSP makes a complete scan of the dataset for each iteration, SPADE usually only performs three scans. In Table 2.3 we show the vertical dataset layout used by SPADE corresponding to the dataset from Table 2.1. In each sequence we record, for each occurrence of an item, a tuple containing the item and the time step where it occurs (thus also ends) in the sequence (eid).

Table 2.3: On the left we show the example dataset from Table 2.1. On the right we show the vertical representation used by the SPADE algorithm.

Example from Table 2.1	
sid	Sequence
1	$\langle(ac)c(ac)\rangle$
2	$\langle d(ab)a(ab)\rangle$
3	$\langle(ac)(de)bc\rangle$
4	$\langle(ac)ca\rangle$
5	$\langle abc(ac)c\rangle$

Vertical representation	
sid	$(item, eid)$ pairs
1	(a 1) (a 3) (c 1) (c 2) (c 3)
2	(a 2) (a 3) (a 4) (b 2) (b 4) (d 1)
3	(a 1) (b 3) (c 1) (c 4) (d 2) (e 2)
4	(a 1) (a 3) (c 1) (c 2)
5	(a 1) (a 4) (b 2) (c 3) (c 4) (c 5)

To decompose the search space SPADE makes use of prefix-based equivalence classes, such that only the patterns within each class need to be joined. That is, two sequences are in the same class if they share a common k-length prefix. For example, the set of first level or parent equivalence classes have common prefixes of length 1. Moreover, the SPADE algorithm (Algorithm 2) works as follows. In the first two scans of the dataset we find the frequent 1_s- and 2_s-sequences (line 1-2). Thereafter, the dataset is decomposed into prefix-based parent equivalence classes (line 3). For each equivalence class the frequent sequences are enumerated by the Enumerate-Frequent-Seq algorithm (line 4-5), which recursively decomposes each parent class into smaller classes. In Enumerate-Frequent-Seq, see Algorithm 3, the supports for all combinations between patterns (atoms) are computed by quick joins between the vertical representations (id-lists) of these patterns. These id-lists comprise for each occurrence of the pattern a sequence identifier (sid) and time step where it ends in the sequence (eid). We use $\mathcal{L}(R)$ to refer to the id-list for atom R. A pruning step (line 5) can be inserted to ensure that all subsequences of a pattern are frequent before computing a join. Pruning does, however, come at the cost of a significant memory overhead. Lastly, the SPADE algorithm offers the choice between a breadth-first or depth-first search. The former is more suitable when we want to prune joins of infrequent patterns and the latter requires less main memory.

As an example, consider the id-lists for the patterns $\langle(ac)a\rangle$ and $\langle(ac)c\rangle$ on the left in Table 2.4, corresponding to the dataset in Table 2.3. The result of the very fast join between these two atoms, given min_sup = 2, is presented on the right in Table 2.4. That is, SPADE only needs to consider three possible joins for these patterns. The first join combines occurrences of the two sequential patterns with similar sid and eid, leading to the sequence

Algorithm 2 The SPADE algorithm [108]

Input: A minimum support threshold min_sup and a dataset D
Output: Sequential patterns $\mathcal{F}$ in D

```
1: F_1 ← { frequent items or 1_s-sequences }
2: F_2 ← { frequent 2_s-sequences }
3: ε ← one level decomposition of D
4: for all S ∈ ε do
5:     Enumerate-Frequent-Seq(S)
```

Algorithm 3 Enumerate-Frequent-Seq(S) [108]

Input: A set of atoms of a sub-lattice S, along with their id-lists
Output: Sequential patterns $\mathcal{F}$ in S

```
1: for all atoms A_i ∈ S do
2:     T_i ← ∅
3:     for all atoms A_j ∈ S with j ≥ i do
4:         R ← A_i ∨ A_j
5:         if not Prune(R) then
6:             L(R) ← L(A_i) ∩ L(A_j)
7:             if support(R) ≥ min_sup then
8:                 T_i ← T_i ∪ {R}
9:                 F_|R| ← F_|R| ∪ {R}
10:    if Depth-First-Search then
11:        Enumerate-Frequent-Seq(T_i)
12: if Breadth-First-Search then
13:    for all T_i ≠ ∅ do
14:        Enumerate-Frequent-Seq(T_i)
```

$\langle(ac)(ac)\rangle$. The second join combines occurrences with similar sid and a larger eid for the second pattern, thus creating the sequence $\langle(ac)ac\rangle$, coincidentally not present in the example dataset. Similarly, but in a reversed order, the sequence $\langle(ac)ca\rangle$) is created as the third join.

Projections

The FreeSpan algorithm, introduced by Han et al. [38], mines the complete set of frequent sequences more efficiently than GSP by partitioning the dataset into smaller projected datasets. These projections confine the search as only the projected parts of the dataset need to be scanned to grow larger patterns. The first step of the FreeSpan algorithm is to scan the complete dataset once to find the set of frequent items called f_list. These are then ordered descendingly on their support. For our example dataset from Table 2.1 we have

$$f\_list = [a : 5,\ c : 4,\ b : 3,\ d : 2,\ e : 1] \quad ,$$

Table 2.4: Example of the quick join on id-lists of two sequential patterns from which we can create the id-lists for three larger sequences, only using sid and eid for each occurrence of a pattern.

Id-lists for two patterns

$\langle(ac)a\rangle$		$\langle(ac)c\rangle$	
sid	eid	sid	eid
1	3	1	2
4	3	1	3
		3	4
		4	2
		5	5

Id-lists for three possible joins

$\langle(ac)(ac)\rangle$		$\langle(ac)ac\rangle$		$\langle(ac)ca\rangle$	
sid	eid	sid	eid	sid	eid
1	3			1	3
				4	3

where each item is followed by its support. Then projections are constructed based on the frequent items from f_list to partition the search space. An α-projected dataset comprises the sequences where pattern α occurs without infrequent items and without patterns following α in f_list. See Table 2.5, for the $\langle b\rangle$- and $\langle c\rangle$-projected datasets corresponding to the example from Table 2.1. As a result of dividing the search space using projections, instead of the complete dataset only these projections need to be scanned to grow the frequent patterns, whereafter new projection are constructed recursively.

Table 2.5: Example of the FreeSpan $\langle b\rangle$- and $\langle c\rangle$-projected datasets corresponding to the dataset from Table 2.1 (repeated on the right).

$\langle b\rangle$-projected

sid	Sequence
2	$\langle(ab)a(ab)\rangle$
3	$\langle(ac)bc\rangle$
5	$\langle abc(ac)c\rangle$

$\langle c\rangle$-projected

sid	Sequence
1	$\langle(ac)c(ac)\rangle$
3	$\langle(ac)c\rangle$
4	$\langle(ac)ca\rangle$
5	$\langle ac(ac)c\rangle$

Example from Table 2.1

sid	Sequence
1	$\langle(ac)c(ac)\rangle$
2	$\langle d(ab)a(ab)\rangle$
3	$\langle(ac)(de)bc\rangle$
4	$\langle(ac)ca\rangle$
5	$\langle abc(ac)c\rangle$

Besides the benefit of scanning only projections instead of the complete dataset FreeSpan also considers a smaller number of combinations of patterns than GSP. The expensive part of the algorithm is the construction of the projections. The more dense a dataset, however, the less the proposed projections shrink. As a result, FreeSpan was quickly followed by the celebrated PrefixSpan algorithm introduced by Pei et al. [70, 71].

For PrefixSpan the frequent pattern-guided projection employed in FreeSpan is substituted by a prefix-guided projection. This prefix-based approach makes sure that projected datasets always shrink, leading to an even more focussed search. For PrefixSpan an α-projected dataset,

Algorithm 4 The PrefixSpan Subroutine [70]

Input: A sequential pattern α, the size $||\alpha||$ of α, and a dataset $D|_\alpha$
Output: Sequential patterns of size $||\alpha||$+1

1: Scan $D|_\alpha$ once, find the frequent items b such that
   - b can be assembled to the last element of α to form a pattern α'; or
   - b can be appended to α to form a pattern α'
2: **for all** such α' **do**
3: Output α'
4: PrefixSpan(α', $||\alpha||$+1, $D|_{\alpha'}$)

denoted as $D|_\alpha$ is the collection of postfixes of sequences in D with respect to the prefix pattern α. The postfix is the part of a sequence that occurs after a given pattern in the sequence. For example, in Table 2.6 we show the $\langle b \rangle$- and $\langle (ac) \rangle$-projected datasets of the dataset from Table 2.1. Note that the underscore indicates the presence of the last element of the prefix in an itemset of the sequence.

Table 2.6: Example of the PrefixSpan $\langle b \rangle$- and $\langle (ac) \rangle$-projected datasets corresponding to the dataset from Table 2.1 (repeated on the right).

$\langle b \rangle$-projected

sid	Sequence
2	$\langle (\_a)a(ab) \rangle$
3	$\langle c \rangle$
5	$\langle c(ac)c \rangle$

$\langle (ac) \rangle$-projected

sid	Sequence
1	$\langle c(ac) \rangle$
3	$\langle (de)bc \rangle$
4	$\langle ca \rangle$
5	$\langle c \rangle$

Example from Table 2.1

sid	Sequence
1	$\langle (ac)c(ac) \rangle$
2	$\langle d(ab)a(ab) \rangle$
3	$\langle (ac)(de)bc \rangle$
4	$\langle (ac)ca \rangle$
5	$\langle abc(ac)c \rangle$

The input of the PrefixSpan algorithm is a sequence dataset and a minimum support threshold and it outputs the complete set of sequential patterns. The algorithm starts with the following initial call to the PrefixSpan Subroutine (Algorithm 4): PrefixSpan($\langle \rangle$, 0, D). This subroutine takes as input a sequential pattern, the size of that pattern, and the projected dataset for that pattern. Then it works as follows, it scans the projected dataset to find sequential patterns that are one item larger (line 1), whereafter it recursively calls the subroutine on the projected dataset for this new pattern (line 4). This approach requires no candidate generation, but the most expensive step is again to construct projected datasets. To reduce these costs different projection methods are used. Firstly, bi-level projections reduce the number of projections by also counting supports for 2_s-sequences in a single iteration. As a result, only projections for frequent 2_s-sequences have to be constructed, thereby skipping all 1_s-sequences. Of course this comes at the cost of a more expensive counting step. Secondly, pseudo-projections use pointers and offset to identify projections in-memory.

Longer Patterns

Focusing on longer patterns Ayres et al. [9] introduced the depth-first search algorithm SPAM. It assumes that the entire dataset fits into main memory and by employing a vertical bitmap representation of the dataset it allows for very efficient support counting. As a result it improves upon SPADE and PrefixSpan in terms of runtime, especially for longer sequences.

Additional Constraints

Besides setting a minimum support threshold, other constraints can be enforced to reduce the redundancy and the size of the set of sequential patterns. That is, instead of mining all frequent sequences we can look for maximal, closed or the top-k most frequent sequences. In a set of sequences, a sequence is *maximal* if it is not a subsequence of any other sequence in the set. A set of sequences is *closed* when for each sequence it only contains subsequences of this pattern when the support of these subsequences is higher than its own support. Note that the set of closed sequential patterns is a lossless compression of the complete set of sequential patterns. Both maximal and closed patterns were first coined in the domain of frequent itemset mining, by Bayardo [12] and by Pasquier et al. [69], respectively. In Table 2.7 we show the impact of mining only constrained sets of sequential patterns.

Table 2.7: The maximal, closed and complete set of sequential patterns corresponding to the dataset in Table 2.1 and min_sup = 4, where each pattern is followed by its support.

Maximal	Closed	All
$\langle(ac)c\rangle$: 4	$\langle(ac)c\rangle$: 4	$\langle(ac)c\rangle$: 4
$\langle aa\rangle$: 4	$\langle aa\rangle$: 4	$\langle aa\rangle$: 4
	$\langle a\rangle$: 5	$\langle ac\rangle$: 4
		$\langle cc\rangle$: 4
		$\langle(ac)\rangle$: 4
		$\langle a\rangle$: 5
		$\langle c\rangle$: 4

Closed Patterns

To mine closed sequential patterns in large datasets Yan et al. [105] introduced CloSpan. They use global optimization techniques to discover significantly smaller sets of frequent sequences, with the same expressive power as all frequent sequences. As a result, much larger sequences can be discovered in a shorter time. Their recursive approach first generates a superset of closed frequent sequences whereafter it performs post-pruning to eliminate non-closed sequences. It utilises successful concepts of both PrefixSpan and SPAM together with other techniques such as hashing to quickly mine the set of closed sequential patterns.

Wang and Han [100] introduced the BIDE algorithm which stands for BI-Directional Extension based frequent closed sequence mining. As its name suggests it comprises the extension of patterns in two directions. The forward directional extension is used to both grow and check the closure of prefix patterns. The backward directional extension is also used to check the closure of a prefix pattern, but additionally it is used to prune the search space. A forward extension of a sequence adds an item to the end of the sequence, where a backward extension adds an item before the last time step. When a pattern cannot be extended in either way it must be closed. As a result, there is no need for sub- or superpattern checking. Moreover, this new approach dispenses the need for candidate maintenance, as applied in CloSpan, which can be very expensive, especially when the number of frequent closed sequences is large. Thus, leading to an even faster and more memory efficient algorithm.

Top-k Patterns

Tzvetkov et al. [91] introduced a method to mine the top-k closed sequential patterns which are all not smaller than a predefined size. They propose a heuristic algorithm to quickly raise the minimum support threshold used for mining, thereby rapidly pruning the search space. It further builds upon PrefixSpan and CloSpan to efficiently compute the correct result. Moreover, given their approach the user only needs to specify a minimal pattern size instead of a minimum support threshold.

2.2 Frequent Episode Mining

Since we have already discussed sequential patterns in Section 2.1, we now switch our attention to frequent episodes, which are collections of events that occur relatively close to each other in a given partial order [61]. Compared to a sequential pattern a frequent episode can have multiple occurrences within a single sequence. To determine the total number of occurrences we can either use a sliding window approach or we can count the disjoint set of minimal windows [62]. We explain both approaches in more detail later. After introducing the notation we discuss several different kinds of patterns such as parallel, serial and strict episodes. We end this section with a small discussion of other related work.

Notation

In this section we consider datasets $D = \{S_1 S_2 ... S_{|D|}\}$ comprising $|D|$ event sequences, where each sequence $S_i = \langle e_1 e_2 ... e_{||S_i||} \rangle$ consists of $||S_i||$ events, also referred to as its size. When multiple events can occur simultaneously the length $t(S_i)$ of the sequence, i.e. the number of different time steps on which events occur, differs from its size. Each event e is a triple $(ts(e), lab(e), id(e))$ with a time step integer $ts(e)$, a label $lab(e)$ and a unique id number $id(e)$ per time step. The id number is used to differentiate between multiple events that occur simultaneously. The label indicates the type of event from the alphabet Ω, i.e. $lab(e) \in \Omega$. The time step defines the order between the events in the sequence. Since our datasets are just strings of events we simply represent them without time steps in our examples.

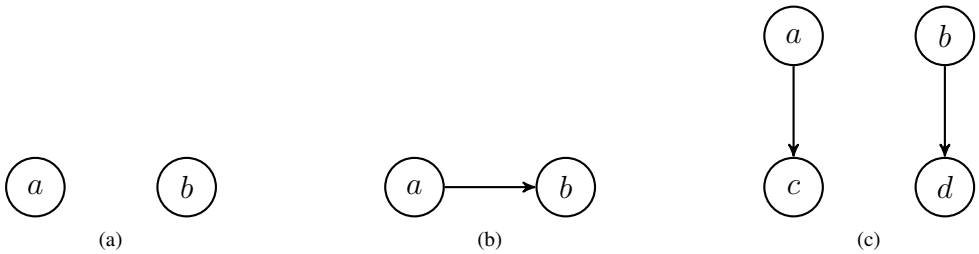


Figure 2.1: Examples of a parallel (a), serial (b) and general episode (c). The parallel episode matches sequences containing $\langle ab \rangle$, $\langle ba \rangle$ and $\langle (ab) \rangle$, and the serial episode only matches sequences containing $\langle ab \rangle$.

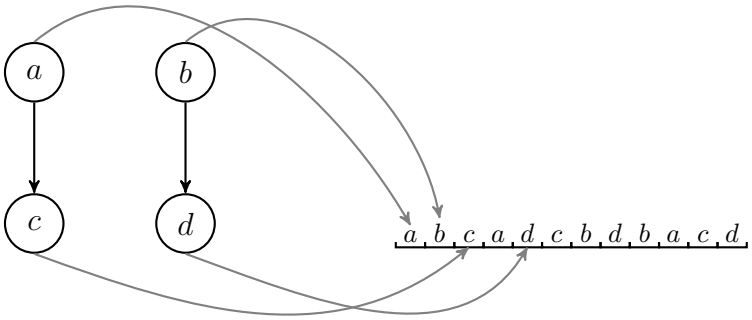
Parallel, Serial and General Episodes

A general episode is a partially ordered collection of events occurring closely together. A parallel episode has no constraints on the order of its events, whereas the events in a serial episode form a total order. We can represent episodes as directed acyclic graphs (DAG), since they can describe the order restrictions of an episode. The graph corresponding to a parallel episode contains only nodes and no arcs, whereas the graph for a serial episode contains a chain of events connected with directed arcs. Other episodes that are a combination between parallel and serial episodes are called general episodes. For example, consider the parallel, serial, and general episodes from Figure 2.1, where the parallel episode matches sequences containing $\langle ab \rangle$, $\langle ba \rangle$ and $\langle (ab) \rangle$, and the serial episode only matches sequences containing $\langle ab \rangle$.

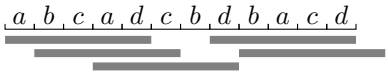
An episode occurs in a sequence when each event from the DAG only occurs after all its parent events (w.r.t. the DAG) in the sequence. For example, we indicated the first occurrence of an episode with arrows in the dataset from Figure 2.2. We call an episode frequent when it occurs more than a predefined number of times (minimum support) in the dataset.

To determine the support of an episode we need to define how we count its occurrences. We can, for example, take a sliding window (contiguous fixed-size interval of the sequence) approach and count the number of windows that fully contain the episode [62]. To ensure that events close to either end of the sequence are contained in equally many windows we also consider windows that extend partly outside the sequence. For example, the first window only contains the first time step and the last window only the last time step. The five windows of length five containing the example episode are highlighted in Figure 2.2b. With a total of 16 windows of length of five this results in a support of 5 (out of 16) for the example episode.

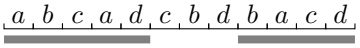
Another approach is to count the maximal number of disjoint (non-overlapping) minimal windows in which the episode occurs, also called minimal occurrences [62]. A minimal window is an interval in the dataset in which an episode occurs for which there is no proper subinterval in which the episode also occurs [51, 86]. A maximum of two minimal windows for the example episode is highlighted in Figure 2.2c. Note that there is an alternative to



(a) The arrows highlight the first occurrence of the episode in the dataset.



(b) 5 fixed-size windows.



(c) A maximum of two minimal windows.

Figure 2.2: The first occurrence of the example episode is highlighted by arrows pointing to the dataset (a). The windows that contain the example episode are highlighted for both a sliding window (b) and a minimal window approach (c). Example borrowed from Tatti and Cule [88].

highlight two other disjoint minimal windows in this example.

To mine all parallel or serial episodes Mannila et al. [62] introduced two breadth-first search Apriori style algorithms called WINEPI and MINEPI. They perform a level-wise search, generating candidate episodes and selecting only the frequent ones. WINEPI uses a sliding window approach and MINEPI employs the concept of minimal windows. Both algorithms can find either all parallel or all serial frequent episodes. Mannila and Toivonen [59] proposed a more general framework for episode discovery, where episodes are defined as combinations of events satisfying certain predefined conditions.

Closed Strict Episodes

A frequent episode is closed when there exists no superpattern with the same support. It is not possible to define a unique closure based on frequency since an episode may have multiple closed superepisodes. To be able to define the concept of closedness for episodes Tatti and Cule introduced strict episodes [88]. Strict episodes are a subclass of general episodes, and we call an episode strict when all nodes with the same label are connected in the DAG. See Figure 2.3 for an example, where the non-strict episode matches the sequence $\langle (aa)bc \rangle$ and

the strict episode does not. Their algorithm efficiently mines closed strict episodes thereby removing a lot of redundancy from the resulting pattern set. As a result, much higher minimum support thresholds can be used for which finding all frequent episodes would be infeasible due to the prohibitively large number of returned patterns.

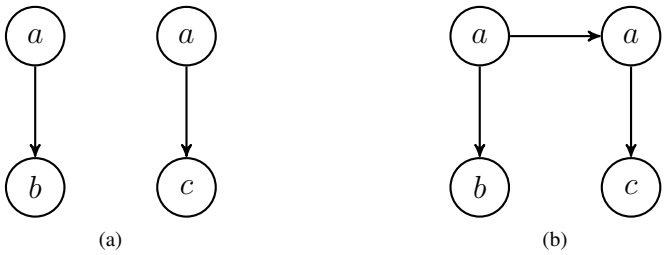


Figure 2.3: Example of a non-strict (a) and a strict episode (b). The former matches the sequence $\langle (aa)bc \rangle$ and the latter does not.

Closed Episodes with Simultaneous Events

Tatti and Cule extended their definition of an episode to be able to mine closed general episodes with simultaneous events [87]. Their method captures four different relationships between the events a and b: (1) their order does not matter, (2) they occur at the same time, (3) event b occurs after event a, and (4) event b occurs after or at same time as event a. To represent these relationships in a DAG they use proper (solid) and weak (dashed) edges. A proper edge from a to b means that event a must occur before event b. A weak edge from a to b indicates that event a occurs at the same time or before event b. In Figure 2.4 we show an example episode and we highlighted its two minimal windows in a dataset. It first occurs with and thereafter without simultaneous events.

Figure 2.4: Example episode that first occurs with and thereafter without simultaneous events in the dataset. Both minimal windows are highlighted.

To counter the well-known pattern explosion, Tatti and Cule only mine closed episodes. They introduce a subset relationship between episodes based on coverage instead of based on DAGs, since there can be multiple DAGs representing the same episode. For example, consider the DAGs in Figure 2.5 both representing the same episode. A single window may contain multiple instances of an episode, see Figure 2.6 for an example. To efficiently grow episodes Tatti and Cule proposed to keep track of all these instances. Further, they introduce

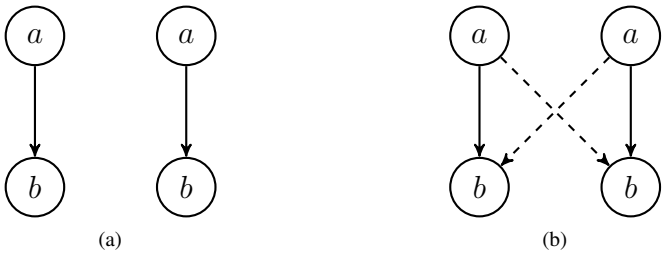


Figure 2.5: Both DAGs represent the same episode and match all sequences containing $\langle abab \rangle$, $\langle a(ab)b \rangle$, $\langle (aa)(bb) \rangle$, $\langle aa(bb) \rangle$, $\langle (aa)bb \rangle$, or $\langle aabb \rangle$.

the concept of instance-closure to reduce the search space. Since an instance-closed episode is not guaranteed to be actually closed they filter the episodes in a post-processing step resulting in an efficient depth-first search algorithm.

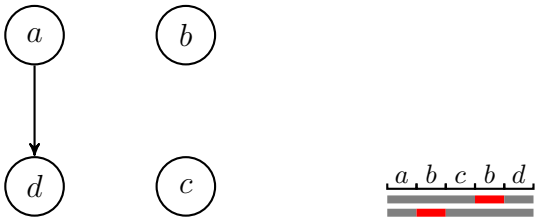


Figure 2.6: A window containing two instances of the example episode. In each window the event highlighted in red does not belong to the instance of the episode.

Other Related Works

There are many other related works in this field. For example, to get an overview of the ordering relationships in the data Mannila and Meek [58] introduced a method to discover partial orders from sequences of events. However, they only look at parallel and serial partial orders and each type of event may occur only once. Similarly, Achar et al. [1] mine general episodes but restrict their search to injective episodes, that is comprising only unique labels. Chen et al. [25] mine complex interval-based patterns and introduce an efficient algorithm called CTMiner.

2.3 Motif Discovery in Time Series

For completeness we briefly discuss motif discovery in time series data, since continuous time series are inherently different from the previously described sequences.

The discovery of motifs in time series data was formally introduced by Lin et al. [55] where motifs are defined as approximately repeated subsequences. That is, a motif is a subsequence that approximately matches many other subsequences in the data, and two subsequences in the dataset match each other when their distance is smaller than some predefined threshold. In Figure 2.7 we show an example from [55] containing a motif occurring three times in the dataset. Chiu et al. [27] improved the scalability of this method and extended the flexibility of the pattern language by allowing for small gaps (noise) in a motif.

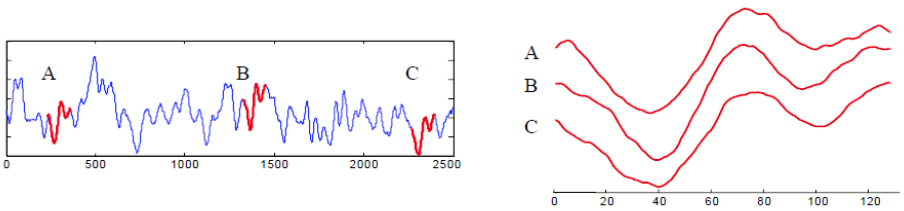


Figure 2.7: A time series containing three nearly identical subsequences A, B, and C [55].

To be able to efficiently discover these motifs Lin et al. [56] introduced SAX, which transforms the original time series into a symbolic sequence that allows a distance measure that lower bounds a distance measure defined on the original time series. Since the resulting discretised sequence fits the previously discussed framework of serial episodes, we do not further investigate motif discovery. We do, however, explain in detail how to discretise a time series using SAX because we use it in the experiments of the following chapters.

SAX

SAX is short for Symbolic Aggregate approXimation. Next to discretising a time series it is also able to reduce its length. The first step is to transform the data by Piecewise Aggregate Approximation (PAA) [106] into an intermediate representation, thereby reducing the dimensionality of the data. This step is also important in the proof that the distance between two symbolic sequences is lower bounded by the distance between the two original time series. Consider a time series $C = \langle c_1 c_2 ... c_n \rangle$ of length n and its PAA representation $\bar{C}$ of length w. We can simply compute the value for each of its elements, by taking the mean value of the data within w equal sized intervals, as follows.

$$\bar{c}_i = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} c_j$$

In Figure 2.8a we show an example sequence C and its PAA representation $\bar{C}$. Note that before transforming a time series into its PAA representation it is first normalised to have zero mean and standard deviation one.

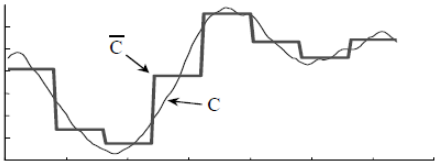
The next step is to assign a symbol to each interval such that each symbol has a similar frequency in the resulting sequence. Since SAX assumes that the values of time series are normally distributed, the breakpoints leading to Ω equal-sized areas under the Gaussian curve can be easily computed. For example, the breakpoints (β_i) that divide a Gaussian distribution in an arbitrary number ($|\Omega|$) of equiprobable regions can be found in Table 2.8. To compute the symbol $\hat{c}_i$ corresponding to an element $\bar{c}_i$ from the PAA representation we have

$$\hat{c}_i = \Omega_j, \quad \text{iff} \;\; \beta_{j-1} \leq \bar{c}_i < \beta_j \quad ,$$

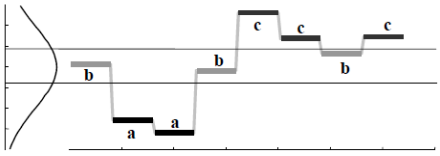
where Ω_j is the j^{th} symbol from the alphabet Ω in lexicographical order. As a result, these symbols retain the order between the PAA intervals. In Figure 2.8b we show the symbolic representation of the sequence from Figure 2.8a.

Table 2.8: A lookup table that contains the breakpoints (β_i) that divide a Gaussian distribution in an arbitrary number ($|\Omega|$) of equiprobable regions [56].

$\|\Omega\|$	3	4	5	6	7	8	9	10
β_1	-0.43	-0.67	-0.84	-0.97	-1.07	-1.15	-1.22	-1.28
β_2	0.43	0	-0.25	-0.43	-0.57	-0.67	-0.76	-0.84
β_3		0.67	0.25	0	-0.18	-0.32	-0.43	-0.52
β_4			0.84	0.43	0.18	0	-0.14	-0.25
β_5				0.97	0.57	0.32	0.14	0
β_6					1.07	0.67	0.43	0.25
β_7						1.15	0.76	0.52
β_8							1.22	0.84
β_9								1.28



(a) A dimensionality reduced PAA representation $\bar{C}$ of a time series C.



(b) The SAX representation $\hat{C} = \langle baabccbc \rangle$ of the time series from Figure 2.8a, using an alphabet of size 3.

Figure 2.8: The dimensionality reduced PAA representation (a) and discretisation (b) of SAX in action [55, 56].

CHAPTER 3

Summarising Datasets using MDL

Summarising a dataset means describing it in a more succinct way, for example by a small set of descriptive patterns. Since virtually all datasets contain enormous piles of patterns, selecting only the interesting ones is not easy. In this chapter we show how we can employ the Minimum Description Length (MDL) principle to do exactly this. More specifically, we use it to measure how well the selected set of patterns describes the dataset at hand in search for the best summary. In the remainder we first give a short introduction into MDL in Section 3.1 and continue with examples of summarising different types of datasets using MDL in Section 3.2. The work described in this chapter serves as a basis on which the next chapters build. Therefore, our goal is not to provide a complete overview, but to provide a gentle introduction to the following chapters.

3.1 An Introduction to MDL

A Brief History

In the field of information theory the Minimum Description Length (MDL) principle was pioneered by Rissanen [75]. Information theory studies the compression and transmission of data and can for a great part be described by learning through compression. In the 1960's Solomonoff [80], Kolmogorov [46], and Chaitin [23] independently developed *Kolomogorov complexity* [53], which laid the foundation for the theory of descriptive complexity. More simply, it can be regarded as measuring the ultimate data compression. A few years later, Wallace [98], not aware of the notion of Kolmogorov complexity, introduced the Minimum Message Length principle which is very closely related to MDL. About another 10 years later Kolomogorov's work, together with Akaike's AIC method for model selection [6], inspired Rissanen [75] to develop MDL. More recently, Grünwald [35] wrote a nice introduction and a very complete overview of the MDL principle.

The MDL Principle

The fundamental idea underlying MDL is equating learning to compression. That is, we can use compression to learn the laws and regularities that reside in our data. As an example consider the following two sequences both comprising 10 000 bits, where the first sequence consists of alternating zeros and ones and the second sequence is random. That is, it can be regarded as the result of 10 000 flips of a fair coin.

sequence 1: 01010101010...10101010101
sequence 2: 11010001100...01001011111

What we can learn from the first sequence is that it simply contains 5 000 repetitions of the pattern 01 and that it will most likely continue similarly following this 'law'. For the second sequence, however, we cannot identify any regularity since it is random.

Moving to compression, we first recall that the Kolomogorov complexity of a string of symbols x is the length of the shortest program that produces the output x and halts. Thus, the more regular a sequence, the less complex and the shorter we can describe it. We consider a general-purpose computer language as description method to compress sequences. More specifically, a description of a sequence can be regarded as a computer program that prints the sequence and halts. To compress the first sequence we can write the following program:

```
for i = 1 to 5 000 do
    print '01'
return
```

It is much shorter than the original sequence because it makes use of the repeating pattern in the data. For the second sequence, however, the best we can do is to write a program that contains the complete sequence as follows.

```
print '11010001100...01001011111'
return
```

Since it contains no regularity we also have no way to describe it more succinctly.

From this example we can conclude that we can use any regularity in the data to find shorter descriptions of the data, i.e. using fewer symbols than describing the data literally. Further, it is also this regularity that learns us something about the data. Hence, the apt description *induction by compression*.

To learn as much as possible our goal is to find the shortest program that prints the data. The problem, however, is that we cannot compute such a shortest program [53]. As a result, we focus on practical versions of MDL that use description methods that are less expressive than general-purpose computer languages. We do this by making sure that for any sequence we can compute the length of the shortest description, which comes at the cost of not being able to always compress all sequences. Which seems reasonable, because, as already mentioned, there is no method for inductive inference that will always give us all regularity.

To summarise, we try to find regularity in the data, where regularity can be expressed by the ability to compress. MDL combines these concepts and can be applied to the problem of model selection. That is, among a set of candidate models select the one that is most likely to have generated the dataset at hand. Employing the MDL principle we can use compression to select the best model as the model that gives the best compression. More formally, given a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ is the one that minimises

$$L(M) + L(D \mid M)$$

where $L(M)$ is the length of the description of M, and $L(D \mid M)$ is the length of the description of the data. This is called two-part (or crude) MDL, which differs from refined MDL by encoding the data and the model separately. We focus on this version of MDL, because we are especially interested in the model, which gives us the summary we are after.

Note that MDL is strongly related to the maximum a posteriori probability (MAP) estimate in Bayesian statistics. Given a prior probability on the model we can look for the model that maximises the posterior probability on the model given the data. That is, we have

$$\arg\max_M P(M \mid D) = \arg\max_M \frac{P(M) \times P(D \mid M)}{P(D)} \quad .$$

Since the data is the same for each model we can disregard the denominator. The main difference with MDL is that using Bayes' rule we have to estimate a prior probability for the model instead of defining an encoding/decoding schema. That is, using MDL the probability $P(M)$ is simply the description length of M given the chosen description method.

MDL exhibits a form of Occam's Razor since it balances the fit of the model with its complexity. That is, MDL picks the simplest model when two models describe the data equally well. Further, it automatically protects against overfitting by taking the size of the model into account. Moreover, compared to many other statistical methods, the concept of MDL does not depend on any underlying 'true' model for the data. For example, there is no need to make assumptions about the process generating the data by describing a set of probability distributions. MDL models have a clear interpretation independent of the fact whether or not there exists some underlying 'true' model.

3.2 Summarising Datasets

In this section we describe how MDL can be employed to summarise a dataset. Firstly, we focus on transaction data, whereafter we shift our attention to event sequences. For both domains we describe a successful algorithm in more depth.

Summarising Transaction Datasets

The first to use MDL to summarise a transaction dataset were Siebes et al. [77], which resulted in the KRIMP algorithm [97]. This research originated from the MDL theory and shifted focus from the long-standing goal of selecting a collection of patterns that describes the set of all

frequent patterns (such as, closed [69], non-derivable [21], maximal [12] frequent patterns, and δ-free sets [29]) to a *pattern set* that describes the entire dataset.

Next, we first define how to use MDL to summarise a transaction dataset, whereafter we present the successful KRIMP algorithm. We end with a short discussion of other related works in this domain.

MDL for Transaction Datasets

A transaction dataset D is a bag of transactions where each transaction t is a subset of the alphabet $\mathcal{I}$, i.e. $t \subseteq \mathcal{I}$. A pattern or itemset X is just a set of $|X|$ items $X \subseteq \mathcal{I}$ and the support of an itemset is the number of transactions in which it occurs, i.e. $support(X) = |\{t \in D \mid X \subseteq t\}|$. Throughout this thesis all logarithms have base 2 and by convention we have $0 \log 0 = 0$.

As summaries for these datasets Vreeken et al. [97] use the concept of a code table.

Definition 1. *Let $\mathcal{I}$ be a set of items and $\mathcal{C}$ a set of code words [28]. A code table CT over $\mathcal{I}$ and $\mathcal{C}$ is a two-column table such that:*

1. *The first column contains itemsets, that is, subsets over $\mathcal{I}$. This column contains at least all singleton itemsets.*
2. *The second column contains elements from $\mathcal{C}$, such that each element of $\mathcal{C}$ occurs at most once.*

An itemset X, drawn from the power set of $\mathcal{I}$, i.e. $X \in \mathcal{P}(\mathcal{I})$, occurs in CT, denoted by $X \in CT$ iff X occurs in the first column of CT, similarly for a code $C \in \mathcal{C}$. For $X \in CT$, $code_{CT}(X)$ denotes its code, i.e. the corresponding element in the second column. We call the set of itemsets $\{X \in CT\}$ the coding set PS. [97]

See Figure 3.1 for an example code table. Taking these code tables as our models we can employ MDL to search for the code table that gives the best possible compression. Note that we use the terms model, code table, and summary interchangeably.

To be able to fairly compare models MDL requires that we consider the lengths of lossless descriptions of the data. Moreover, this implies that we are not interested in the actual code words, but only in the optimal code lengths. Next, we discuss how to describe or compress a dataset given a code table. That is, we need to be able to encode and decode the data using our model. Only then can we compute the total compressed size of the dataset and the code table to be able to compare different models. To encode a dataset given a code table we need to determine where we use which pattern to describe the dataset. That is, we need a *cover* function $cover(CT, t)$ that chooses for each transaction which patterns from CT to use to compress it, thereby describing each item in the transaction exactly once. More formally we have the following definition for a cover.

Definition 2. *Let D be a dataset over a set of items $\mathcal{I}$, t a transaction drawn from D, let $\mathcal{CT}$ be the set of all possible code tables over $\mathcal{I}$, and CT a code table with $CT \in \mathcal{CT}$. Then, $cover : \mathcal{CT} \times \mathcal{P}(\mathcal{I}) \to \mathcal{P}(\mathcal{P}(\mathcal{I}))$ is a* cover function *iff it returns a set of itemsets such that*

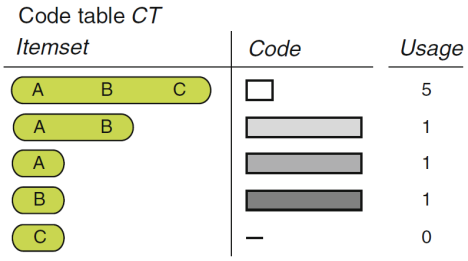


Figure 3.1: Example code table. The widths of the codes represent their lengths. $\mathcal{I} = \{A, B, C\}$. Note that the usage column is not part of the code table, but shown here as illustration: for optimal compression, codes should be shorter the more often they are used. [97]

1. *$cover(CT, t)$ is a subset of PS, the coding set of CT, i.e.* $X \in cover(CT, t) \rightarrow X \in CT$
2. *if $X, Y \in cover(CT, t)$, then either $X = Y$ or $X \cap Y = \emptyset$*
3. *the union of all $X \in cover(CT, t)$ equals t, i.e.* $t = \bigcup_{X \in cover(CT,t)} X$

We say that $cover(CT, t)$ covers t. Note that there exists at least one well-defined $cover$ function on any code table CT over $\mathcal{I}$ and any transaction $t \in \mathcal{P}(\mathcal{I})$, since CT contains at least the singleton itemsets from $\mathcal{I}$. [97]

In Figure 3.2 we see a possible cover of the dataset given the code table in Figure 3.1. Now to encode a dataset given a code table we just replace all patterns in the cover of a transaction with their corresponding code in the code table. Since we are after the best compression of the dataset we use optimal prefix codes [28], the length of which we can easily compute by Shannon entropy. Intuitively, we assign shorter codes to more frequently used patterns. Note that using such prefix codes we are also able to decode the encoded dataset unambiguously, thus resulting in a lossless compression.

Theorem 1. *Let P be a distribution on some finite set D. There exists an optimal prefix code $\mathcal{C}$ on D such that the length of the code for $d \in D$, denoted by $L(d)$, is given by*

$$L(d) = -\log(P(d)).$$

Moreover, this code is optimal in the sense that it gives the smallest expected code size for data sets drawn according to P. (For the proof, please refer to Theorem 5.4.1 in [28].) [97]

The probability that the cover function assigns to each pattern from the code table is given by its relative usage in the cover. Moreover, the code length of a pattern in the code table is equal to the negative logarithm of this probability [53]. More formally, to compute the optimal codes for the patterns in the code table we have the following definition.

Definition 3. *Let D be a transaction dataset over a set of items $\mathcal{I}$, $\mathcal{C}$ a prefix code, $cover$ a cover function, and CT a code table over $\mathcal{I}$ and $\mathcal{C}$. The* usage *count of an itemset $X \in CT$ is defined as*

$$usage(X) = |\{\, t \in D \mid X \in cover(CT, t) \,\}|.$$

The probability *of $X \in CT$ being used in the cover of an arbitrary transaction $t \in D$ is thus given by*

$$P(X \mid D) = \frac{usage(X)}{\sum_{Y \in CT} usage(Y)}.$$

The $code_{CT}(X)$ for $X \in CT$ is optimal *for D iff*

$$L(code_{CT}(X)) = |code_{CT}(X)| = -\log(P(X|D)).$$

A code table CT is code-optimal *for D iff all its codes,*

$$\{\, code_{CT}(X) \mid X \in CT \,\},$$

are optimal for D. [97]

For example, the code length for the pattern $X = \{A, B, C\}$ in Figure 3.1 corresponding to the dataset and cover in Figure 3.2 is $L(code_{CT}(X)) = -\log(5/8) \approx 0.68$.

Now we can easily compute the encoded length of the dataset given a code table by taking the sum over the code lengths of the patterns in the cover of the dataset. This leads to the following trivial lemma.

Lemma 2. *Let D be a transaction dataset over $\mathcal{I}$, CT be a code table over $\mathcal{I}$ and code-optimal for D,* cover *a cover function, and* usage *the usage function for* cover. *[97]*

1. *For any $t \in D$ its encoded length, in bits, denoted by $L(t \mid CT)$, is*

$$L(t \mid CT) \quad = \quad \sum_{X \in cover(CT, t)} L(code_{CT}(X)).$$

2. *The encoded size of D, in bits, when encoded by CT, denoted by $L(D \mid CT)$, is*

$$L(D \mid CT) \quad = \quad \sum_{t \in D} L(t \mid CT).$$

Following the MDL principle we also have to account for the size of the model when scoring the quality of a code table. The description length of the code table is the sum of the code lengths in the right column and the compressed size of the patterns in the left column. Since a binary integer encoding using $\log(\mathcal{I})$ bits per item is not optimal we use the simplest possible code table, containing only singleton patterns, to encode the patterns in the left column. We refer to this *standard code table* containing only singleton patterns as ST.

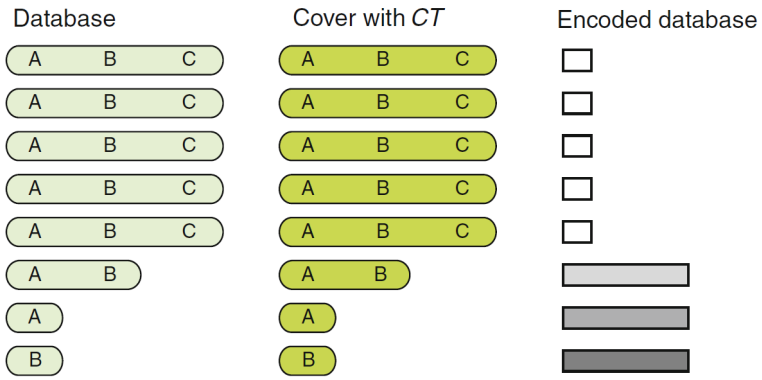


Figure 3.2: Example dataset, and the cover and encoded dataset obtained by using the code table shown in Figure 3.1. $\mathcal{I} = \{A, B, C\}$. [97]

Definition 4. *Let D be a transaction dataset over $\mathcal{I}$ and CT a code table that is code-optimal for D. The size of CT in bits, denoted by $L(CT \mid D)$, is given by*

$$L(CT \mid D) = \sum_{X \in CT : usage(X) \neq 0} L(code_{ST}(X)) + L(code_{CT}(X)).$$

Note that we do not take itemsets with zero usage into account. Such itemsets are not used to code. [97]

Now that we know how to compute the size of the dataset given the code table and the size of the model itself we arrive at the final definition.

Definition 5. *Let D be a transaction dataset over $\mathcal{I}$, let CT be a code table that is code-optimal for D and* cover *a cover function. The* total compressed size *of the encoded dataset and the code table, in bits, denoted by $L(D, CT)$ is given by*

$$L(D, CT) = L(D \mid CT) + L(CT \mid D).$$

[97]

Our goal is to find the code table that minimises $L(D, CT)$, since it will give us the most succinct description of our dataset. Next, we discuss the KRIMP algorithm which approximates the optimal result.

The KRIMP Algorithm

The KRIMP algorithm introduced by Vreeken et al. [97] battles the pattern explosion by mining only small sets of characteristic patterns that together succinctly summarise a transaction

dataset. Ideally, we want to find the optimal code table, that is, the one that leads to the best compression. Unfortunately, the search space is too big to consider exhaustively and it exhibits no useable structure that can be used to prune large parts of it. To give an idea, there are already

$$\sum_{k=0}^{2^{|\mathcal{I}|}-|\mathcal{I}|-1} \binom{2^{|\mathcal{I}|}-|\mathcal{I}|-1}{k}$$

possible different coding sets, since we only know it must contain the $|\mathcal{I}|$ singleton patterns. To find the one giving the best compression we also have to consider each possible cover order given a code table. That is, the order in which the patterns are used to cover a transaction affect the usages of the patterns and thus their code lengths. This, in turn, affects the total compressed size of the dataset. To quickly find good code tables Vreeken et al. introduced the heuristic KRIMP algorithm which makes a few greedy choices. Firstly, it only considers each candidate pattern once in a predefined order, and secondly, it covers the transactions using the most promising patterns first. Moreover, to cover the transactions KRIMP uses the **Standard Cover Order**, which is defined as follows

$$|X| \downarrow \quad support(X) \downarrow \quad \text{lexicographically} \uparrow \quad .$$

Thus preferring longer and more frequent patterns. Further, the order in which KRIMP considers all candidate patterns to add to the code table is defined as the **Standard Candidate Order** as follows

$$support(X) \downarrow \quad |X| \downarrow \quad \text{lexicographically} \uparrow \quad .$$

Here the most frequent patterns are preferred since a lot of gain can be made by replacing many occurrences with shorter codes. A schematic overview of the KRIMP algorithm is given in Figure 3.3. In Algorithm 5 we describe KRIMP more precisely. As input it takes a transaction dataset D and a set of candidate patterns $\mathcal{F}$ to produce a code table CT that succinctly describes D. It starts with the standard code table ST (line 1) and iteratively considers the patterns from $\mathcal{F}$ in **Standard Candidate Order** (line 2-3). Each pattern is added to CT (line 4) and when it improves compression (line 5) it is kept in CT (line 6), otherwise it is discarded. Because the addition of a new pattern to the code table might lead to redundancy in the code table, a pruning step can be inserted after the acceptance of a new pattern. In this pruning step all patterns in the code table for which their usage decreased are considered for removal. For example, consider a code table containing the pattern $X = \{A, B\}$ and the recently accepted pattern $Y = \{A, B, C\}$. When (almost) all occurrences of the pattern X in the dataset are covered by pattern Y it might be cheaper to remove X from the code table.

Other Related Works

Among other related works to summarise a transaction dataset Geerts et al. [33] proposed to find the smallest set of patterns that covers as much of the data as possible. Since their TILING algorithm does not take the complexity of the pattern set into account, the results

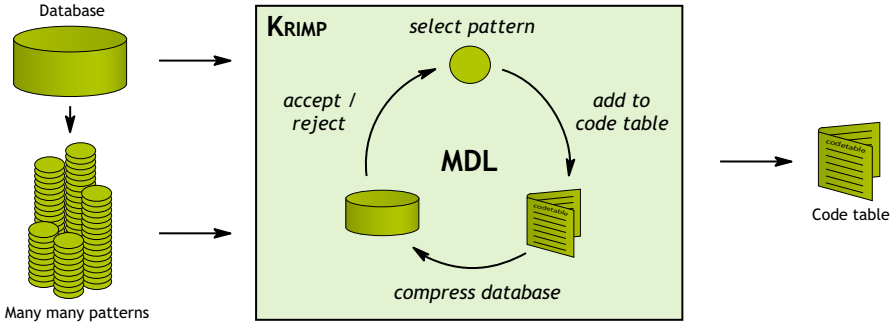


Figure 3.3: KRIMP in action [97].

Algorithm 5 The KRIMP Algorithm [97]

Input: A transaction dataset D and a candidate set $\mathcal{F}$, both over a set of items $\mathcal{I}$
Output: A code table CT

1: $CT \leftarrow$ **Standard Code Table**(D)
2: $\mathcal{F}_o \leftarrow \mathcal{F}$ in **Standard Candidate Order**
3: **for all** $F \in \mathcal{F}_o \setminus \mathcal{I}$ **do**
4: $\quad CT_c \leftarrow (CT \oplus F)$ in **Standard Cover Order**
5: $\quad$ **if** $L(D, CT_c) < L(D, CT)$ **then**
6: $\quad\quad CT \leftarrow CT_c$
7: **return** CT

often overfit (e.g. include complete rows or columns). Other cover-based approaches that try to remove redundant patterns from the selected pattern set include those of Bringmann and Zimmermann [19], Knobbe and Ho [44], Xiang et al. [103], and Yan et al. [104].

The MTV algorithm by Mampaey et al. [57] can be seen as an idealised version of KRIMP, as instead of estimating probabilities through a greedy cover function it calculates the complexity of the data under the pattern set using a Maximum Entropy model [43]. While this results in better estimates, it also renders the algorithm exponential in the size of the pattern set. Two approaches inspired by KRIMP that regard both the presence and absence of items in a transaction, i.e. both the zeros and the ones, are PACK by Tatti and Vreeken [89] and LESS by Heikinheimo et al. [42].

An improved version of KRIMP called SLIM, introduced by Smets and Vreeken [79], mines patterns directly from the data thereby skipping the expensive generation of all frequent patterns as a preprocessing step. Instead, in each step all current patterns are combined to form bigger patterns and the most promising one is added to the code table. To determine which candidate pattern $(X \cup Y)$ is most promising we estimate its gain when it would be added to the code table, denoted by $\Delta L(CT \oplus (X \cup Y), D)$.

More precisely, following the pseudo-code in Algorithm 6, SLIM works as follows. We

start with the singleton code table (line 1) and in every iteration we consider all pairwise combinations of $X, Y \in CT$ as candidates in **Gain Order**, i.e. descending on $\Delta L(CT \oplus (X \cup Y), D)$ (line 2). Iteratively, we add a candidate to CT and cover the data (line 3), and compute the total encoded size (line 4). If compression improves, we accept the candidate, otherwise we reject it. When accepted, we perform a post-pruning step (line 5) and update the candidate list. This process continues until there are no more candidates that decrease the total compressed size.

Algorithm 6 The SLIM Algorithm [79]

Input: A transaction dataset D over a set of items $\mathcal{I}$
Output: A code table CT

1: $CT \leftarrow$ **Standard Code Table**(D)
2: **for** $F \in \{X \cup Y : X, Y \in CT\}$ in **Gain Order do**
3: $\quad CT_c \leftarrow (CT \oplus F)$ in **Standard Cover Order**
4: $\quad$ **if** $L(D, CT_c) < L(D, CT)$ **then**
5: $\quad\quad CT \leftarrow$ *post-prune*(CT_c)
6: **return** CT

Summarising Event Sequences

Similar to the advancements in transaction datasets, summarisation of sequential data also developed from frequent pattern mining. Chapter 2 gives an overview of sequence mining and the different types of patterns that can be studied. As with all traditional pattern mining approaches, redundancy is also a key problem when mining frequent patterns in sequential data. To this end, Tatti and Vreeken [90] proposed instead to approximate the MDL-optimal summarisation of event sequence data using serial episodes. Their SQS algorithm deals with many challenges inherent to this type of data, such as the importance of the order of events and the possibility for patterns to allow gaps in their occurrences. Other methods exist, but those either do not consider [11, 58] or do not punish gaps [48].

Next, we describe how MDL can be applied to compress event sequences, whereafter we discuss the algorithms that quickly mine good summaries for this data.

MDL for Event Sequences

Recall from Chapter 2 that an event sequence dataset D comprises $|D|$ sequences $S \in D$ over an alphabet Ω. Note that in this chapter we only consider sequences without simultaneous events. That is, each sequence S contains $t(S)$ events $e \in \Omega$ ($t(\cdot)$ for the number of time steps), and the total number of events in the dataset is $t(D)$. As models, again, we consider code tables. However, these code tables contain two extra columns next to the columns containing the patterns and their codes. That is, the third and fourth column contain two additional pattern-dependent codes to identify the presence or absence of gaps in the occurrence of a

pattern in the data [90]. Given three different codes for each pattern X in code table CT, we refer to its pattern code as $code_p(X \mid CT)$, to its gap code as $code_g(X \mid CT)$, and to its no-gap or fill code as $code_f(X \mid CT)$. For readability, we omit CT whenever clear from context. The example in Figure 3.4 shows a dataset together with the singleton code table, here called CT_1, and another code table CT_2 containing two additional patterns P and Q. Note that singleton patterns cannot have gaps and therefore lack gap and fill codes. In the remainder we refer to all patterns in the first column of CT as PS_{CT} and to the non-singletons patterns as $\mathcal{P}_{CT}$ ($= PS_{CT} \setminus \Omega$), again omitting CT whenever clear from context.

The cover $\mathcal{C}$ of a dataset describes where we use which patterns from the code table to encode the dataset and where pattern instances contain gaps. Before we describe how to cover a dataset given a code table, we first describe how to encode it given a cover and how to decode an encoded dataset. Finally, we describe how to efficiently find high quality code tables that succinctly summarise the dataset.

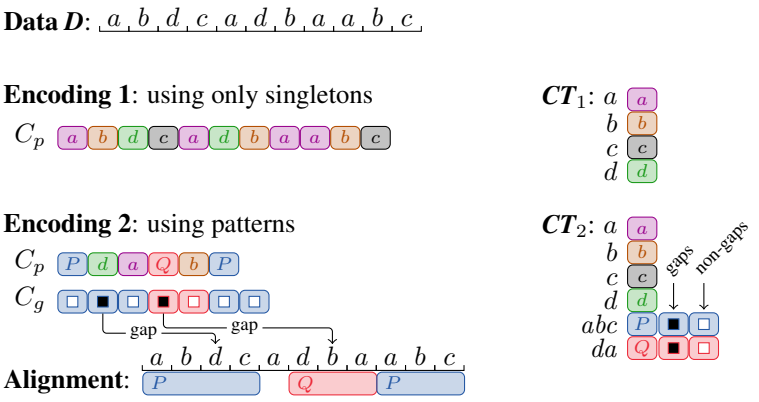


Figure 3.4: Toy example of two possible encodings. The first encoding uses only singletons. The second encoding uses singletons and the two patterns $P = \langle abc \rangle$ and $Q = \langle da \rangle$. [90]

Encoding

An encoded sequence dataset comprises two data streams, the pattern and gap stream, together describing the cover $\mathcal{C}$ of the data using the patterns from the code table. The pattern stream C_p defines where we use which patterns to describe the dataset and the gap stream C_g indicates when and where gaps in these pattern instances occur. These two streams are a result of the cover that is chosen to describe the dataset which we discuss in more depth later. To be able to determine the compressed size of the encoded dataset we need to compute the lengths of the used codes. The length of the optimal pattern code for a pattern X is similar to that in the

transaction domain, recall that we have

$$L(code_p(X \mid CT)) = -\log\left(\frac{usage(X)}{\sum_{Y \in PS} usage(Y)}\right) \quad .$$

The code lengths to describe the gaps and fills within a pattern occurrence can be computed similarly by their negative log-likelihood. For a pattern X we define the number of gap events in the encoding as $gaps(X)$. Further, the number of fill events is solely determined by the usage of pattern X in the encoding, i.e. $fills(X) = usage(X) \times (|X| - 1)$. Since these codes are pattern-dependent for each pattern X we have

$$L(code_g(X \mid CT)) = -\log\left(\frac{gaps(X)}{gaps(X) + fills(X)}\right) \quad ,$$

for its gap code length and

$$L(code_f(X \mid CT)) = -\log\left(\frac{fills(X)}{gaps(X) + fills(X)}\right) \quad ,$$

for its fill code length. Now to compute the encoded length of the code streams we simply sum over the lengths of the used codes, we thus have

$$L(C_p \mid CT) = \sum_{X \in PS} usage(X) \times L(code_p(X)) \quad ,$$

for the encoded length of the pattern stream. And for the encoded length of the gap stream we have

$$L(C_g \mid CT) = \sum_{X \in \mathcal{P}} \Big(gaps(X) \times L(code_g(X)) \; + \; fills(X) \times L(code_f(X))\Big) \quad .$$

To be able to decode or reconstruct the dataset from the two code streams we need to know the number and length of all sequences. Given this information we do not need any extra codes to mark the end of each sequence in the code streams. We encode these details using the MDL optimal Universal code for integers [35, 76], denoted by $L_{\mathbb{N}}$. This gives us the total encoded length of the dataset given a code table as follows

$$L(D \mid CT) = L_{\mathbb{N}}(|D|) + \sum_{S \in D} L_{\mathbb{N}}(t(S)) + L(C_p \mid CT) + L(C_g \mid CT) \quad .$$

To compute the encoded length of the code table Tatti and Vreeken [90] take a different approach than Vreeken et al. [97]. That is, they do not sum over the lengths of the codes in the code table, but they encode the minimal information needed to reconstruct it. The encoded length of the first column for each pattern X is the length of the pattern $|X|$ plus the encoded length of X given the singleton code table ST, that is

$$L_{\mathbb{N}}(|X|) + \sum_{x \in X} L(code_p(x \mid ST)) \quad .$$

To be able to use these ST code lengths we first need to encode the supports of all singletons. Since given the support of a singleton pattern Y we can easily determine its ST code length as the negative log-likelihood of Y in D

$$L(code_p(Y \mid ST)) = -\log \frac{support(Y)}{t(D)} \quad .$$

Instead of encoding all these supports separately we use a composition of the sum over all singleton supports, $t(D)$, into $|\Omega|$ terms. The total number of such compositions is given by $\binom{t(D)-1}{|\Omega|-1}$, and by taking the logarithm we have the number of bits needed to identify a single composition. We thus need

$$L_{\mathbb{N}}(|\Omega|) + \log \binom{t(D)-1}{|\Omega|-1} \quad ,$$

bits to encode all singleton supports, from which we can reconstruct ST.

To be able to reconstruct the code lengths in the second column of the code table we need to encode the usages of all patterns. Given the usages of all non-singletons patterns $\mathcal{P}$ and the singleton supports we already have from ST we can reconstruct the singleton usages. Therefore, we only encode the usages of $\mathcal{P}$, for which we need $L_{\mathbb{N}}(|\mathcal{P}|+1)$ bits for the number of non-singletons, $L_{\mathbb{N}}(usage(\mathcal{P})+1)$ bits for the sum over all their usages, and $\log \binom{usage(\mathcal{P})-1}{|\mathcal{P}|-1}$ bits for their individual usages. Because $|\mathcal{P}|$ and $usage(\mathcal{P})$ can be zero, for which $L_{\mathbb{N}}$ is not defined, we apply a +1 shift.

For the gap and fill codes in the last two columns we only need to encode the number of gaps per pattern (again applying a +1 shift when necessary). Since we already have the usage and length of each pattern, the number of fills follows automatically. As a result to compute the encoded length of the code table we have

$$\begin{aligned} L(CT \mid \mathcal{C}) = \; & L_{\mathbb{N}}(|\Omega|) + \log \binom{t(D)-1}{|\Omega|-1} \\ & + L_{\mathbb{N}}(|\mathcal{P}|+1) + L_{\mathbb{N}}(usage(\mathcal{P})+1) \\ & + \log \binom{usage(\mathcal{P})-1}{|\mathcal{P}|-1} + \sum_{X \in \mathcal{P}} L(X, CT) \quad , \end{aligned}$$

where $L(X, CT)$ is the number of bits to encode for each non-singleton pattern X: its length, its events, and its gaps in $\mathcal{C}$. That is, we have

$$L(X, CT) = L_{\mathbb{N}}(|X|) + \sum_{x \in X} L(code_p(x \mid ST)) + L_{\mathbb{N}}(gaps(X)+1) \quad .$$

Ultimately, to find a good summary, we are again after the cover $\mathcal{C}$ and code table CT that minimise the total encoded length of the dataset and the code table

$$L(CT, D) = L(CT \mid \mathcal{C}) + L(D \mid CT) \quad .$$

Decoding

Given an encoded dataset it is easy to recover the original dataset by alternatively reading codes from the pattern and gap stream. This works as follows, first we read a code $code_p(X)$ from the pattern stream to find the first event $x_1 \in X$ that we can append to the decoded dataset D_{dec}. If pattern X is a singleton we read the next code from the pattern stream, otherwise we read a code from the gap stream. If we read a fill code $code_f(X)$ then we can append the second event $x_2 \in X$ to D_{dec}. If we read a gap code $code_g(X)$ then we have to read another code from the pattern stream, e.g. $code_p(Y)$, to fill this gap with the first event $y_1 \in Y$ from this new pattern. Thereafter, if we have not completely decoded pattern X, we first read from the gapstream for pattern X and afterwards for pattern Y. We continue in this fashion until we have read all codes from both streams, that is when $D_{dec} = D$.

As an example consider the dataset D, Encoding 2 and CT_2 in Figure 3.4. To decode this dataset we start with reading pattern code $code_p(P)$ from C_p after which we can append event a to the decoded dataset D_{dec}. We then read fill code $code_f(P)$ from C_g and append event b to D_{dec}. Subsequently, we read gap code $code_g(P)$ from C_g and pattern code $code_p(d)$ from C_p, after which we append event d to D_{dec}. We finish the decoding of pattern P after reading fill code $code_f(P)$ from C_g and adding event c to D_{dec}. We then continue to read from the pattern and gap stream as from the start to decode the rest of the dataset.

Covering

Opposed to the decoding of a dataset, the cover (which determines the encoding) is not unambiguous, since we can often use multiple patterns to describe the same event. A window is defined as an interval in the data in which a pattern occurs. An optimal cover will only contain minimal windows, which are windows that cannot be shortened while still containing the pattern [90]. An alignment is the set of disjoint minimal windows for all non-singleton patterns used in the cover of the dataset. All gap events between and within these windows are covered by singletons. Note that pattern occurrences are not allowed to interleave in an alignment. Given such an alignment we can determine the usages for all pattern and gap codes to find the corresponding optimal code table. We do, however, also need to find the optimal alignment minimising $L(CT, D)$. When we fix the code lengths in the code table we can find the corresponding optimal alignment by a simple dynamic program called ALIGN (see [90] for more details), which takes as input all minimal windows sorted on starting time step. Given this alignment we can update the code lengths to be optimal again corresponding to the new alignment and we can repeat this process until it converges to a local optimum in finite time.

In Algorithm 7 we give the pseudocode for the SQS algorithm, which takes as input a dataset and a set of patterns and outputs an alignment. We start by setting the singleton usages equal to their supports (line 1-2). Then for all non-singletons we find their minimal windows by FINDWINDOWS (see [88]) and set their usage and number of gaps (line 3-6). In line 7 we mergesort all these minimal windows whereafter we start the loop of finding optimal alignments and code lengths (line 8-11) which ends after convergence by returning an alignment (line 12).

Algorithm 7 Summarising event seQuenceS with the SQS algorithm [90]

Input: Database of sequences D and set of patterns $\mathcal{P}$
Output: Alignment A

1: **for** $s \in \Omega$ **do**
2: $usage(s) \leftarrow support(s \mid D)$
3: **for** $X \in \mathcal{P}, |X| > 1$ **do**
4: $W_X \leftarrow$ FINDWINDOWS(X, D)
5: $usage(X) \leftarrow |W_X|$
6: $gaps(X) \leftarrow |X| - 1$
7: $W \leftarrow$ merge sort $\{W_X\}_{X \in \mathcal{P}}$ based on first event
8: **while** changes **do**
9: compute gain for each $w \in W$
10: $A \leftarrow$ ALIGN(W)
11: recompute usage and gaps from A
12: **return** A

Mining Summaries

We now know how to score a code table but how do we select the set of patterns that minimise $L(CT, D)$? Again, we face an enormous search space. The first approach introduced by Tatti and Vreeken is SQS-CANDIDATES and resembles the ideas of KRIMP. As input SQS-CANDIDATES takes a candidate set of patterns, e.g. all frequent or closed patterns, and it outputs a small set of patterns that together succinctly describe the dataset. First the candidate patterns are ordered based on their individual gain in compression (line 1). We then iteratively loop through the ordered candidate patterns (line 3) and check if adding them to the result set improves compression (line 4). If a pattern is accepted and at the end of the loop we perform a pruning step (line 5-6). The algorithm ends by ordering the result set by the contribution in compression for each pattern (line 7) and returning a pattern set (line 8).

Algorithm 8 The SQS-CANDIDATES algorithm [90]

Input: Database of sequences D and candidate patterns $\mathcal{F}$
Output: set of non-singleton patterns $\mathcal{P}$ that heuristically minimise $L(CT, D)$

1: order patterns $X \in \mathcal{F}$ based on $L(D, \{X\})$
2: $\mathcal{P} \leftarrow \emptyset$
3: **for** $X \in \mathcal{F}$ in order **do**
4: **if** $L(D, \mathcal{P} \cup X) < L(D, \mathcal{P})$ **then**
5: $\mathcal{P} \leftarrow$ PRUNE$(\mathcal{P} \cup X, D,$ **false**$)$
6: $\mathcal{P} \leftarrow$ PRUNE$(\mathcal{P}, D,$ **true**$)$
7: order patterns $X \in \mathcal{P}$ by $L(D, \mathcal{P}) - L(D, \mathcal{P} \setminus X)$
8: **return** $\mathcal{P}$

Tatti and Vreeken also introduced SQS-SEARCH that mines patterns directly from the data without the need to first mine a candidate pattern set, similar to the SLIM algorithm.

CHAPTER 4

Characterising Seismic Data

When a seismologist analyses a new seismogram it is often useful to have access to a set of similar seismograms. For example if she tries to determine the event, if any, that caused the particular readings on her seismogram. So, the question is: when are two seismograms similar?

To define such a notion of similarity, we first preprocess the seismogram by a wavelet decomposition, followed by a discretisation of the wavelet coefficients. Next we introduce a new type of patterns on the resulting set of aligned symbolic time series. These patterns, called block patterns, satisfy an Apriori property and can thus be found with a levelwise search. Next we use MDL to define when a set of such patterns is characteristic for the data. We introduce the MULTI-KRIMP algorithm to find such *code sets*.

In experiments we show that these code sets are both good at distinguishing between dissimilar seismograms and good at recognizing similar seismograms. Moreover, we show how such a code set can be used to generate a synthetic seismogram that shows what all seismograms in a cluster have in common.[1]

[1] This work was originally published as [13]:
R. Bertens, J. Vreeken, and A. Siebes. Characterising Seismic Data. *SDM'14*, SIAM.

4.1 Introduction

One of the goals of seismology is to detect and understand the sources and the causes – e.g., earthquakes, volcano eruptions or (man made) explosions – of seismic waves that travel through the earth. One of the main tools for this is the seismometer which produces recordings of earth motion at its location, as a function of time in a so-called seismogram. The type and the location of a (seismic) event is determined by analysing and combining seismograms from multiple seismometers. To a large extend, this identification is still manual labour by a seismologist.

There are large collections of explicitly and/or implicitly labelled seismograms, produced by seismometers all over the globe, of previous events that have been identified. So, it is natural to wonder whether or not the task of the seismologist can be simplified by giving her access to similar, identified, seismograms. That is, given a new, unidentified, seismogram, return a set of similar, identified, seismograms, that help identifying the new seismogram.

To explain what we mean by this we have to make precise what we mean by both "similar seismograms" and by "help identifying". Similar seismograms does *not* mean that their graphs should be (almost) identical. Firstly, because for our purposes seismograms are inherently noisy. That is, the graph is the weighted sum of all events that make the earth move at the location of the seismometer, from the intended event – such as an earthquake – to passing trucks and nearby road-works.

Secondly, even if we would have a clean signal, many of its characteristics depend on much more than just the event. For example, the amplitude measured depends not only on the size of the event, but also on the distance between the event and the location of the seismometer. In fact, many aspects of the graph depend, among others, on the composition of the earth's crust between the event and the seismometer.

The "noise" problem means that we have to somehow clean the seismogram, i.e, we have to filter the intended signal from the noisy graph. Fortunately, it is well known in seismology that the signal in the range from roughly 4 Hz to $1/4$ Hz has a fairly good signal to noise ratio [7, 74]. That is, signals in that range are predominately of seismic origin. To decompose a seismogram, we use the discrete Haar wavelet and discard all components outside the 4 Hz to $1/4$ range. Note that we use a wavelet decomposition rather than a Fourier decomposition because seismic events are limited in time.

This decomposition gives us a time series of multi-level detail coefficients. Since these coefficients still represent characteristics such as the amplitude of the original signal, we next discretise the wavelet coefficients. That is, we retain that the signal goes up or down, but we do not retain exactly by how much. Since the range of the wavelet coefficients will in general differ for the different frequency levels, we discretise each level separately. The discretisation is based on MDL histogram density estimation [47]; a method which finds the MDL-optimal bin count and cut point locations and is known to have good characteristics.

Preprocessing the data, the details of which are given in Section 4.2, transforms the original seismogram in a set of aligned categorical time series. Hence the question of when two seismograms are similar is transformed in the question when two such sets are similar.

Our answer is simple: when they exhibit similar patterns. The question is then, of course, which patterns?

To answer this question recall that our goal is that the retrieved identified seismograms should help in identifying the new seismogram. When can seismogram x help in identifying seismogram y? Clearly, if knowing seismogram x helps in predicting what will happen next in seismogram y. The more accurate x helps us in predicting what happens next in y, the more similar and useful it is.

Hence, the patterns we are interested in should be such predictive patterns. For example, assume that we have a symbol a on level l at time t. To predict a, physics tells us [74] we can use a "block" of symbols on all levels at all time points prior to t provided this block contains no holes and it is adjacent to a. Clearly these patterns satisfy an Apriori property and thus can be found with a standard levelwise algorithm.

As for (almost) any kind of pattern, the number of these patterns will quickly explode and most patterns will not be very descriptive of the (transformed) seismogram. To select a small set of descriptive patterns, we use the Minimum Description Length principle (Chapter 3). That is, similar to the KRIMP algorithm [77], we encode the (transformed) seismogram with a set of patterns. The better a set of patterns compresses the data, the better they (collectively) describe the data. Finding an optimal set of patterns is again intractable and, hence, a heuristic algorithm called MULTI-KRIMP is used. Note that since the behaviour of the seismogram on different frequency levels can be different, *code sets* are computed for each level separately. The details of both the patterns, their discovery and the MULTI-KRIMP algorithm are given in Section 4.3.

Our claim is now that similar seismograms are seismograms that yield similar code sets. That is, if x and y are similar seismograms, then compressing x with the code set computed from y should be almost as good as compressing it with its own code set and vice versa, of course. Moreover, we claim that if the seismograms are similar – i.e., they compress each other well – their identification is similar – i.e., they indicate similar seismic events.

Since the classification of a seismic event by a seismogram is not a mathematically defined property – in which case devising algorithms for the task would have been easy – we can not verify our claims formally. Hence, we use experiments to substantiate our claims. But before we describe these experiments, related work is first discussed in Section 4.4. Most notably *shapelets* [107] are discussed there. Not only because they are a well-known technique in time series analysis, but also because they are suitable for part of what we aim to achieve.

To substantiate our claims, we use data from different seismic events and seismometers at different locations. We show that our technique clusters the events correctly regardless of the location (and size) of the event and the location of the seismometers. Moreover, we illustrate visually that our technique captures the shape of a seismogram. For, given a code set, we can generate artificial seismograms. By plotting both the real seismogram and a generated one, one can see that they are similar. The details are given in Section 4.5. The discussion of these results is given in Section 4.6. Finally, the conclusions are formulated in Section 4.7.

4.2 Preprocessing the Data

Seismograms can be seen as functions from time to the real numbers, or more formally, $S : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$. In reality, of course, the signal at a location is only sampled at some frequency rather than measured continuously.

In the Introduction of this chapter we already explained why we cannot use S directly, it is noisy and characteristics such as the amplitude should be removed. Hence, S is first decomposed using a wavelet transform and subsequently the coefficients are further discretised. Both steps are discussed in detail in this Section.

Wavelet Decomposition

There are a number of techniques to decompose a function in frequency components, such as the Fourier decomposition and wavelet decompositions [30]. The advantage of a wavelet decomposition over the Fourier decomposition is that wavelets are localised and, thus, better at describing local behaviour in the signal. Since seismic events are by nature local events in the time series, we use a wavelet transform rather than Fourier analysis.

Formally, a wavelet transformation is the convolution, i.e., an inner product, of the function f with a scaled (parameter s) and translated (parameter b) wavelet ϕ:

$$Wf(s,b) = \langle f, \phi_{s,b} \rangle = \frac{1}{s} \int f(x) \phi \left(\frac{x-b}{s} \right) dx$$

A wavelet decomposition is computed by a set of wavelet transforms. Since we have sampled data, we use a discrete wavelet transform and in that case the decomposition is computed as follows. We assume that we have (a window of) data with 2^N data samples $f(t_1), \ldots f(t_{2^N})$ of f, (with time normalised such that $[t_1, t_{2^N}] = [0, 1]$). The discrete wavelet decomposition is now given by:

$$f = f^0 + \sum_{m=0}^{N} \sum_{l=0}^{2^m} \langle f, \phi_{m,l} \rangle \phi_{m,l}$$

where f^0 is the coarsest approximation of the time series.

This decomposition allows us to built a ladder of approximations to f given by:

$$f^{j-1} = f^j + \sum_{k=0}^{2^j} \langle f, \phi_{j,k} \rangle \phi_{j,k}.$$

This ladder gives us two sets of coefficients, the approximation coefficients (the f^j) and the detail coefficients ($f^{j-1} - f^j$). The detail coefficients encode the local behaviour of the time series at level j. Hence, these are the coefficients we will be using.

Each wavelet family has different properties, bringing out different aspects of the time series. Since we are interested in the shape of the graph, we use the Discrete Haar wavelet:

$$\phi(x) = \begin{cases} 1 & \text{for } 0 \leq x < 0.5 \\ -1 & \text{for } 0.5 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Using the Haar wavelet the approximation coefficients are averages on progressively smaller subsets of (a window of) the time series, while the detail coefficients are the local deviations from those averages.

To be more concrete, consider the toy example of a time series and its coefficients from Table 4.1.

Table 4.1: Toy example of a time series and its coefficients.

Decomp. level	**Detail coef.**	**Approx. coef.**
1^{st} level	1 -1 -1 1	8 4
2^{nd} level	2 -2	6

Original signal
9 7 3 5

The average of 9 and 7 is 8, and the deviation of 9 from this average is $9 - 8 = 1$. Similarly, the average of 3 and 5 is 4 and the deviation of 3 from this average is 3 - 4 = -1. Finally, the average of 8 and 4 is 6 and the deviation of 4 from this average is 4 - 6 = -2. Note that each original data point can be reconstructed using the coarsest approximation, the overall average, 6 and a sequence of detail coefficients, e.g., for the rightmost point we add the rightmost coefficients: $6 - 2 + 1 = 5$.

Further note that for technical reasons the actual coefficients at each level should be multiplied by $\sqrt{2}$, but that does not concern us here.

Given a seismogram S which may have any length, we compute the wavelet coefficients using a sliding window w, which has some dyadic (power of 2) length. Note that this implies that we do not compute coefficients for the first $|w| - 1$ data points.

Assume that w starts at position 1 in S and ends at $|w|$. We then compute the detail coefficients from the Haar decomposition of $S_1, \ldots, S_{|w|}$ as indicated above. We also noted above that it is the right most set of detail coefficients that tell something about the local behaviour of S at $S_{|w|}$. Hence we start our aligned (transformed) time series with these rightmost detail coefficients. Next we shift the window by a step (for us, always a step of size 1) and repeat the procedure to compute the next set of coefficients for $S_{|w|+1}$.

As an example, consider the time series from Table 4.2. If we use a window size of 8

Table 4.2: Example time series.

1	1	4	4	4	8	1	4	7	7	9

and a step size of 1, we have four windows on this signal. For real seismic data, we are only interested in the detail coefficients that represent the frequencies in or close to the interval 4 Hz to $1/4$ Hz, as discussed in the Introduction. Here, however, we simply retain all detail coefficients. Hence, our time series – which allows for four windows – is transformed to the set of aligned time series from Table 4.3.

Table 4.3: Transformed set of aligned time series for the example from Table 4.2.

Window 1	**Window 2**	**Window 3**	**Window 4**
-2.12	-2.12	0	-1.41
3.5	-1	-4.5	-2.5
-2.47	-2.47	0.35	-3.54

Computing all coefficients for all windows is wasteful, however optimising this is not the focus of this chapter.

Discretisation

Together with the coarsest approximation coefficient the detail coefficients are sufficient to reconstruct the original time series. As already stated, we will not use all detail coefficients and neither will we use the coarsest approximation coefficient, but this observation means that the detail coefficients are too detailed for our purposes. We are interested in the general shape of the graph, not in its exact values. That is, we are interested in whether it is ascending or descending at various frequency levels. We may even be interested in whether it is ascending steeply or barely on a given frequency level, but we are not interested in how steeply it is rising exactly.

Hence, we discretise the detail coefficients we just computed as the next preprocessing step. Since the spread of the coefficients may very well be different for each of the levels, we discretise the levels separately. But, of course, we use the same discretisation for all seismograms.

There are many techniques to discretise data, each with its own strong and weak points. For this chapter, we use one based on the Minimum Description Length principle (Chapter 3). More in particular, we use MDL histogram density estimation [47]. This method finds the MDL-optimal number of bins and cut point locations automatically.

For each level l we have a set of symbols – alphabet – A_l, which has one symbol for each bin the discretisation produces for l. Each value on level l of the aligned time series, produced by the wavelet transformation, is replaced by its corresponding element from A_l. In the end our original time series S is thus transformed in a set of aligned symbolic time series.

4.3 Patterns and Code Sets

Given the preprocessed data – the set of aligned symbolic time series – we now have to discover a small set of characteristic patterns for each level of the preprocessed data. The definition and discovery of the patterns is a standard pattern mining problem. For the second step, the discovery of a small set of *characteristic* patterns, we use MDL.

Patterns

As noted in the Introduction, our predictive pattern occurrences should have no holes across either the time or the level axes. For our patterns that means the following.

A set of sequences $X = \{x_l, \ldots, x_m\}$ is a *block pattern* over the level alphabets $A_1, \ldots, A_n$ iff

- For each $j \in \{l, \ldots, m\}$, x_j is made from symbols in A_j.
- $\{l, \ldots, m\}$ is a consecutive subset of $\{1, \ldots, n\}$.

The second requirement goes a long way to ensure that we get no holes in occurrences, but we need one more requirement to ensure it properly. This is a requirement on what constitutes an occurrence of X.

Let S be a set of aligned time series over the level alphabets $A_1, \ldots, A_n$ and $X = \{x_l, \ldots, x_m\}$ a block pattern over those same alphabets. X occurs in S at time t iff for every $x_k \in X$ there is a consecutive subsequence of S at level k, $[S_k[t - |x_k|], \ldots, S_k[t]]$ such that:

$$\forall j \in \{1, \ldots, |x_k|\} : x_k[j] = S_k[t - |x_k| + j]$$

That is the pattern sequences should, of course, match their respective elements in S exactly and all the pattern sequences occurrences should end at the same time t.

To give an example, assume that in our example transformed time series from Section 2.1, the number -2.1 is replaced by the abstract symbol -2.1 and so on (in other words, assume for a moment that the numbers are labels). Then the patterns 1 and 2 from Table 4.4 occur and the patterns 3 and 4 do not; their only potential occurrences exhibit holes.

Table 4.4: Example patterns.

Pattern 1		**Pattern 2**		**Pattern 3**		**Pattern 4**
-2.12	0	-1	-4.5	-1	-2.5	-1.41
	-4.5	-2.47	0.35			-3.54

With these definitions, the support of X in S is defined in the usual way, viz., its number of occurrences. Moreover it is clear that block patterns satisfy an Apriori principle: if X_1 is a sub-pattern of X_2, the support of X_1 will be at least as big as that of X_2. Hence, all frequent block patterns in S can be discovered with levelwise search [37].

Code Sets

As usual in pattern mining, if we set our threshold low, there are enormous numbers of frequent block patterns. How do we choose which ones are characteristic for a (preprocessed) seismogram? The first clue is that, as noted in the Introduction, our patterns should be predictive. That is, if a pattern occurs at time t in[2] S it should help us in predicting what happens at time $t+1$ in S.

Next observe that this gives us a way to encode S. Let X be some one level pattern, i.e., $X = \{x_l\}$. Furthermore, assume that we observe that if X occurs in S at a time t, $S_l[t+1]$ is either the symbol a or the symbol b with probability p_a and p_b respectively. To encode, or compress, S we could now replace a's and b's that occur right after X in S with a codeword of length $-\log(p_a)$ and $-\log(p_b)$ respectively, [35].

Clearly, it is slightly more complicated as there will be many patterns that are followed by an a and patterns will span multiple levels. But, the main idea that we can use patterns to encode the data is obviously valid. And that gives our second clue to how to choose: we can use the Minimum Description Length principle (Chapter 3).

Given that each level S_l of S has its own alphabet A_l, we encode each level separately. The simplest way to encode S is by disregarding all patterns and simply use a (prefix) code that reflects how often each $a \in A_l$ occurs in S_l. That is, we give it a code of length $-\log P(a \mid S_l)$. This is what we call the standard encoding ST_l of S_l and by doing this on all levels we have the standard encoding ST of S.

Let $\mathcal{X}^S$ be the set of all frequent block patterns on S. A $X \in \mathcal{X}^S$ is said to be a level l pattern if one of the sequences in X is built from A_l. The set of all frequent level l patterns in S is denoted by $\mathcal{X}_l^S$. We will simply write $\mathcal{X}$ and $\mathcal{X}_l$ if S is clear from the context.

To simplify the remainder of this section, and indeed the rest of this chapter, we augment each $\mathcal{X}_l$ with the special pattern $\emptyset$, which matches every element of S_l and even no element at all.

A *covering set* C_l for S_l is an ordered subset of $\mathcal{X}_l$ which contains $\emptyset$ as its last element. The *cover* of S by C_l is again a time series, denoted by $C_l(S_l)$, in which

> $C_l(S_l)[t]$ is the first element of C_l that occurs at time $t-1$ if $t > 1$, otherwise $C_l(S_l)[t] = \emptyset$.

To turn a cover of S_l into an encoding of S_l, we augment each element of a covering set C_l with a *code table* (Chapter 3). This code table consists of two columns. The first contains the elements of A_l in some order. The second contains a code word from some prefix code $\mathcal{C}_{C_l}$. Since we want to compress S_l, $\mathcal{C}_{C_l}$ has to be optimal for this compression. This is determined as follows.

For $a \in A_l$ and $X \in C_l$, define:

$$usage(a \mid X) = |\{t \mid S_l[t] = a \wedge C_l(S_l)[t] = X\}| + 1$$

[2]We will use S both to denote the original and the preprocessed time series.

which gives us:

$$P(a \mid X) = \frac{usage(a \mid X)}{\sum_{b \in A_l} usage(b \mid X)}$$

And thus, the code $\mathcal{C}_{C_l}$ should assign to a in the code table of $X \in C_l$ has length $-\log(P(a \mid X))$.

A *level code set* CS_l is a covering set C_l for S_l in which each pattern in C_l is augmented with a code table for A_l with the optimal codes as constructed above. A *code set* CS for S is simply a set of level code sets, one for each level S_l in S.

Coding S_l with CS_l is simple. First compute $C_l(S_l)$, and then replace $S_l[t]$ by its code as given in CS_l in the code table of $C_l(S_l)[t]$. Note that the standard encoding ST_l of S_l that we introduced above is simply the encoding induced by the covering set $C_l = \{\emptyset\}$ plus a Laplace correction. From now on, we use this Laplace corrected version as the standard encoding.

The encoded size of the data given CS_l, denoted by $L(S_l \mid CS_l)$ is now simply the sum of the code lengths of the codes in the encoded string.

To find the optimal encoding for S_l according to the MDL principle, we also have to determine the size of CS_l. This is determined as follows:

- For each of the codes in the code tables we have a length
- For each of the elements of A_l in those code tables we use the standard encoding ST_l
- Each of the patterns X is also encoded with the standard encoding. Each sequence in X is, of course, encoded by the standard encoding at the appropriate level.

By summing all these encoded lengths, we get the total encoded size of the model, denoted by $L(CS_l \mid S)$.

MDL now tells us that we need to find the code set CS_l such that

$$l(CS_l, S_l) = L(CS_l \mid S) + L(S_l \mid CS_l)$$

is minimised. Unfortunately, as explained in Chapter 3, this is an intractable problem. Firstly because of the order used in covering, every permutation will lead to another compressed size. Secondly because adding a pattern to the level code set may both increase and decrease the compressed size. In other words, there is no structure in the search space that allows us to prune large parts of it. Hence we need to resort to heuristics.

MuLTi-Krimp

We adapt the heuristics used in Chapter 3, if only because they proved to work well. The first heuristic is that we define the order of the patterns in a level code set. These patterns are assumed to be ordered by the **Standard Cover Order**. Which is descending on cardinality first, descending on support second, ascending on height third, descending on top-level-length fourth, and last lexicographically ascending to make it a total order.

Secondly, we use a simple greedy algorithm to find good level code sets, called MULTI-KRIMP, the pseudo-code is given in Algorithm 9. As input it takes the (preprocessed) time series S, a level l and the frequent block patterns X_l.

We start with the level code set containing only the empty set (1). We loop through all frequent patterns in **Standard Candidate Order** (like the Standard Cover Order, only sorted on support before cardinality) (2); we then add each pattern to our code set (3) and test if this new code set compresses our data better than before (4). If it does compress better, we keep the pattern and consider if other patterns in our code set can be pruned (5). After we have considered all patterns, we return the final level code set (8).

Algorithm 9 The MULTI-KRIMP Algorithm

Input: A preprocessed seismogram S, a level l, and a set of frequent patterns for level l, $\mathcal{X}_l$.
Output: A level CS_l

1: $CS_l \leftarrow \{\emptyset\}$
2: **for** $X \in \mathcal{X}_l$ in **Standard Candidate Order do**
3: $\quad CS_l^c \leftarrow (CS_l \oplus X)$ in **Standard Cover Order**
4: $\quad$ **if** $L(CS_l^c, S_l) < L(CS_l, S_l)$ **then**
5: $\quad\quad CS_l \leftarrow$ *post-prune*(CS_l^c)
6: **return** CT_l

4.4 Related Work

Our use of wavelets to decompose a time series is, of course, far from unique. In fact, we have used the discrete Haar wavelet ourselves before in [83]. An overview of all possible ways to decompose time series data is far beyond the scope of this chapter. For more on wavelets, we refer to [30]. Also for the discretisation of real valued data there are far too many techniques to even attempt an overview here. We chose MDL histogram density estimation [47] because it finds both the number of bins and the bins themselves automatically in a well-founded manner. Moreover, it is known to work well in practice.

Frequent pattern mining in time series data is useful for tasks such as clustering, classification, prediction, and many more. All these fields have attracted much research from the data mining community. Our block patterns are somewhat unusual in that they span multiple frequency scales of a decomposed and discretised time series, but mining them is completely standard. For a brief overview of frequent pattern mining we refer to [37].

Techniques for clustering time series data fall mainly in three groups: those that work directly with the raw data, those that work with features extracted from the raw data, and those that work with models build from the data; an overview of time series clustering can be found in [54]. Our work fits in the intersection of the second and third category, because we cluster data samples based on their size when compressed with a range of code sets computed on discretised wavelet coefficients of the original data.

A good example of a time series clustering approach that works directly on the raw data is [107]. It uses *shapelets*, which are time series snippets characteristic for a certain cluster of data. Time series chunks are clustered using the distance to a shapelet, rather than the distance to the nearest neighbour chunk. This technique has proven to work very well in many different domains. Unfortunately, for our data – which can be very noisy and repetitive – it did not always do very well; for more details see the experiments and the discussion thereof.

The way we use MDL to identify characteristic patterns is, of course, reminiscent of the work on KRIMP, described in Chapter 3. While the high-level approach here is similar to that, many aspects are very different. Probably the biggest difference is that the current encoding is completely different. Here we compress for predictive capabilities rather than for descriptive ones. This means firstly, that the patterns we use to cover a value $S_l[t]$ do *not* contain $S_l[t]$ itself. Rather a pattern describes the behaviour of S just before $S[t]$. Secondly, it means that the code lengths are determined by conditional probabilities rather than by simple relative occurrence frequencies. Third and finally, it means that patterns cover exactly a single value $S_l[t]$ and never a larger subset of a preprocessed time series.

Application-wise, the Dynamic Bayesian Network (DBN) approach to seismic data classification in [74] is closely related to our research. They also decompose the signal using wavelets – albeit a different one: the Morlet wavelet – and then use a DBN to classify the incoming data as either "Earthquake" or "Noise". The most important difference is that we do not need pre-defined classes. Using MULTI-KRIMP we can both determine which clusters there are in the data and classify new data as belonging to any of these clusters or being something not seen before.

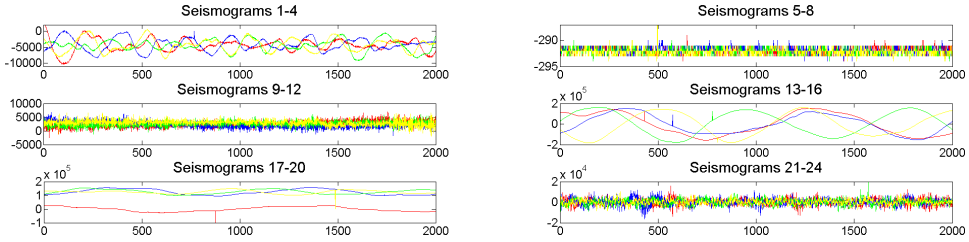


Figure 4.1: Six clusters, all containing four seismograms.

4.5 Experiments

To evaluate the code sets produced by MULTI-KRIMP experimentally, we perform three (sets of) experiments. Firstly to show their ability to distinguish between different types of seismograms. secondly to show that they can be used to identify similar seismograms. Finally we show that code sets can be used to generate synthetic seismograms and that these generated seismograms are visually similar to the seismogram the code set was computed from.

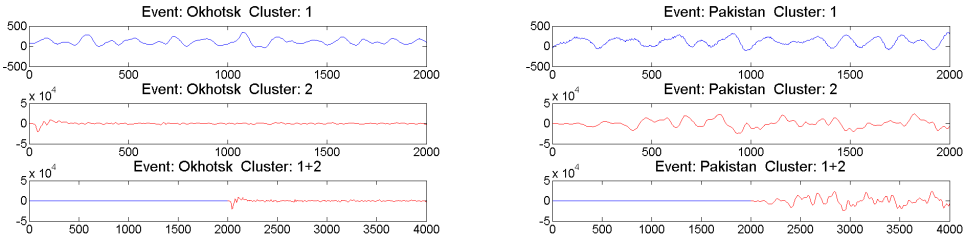


Figure 4.2: Seismograms from the Brant station for both events.

For reproducibility and to stimulate future research, both the code and the datasets are available through the first author.

Setup

For the wavelet transformation we use a step size of 1 and a window of 256 data points. Since all the seismograms used are sampled at 40 Hz, we use the levels 4 (corresponding to 2.5 Hz) to 8 (corresponding to 0.16 Hz). As noted before, the discretisation requires no parameters.

Since experiments show that larger patterns are hardly, if ever, used, we limit the patterns we mine to those that span across at most three frequency levels and two time points.

Datasets

In cooperation with a domain expert, we manually gathered seismograms from the publicly available ORFEUS POND data repository[3].

For the first experiments we collected seismograms of 2000 data points with varying frequencies and amplitudes. It serves to show the ability our method has to distinguish between only slightly different seismograms. It consists of seismograms at various moments and at different locations. We manually distinguished six different clusters, all containing four very similar seismograms, see Figure 4.1.

For the second experiment we gathered seismograms from two different events both measured at the same 12 stations. The first was a quake in the sea of Okhotsk of magnitude 6.4, the second was a quake in Pakistan of magnitude 7.7. We split each of these seismograms into 2000 data points before the quake and 2000 beginning at the start of the quake. That is, for each event we have two clusters of seismograms, Cluster 1: no event, Cluster 2: quake.

Data from both events are plotted for one station in Figure 4.2. Due to space limitations, the data from all other stations can be found in [14]. The seismograms from the event at Okhotsk in cluster 1 are numbered 1-12 and those in cluster 2 are numbered 13-24. For the event in Pakistan the seismograms 25-36 are in cluster 1 and 37-48 in cluster 2. In Figure 4.3 the locations of the two events and of all stations involved are given.

[3] ftp://www.orfeus-eu.org/pub/data/POND/

Figure 4.3: The location of the events and the stations.

The Power to Distinguish

Using MULTI-KRIMP we build a code set for each of the seismograms from the first dataset. Then each seismogram was encoded with the code set of every other seismogram. This gave a table in which a row indicates a seismogram and a column a code set of one of the seismograms. In this table, each cell contained the size of the corresponding seismogram compressed with the corresponding code set. This table was used to hierarchically cluster the seismograms using single linkage and the Spearman metric. The resulting dendrogram is shown in Figure 4.4. The numbers on the x-axis represent seismograms, where every four subsequent numbers (1-4, 5-8, 9-12, 13-16, 17-20, 21-24) represent seismograms that we manually identified to be very similar, see also Figure 4.1. Note that all seismograms are clustered as we would expect, the distance between seismograms which are alike is relatively small compared to the distance to other seismograms. Clustering the seismograms using shapelets and k-means clustering gave rather less convincing results.

To further substantiate our claim, we also did a leave-one out validation. That is for each time series, we clustered the remaining 23 time series manually, computed a code set for each of the clusters and determined the "right cluster" for the left out time series by choosing the cluster whose code set compressed it most. Each of the seismograms was assigned to its own cluster. This confirms our claim that code sets are good in distinguishing between different seismograms.

Identifying Similar Seismograms

With the second dataset – the seismograms related to the two quakes – we first redid the first experiment. That is we again build a table of the respective compressed sizes and performed a clustering based on that table, see Figure 4.5. All seismograms from cluster 1 are easily distinguished from cluster 2. However, there are four mistakes; the seismograms 2, 4, 26

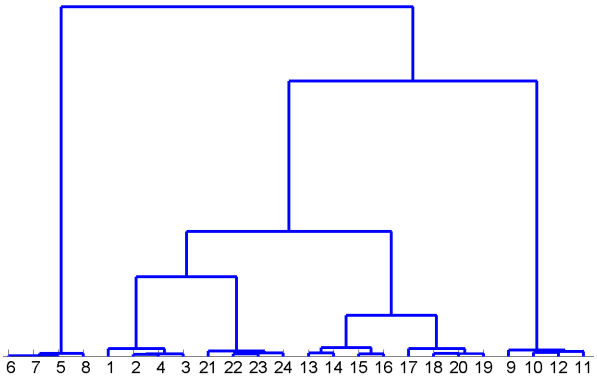


Figure 4.4: A dendrogram clearly representing the six clusters.

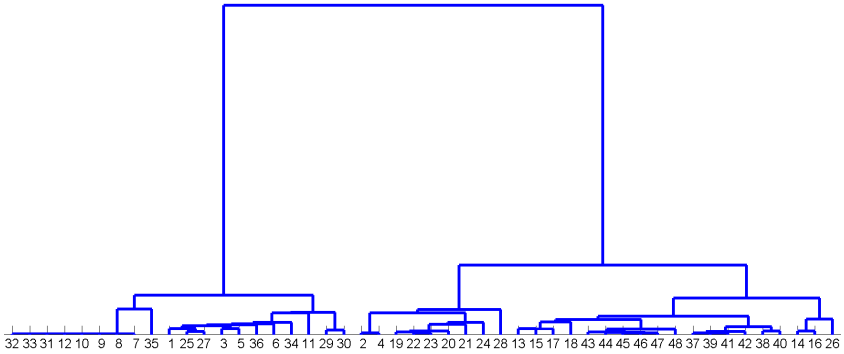


Figure 4.5: A dendrogram for all seismograms from two clusters of two events.

and 28 from cluster 1 are clustered with the data from cluster 2. This can be explained by the fact that these seismograms show a sizeable amount of movement, as can be seen in [14]. Experiments using shapelets gave comparable results.

Next we assigned all seismograms manually to three clusters, no event, quake in the sea of Okhotsk and quake in Pakistan respectively. Then we performed again a leave one out validation. This gave near perfect results. Only one non-event, seismogram 26 (which shows significant movement) was assigned to the event in Okhotsk. Moreover, two seismograms from the event at Okhotsk are clustered with the event in Pakistan, to which they apparently are more similar. Note that from the point of view of identification only the mistake with seismogram 26 is a real mistake.

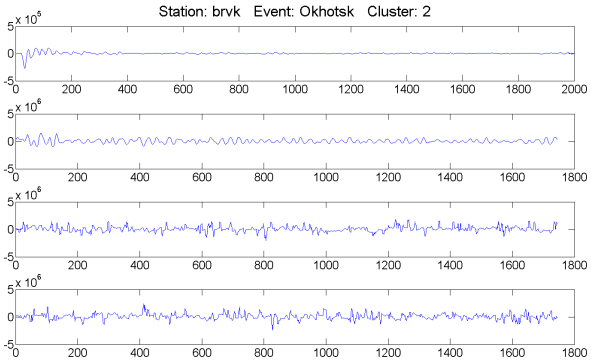


Figure 4.6: Visualisation of the code set for a seismogram.

Generating Seismograms

Given that the patterns in a code set predict what is going to happen next, we can use a code set CS to generate a seismogram. Recall that $\emptyset$ matches anything – including nothing – so, for $t = 1$ we choose random symbols drawn with probabilities according to its (i.e., $\emptyset$) code table on each level. For each next step, on each level, we use its level code set to determine which pattern matches the preceding time series and draw a new symbol according to the code table of that pattern.

This gives a series of aligned symbolic time series. Next we translate each symbol into a number by replacing it with the middle-value of its associated bin. Finally, we sum the values at all levels at the same time point and smooth this single level signal by combining each five consecutive values as compensation for using only the middle-value of each discretisation bin. This gives us a synthetic time series, say T.

Now CS is, of course, computed from a seismogram, say S. If CS is characteristic of S, T should resemble S. Well, it should resemble S, when all the details of the frequency levels not in $\{4, 5, 6, 7, 8\}$ are filtered out of S.

To test this we did this experiment for a few of the seismograms. One of these tests is pictured in Figure 4.6. The top plot is the original seismogram. The second one is the signal as present on the frequency levels 4 to 8, i.e., 2.5 Hz to 0.16 Hz. The next two plots are generated using the procedure above.

4.6 Discussion

The whole procedure of wavelet decomposition, discretisation, and building a code set using MULTI-KRIMP serves one purpose: to characterise a seismogram. The experiments substantiate that this is indeed the case.

The experiments on the first dataset show that our code sets perform well in distinguishing between visually different seismograms. In fact, as noted in the previous section, they perform

better than the state of the art method [107] based on shapelets.

The experiments on the second set of data show that the code sets perform also well in clustering similar events together, while simultaneously distinguishing between different events. In this case it performed on par with the shapelet based method from [107]. An important aspect of this experiment is that it shows that, at least in this case, the characterisation is independent of, the event size, the location of the event, and of the location of the seismometers. This is important since as we already discussed in the Introduction many of the characteristics of a seismic signal *are* dependent on these three aspects.

Hence, code sets are both good in the recognition of similar seismograms and in the distinction between dissimilar seismograms. In other words, code sets are characteristic for seismograms.

This is further corroborated by our third experiment. Here we see that the artificial seismograms one can generate from a code set visually resemble the seismograms from which the code set itself was computed. This is important to show to the seismologist what exactly are the characteristics of a given cluster of seismograms. Clearly, one can always present the seismologist with a centre of a cluster. The strong point of a generated seismogram is, however, that it shows what all the seismograms have in common. That is something that is hard to infer from a representative seismogram.

4.7 Conclusion

The goal of this chapter is to find a method to characterise seismograms, such that the identification of an event in a new seismogram (is it an earthquake, a passing truck, or something else) can be facilitated by finding similar seismograms that have already been identified.

To achieve this goal, we devised a method that first preprocesses a seismogram using a wavelet decomposition and discretisation and then uses a new algorithm called MULTI-KRIMP, to discover a code set that contains characteristic patterns for that seismogram.

The experiments show that these resulting code sets are indeed characteristic of a seismogram. They are good in both the recognition of similar seismograms and in the distinction between dissimilar seismograms. Moreover, they can be used to generate a synthetic seismogram that shows what all seismograms in a cluster have in common.

The final conclusion is two seismograms S_1 and S_2 are similar if we have for their MULTI-KRIMP's code sets CS_1 and CS_2 that $CS_1(S_1) \approx CS_2(S_1)$ and $CS_1(S_2) \approx CS_2(S_2)$.

CHAPTER 5

Keeping it Short and Simple: Summarising Complex Event Sequences with Multivariate Patterns

We study how to obtain concise descriptions of discrete multivariate sequential data. In particular, how to do so in terms of rich multivariate sequential patterns that can capture potentially highly interesting (cor)relations between sequences. To this end we allow our pattern language to span over the domains (alphabets) of all sequences, allow patterns to overlap temporally, as well as allow for gaps in their occurrences.

We formalise our goal by the Minimum Description Length principle, by which our objective is to discover the set of patterns that provides the most succinct description of the data. To discover high-quality pattern sets directly from data, we introduce DITTO, a highly efficient algorithm that approximates the ideal result very well.

Experiments show that DITTO correctly discovers the patterns planted in synthetic data. Moreover, it scales favourably with the length of the data, the number of attributes, the alphabet sizes. On real data, ranging from sensor networks to annotated text, DITTO discovers easily interpretable summaries that provide clear insight in both the univariate and multivariate structure.[1]

[1]This work was originally published as [15]:
R. Bertens, J. Vreeken, and A. Siebes. Keeping it Short and Simple: Summarising Complex Event Sequences with Multivariate Patterns. *KDD'16*, ACM.

5.1 Introduction

Most real sequential data is multivariate, as when we measure data over time, we typically do so over multiple attributes. Examples include sensors in a network, frequency bands in seismic or ECG data, transaction records, and annotated text. In this chapter we investigate how to succinctly summarise such data in terms of those patterns that are most characteristic for the data.

To capture these characteristics, our pattern language needs to be rich, i.e., patterns may span multiple aligned sequences (attributes) in which no order between these attributes is assumed and occurrences may contain gaps. For example, if we consider a sensor network, a pattern could be a characteristic set of values for multiple attributes at one point in time, or, more complex, a specific value for one sensor, temporally followed by certain readings of one or more other sensors. That is, patterns that are able to identify associations and correlations between one or multiple attributes.

Having such a rich pattern language has as immediate consequence that the well-known frequent pattern explosion hits particularly hard. Even for trivial data sets one easily finds enormous numbers of frequent patterns, even at modest minimal frequency thresholds [87]; the simplest synthetic dataset we consider contains only five true patterns, yet the *lower bound* on the number of frequent patterns is more than a billion.

Clearly, returning collections of such magnitude is useless, as they cannot be inspected in any meaningful way. Rather, we want a small set of such patterns that collectively describe the data well. To find such sets, we employ the Minimum Description Length (MDL) principle. In Chapter 3 we showed that this approach has a proven track record in the domain of transaction data mining and has also been successfully applied to, e.g., sequential data. Because complex multivariate sequential data is ubiquitous and correlation between multiple sequences can give much insight to domain experts, here we take this research further by defining a framework to summarise this data, while able to efficiently deal with the enormous space of frequent patterns. Note that all real-valued time series can easily be discretised to fit our framework, for example using SAX [56].

The humongous number of frequent patterns makes it intractable to discover a small set of characteristic patterns by post-processing the set of all frequent patterns; all the more because the MDL objective function on pattern sets is neither monotone nor sub-modular. Hence we introduce a heuristic algorithm, called DITTO,[2] that mines characteristic sets of patterns directly from the data. In a nutshell, DITTO mines good models by iteratively adding those patterns to the model that maximally reduce redundancy. This approach resembles the SLIM and SQS-SEARCH algorithms discussed in Chapter 3. That is, it iteratively considers the current description of the data and searches for patterns that most frequently occur one after the other, and considers the union of such pairs as candidates to add to the model. As such, DITTO focusses on exactly those patterns that are likely to improve our score, pruning large parts of the search space, and hence only needs seconds to mine high-quality summaries.

[2]The ditto mark (") is used in lists to indicate that an item is repeated, i.e., a multivariate pattern.

Our extensive empirical evaluation of DITTO shows it ably discovers succinct summaries that contain the key patterns of the data at hand, and, importantly, that it does not pick up on noise. An example of what we discover includes the following. When applied to the novel Moby Dick by Herman Melville (one attribute) aligned with the corresponding part-of-speech tags (another attribute), the summary DITTO discovers consists of meaningful non-trivial linguistic patterns including "⟨noun⟩ of the ⟨noun⟩", e.g. "Man of the World", and "the ⟨noun⟩ of", as used in "the ship of (somebody)". These summaries hence give clear insight in the writing style of the author(s). Besides giving direct insight, the summaries we discover have high downstream potential. One could, for example, use them for comparative analysis [20]. For example, for text identifying similarities and differences between authors, for sensor networks detecting and describing concept drift over time, and characterising differences between patients. Such analysis is left for future work, as we first have to be able to efficiently discover high quality pattern-based summaries from multivariate sequential data. That is what we focus on in this chapter.

The remainder of this chapter is organised as follows. We first cover preliminaries and introduce notation in Section 5.2. In Section 5.3 we formally introduce the problem. Section 5.4 gives the details of the DITTO algorithm. We discuss related work in Section 5.5, and empirically evaluate our score in Section 5.6. We round up with discussion and conclusions in Sections 5.7 and 5.8, respectively.

5.2 Preliminaries and Notation

As data type we consider databases D of $|D|$ multivariate event sequences $S \in D$, all over attributes A. We assume that the set of attributes is indexed, such that A_i refers to the i^{th} attribute of D.

A multivariate, or *complex*, event sequence S is a bag of $|A|$ univariate event sequences, $S = \{S^1, \ldots, S^{|A|}\}$. An event sequence $S^i \in \Omega_i^n$ is simply a sequence of n events drawn from discrete domain Ω_i, which we will refer to as its *alphabet*. An event is hence simply an attribute–value pair. That is, the j^{th} event of S^i corresponds to the value of attribute A_i at time j. We will write Ω for the union over these attribute alphabets, i.e. $\Omega = \cup_{i \in |A|} \Omega_i$.

By $||S||$ we indicate the number of events in a multivariate sequence S, and by $t(S)$ the length of the sequence, i.e. the number of time steps. We will refer to the set of events at a single time step j as a *multi-event*, writing $S[j]$ for the j^{th} multi-event of S. To project a multivariate event sequence S onto an individual attribute, we write S^i for the univariate event sequence on the i^{th} attribute, and analogously define D^i. For completeness, we define $||D|| = \sum_{S \in D} ||S||$ for the total number of events, and $t(D) = \sum_S t(S)$ for the total number of time steps in D.

Our framework fits both categorical and transaction (item set) data. With categorical data the number of events at each time step is equal to the number of attributes. For simplicity, w.l.o.g., we consider categorical data in the remainder.

As patterns we consider partial orders of multi-events, i.e. sequences of multi-events, allowing for gaps between time steps in their occurrences. By $t(X)$ we denote the *length* of a

pattern X, i.e. the number of time steps for which a pattern X defines a value. In addition, we write $||X||$ to denote the *size* of a pattern X, the total number of values it defines, i.e. $\sum_{X[i] \in X} \sum_{x \in X[i]} 1$. For example, in Figure 5.1 it holds that $||X|| = 4$ and $t(X) = 3$.

We say a pattern X is *present* in a sequence $S \in D$ of the data when there exists an interval $[t_{start}, \cdots, t_{end}]$ in which all multi-events $X[i] \in X$ are contained in the order specified by X, allowing for gaps in time. That is, $\forall_{X[i] \in X} \exists_{j \in [start,end]} X[i] \subseteq S[j]$, and $k \geq j + 1$ for $X[i] \subseteq S[j]$ and $X[i+1] \subseteq S[k]$. A singleton pattern is a single event $e \in \Omega$ and $\mathcal{P}$ is the set of all non-singleton patterns. A minimal window for a pattern is an interval in the data in which a pattern occurs which cannot be shortened while still containing the whole pattern [86]. In the remainder of this chapter when we use the term pattern occurrence we always expect it to be a minimal window. Further, we use $occs(X, S)$ for the disjoint set of all occurrences of X in S. Figure 5.1 shows examples of two categorical patterns occurring both with and without gaps.

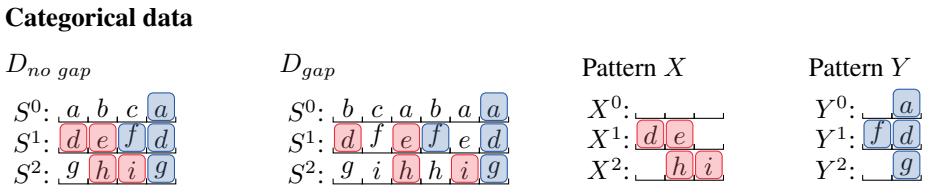


Figure 5.1: Patterns X and Y occurring with (D_{gap}) and without ($D_{no\ gap}$) gaps in the data.

Our method operates on categorical data. To find patterns in continuous real-valued time series we first have to discretise it. In Chapter 2 we discussed SAX, which is a celebrated approach for doing so, and we will use it in our experiments – though we note that any discretisation scheme can be employed. This ordinal data can either be represented by absolute values per time step, or by relative values that represent the difference between each pair of subsequent values. These relative values describe the changes in the data rather than the exact values – by which different types of patterns can be discovered.

5.3 Theory

In Chapter 3 we already gave an introduction to the MDL principle. In this section we define our objective function and analyse the complexity of the optimization problem.

MDL for Multivariate Sequential Data

As models for our data we consider code tables (CT); these are simple four-column look-up tables (or, dictionaries) between patterns on the left hand side, and associated codes on the right hand side (Chapter 3). In this chapter we consider three types of codes, i.e. pattern codes, gap codes and fill codes. We will explain the use of these below, in Section 5.3 and Example 2.

Loosely speaking, whenever we read a pattern code $code_p(X \mid CT)$ we know we can start to decode the events of pattern X. A fill code $code_f(X \mid CT)$ tells us we can decode the next time step of pattern X, whereas a gap code $code_g(X \mid CT)$ tells us there is a gap in this occurrence of pattern X. To fill such a gap we read the next pattern code. Note that our approach thus allows for patterns to interleave. For readability, we do not write CT wherever clear from context. To make sure we can encode any multivariate event sequence over Ω, we require that a code table at least contains all singleton events. Next, we will describe how to use these code tables to cover and encode a dataset. Further, we define how to compute $L(CT)$ and $L(D \mid CT)$ and we conclude with our formal problem definition.

Covering

A cover determines where we use each pattern when encoding a dataset and also where gaps in patterns occur. More formally, a cover $\mathcal{C}$ defines the set of all pattern occurrences that are used to cover the complete dataset. Therefore, if one of the events from pattern occurrence o is already covered by another pattern occurrence $o' \in \mathcal{C}$ we say that the occurrence o *overlaps* with the current cover $\mathcal{C}$, i.e. $\mathcal{C} \cap o \neq \emptyset$, which we do not allow.

As we are after the minimal description, we want to use optimal codes. Clearly, the more often a code is used, the shorter it should be. This, however, creates a non-trivial connection between how often a pattern is used to describe part of the data, and its code length. This, in turn, makes the complete optimisation process, and in particular that of finding a good cover of the data given a set of patterns non-trivial. For the univariate case, we showed [90] that when we know the optimal code lengths we can determine the optimal cover by dynamic programming, and that we can approximate the optimal cover by iteratively optimising the code lengths and the cover. Although computationally expensive, for univariate data this strategy works well. For multivariate data, naively seen the search space grows enormously. More importantly, it is not clear how this strategy can be generalised, as now the usages of two patterns are allowed to temporally overlap as long as they do not cover the same events. We therefore take a more general and faster greedy strategy.

To cover a dataset we need the set of patterns from the first column of a code table CT, which we will refer to as PS (Patterns and Singletons). In Algorithm 10 we specify our COVER algorithm, which iterates through PS in a pre-specified order that we will detail later. For each pattern X the algorithm greedily covers all occurrences that do not overlap with already covered data, and have fewer than $t(X)$ gaps. To bound the number and size of gaps we only allow occurrences where the number of gap positions is smaller than the length of the pattern itself. This avoids gap codes becoming cheaper than fill codes, by which occurrences with more gaps would be preferred over compact occurrences. This process continues until all data is covered. In Figure 5.2 we show in 3 steps how a PS is used to cover the example dataset.

Example 1. In Algorithm 10, we start with an empty cover, corresponding to an uncovered dataset as in the top left of Fig 5.2. Next, we cover the data using the first pattern from the PS (line 1). For each occurrence of the pattern $\langle {}^{a}_{d} \rangle$ (line 2), we add it to our cover (line 4) when it

does not overlap previously added elements and has a limited number of gaps (line 3). In Step 1 of Fig 5.2 we show the result of covering the data with the pattern $\langle {}^{a}_{d} \rangle$. We continue to do the same for the next pattern $\langle h, e \rangle$ in CT. This gives us the cover as shown in Step 2. With only the singleton patterns left to consider in CT there is only one way to complete our cover by using them for all so far uncovered events. Now all events in the dataset are covered, thus we can break out of our loop (line 6) and return the final cover of Step 3 in Fig. 5.2 (line 7).

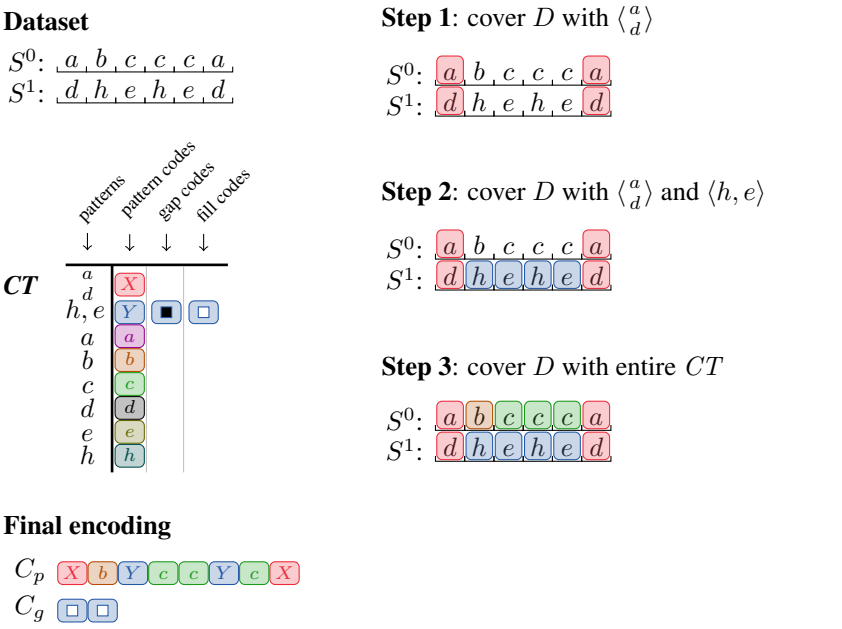


Figure 5.2: **Example of how a *CT* is used to cover (and encode) a dataset.** Note that for all patterns but $\langle h, e \rangle$ there is no need for gap and fill codes. See Example 1 and Example 2 for a more detailed description of the cover and encoding process, respectively.

Encoding

A cover $\mathcal{C}$ specifies how the data will be encoded. Conceptually, we can decompose it into a pattern code stream and a gap code stream. The pattern stream C_p contains a sequence of pattern codes, $code_p(X)$ for pattern $X \in CT$, used to describe the data, whereas the gap code stream C_g consists of codes $code_g(X)$ and $code_f(X)$ indicating whether and where a pattern instance contains gaps or not, see also Chapter 3.

To encode a dataset, see Algorithm 11, we traverse our dataset from left-to-right and top-to-bottom and each time we encounter a new pattern in our cover we add the code for this pattern to C_p. When moving to the next multi-event in the dataset we add for each pattern, that

Algorithm 10 The COVER Algorithm

Input: A sequence $S \in D$, a set of patterns PS, and $\mathcal{C} = \emptyset$
Output: A cover $\mathcal{C}$ of $S \in D$ using PS

```
1: for each X ∈ PS in order do                                  // see Sec. 5.4 for order
2:     for all o ∈ occs(X, S) do                                // all occurrences of X
3:         if C ∩ o = ∅ and t(o) < 2t(X) then                   // no overlap, and no gaps longer than t(X)
4:             C ← C ∪ o
5:     if C completely covers S then
6:         break
7: return C
```

we already encountered but not yet completely passed by, a gap or fill code to C_g. We choose a gap code for a pattern if the current multi-event does not contain any event from the pattern or a fill code if it does. We are finished when we have encoded all multi-events in the data.

Algorithm 11 The ENCODE Algorithm

Input: A sequence $S \in D$, a cover $\mathcal{C}$, and the corresponding code table CT
Output: A pattern code stream C_p and a gap code stream C_g

```
1: C_p ← ∅
2: C_g ← ∅
3: for all S[i] ∈ S do                                          // Multi-events from left-to-right
4:     for all S^k[i] ∈ S[i] do                                 // attributes from top-to-bottom
5:         o ← occurrence of pattern X that covers S^k[i] in C
6:         if ∀S^l[j] ∈ o it holds that S^l[j] is not covered then
7:             mark the events of o as covered
8:             C_p ← C_p + code_p(X)
9:     for all o' ∈ C for which start(o') < i and end(o') ≥ i do   // all partially encoded occurrences
10:        Y ← pattern(o')
11:        if S[i] ∩ o' = ∅ then                                // this multi-event is not covered by this occurrence
12:            C_g ← C_g + code_g(Y)                            // add a gap code
13:        else                                                 // pattern Y continues in this multi-event
14:            C_g ← C_g + code_f(Y)                            // add a fill code
15: return C_p and C_g
```

Example 2. Continuing from Example 1, to encode the dataset from Figure 5.2 we start with an empty pattern (line 1) and gap (line 2) stream. We then walk through the covered data from left-to-right (line 3) and top-to-bottom (line 4). First, we encounter the pattern $\langle {a \atop d} \rangle$ (line 5), which is not yet covered (line 6), thus we add $code_p({a \atop d})$ to the pattern stream (line 8). We move to the next time step and subsequently add $code_p(b)$ and $code_p(h, e)$ to C_p. In the next time step we add $code_p(c)$ to C_p, whereafter we encounter the second part of the partially

encoded pattern $\langle h, e\rangle$ (line 9). This indicates that there is no gap for this pattern, thus we add $code_f(h, e)$ to the gap stream (line 14). In a similar fashion we encode the rest of the data until we reach the end of it. This leads to the encoding as in the top left of Figure 5.2. Note that in our example only the pattern $\langle h, e\rangle$ can contain gaps, but because in both usages of the pattern there are no gaps we only find two fill codes for this pattern in the gap stream.

Example 3. In Figure 5.3 we show two different encodings for the same dataset. The first encoding only uses the singleton patterns from CT_1 and the second encoding also uses the larger patterns $X = \langle {}^{a,\ b}_{d,} \rangle$ and $Y = \langle c, e\rangle$. For the first encoding our pattern stream consists of the pattern codes for all events ordered from left-to-right and top-to-bottom. Its gap stream stays empty because singleton patterns can not have gaps. The second encoding is a bit more complex. To encode the first multi-event we first add $code_p(a)$ and then $code_p(Y)$ to C_p. For the second multi-event we add $code_p(X)$ to C_p and $code_g(Y)$ to C_g, because event e from pattern Y does not yet occur in this multi-event. For the third multi-event we add $code_p(a)$ to C_p, and $code_g(X)$ and $code_f(Y)$ to C_g. Code $code_g(X)$ marks the gap for pattern X and $code_f(Y)$ indicates that event e from pattern Y does occur in this multi-event. The fourth multi-event results in the addition of $code_p(Y)$ to C_p and $code_f(X)$ to C_g, where the latter marks the presence of event b from pattern X in this multi-event. For the last multi-event we add $code_p(a)$ to C_p and $code_f(Y)$ to C_g, which completes the encoding.

Using C_p and C_g we can compute the actual codes to construct the code table CT corresponding to PS used to cover the data. We will encode the data using optimal prefix codes [28], the length of which we can compute by Shannon entropy. In our case this means that the length of an optimal pattern code for pattern X is the negative log-likelihood of a pattern in the cover [35], that is

$$L(code_p(X \mid CT)) = -\log\left(\frac{usage(X)}{\sum_{Y \in CT} usage(Y)}\right) \quad ,$$

where $usage(X)$ is the number of times a pattern X is used to describe the data, i.e. $usage(X) = |\{Y \in C_p | Y = code_p(X)\}|$.

Gap and fill code lengths are computed similarly, corresponding to the negative log-likelihood of these codes within the usages of the corresponding pattern. That is, we have

$$L(code_g(X \mid CT)) = -\log\left(\frac{gaps(X)}{gaps(X) + \mathit{fills}(X)}\right) \quad ,$$

$$L(code_f(X \mid CT)) = -\log\left(\frac{\mathit{fills}(X)}{gaps(X) + \mathit{fills}(X)}\right) \quad .$$

where $gaps(X)$ and $\mathit{fills}(X)$ are the number of times a gap, resp. fill-code of pattern X is used in the cover of the data, i.e. $gaps(X) = |\{Y \in C_g \mid Y = code_g(X)\}|$ and analogue for $\mathit{fills}(X)$.

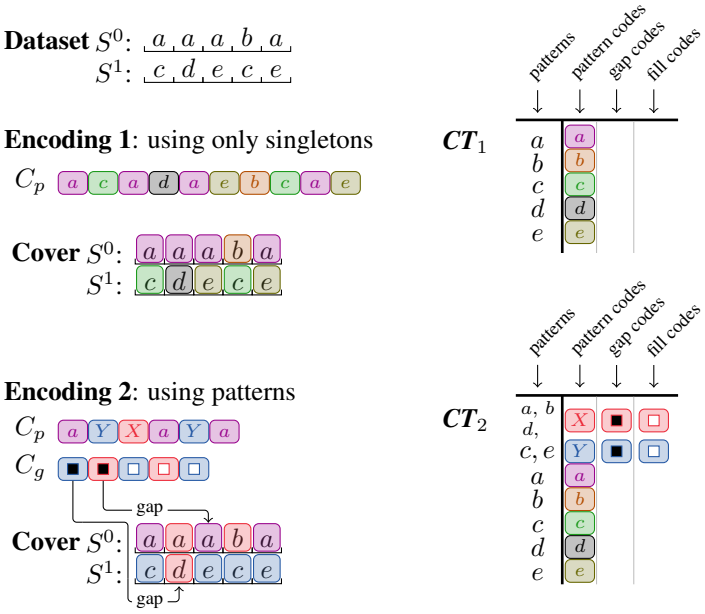


Figure 5.3: **Example of two possible encodings of the same dataset.** The first encoding uses only singletons (CT_1) and the second uses the singletons and the two patterns X and Y from CT_2. See Example 3 for a more detailed description.

Decoding

As we are using a lossless coding scheme we are able to reconstruct the original data from the code table and the encoded data together with the dimensions of the original data. In short (see Algorithm 12), we reconstruct the original dataset from left-to-right (line 2) and top-to-bottom (line 6), reading from the pattern stream (line 7) until the current multi-event is completely decoded (line 6). Thereafter, for the next multi-event (line 2) we read a code from the gap stream (line 4) for each pattern that did not yet end in the previous multi-event (line 3).

Example 4. In Figure 5.4 we show step-by-step how to decode the second encoding from Example 3 using the corresponding CT_2. For each step in the decoding we state which codes are read and how these are used to reconstruct the original dataset. We start with step 1 by reading $code_p(a)$ from the pattern stream (line 7), whereafter we can add event a to S^0 in the first time step (line 8). Because this time step is not yet completely decoded (line 6), we read $code_p(Y)$ from C_p (line 7) and finish step 2 by adding event c from the first time step of pattern Y to our dataset. Now the first time step of our dataset is decoded we continue to the next (line 2) in step 3, where we read $code_g(Y)$ from the gap stream (line 4), because pattern Y was not yet completely decoded (line 3). As we read a gap code for pattern Y there

Algorithm 12 The DECODE Algorithm

Input: A pattern code stream C_p and gap code stream C_g with corresponding code table CT
Output: A sequence $S \in D$

1: $S \leftarrow \emptyset$
2: **for all** $i \leftarrow 0$ to $t(S)$ **do** // decode all multi-events from left-to-right
3: **for all** partially decoded patterns X **do** // reconstruct events using partially decoded patterns
4: $X[j] \leftarrow$ determine next multi-event using C_g // $X[j]$ points to X's first undecoded time step
5: $S[i] \leftarrow S[i] \cup X[j]$ // decode values, if a gap code then $X[j] = \emptyset$
6: **while** $|S[i]| \neq |A|$ **do** // all events in this multi-event must be decoded (top-to-bottom)
7: $X \leftarrow$ next pattern from C_p // read the next pattern from the pattern stream
8: $S[i] \leftarrow S[i] \cup X[0]$ // add the first multi-event from X to S
9: **return** S

is nothing to add to our dataset (line 5) and thus we have to read again from C_p, which gives us $code_p(X)$. We can now finish the third step by adding the events a and d, from the first time step of pattern X, to S^0 and S^1 respectively. In step 4 we want to decode the event for the third time step on attribute A^0, but because the undecoded part of pattern X holds events on this attribute we first have to read from the gap stream. We read $code_g(X)$ from C_g and therefore we also have to read from the pattern stream to find the event we are looking for. After we read $code_p(a)$ from C_p we can finally append event a to S^0 of the dataset. In step 5 we again have to read from the gap stream, however, this time we read a fill code for Y, after which we can immediately append event e to S^1 (line 5). With steps 6 to 9 we continue the decoding in a similar fashion to correctly find the original dataset.

Encoded Length of Data Given Code Table

Now that we have determined the cover and encoding scheme, we can formalise the calculation of the encoded length of the data given the CT. This encoded length is the sum of the encoded length of following terms: the pattern stream, the gap stream, the number of attributes, the number of sequences and the length of each sequence. Formally, we have

$$L(D \mid CT) = L(C_p \mid CT) + L(C_g \mid CT) + L_{\mathbb{N}}(|A|) + L_{\mathbb{N}}(|D|) + \sum_{S \in D} L_{\mathbb{N}}(t(S)) \quad ,$$

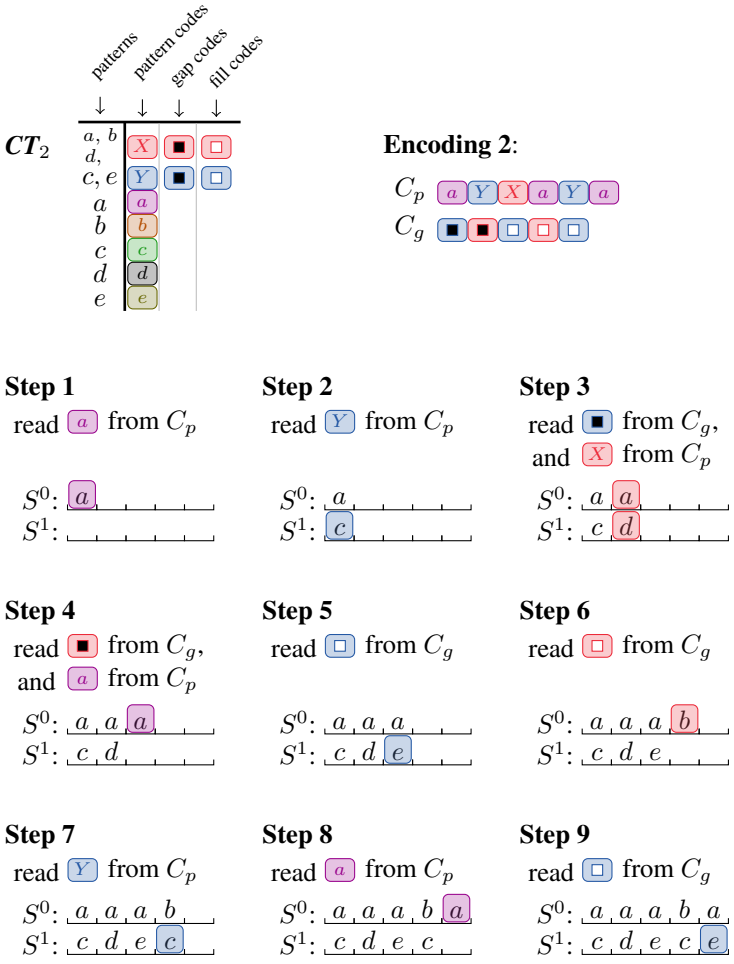


Figure 5.4: **The step-by-step decoding of example encoding 2 using CT_2, both as introduced in Example 3.** See Example 4 for a step-by-step description.

where $L_{\mathbb{N}}$ is the MDL optimal Universal code for integers [35] and the encoded length of the pattern and gap stream are simply the code lengths of all codes in the streams,

$$L(C_p \mid CT) = \sum_{X \in CT} usage(X) L(code_p(X))$$

$$L(C_g \mid CT) = \sum_{X \in CT}^{t(X)>1} \big[gaps(X) L(code_g(X)) + fills(X) L(code_f(X)) \big] \quad .$$

Encoded Length of Code Table

The encoded length of the code table consists of the following parts. For each attribute A_j we encode the number of singletons and their support in D^j. Then we encode the number of non-singleton patterns in the code table, the sum of their usages, and then using a strong number composition their individual usages. Last, we encode the patterns themselves using $L(X \in CT)$. We hence have

$$\begin{aligned} L(CT \mid \mathcal{C}) = \sum_{j \in |A|} \left(L_{\mathbb{N}}(|\Omega_j|) + \log \binom{|D^j| - 1}{|\Omega_j| - 1} \right) \\ + L_{\mathbb{N}}(|\mathcal{P}| + 1) + L_{\mathbb{N}}(usage(\mathcal{P}) + 1) \\ + \log \binom{usage(\mathcal{P}) - 1}{|\mathcal{P}| - 1} + \sum_{X \in \mathcal{P}} L(X \in CT) \quad . \end{aligned}$$

For the encoded length of a non-singleton pattern $X \in CT$ we have

$$\begin{aligned} L(X \in CT) = L_{\mathbb{N}}(t(X)) \; + L_{\mathbb{N}}(gaps(X) + 1) \\ + \sum_{t(X)} \log(|A|) + \sum_{x \in X} L(code_p(x \mid ST)) \quad , \end{aligned}$$

where we first encode its length, and its total number of gaps – note that we can derive the total number of fills for this pattern from these two values. As $L_{\mathbb{N}}$ is defined for integers $z \geq 1$, we apply a $+1$ shift wherever z can be zero [35]. Then, per time step, we encode for how many attributes the pattern defines a value, and what these values are using the singleton-only, or Standard Code Table (ST). For the encoded length of an event given the ST we have

$$L(code_p(x \mid ST)) = -\log \left(\frac{support(x \mid D)}{||D||} \right) \quad ,$$

which is simply the negative log-likelihood of the event under an independence assumption.

Formal Problem Definition

Loosely speaking, our goal is to find the most succinct description of the data. By MDL, we define the optimal set of patterns for the given data as the set for which the optimal cover and associated optimal code table minimise the total encoded length. As such we have the following problem definition.

Minimal Pattern Set Problem *Let Ω be a set of events and let D be a multivariate sequential dataset over Ω, find the minimal set of multivariate sequential patterns $\mathcal{P}$ and cover $\mathcal{C}$ of D using $\mathcal{P}$ and Ω, such that the encoded length $L(CT, D) = L(CT \mid \mathcal{C}) + L(D \mid CT)$ is minimal, where CT is the code-optimal code table for $\mathcal{C}$.*

Let us consider how complex this problem is. Firstly, the number of possible pattern sets (with a maximum pattern length of n) is

$$\sum_{k=1}^{2^{|\Omega|^n}-|\Omega|-1} \binom{2^{|\Omega|^n} - |\Omega| - 1}{k} \quad .$$

Secondly, to use a pattern set to cover the data we also need to specify the order of the patterns in the set. That is, we need to find the optimal order for the elements in the pattern set to find the one that minimises the total encoded length. The total number of ways to cover the dataset using one of the possible ordered pattern sets is

$$\sum_{k=1}^{2^{|\Omega|^n}-|\Omega|-1} \binom{2^{|\Omega|^n} - |\Omega| - 1}{k} \times (k + |\Omega|)! \quad .$$

Moreover, unfortunately, it does not show submodular structure nor (weak) (anti-)monotonicity properties by which we would be able to prune large parts of it. Hence, we resort to heuristics.

5.4 The DITTO Algorithm

In this section we present DITTO, an efficient algorithm to heuristically approximate the MDL optimal summary of the data. In particular, it avoids enumerating all multivariate patterns, let alone all possible subsets of those. Instead, it considers a small but highly promising part of this search space by iterative bottom-up search for those patterns that maximally improve the description. More specifically, we build on the idea of SLIM and SQS, which are discussed in Chapter 3. That is, as candidates to add to our model, we only consider the most promising combinations of already chosen patterns – as identified by their estimated gain in compression.

We give the pseudo code of DITTO as Algorithm 13. We start with singleton code table ST (line 1) and a set of candidate patterns of all pairwise combinations of singletons (line 2). We then iteratively add patterns from the candidate set to code table CT in **Candidate Order**: $\downarrow$ *estimated gain*(X), $\downarrow$ *support*$(X \mid D)$, $\downarrow$ $||X||$, $\downarrow$ $L(X \mid ST)$ and $\uparrow$ lexicographically (line 3). This order prefers the most promising candidates in terms of compression gain. When a new pattern improves the total encoded length $L(D, CT)$ we keep it, otherwise we discard it (line 4). After acceptance of a new pattern we prune (line 5) CT and recursively test whether to add variations (line 6) of the accepted pattern in combination with its gap events. When all variations are tested recursively, we update the candidate set by combining $CT \times CT$ (line 7).

We give the details of each of these steps below, as well as explain how to gain efficiency through smart caching, and discuss the computational complexity of DITTO.

Covering

As detailed in the previous sections, to compute $L(D, CT)$, we first need to cover the data with CT. The COVER algorithm covers the data using the patterns as they are ordered in

Algorithm 13 The DITTO Algorithm

Input: The dataset D and singleton code table ST
Output: An approximation to the **Minimal Pattern Set Problem**

```
1: CT ← ST
2: Cand ← CT × CT
3: for X ∈ Cand in Candidate Order do
4:     if L(D, CT ⊕ X) < L(D, CT) then
5:         CT ← PRUNE(D, CT ⊕ X)
6:         CT ← VARIATIONS(D, X, CT)
7:         Cand ← CT × CT
8: return CT
```

CT. To find the optimal cover we need to identify the cover order that leads to the smallest encoded length of the data. We do so greedily, by considering the pattern set in a fixed order. As our goal is to compress the data, we prefer patterns that cover many events with just a short code length. We hence define the **Cover Order** as follows: $\downarrow ||X||$, $\downarrow support(X \mid D)$, $\downarrow L(X \mid ST)$ and $\uparrow$ lexicographically. It follows the intuition that we give preference to larger and more frequent patterns, for which we expect a higher compression gain, to cover the data first, as these maximise the likelihood of the cover.

Candidates and Estimation

Conceptually, at every iteration the set of candidates from which we can choose, consists of the Cartesian product of the code table with itself. Two patterns X and Y can be combined to form new candidate patterns. Each alignment of X and Y without gaps in either X, Y and the resulting pattern, forms a new candidate, with the exception of alignments in which X and Y overlap. See Figure 5.5 for an example.

Selecting the candidate with the highest gain is very expensive – we would need to cover the whole data for every candidate. Instead, we select the candidate with the highest *estimated* gain – which can be done much more efficiently. Intuitively, based on the estimated gain ($\Delta L'$) we only consider candidate patterns in a lazy fashion based on their usage and do not consider patterns with a lower usage than the current best candidate. For notational brevity, for a pattern $X \in CT$ we use $x = usage(X)$. Further, let s be the total usage count of all patterns in CT, i.e. $s = \sum_{X \in CT} x$. For $CT' = CT \oplus Z$, we use x' and s' similarly. We estimate z, the usage of Z, optimistically as the minimum of the usages of X and Y – and as $x/2$ when $X = Y$ (because the usage of XX cannot be higher). Formally, our gain estimate $\Delta L'(CT \oplus Z, D)$

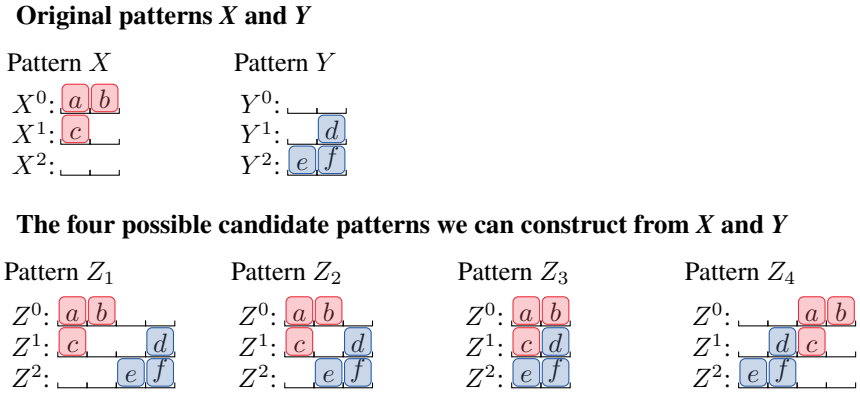


Figure 5.5: The four possible candidate patterns constructed from different alignments of X and Y.

of the true gain $\Delta L(CT \oplus Z, D)$ for adding pattern $Z = X \cup Y$ to CT is as follows,

$$\Delta L'(CT',\ D) = \Delta L'(CT' \mid D) + \Delta L'(D \mid CT') \quad ,$$

$$\Delta L'(CT' \mid D) = -L_{\mathbb{N}}(|Z|) - \sum_{l(Z)} \log(|A|) - \sum_{Z[i] \in Z} \sum_{z \in Z[i]} L(code_p(z \mid ST)) \quad ,$$

$$\Delta L'(D \mid CT') = s \log s - s' \log s' + z \log z - x \log x + x' \log x' - y \log y + y' \log y' \quad .$$

That is, the estimated gain of adding pattern Z to CT thus consists of the estimated gain in the size of the data, minus the increase in the size of CT. Note that $\Delta L'$ is an estimate and for simplicity we ignore the effects of adding pattern Z to code table CT on the pattern and (no-)gap usages of patterns other than X and Y.

Pruning

After the acceptance of a new pattern in our code table other patterns may have become redundant as their role may have been overtaken by the newer pattern. Therefore, each time a pattern X is successfully added to the code table, we consider removing those $Y \in CT$ for which the usage decreased and hence the pattern code length increased. Algorithm 14 describes how a code table is pruned.

Algorithm 14 The PRUNE Algorithm

Input: The dataset D and a code table CT
Output: A pruned code table

```
Cand ← X ∈ CT with decreased usage
for X ∈ Cand in Prune Order do
    if L(D, CT \ X) < L(D, CT) then
        CT ← CT \ X
        Cand ← Cand ∪ {Y ∈ CT | usage decreased}
    Cand ← Cand \ X
return CT
```

Generating Pattern Variations

To efficiently discover a large and diverse set of promising patterns – without breadth-first-search, which takes long to find large patterns, and without depth-first-search, which would be prohibitively costly – we consider *variations* of each accepted pattern to the code table. That is, when a pattern leads to a gain in compression, we consider all ways by which we can extend it using events that occur in the gaps of its usages. This way we consider a rich set of candidates, plus speed up the search as we are automatically directed to patterns that actually exist in the data. Algorithm 15 outputs a code table possibly containing variations of the lastly added pattern Y.

Algorithm 15 The VARIATIONS Algorithm

Input: The dataset D, a pattern Y and a code table CT
Output: A code table possibly containing variations of Y

```
Cand ← Y × gap events(Y)
for X ∈ Cand do
    if L(D, CT ⊕ X) < L(D, CT) then
        CT ← PRUNE(D, CT ⊕ X)
        CT ← VARIATIONS(D, X, CT)
    Cand ← Cand \ X
return CT
```

For example, consider the dataset $\{a, b, a, b, c, a, c, a\}$ where pattern $\{a, a\}$ occurs twice with a gap of length one. After adding pattern $\{a, a\}$ to CT we consider the patterns $\{a, b, a\}$ and $\{a, c, a\}$ for addition to CT.

Faster Search through Caching

The space of all possible multivariate patterns is extremely rich. Moreover, in practice many candidate patterns will not lead to any gain in compression. In particular, those that occur only

very infrequently in the data are unlikely to provide us any (or much) gain in compression. We thus can increase the efficiency of DITTO by allowing the user to impose a minimum support threshold for candidates. That is, only patterns X will be evaluated if they occur at least σ times in the data. To avoid time and again re-evaluating candidates of which we already know that they are infrequent, we cache these in a tree-based data structure. Only *materialised* infrequent patterns are added to the tree. Future candidates are only materialised when none of its subsets are present in the tree, as by the a priori principle we know it cannot be frequent [60].

Even though this tree can theoretically grow very large, in practice it stays relatively small because we only consider a small part of the candidate space. That is, we only combine patterns we know to be frequent to form new candidate patterns. In practice, we found that DITTO only has to cache up to a few thousand candidates. Using this tree we see speed ups in computation of 2 to 4 times, while also memory consumption is strongly reduced. For some datasets the difference is even bigger, up to an order of magnitude.

In this work we only consider keeping track of infrequent candidates. Note, however, that at the expense of some optimality in the search additional efficiency can be gained by also storing *rejected* candidates in the tree. In both theory and practice, however, candidates rejected in one iteration may lead to compression gain later in the process [79].

Complexity

The time complexity of DITTO has a simple upper bound as follows. In the worst-case we cover the data for each frequent pattern from the set of all frequent patterns $\mathcal{F}$ in each iteration. Covering takes $O(|CT| \times ||D||)$ and the number of iterations is worst-case $O(|\mathcal{F}|)$. Together, the worst-case time complexity is

$$O(|\mathcal{F}|^2 \times |CT| \times ||D||) \quad .$$

From the experiments in Section 5.6, however, we will learn that this estimate is rather pessimistic. In practice the code table stays small ($|CT| \ll |\mathcal{F}|$), we only consider a subset of all frequent patterns and we do this greedily. In practice the runtime of DITTO therefore stays in the order of seconds to minutes.

5.5 Related Work

The first to use MDL to summarise transaction data were Siebes et al. [77], resulting in KRIMP [97]. It shifted focus from the long-standing goal of mining collections of patterns that describe the set of all frequent patterns, e.g. closed [69] frequent patterns, to a *pattern set* that describes the data.

Similar to the transaction data domain, summarisation of sequential data also developed from frequent pattern mining. For sequential data, the key types of patterns studied are frequent subsequences [71], and frequent episodes [4, 62]. As with all traditional pattern mining approaches, redundancy is also a key problem when mining sequential patterns [40, 100].

To this end, Tatti and Vreeken [90] proposed to instead approximate the MDL-optimal summarisation of event sequence data using serial episodes. Their method SQS deals with many challenges inherent to this type of data, such as the importance of the order of events and the possibility for patterns to allow gaps in their occurrences – aspects we build upon and extend. Other methods exist, but either do not consider [11, 58] or do not punish gaps [48, 49] with optimal codes. None of these methods consider, or are easily extended to multivariate sequential data.

One of the first to consider multivariate sequential patterns, by searching for multi-stream dependencies, were Oates et al. [68]. Tatti and Cule [87] formalised how to mine the set of closed frequent patterns from multivariate sequential data, where patterns allow simultaneous events. In [102] the mining of high utility sequential patterns is studied, where they allow simultaneous events. Chen et al. [25] and Moerchen et al. [66] study mining interval-based patterns, where frequency is determined by how often univariate patterns co-occur within a given interval. All these methods are traditional pattern mining techniques in the sense that they return all patterns that pass an interestingness threshold.

Whereas traditional pattern mining techniques often only consider discrete data, there does exist extensive literature on mining patterns in continuous valued time series. These patterns are often called 'motifs' [27, 55]. For computational reasons, many of these methods first discretise the data [56]. Most motif discovery algorithms consider only univariate data. Example proposals for motif discovery in a multivariate setting include that by Tanaka et al. [85], who first transform the data into one dimension before the pattern discovery process and do not consider gaps, and by [64], who do not allow patterns to temporally overlap even if they span different dimensions and do not consider variable-length motifs. More recently Vespier et al. [93], mine characteristic multi-scale motifs in sensor-based time series but aim at mining all motifs, not a succinct summary.

To the best of our knowledge there are no methods yet to summarise *multivariate* sequential data, other than regarding each attribute separately or with restrictions on the pattern language [13]. In this work we introduce DITTO to discover important sequential associations between attributes by mining succinct summaries using rich multivariate patterns.

5.6 Experiments

We implemented DITTO in C++ and generated our synthetic data and patterns using Python. We make our code available for research purposes.[3] All experiments were conducted on a 2.6 GHz system with 64 GB of memory. For our experiments on real data we always set the minimum support threshold as low as feasible, unless domain knowledge suggests otherwise.

We evaluate DITTO on a wide range of synthetic and real world data. As discussed in Sec. 5.5, there exist no direct competitors to DITTO. Traditional pattern mining and motif discovery methods 'simply' mine all patterns satisfying some constraints. For summarising sequential data, most existing methods consider univariate data [48, 90]. The only summarisation approach for multivariate sequential data considers the special case where attributes

[3] Code – `http://eda.mmci.uni-saarland.de/ditto/`

are ordered (e.g. frequency bands) [13], whereas we consider multivariate sequential data in general. We empirically compare DITTO to SQS, the latter was discussed in Chapter 3. We do so by applying SQS to each univariate sequence $S^i \in D$, combining these results into one pattern set.

Synthetic Data

To validate whether DITTO correctly identifies true multivariate sequential patterns from a dataset, we first consider synthetic data. In particular, we generate random data in which we plant a number of randomly generated patterns of different characteristics. Clearly, ideally the true patterns are recovered. Moreover, ideally no other, spurious patterns that are only due to noise are returned. To this end we perform an extensive set of experiments varying the number of events, the number of attributes and the alphabet size of the dataset, and the number, frequency and size of the planted patterns.

Data Generation

As noted in Table 7.1, for each experiment we generated $t(D)$ random multi-events on $|A|$ attributes (i.e. a total of $||D||$ events) with an alphabet size per attribute of $|\Omega_i|$. Further, after the data generation $|\mathcal{P}|$ patterns are planted, where each pattern X has a size $||X||$, a 5% chance on a gap between subsequent multi-events, and a support such that each pattern spans $support\%$ of all events in the dataset. An example of an insertion of a pattern in a random dataset that does not lead to an actual occurrence of that pattern is when due to the gap chance the minimal window of the pattern contains too many gaps. We do not allow patterns to overwrite each other during generation, as this makes evaluation much more complicated – i.e. it becomes unclear whether not recovering a pattern is an artefact of the search or of the data generation process. Further, only for experiments with 50 attributes, we prevented that pattern occurrences interleave and did not allow an event to be used in more than one pattern to assure that the planted patterns are actually present in the data. This restriction is justified because we are merely testing whether our algorithm is able to find multivariate patterns in multivariate data and without it patterns will easily crowd each other because of the high number of attributes.

Evaluation

We evaluate the quality of the pattern set discovered by considering how close they represent the planted patterns. In particular, following [101] we consider both exact ($=$) and subset ($\subset$) matches. Exact indicates that the pattern exactly corresponds with a planted pattern, whereas subset implies that it is only part of a planted pattern. In addition, we consider how well the planted patterns are recovered; we report how many of the events of the ground truth pattern set $\mathcal{P}$ we can cover with the discovered patterns. The higher this ratio, the better this result. Last, we consider the gain in compression of the discovered model over the initial, Standard

Code Table. The higher this number, the better – the best score is attained when we recover all patterns exactly, and no further noise.

Results

We first consider the traditional approach of mining all (closed) frequent multivariate patterns. We do so using the implementation of Tatti and Cule [87]. We use a minimal support of 90% of the lower-support planted pattern. This choice is made to ensure that even when not all insertions of a pattern result in an actual occurrences, it can still be discovered. For the most simple synthetic dataset we consider, corresponding to the first row of Table 7.1, this takes a few days, finally reporting a *lower bound* of 14 092 944 394 frequent patterns, and returning 6 865 closed frequent patterns – hardly a summary, knowing there are only 5 true patterns. In the remainder we therefore do not consider traditional pattern mining.

Next, we consider DITTO and SQS. We report the results in Table 7.1. On the right hand side of the table we see that DITTO recovers all planted patterns, and does not report a single spurious pattern (!). In all cases it recovers the ground truth model, and obtains the best possible gain in compression. Next to the exactly identified planted pattern sometimes it also identifies some subsets of the planted patterns. This is a result of the data generation, i.e. subsets are sometimes included in the code table when planted occurrences contain too many gaps to be covered with the exact pattern. The patterns SQS discovers, on the other hand, are only small univariate fragments of the ground truth, recovering roughly only 10% to 30% of the ground truth. The near-zero gains in compression corroborate it is not able to detect much structure.

Regarding runtime and scalability, DITTO scales very favourably. Although SQS is faster it considers only the much simpler case of univariate data and patterns. DITTO requires seconds, up to a few minutes for the largest data, even for very low minimal support thresholds. Analysing the runtime of DITTO in more detail show how well its heuristics work; most time is spent on computing the minimal windows for candidates, of which up to ten thousand are materialised. Only for a few hundred of these a full cover of the data is required to evaluate their contribution to the model. Smart implementation for computing the minimal windows of all candidates in one pass will hence likely speed up DITTO tremendously.

Real World Data

As case studies we consider 4 datasets. An ECG sensor dataset, a structural integrity sensor dataset of a Dutch bridge, the text of the novel Moby Dick tagged with part-of-speech tags, and a multilingual text of an European Parliament resolution. See Table 5.2 for their characteristics.

ECG To investigate whether DITTO can find meaningful patterns on real-valued time series we consider a well-known ECG dataset.[4] For ease of presentation, and as our main goal is to show that DITTO can discover multivariate patterns, we consider only two sensors. As

[4]ECG – `http://physionet.org/physiobank/database/stdb`

Table 5.1: DITTO discovers all planted patterns on all synthetic datasets, without picking up on noise. Given are the base statistics of the synthetic datasets, and the results of SQS and DITTO. For SQS and DITTO we give the number of exactly recovered patterns (=) and the number of discovered patterns that are subsets of planted patterns (⊂). Further, we report how much of the ground truth is recovered ($R\%$), as well as the gain in compression over the singleton-only model ($L\%$), for both higher is better. Last, we give the runtime in seconds.

Data				**Planted Patterns**			**SQS**				
$\|\|D\|\|$	$t(D)$	$\|A\|$	$\|\Omega_i\|$	$\|\mathcal{P}\|$	$\|\|X\|\|$	*support*	$=$	$\subset$	$R\%$	$\Delta L\%$	time
100 000	10 000	10	100	5	3–7	1%	0	1	9.5	0.1	3
100 000	10 000	10	100	5	**5**	**10%**	0	0	0	0	3
100 000	10 000	10	**1 000**	5	3–7	1%	0	1	8.0	0	12
100 000	10 000	10	100	**20**	3–7	1%	0	9	19.5	0.9	4
500 000	10 000	**50**	100	5	3–7	1%	0	0	0	0	15
500 000	**50 000**	10	100	5	3–7	1%	0	1	10.0	0.2	4
1 000 000	**100 000**	10	100	5	3–7	1%	0	4	31.8	0.3	4
Data				**Planted Patterns**			**DITTO**				
$\|\|D\|\|$	$t(D)$	$\|A\|$	$\|\Omega_i\|$	$\|\mathcal{P}\|$	$\|\|X\|\|$	*support*	$=$	$\subset$	$R\%$	$\Delta L\%$	time
100 000	10 000	10	100	5	3–7	1%	5	0	100.0	3.0	2
100 000	10 000	10	100	5	**5**	**10%**	5	5	100.0	31.6	31
100 000	10 000	10	**1 000**	5	3–7	1%	5	0	100.0	2.9	4
100 000	10 000	10	100	**20**	3–7	1%	20	0	100.0	12.9	15
500 000	10 000	**50**	100	5	3–7	1%	5	1	100.0	3.1	41
500 000	**50 000**	10	100	5	3–7	1%	5	0	100.0	3.1	28
1 000 000	**100 000**	10	100	5	3–7	1%	5	14	100.0	3.3	374

preprocessing steps we applied 3 transformations: we subsampled the data, we transformed it from absolute to relative, and we discretised it using SAX [56]. For the subsampling we replaced each 5 subsequent values with their average, thus creating a sequence 5 times shorter. Thereafter, we transformed the absolute data into relative data by replacing each value by the difference of its successor and its own value. Lastly, we discretised each attribute into 3 intervals using SAX. Using a minimum support of 10, within 360 seconds DITTO discovers a code table containing 11 non-singleton patterns that capture the main structure of the data. In Figure 5.6 we plotted 2 occurrences of the top ordered pattern of this code table. This is a multivariate pattern which shows a very characteristic repeating structure in the data comprising a longer flat line ending with a peak on both attributes simultaneously. Showing the power of our pattern language, note that the pattern does not define any values on the second attribute for the first and second-last time steps (indicated with arrows in Figure 5.6), while it does on the first attribute. This flexibility allows us to use this multivariate pattern to describe a larger part of the data.

Like for the synthetic data, we also ran SQS. Again we see that as it cannot reward multivariate structure it does not obtain competitive gains in compression. Close inspection

Table 5.2: Results on 4 real datasets. We give the basic statistics, as well as the number of patterns and relative gain in compression for both SQS and DITTO. The high compression gains show that DITTO discovers much more relevant structure than SQS.

Data						**SQS**		
Dataset	$t(D)$	$\lvert D\rvert$	$\lvert A\rvert$	$\lvert\Omega\rvert$	*support*	$\lvert\mathcal{P}\rvert$	$\Delta L\%$	time (s)
ECG	2 999	1	2	6	10	57	38.8	1
Bridge	5 000	1	2	10	100	21	58.8	1
Moby Dick	2 248	103	2	887	5	20	1.7	3
Text	5 960	115	3	4 250	10	35	1.6	12
Data						**DITTO**		
Dataset	$t(D)$	$\lvert D\rvert$	$\lvert A\rvert$	$\lvert\Omega\rvert$	*support*	$\lvert\mathcal{P}\rvert$	$\Delta L\%$	time (s)
ECG	2 999	1	2	6	10	11	**75.3**	360
Bridge	5 000	1	2	10	100	22	**76.3**	325
Moby Dick	2 248	103	2	887	5	79	**14.3**	102
Text	5 960	115	3	4 250	10	51	**2.2**	136

shows it returns many small patterns, identifying consecutive values.

Bridge Next we consider the setting of monitoring the structural integrity of a bridge in the Netherlands.[5] Amongst the collected sensor data are the strain, vibration and temperature. We selected 2 strain sensors (1 Hz) on consecutive pillars of the bridge and decomposed the signals into low, medium and high frequency components using techniques from [92]. We used the medium frequency components, after preprocessing, as our dataset. As preprocessing, we transformed the absolute values into relative values by replacing each value by the difference of its successor and its own value. We then discretised each z-normalised attribute into 5 intervals using SAX [56].

For a support threshold of 100, it takes DITTO 325 seconds to discover a code table with 22 non-singleton patterns. Although only one more than SQS at the same threshold, the patterns DITTO discovers are more descriptive. That is, they are multivariate and larger, leading to a much higher gain in compression. Moreover, the patterns it discovers correctly show the correlation between the attributes, whereas the patterns SQS discovers only identify univariate patterns.

Moby Dick For more interpretable results, we next evaluate on text data. In particular we considered the first chapter of the book Moby Dick, written by Herman Melville,[6] aligning the

[5] Bridge – `http://infrawatch.liacs.nl`

[6] Moby Dick – `www.gutenberg.org`

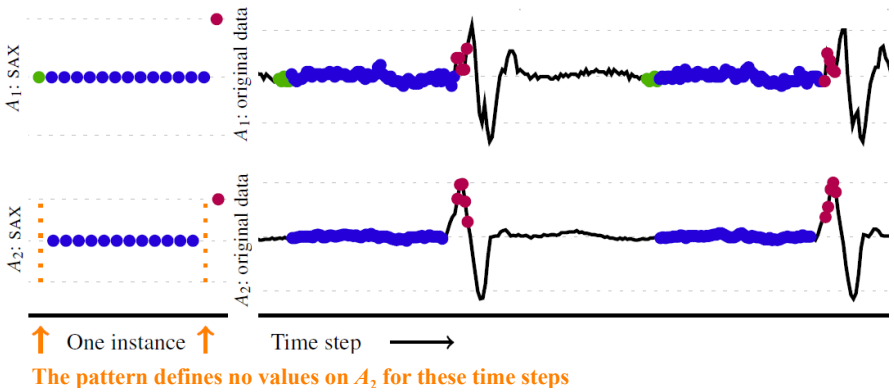


Figure 5.6: The top ordered pattern for the ECG data in the code table (left) and its first 2 occurrences in the data (right). The first time step of an occurrence is marked in green, subsequent ones in blue, and the last in red.

text with part-of-speech tags.[7,8] That is, one attribute comprises a stream of the words used in the book; each sentence is regarded as a sequence. The other attribute contains the tags that identify the type and function of each of these words. For example,

attribute 1:	VB	PRP	NNP
attribute 2:	Call	me	Ishmael

for which we will further use the following notation, where each time step is enclosed by curly brackets and the symbols for different attributes within a time step are divided by a comma: {VB, Call}{PRP, me}{NNP, Ishmael}. A short description for the part-of-speech tags in this example can be found in Table 5.4.

With a support threshold of 5, it takes DITTO 102 seconds to discover 79 non-singleton patterns. After studying the resulting pattern set we found that the identified patterns show highly intuitive multivariate structure. The highest ordered patterns together with examples of text fragments that match these patterns are shown in Table 5.3.

Given the modest compression gain that SQS obtains, see Table 5.2, it is clear there is not much structure in each of the attributes separately; DITTO, however, is able to find a significant amount of multivariate structure.

Multilingual Text As a final experiment, to further corroborate whether DITTO discovers meaningful and interpretable multivariate patterns, we consider mining patterns between the same text in different languages. To this end we collected a text in English, French and German from the European Parliament register of documents.[9] In this data we expect

[7]Part-of-speech tags – `http://nlp.stanford.edu/software/tagger.shtml`
[8]Part-of-speech tags – `https://gate.ac.uk/wiki/twitter-postagger.html`
[9]Text – `www.europarl.europa.eu/RegistreWeb`

Table 5.3: The highest ordered patterns in the code table for the Moby Dick dataset, together with example fragments of from the text which correspond to the patterns.

Pattern	**Example text fragments**
{TO, to}{VB}{DT, a}{NN}	to get a broom, to buy (him) a coat
{DT, the}{JJ}{NN}	the green fields, the poor poet
{DT, a}{JJ}{NN}	a mazy way, a hollow trunk
{DT, the}{JJ}{NNS}	the wild conceits, the old (sea-)captains
{PRP, i}{RB}{VB}	I quietly take, I always go
{EX, there}{VBZ, is}	there is

Table 5.4: A short description of the part-of-speech tags used in the examples in this chapter for the Moby Dick experiment.

Tag	**Explanation**
DT	Determiner
EX	Existential *there*
JJ	Adjective
NN	Noun, singular or mass
NNP	Proper noun, singular
NNS	Noun, plural
PRP	Personal pronoun
RB	Adverb
TO	*to*
VB	Verb, base form
VBZ	Verb, 3rd person singular present

frequent combinations of words within one (or more) language(s), as well as (and of more interest), multivariate patterns in the form of translations between the languages. As a preprocessing step all text data was stemmed and stop words were removed. To keep the different languages aligned we regarded every paragraph as a subsequence and padded shorter aligned subsequences with sentinel events which are ignored by DITTO. This ensures that the difference in length of the sentences in different languages will not lead to very big misalignments.

For a support threshold of 10, DITTO takes 136 seconds to discover 51 non-singleton patterns. The highest ordered pattern, i.e. the one that aids compression most, is a translation pattern; it identifies the correct relation between the French word *relève*, the German phrase *stellt fest dass* and the English word *note*. Other high ordered patterns are the English *EUR (x) million* and the German *(x) Millionen EUR*, and the words *parliament*, *Parlament* and *parlement* in English, German and French, respectively.

The modest compression gain of DITTO over SQS, see Table 5.2, indicates this data is not very structured, neither univariately, nor multivariately. One of the reasons being the

different order of words between different languages which results in very large gaps between translation patterns.

5.7 Discussion

Overall, the experiments show that DITTO works well in practice. In particular, the experiments on synthetic data show that DITTO accurately discovers planted patterns in random data for a wide variety of data and patterns dimensions. That is, DITTO discovers the planted patterns regardless of their support, their size, and the number of planted patterns – without discovering any spurious patterns. DITTO also performed well on real data – efficiently discovering characteristic multivariate patterns.

The results on the annotated text are particularly interesting; they give clear insight in non-trivial linguistic constructs, characterising the style of writing. Besides giving direct insight, these summaries have high downstream potential. One could, for example, use them for comparative analysis [20]. For example, for text identifying similarities and differences between authors, for sensor networks detecting and describing concept drift over time, and characterising differences between patients.

Although DITTO performs very well in practice, we see ways to improve over it. Firstly, MDL is not a magic wand. That is, while our score performs rather well in practice, we carefully constructed it to reward structure in the form of multivariate patterns. It will be interesting to see how our score and algorithms can be adapted to work directly on real-valued data. Secondly, it is worth investigating whether our current encoding can be refined, e.g. using prequential codes [20]. A strong point of our approach is that we allow for noise in the form of gaps in patterns. We postulate that we can further reduce the amount of redundancy in the discovered pattern sets by allowing noise *in the occurrences* of a pattern, as well as when we allow *overlap* between the occurrences of patterns in a cover. For both cases, however, it is not immediately clear how to adapt the score accordingly, and even more so, how to maintain the efficiency of the cover and search algorithms.

Last, but not least, we are interested in applying DITTO on vast time series. To accommodate, the first step would be to investigate parallelisation; the search algorithm is trivially parallelisable, as candidates can be generated and estimated in parallel, as is the covering of the data. More interesting is to investigate more efficient candidate generation schemes, in particular top-k mining, or lazy materialization of candidates.

Previous work has shown that MDL-based methods work particularly well for a wide range of data mining problems, including classification [49, 97] and outlier detection [78]. It will make for interesting future work to investigate how well DITTO solves such problems for multivariate event sequences. Perhaps the most promising direction of further study is that of causal inference [94].

5.8 Conclusion

We studied the problem of mining interesting patterns from multivariate sequential data. We approached the problem from a pattern set mining perspective, by MDL identifying the optimal set of patterns as those that together describe the data most succinctly. We proposed the DITTO algorithm for efficiently discovering high-quality patterns sets from data.

Experiments show that DITTO discovers patterns planted in synthetic data with high accuracy. Moreover, it scales favourably with the length of the data, the number of attributes, and alphabet sizes. For real data, it discovers easily interpretable summaries that provide clear insight in the associations of the data.

As future work, building upon our results on the part-of-speech tagged text data, we are collaborating with colleagues from the linguistics department to apply DITTO for analysis of semantically annotated text and for inferring patterns in morphologically rich languages.

CHAPTER 6

Efficiently Discovering Unexpected Pattern-Co-Occurrences

Our world is filled with both beautiful and brainy people, but how often does a Nobel Prize winner also wins a beauty pageant? Let us assume that someone who is both very beautiful and very smart is more rare than what we would expect from the combination of the number of beautiful and brainy people. Of course there will still always be some individuals that defy this stereotype; these beautiful brainy people are exactly the class of anomaly we focus on in this chapter . They do not possess intrinsically rare qualities, it is the unexpected combination of factors that makes them stand out.

In this chapter we define the above described class of anomaly and propose a method to quickly identify them in transaction data. Further, as we take a pattern set based approach, our method readily explains why a transaction is anomalous. The effectiveness of our method is thoroughly verified with a wide range of experiments on both real world and synthetic data.[1]

[1] This work was originally published as [16]:
R. Bertens, J. Vreeken, and A. Siebes. Efficiently Discovering Unexpected Pattern-Co-Occurrences. *SDM'17*, SIAM.

6.1 Introduction

The recognition of anomalies provides useful application-specific insights [2]. More specifically, the field of anomaly detection focusses on the identification of data that significantly differ from the rest of the dataset — so different that it gives rise to the suspicion that they were generated by a different mechanism. Such an anomaly may, e.g., occur because of an error, it may be an outlier, or it may be a highly unexpected data point. It is hard, if not impossible, to automatically distinguish between such different possible origins. Hence, anomalies should be inspected manually to decide whether it should, e.g., be removed, corrected, or simply remain in the data "as is". One should thus preferably not report an overly large list of potentially anomalous data points and, at the very least, that list should be ordered such that the most anomalous data points appear on top.

In this chapter we briefly shift our focus to the domain of transaction data, but we return to sequences data in the next chapter. For a transactional dataset anomaly detection usually boils down to pointing out those transactions that show unexpected behaviour. This unexpected behaviour can manifest itself in different ways and each detection algorithm is limited to find only those anomalies which fit the corresponding framework. For example, much work has been done to detect unexpected behaviour which can be expressed by the compressed size of a transaction given a pre-processed model [8, 78]. That is, transactions that badly fit the norm of the data are deemed to be anomalous. Another example is to score transactions based on the number of frequent patterns that reside in it [41]. Yet another method scores transactions based on items missing from a transaction which were expected given the set of mined association rules [67]. All these methods have their own advantages, however, none of them is able to detect an anomaly based on the presence of multiple items in a single transaction that are not expected to occur together. Therefore in this work we focus on this class of anomalies, not to improve existing methods, but to improve the field of anomaly detection by making it more comprehensive. Since there are many ways in which a transaction can be anomalous, there should be a wide variety of algorithms detecting complementary sets of anomalies.

In addition to highlighting the transactions that show anomalous behaviour, our method describes anomalies in more detail by providing the most unlikely co-occurrence of patterns in that transaction. As an example consider a dataset containing people's drinking habits where roughly half of the people drinks soft drink $\mathbb{C}$ and the other half drinks soft drink $\mathbb{P}$. Now each individual who drinks $\mathbb{C}$ or $\mathbb{P}$ is not surprising. Moreover, someone drinking both $\mathbb{C}$ and $\mathbb{P}$ also does not seem surprising as it can be compressed well using the methods of [8, 78], it contains multiple frequent patterns [41] and there is nothing missing [67]. However, in this dataset almost everyone drinks either $\mathbb{C}$ or $\mathbb{P}$, but not both. Therefore, someone drinking both $\mathbb{C}$ and $\mathbb{P}$ is an anomaly, as drinking both is unexpected. We propose to score each transaction based on the most unlikely co-occurrence between patterns and therefore our method is able to find the described class of anomalies.

For this example, the score we introduce is based on the well-known interestingness measure *lift* of an association rule [84] (also known as its *interest* [18] and closely related to the *novelty* of an association rule [50]). More precisely, we take minus the log of the minimal lift of the two association rules $\mathbb{C} \to \mathbb{P}$ and $\mathbb{P} \to \mathbb{C}$. So, the difference is that we do not

score a rule, but a transaction and do so by the minimal lift of all the rules that apply to this transaction.

The higher the score, the more anomalous the transaction. So, if both $\mathbb{C}$ and $\mathbb{P}$ are frequent, a transaction t containing both is anomalous if $\{\mathbb{C}, \mathbb{P}\}$ is infrequent, and the more infrequent it is, the more anomalous t is. Note that this is the opposite of *rare* patterns, or rare association rules [45]. For rare rules either $\mathbb{C}$, $\mathbb{P}$, or both should be infrequent, while the confidence of, e.g., $\mathbb{C} \rightarrow \mathbb{P}$ should be high.

For rare association rules either $\mathbb{C}$ or $\mathbb{P}$ is expected to be infrequent, and multiple or adaptive minimal support thresholds can be used for efficient discovery. Since we assume both $\mathbb{C}$ and $\mathbb{P}$ to be frequent and only $\{\mathbb{C}, \mathbb{P}\}$ to be infrequent such ideas cannot be used here. Rather, an exhaustive algorithm requires all frequent sets with a support equal or larger than 1, since any of these may be formed by an unexpected combination of patterns. Finding the most surprising transactions using this humongous set is infeasible on all but the most trivial data sets. For this reason we introduce a heuristic algorithm based on the code tables computed by algorithms such as KRIMP [97] or SLIM [79].

In extensive experiments we firstly show that this heuristic algorithm finds all anomalies we hide in synthetic data. Secondly, we show that the transactions found to be anomalous in real world data sets are indeed strange. For example, in the well-known *Adult* data set, the top-ranked transaction contains the very unexpected co-occurrence of someone whose attribute sex is female yet whose relationship status is husband. It is highly probable that this is a mistake, but it is certainly an anomaly the data scientist should be aware of before analysing the data set.

6.2 Notation

We consider transaction datasets D containing $|D|$ transactions. Each transaction t contains a subset, of size $|t|$, of the items from the alphabet Ω. Categorical data consists of $|A|$ attributes, where each attribute $A_i \in A$ has a domain Ω_i, and can also be regarded as transaction data by mapping each attribute value pair to a different item. We assume there is no missing data. We use $P(\cdot)$ to denote a probability function.

6.3 Anomalies in Transaction Data

What is an Anomaly? Anomalies are also referred to as abnormalities, discordants, deviants, or outliers in the data mining and statistics literature [2]. As we consider transaction data we use the following definition.

Definition 6. *A transaction is anomalous when it deviates from what we expect considering the whole dataset.*

Given this definition an anomaly can manifest itself in different ways, resulting in multiple classes of anomalies for transaction data. In this section we recall two familiar classes of anomalies, define one new class, and we show how to identify all of them by formalising

appropriate anomaly scores. We want to emphasise again that the scores for different classes of anomalies are complementary to each other. Further, for ease of interpretation and computation we take the negative log-likelihood for the scores in each class.

Class 0: Unexpected Transaction Lengths

A transaction can be anomalous not as a result of the patterns it contains, but solely on the basis of its deviating length.

Definition 7. *A class 0 anomaly is a transaction with significantly deviating transaction length.*

We propose an anomaly score which represents the number of bits needed to describe the transaction length given all transaction lengths in the data, i.e. for a transaction t we have

$$score_0(t) = -\log(P(|t|)) = -\log\left(\frac{|\{t' \in D \mid |t'| = |t|\}|}{|D|}\right) .$$

The intuition behind the subscript 0 for this score is that we take no patterns into account to identify these anomalies. As it is a fairly trivial score we will not further evaluate it.

Class 1: Unexpected Transactions

When a transaction contains very little structure, i.e. few or no frequent patterns, it can be regarded to be anomalous.

Definition 8. *A class 1 anomaly is a transaction that contains very little of the regularity conveyed by the rest of the dataset.*

The state of the art in transaction anomaly detection focusses on what we call class 1 anomalies. For example, OC^3 [78] scores transactions using a descriptive pattern set PS, i.e. a summary. Transactions containing few of these patterns but mostly singletons will get a higher score. That is, because such a transaction cannot be explained well by the pattern set that is descriptive for the data. We generalise this idea by defining a score based on the probability of a transaction. More formally, $score_1$ scores each transaction based on the number of bits needed to describe it, i.e. for a transaction t we have

$$score_1(t) = -\log P(t) \quad .$$

For compression based methods such as OC^3 this score is defined by the compressed length of the transaction given the model of the data. However, any method that can assign a probability to a transaction based on the whole data can be used here. Note that as transactions are scored as a whole, this approach will unlikely detect unexpected co-occurrences of patterns. For example, using OC^3, all patterns that describe a transaction will contribute to its score independently. As much work has been done to detect these anomalies we will not further evaluate their identification, but focus on the next class of anomalies.

Class 2: Unexpected Co-occurrences

The focus of this chapter lies on identifying unexpected co-occurrences of patterns.

Definition 9. *A transaction contains a class 2 anomaly when it contains two patterns that occur much less frequently together in the data than what could be expected from their individual supports.*

As this definition is somehow the opposite of that of a pattern, which is formed when two smaller patterns occur together more frequently than expected, we can also use the terms negative pattern or negative interaction pattern.

To identify anomalous transactions based on class 2 anomalies we would like to score a transaction based on the unexpectedness of the co-occurrences of the patterns contained in it. That is, we propose to rank a transaction based on its most unexpected pattern co-occurrence. Intuitively this means that for each transaction we compute the number of bits we need to explain the most unlikely co-occurrence given a pattern set PS and the data. For a transaction t we thus have

$$score_2(t) = \max_{\{X,Y \in PS | X,Y \subseteq t\}} -\log P(XY) + \log\big(P(X) \times P(Y)\big) \ .$$

In the remainder of this paper we refer to $score_2$ as the UPC score, for Unexpected Pattern Co-occurrence. We compute $P(X)$ as X's support or relative frequency in the data.

Given a UPC score for a transaction we can readily explain its anomalousness as we know which co-occurrence of patterns is responsible for the score. Therefore our method has the nice property of producing very interpretable rankings.

Our score is related to the concept of lift [72] used in the context of association rules. In our setting we use it to describe the difference between two patterns appearing together in a transaction and what would be expected if they were statistically independent. Therefore, the higher our score the more unexpected the pattern co-occurrence.

Scores that are constructed to identify class 1 anomalies are not able to detect these class 2 anomalies as they look at all patterns *independently*. For example, OC^3 [78] and COMPREX [8] will not give a class 2 anomaly a higher score as both individual patterns are frequent and will add little to the anomaly score. Similarly, the frequent pattern based method from He et al. [41] and the method from Narita et al. [67] have no means to give higher scores to class 2 anomalies. As a result, methods for identifying class 1 anomalies do not identify unexpected co-occurrences, while these actually do indicate anomalous behaviour.

Which patterns to consider

Given the relation between our score and the lift of association rules, a straightforward way to find high scoring transactions may seem to simply mine for low-lift association rules. However, to maximize the score the individual patterns X and Y should have a support as high as possible while XY should have a support as low as possible. That is, we should mine

for all rules — including those with a support of 1 — to ensure that we do not miss the most interesting, most anomalous transactions.

Clearly this quickly becomes infeasible for all but the smallest data sets. Not only because discovering all these rules will take an inordinate amount of time, but also since the post-processing of all these rules necessary to identify the most surprising transactions becomes a rather daunting task.

The alternative we take is by starting from a set of patterns PS. We compute the score of each pair of patterns from PS and identify those transaction in which pairs with a (very) high score occur.

Clearly, not just any pattern set will do as we want to find the highest scoring transactions. The set of all frequent patterns $\mathcal{F}$ will be far too large to be able to consider the interactions between each pair of patterns. In the worst case we need to consider each co-occurrence of patterns for each transaction, thus leading to a computational complexity of

$$O(|D| \times |\mathcal{F}| \times |\mathcal{F}|) \quad .$$

Choosing a higher minimum support will yield smaller pattern sets but as a result we might miss important patterns. We could use condensed representations such as closed [69] or non-derivable [21] frequent patterns to remove as much redundancy as possible, however these sets will still be too large. By sampling [39] patterns we can attain small sets of patterns, however, the choice of the size of the sample determines which anomalies one will (likely) find. A set that is too small might miss some important patterns, but a set that is too large probably contains redundancy and again becomes a bottleneck in our approach. Since it is not straightforward to choose the right size for the required pattern set, we choose to use KRIMP [97] or SLIM [79] to automatically find small descriptive pattern sets that describe the data well without containing noise or redundancy. Using these pattern sets it will hold that $|PS| \ll |\mathcal{F}|$. We thus dramatically reduce the complexity, making the UPC score practically feasible as we will show in our experiments in Section 6.6. Using such a vastly smaller set induces, of course, the risk that we miss anomalies. However in other research we have seen that the pattern sets chosen by KRIMP and SLIM are highly characteristic for the data. The experiments in Section 6.6 bear out that this is also the case here: all anomalies we inject in synthetic data are discovered using these small sets only.

6.4 How to use our scores

In the process of explorative data mining, one has to consider that all 3 classes of anomalies we identify give different insight, i.e., one should instantiate all 3 scores and investigate the top-ranked anomalies for each class. Here, our focus is of course on class 2 anomalies.

To determine which of the UPC top-ranked transactions to investigate, as well as to verify the significance of the scores, we propose two bootstrap methods, explained later in this section. Recall that bootstrap methods consider the given data as a sample, and generate a number of pseudo-samples from it; for each pseudo-sample calculate the statistic of interest,

and use the distribution of this statistic across pseudo-samples to infer the distribution of the original sample statistic [22].

Significance test

For a synthetic dataset it is easy to test the significance of anomaly scores, as we can generate data with and without anomalies for which the resulting scores must clearly differ. For real world data this is unfortunately not the case as we do not know which and how much (negative) patterns the data comprises. Nevertheless, to give a measure of significance we use the following bootstrap approach. We randomly sample transactions from our original dataset (with replacement) to retrieve an equally sized new dataset. We repeat this a thousand times and save the highest anomaly score for each dataset. Then we repeat this process, but we first remove the transaction with the highest UPC score from the sample set. That is, the top-ranked anomaly is definitely not present in the bootstrap samples of the second kind and may or may not be present in the bootstrap samples of the first kind. The bigger the difference between the distributions of scores with and without the top-ranked transaction, the more significant the top-ranked anomaly.

Which transactions to investigate

Choosing the right parameter value is never easy in explorative data mining. As the UPC score produces a ranking of all transactions, where higher scores indicate a higher chance on being anomalous, it does not need any parameters. To determine which transactions to investigate based on this ranking we employ Cantelli's inequality to identify the transactions that significantly differ from the norm.

Theorem 3. ***Cantelli's inequality*** *[34]. Let X be a random variable with expectation μ_X and standard deviation σ_X. Then for any $k \in \mathcal{R}^+$,*

$$P(X - \mu_X \geq k\sigma_X) \leq \frac{1}{1+k^2} \quad .$$

Smets and Vreeken [78] proposed a well-founded way to determine threshold values to distinguish between 'normal' and anomalous transactions. The positive class comprises anomaly scores for 'normal' transactions and based on the distribution of these scores we can choose a threshold by choosing an upper bound on the false-negative rate (FNR). For example, if we choose a confidence level of 10%, Cantelli's inequality tells us that this corresponds to a threshold θ at 3 standard deviations from the mean, given by $\theta = \mu + k\sigma$, with $k = 3$ in this case. This means that the chance on a future transaction with an anomaly score above the threshold is less than 10%, see Figure 6.1.

To compute these thresholds we need the distribution of the positive class, i.e. the anomaly scores for all 'normal' transactions. Because we have only one dataset available which can contain both transactions from the positive and negative (actual anomalies) class, we use again a bootstrap approach. We generate bootstrap datasets by randomly sampling transactions (with

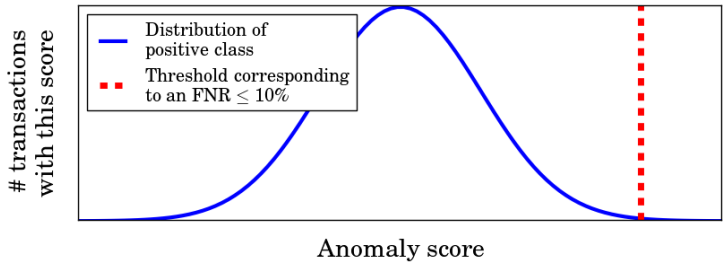


Figure 6.1: **Example of setting a threshold using Cantelli's inequality.** Based on the positive class we compute a threshold corresponding to a false-negative rate of 10%.

replacement) from the original dataset. We then use all anomaly scores from all bootstrap datasets to estimate the distribution.

6.5 Related Work

In this chapter we study anomaly detection in binary transaction data. As anomalies are referred to in many different ways, mostly with slightly different definitions, we refer to [63] and [2] for in-depth overviews on this field of research. In general, most anomaly detection methods rely on distances. Here we focus on discrete data, nominal attributes, for which meaningful distance measures are typically not available.

Of the methods that are applicable on transaction data, that of Smets and Vreeken [78] is perhaps the most relevant. They propose to identify anomalies as those transactions that cannot be described well by the model of the data, where as models they use small descriptive pattern sets. Their method OC^3 works very well for one-class classification, however it is not able to identify unexpected co-occurrences in the data. Akoglu et al. [8] proposed COMPREX which takes a similar approach in that they also rank transactions based on their encoded length. The difference is that they do not use a single code table, but a code table for each partition of correlated features. Although this method achieves very good results it is only suitable for categorical data and not for transaction data in general. Note that following our generalised anomaly score for class 1 anomalies, any method that provides a probability for a transaction can be used. Examples based on pattern sets are those of Wang and Parthasarathy [99] and Mampaey et al. [57].

He et al. [41] rank transactions based on the number of frequent patterns they contain given only the top-k frequent patterns, and Narita et al. [67] rank transactions based on confidence of association rules but need a minimum confidence level as parameter. All these methods have no means to identify class 2 anomalies.

In the Introduction we already mentioned the relation between our score and lift [18]. As stated there, the difference is that we score transactions rather than rules and we give an algorithm to quickly discover the highest scoring transactions. Our notion of anomaly is also

related to the conditional anomalies introduced in [81]. In our running example, $\mathbb{C}$ could be seen as the context that makes a purchase of $\mathbb{P}$ unexpected in their terminology. The difference is that we do not expect the user to define such contexts, they are discovered automatically. Moreover, we use a small set of patterns to discover all the class-2 anomalies rather than probabilistic models on context and other attributes.

To compute the UPC score we need the characteristic patterns of the data. In general, we can use the result of standard frequent pattern mining [3, 69] although this incurs a high computational cost. Instead, we can resort to pattern sampling techniques [17, 39], yet then we have to choose the number of patterns to be sampled. Alternatively, we [77] proposed to mine such pattern sets by the Minimum Description Length principle [35]. That is, they identify the best set of patterns as the set of patterns that together most succinctly describe the data. By definition this set is not redundant and does not contain noise. KRIMP [97] and SLIM [79] are two deterministic algorithms that heuristically optimise this score. Other pattern set mining techniques, especially those that mine patterns characteristic for the data such as [33, 57, 99], are also meaningful choices to be used with UPC.

6.6 Experiments

In this section we evaluate the power of the UPC score to identify class 2 anomalies. Firstly, we show how we generated synthetic data needed for some of the experiments. Secondly, we provide a baseline comparison where we show that the size of the input set of patterns is of great importance. Next we show the performance of UPC on synthetic data and show its statistical power. Lastly, we show some nice results of class 2 anomalies on a wide variety of real world datasets.

We implemented our algorithms in C++ and generated our synthetic data using Python. Our code is available for research purposes.[2]

Generating Synthetic Data

Here we describe how we generated both transaction and categorical synthetic data.

Transaction Data

To generate synthetic datasets we first choose the number of transactions $|D|$ and the size of the alphabet $|\Omega|$. We generate a set $\mathcal{P}$ of random patterns, choosing a random cardinality from 3 to 6 items, and a random support in the range of [5-10%]. In addition we generate 2 patterns with a support of 20%, which we call the anomaly generators, and which we add to the data such that they only occur together in a single transaction; that is the anomaly. Thereafter, we do the following for each transaction. With the probability corresponding to its support each pattern from $\mathcal{P}$ is added to the transaction as long as it does not interfere with the anomaly. In

[2] Code – `http://eda.mmci.uni-saarland.de/upc/`

addition, each singleton from Ω is added to each transaction similarly with a probability of 10%.

Categorical Data

To generate categorical data we take a similar approach. Firstly, we choose the number of transactions $|D|$, the number of attributes $|A|$ and the alphabet size per attributes $|\Omega_i|$. We generate random patterns with the same settings as for transaction data and again first add the anomaly generators to the dataset. We then try to add the other patterns as long as they fit and do not interfere with the anomaly. Then we fill the unspecified attributes for each transaction with random singletons.

Baseline Comparison

Before investigating the reliability of UPC, we first show its efficiency. To emphasise the necessity for using small pattern sets as input, we compare the use of all closed frequent patterns with a minimum support at 5% to the use of SLIM [79] pattern sets with a minimum support of 1. We generated random transaction data with $|D|$ = 5 000, $|\Omega|$ = 50 and we let $|\mathcal{P}|$ range from 10 to 35 patterns. We then ran our method on both input sets keeping track of the runtimes and the size of the input set PS for which we have to consider all $|PS| \times |PS|$ possible combinations. Both approaches always rank the anomaly highest, therefore further we can focus on the runtime and the number of patterns that were considered. The runtimes include the time needed to compute the pattern sets, which are negligible in light of the exponential time the baseline approach takes in the size of PS. Figure 6.2 shows the results which are averages over 5 runs per setting. For higher minimum support thresholds the baseline approach starts missing important patterns and it cannot identify the anomaly. Other settings for generating synthetic data lead to a similar figure. Since using a SLIM pattern set as input set for UPC we attain similar results compared to the baseline approach, that is we correctly identify the planted anomaly, in the remainder of this chapter we always use the SLIM pattern set to compute the UPC score.

Performance on Synthetic Transaction Data

The goal of this experiment is twofold. Firstly, we show that our method is able to identify class 2 anomalies in transaction data. Secondly, we justify the definition of the different classes of anomalies as we show that the class 2 anomalies are not identified by the state of the art class 1 anomaly detector, which is OC^3 [78]. We emphasise again that as a result both scores should not be further compared as they are complementary to each other.

We generated random datasets as described in Section 6.6 with various settings. The results in Table 6.1 show that UPC always ranks the anomaly highest and that OC^3 does not identify them.

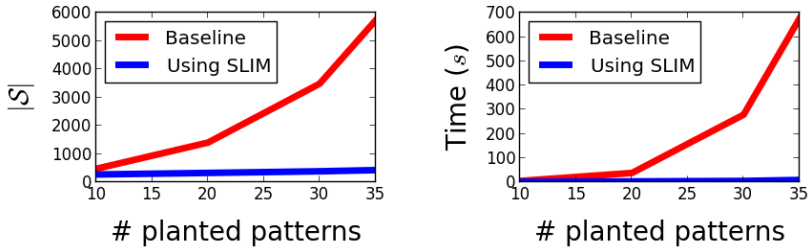


Figure 6.2: **SLIM pattern set versus closed frequent patterns (baseline).** We computed the UPC scores on 5 000 transactions using both input sets with a minimum support of 1 for SLIM and at 5% for the baseline. Using both input sets $\mathcal{S}$ the anomaly is always ranked first, but for the baseline both $|\mathcal{S}|$ and the runtime quickly explode when we increase the number of planted patterns in the data. The runtimes include the time needed to compute the used input set.

Table 6.1: **The performance of UPC on transaction data.** The number of generated transactions is represented by $|D|$, the alphabet size by $|\Omega|$, and the number of synthetic patterns by $|\mathcal{P}|$. All experiments were performed 10 times and the average ranks and runtimes (in seconds) are reported.

Generated Data			**UPC**		**OC^3**	
$\|D\|$	$\|\Omega\|$	$\|\mathcal{P}\|$	rank	time (s)	rank	time (s)
5 000	50	100	1	4	2 420	1
5 000	100	100	1	6	2 757	2
5 000	100	200	1	18	2 433	4
10 000	100	100	1	11	5 464	4
20 000	50	100	1	18	8 281	4

Performance on Synthetic Categorical Data

Knowing that UPC correctly identifies class 2 anomalies for transaction data, here we compared it to the state of the art on categorical data, which is COMPREX [8]. Again we note that we only compare these methods to show that class 2 anomalies are different from class 1 anomalies and that these two methods thus should be used complementary to each other.

We generated random datasets as described in Section 6.6 with various settings. The results in Table 6.2 show that UPC always ranks the anomalous transaction first and COMPREX is not able to identify it (gives it a much lower rank).

Statistical Power

Our aim here is to examine the power of the UPC score for identifying class 2 anomalies. For this purpose, we perform statistical tests using synthetic data. To this end, the null hypothesis

Table 6.2: **The performance of UPC on categorical data.** Each dataset contains 5 000 transactions over $|A|$ attributes, each with an alphabet size of $|\Omega_i|$. $|\mathcal{P}|$ refers to the number of synthetic patterns. All experiments were performed 10 times and average ranks and runtimes (in seconds) are reported.

Generated Data			UPC		COMPREX	
$\|A\|$	$\|\Omega_i\|$	$\|\mathcal{P}\|$	rank	time (s)	rank	time (s)
20	5	100	1	1	3 119	221
50	5	100	1	6	2 028	885
100	5	100	1	29	3 121	5 477
20	10	100	1	1	2 244	429
50	10	200	1	5	2 714	1 978

is that the data contains no class 2 anomalies. To determine the cutoff for testing the null hypothesis, we first generate 100 transaction datasets without the single co-occurrence between the 2 anomaly generators, whereafter we generate another 100 datasets with this co-occurrence included. For all datasets we choose $|D|$ = 5 000, $|\Omega|$ = 25 items and $|\mathcal{P}|$ = 100. Next, we report the highest UPC score for all 100 datasets without anomaly. Subsequently, we set the cutoff according to the significance level $\alpha = 0.05$. The power of the UPC score is the proportion of the highest scores from the 100 datasets with anomaly that exceed the cutoff. Note that we only look at the highest score for each dataset as we know that this must be the anomaly for the datasets containing it. We show the results in Figure 6.3 while varying the range from which we randomly choose the supports for the patterns in $\mathcal{P}$ from [4-8%] to [8-16%] and the support for the anomaly generators from 16% to 32%. In Figure 6.3 we label these linearly growing supports with their growth factor from 1 to 2. With other settings to generate the data we observe the same trend. Again, only to emphasise that methods to identify class 1 anomalies are not suitable to discover class 2 anomalies, in Figure 6.3 we also plotted the statistical power of OC^3 regarding class 2 anomalies. As COMPREX is not applicable to transaction data we performed a similar experiment on categorical data. This resulted in a similar plot with UPC at the top and COMPREX at the bottom.

In Figure 6.4 we show the distribution of the highest scores for both the datasets with and without an anomaly and with pattern supports in range [7-14%] and an anomaly generator support at 28%. We can see a clear distinction between the scores for 'normal' and anomalous transactions.

Real World Data

To show that class 2 anomalies actually exist, are not identified by the state of the art in anomaly detection, and can give much insight we performed multiple experiments on real world datasets from various domains. We used the *Adult* and *Zoo* datasets from the UCI

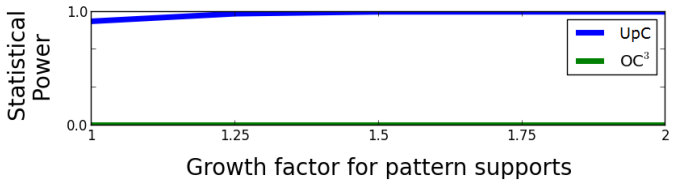


Figure 6.3: [Higher is better] **Statistical power of UPC.** Whereas OC^3 does not identify any class 2 anomalies, UPC does perfectly with large enough supports. We observe the same behaviour for categorical data comparing UPC and COMPREX (not shown in this plot). The growth factor on the x-axis describes the increase of the pattern supports.

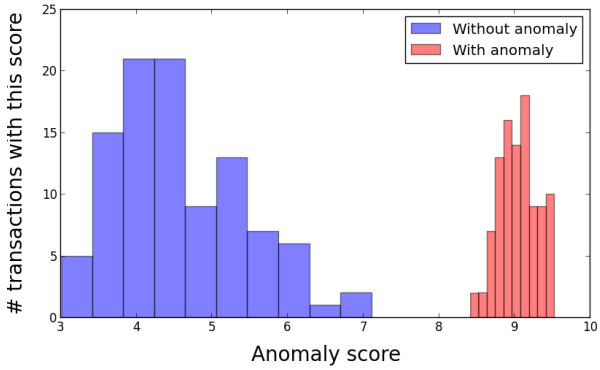


Figure 6.4: **Significance of UPC scores.** The plot shows a clear separation between the highest UPC scores for random synthetic datasets with and without class 2 anomalies.

Machine Learning Repository,[3] together with the *Mammals* [65] and *ICDM Abstracts* [31] datasets. Note that we cannot provide the reader with the accuracy of our method because anomalies can manifest themselves in many different forms, e.g. unexpected behaviour or mistakes in the data, and no ground truth for these datasets is available. However, we aim to give insightful examples instead.

Adult

The *Adult* dataset contains information about 48 842 people such as age, education, and marital-status and is used to predict whether someone's income exceeds \$50K a year.

We computed a ranking based on the UPC score and found some interesting anomalies. The top-ranked transaction contains the very unexpected co-occurrence of someone for which the attribute sex is female yet for whom the relationship status has the value of husband. The

[3] UCI Machine Learning Repository – `http://archive.ics.uci.edu/ml/datasets.html`

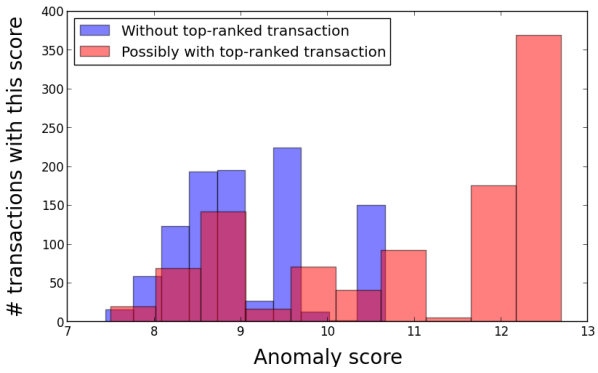


Figure 6.5: **Significance test on *Adult* dataset.** This plot shows the difference in the distribution of highest scores for bootstrap samples without (blue) and possibly with (red) the highest ranked transaction from the original dataset.

following 3 anomalies are persons with a similar situation but with the patterns reversed. That is, the dataset contains 3 persons whose sex is male and whose relationship is wife. The OC^3 rankings of these first 4 people are 115, 148, 89 and 4 090, respectively. These examples show that class-2 anomalies indeed exist in real datasets, and that UPC is effective at identifying these. Clearly, each of these four anomalies is an error, but that does not make them less of an outlier. In fact, one of the goals of outlier detection is to find errors.

To get an idea of the significance of the results we performed the significance test as described in Section 6.4. Figure 6.5 shows the difference in the distribution of highest scores for bootstrap samples without (blue), resp. from data including (red) the top-ranked transaction from the original dataset. Figure 6.5 gives insight in how much this transaction deviates from the norm, as the difference between the two distributions can only be caused by this transaction.

Zoo

The *Zoo* data contains 17 attributes describing 101 different animals.

We performed the bootstrap method described in Section 6.4 to determine which transactions are worth investigating. To this end, we generated 1 000 bootstrap samples for which we computed the anomaly scores for all transactions. In Figure 6.6 we show the distribution of all these scores with a histogram. Further, in the left plot we show the threshold (θ) values corresponding to false-negative rates (FNR) of 50%, 20%, 10% and 5% respectively, together with the number of transactions from the original dataset that score above θ. In the right plot we show the anomaly scores of the 5 highest ranked transactions in the original dataset together with the FNR corresponding to a θ equal to their score.

In Figure 6.6 in the left plot we see that with an FNR below 10% only the top-ranked transaction scores above θ. This transaction contains information about the platypus (duck

bill) and from our results we found that the co-occurrence causing this high score is that the platypus is the only oviparous mammal in the dataset. In Figure 6.6 in the right plot we see that the chance that the second ranked animal belongs to the positive class is less than 11%. This is the scorpion for which UPC found that it is the only animal without teeth that is not oviparous.

Clearly, both these anomalies are well-known as somewhat weird species and, so, these finds may not seem that interesting. However, the algorithm does not know much biology and yet it finds both anomalous species as well as the explanation for why they are seen as somewhat weird.

ICDM Abstracts

Next, we ran our algorithm on a dataset comprising the abstracts from the ICDM conference, after stemming, and removing stopwords.[4]

For this data we expect co-occurrences of terms used in different research fields to rank highly. In Table 6.3 we show the top 5 highest ranked abstracts with their explanation. That is, we show the unexpected co-occurrence responsible for the high UPC score. Further, only to show that these class 2 anomalies are not identified by the state-of-of-the-art, we show their OC^3 rank. The abstract with the highest UPC rank contains both the frequently used words 'pattern mining' and 'training'. Given that (frequent) pattern mining is unsupervised while methods like "train and test" are usually applied in a supervised setting, their combination is genuinely surprising. From the corresponding abstract it transpires that the term 'training' is used to refer to physical exercise rather than that of an algorithm. Hence, the

[4]The data is available upon request from the author of [31].

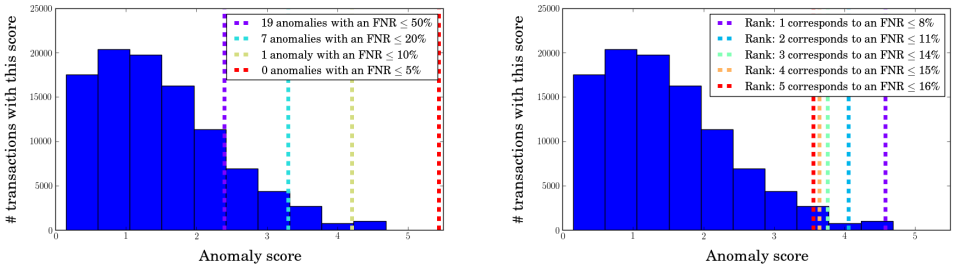


Figure 6.6: **Anomalies in the *Zoo* dataset.** The histogram shows the estimated distribution of anomaly scores. (left) The vertical lines show the decision thresholds at false-negative rates of 50%, 20%, 10% and 5%, together with the number of original transactions that score above the threshold. (right) The vertical lines show the scores of the top-5 ranked anomalies in the original dataset, together with the false-negative rates corresponding to the decision threshold for their score.

discovered anomaly points to an unusual application rather than to an unexpected combination of techniques.

Other highly ranked abstracts show similar unexpected co-occurrences, for example 'learning' on one side and 'frequent pattern mining' on the other or 'frequent pattern mining' and 'compare', which suggest that exploratory algorithms are difficult to compare.

Table 6.3: **The top 5 out of 859 abstracts from *ICDM Abstracts*, with the corresponding unexpected co-occurrences explaining the high UPC scores.** Next to the UPC rank also the OC^3 rank is reported to show that class 2 anomalies are not identified, but ranked low, using an algorithm constructed for class 1 anomalies.

UPC			**OC^3**
	Most unexpected co-occurrence explaining the anomaly score		
Rank	Pattern A	Pattern B	Rank
1	['mine', 'pattern']	['train']	165
2	['algorithm', 'mine', 'pattern', 'frequent']	['learn']	183
3	['rule']	['vector']	132
4	['frequent', 'itemset']	['learn']	193
5	['mine', 'pattern', 'frequent']	['compar']	556

Mammals

The *Mammals* dataset consists of presence/absence records of 121 European mammals within 2 183 geographical areas of 50 $\times$ 50 kilometres.[5] In this dataset an anomaly constitutes two large territories of (groups of) animals which only overlap in a small region.

Figure 6.7 shows two top-ranked areas (in red and pointed to by arrows) and readily explains why these are anomalous. For each of these two areas two groups of animals share this territory where the rest of their territory is completely separated. On the left in Figure 6.7 we see that the large habitat of the beech marten intersects with that of the moose, the European hedgehog and the mountain hare only in this single area. On the right in Figure 6.7 we see a similar phenomenon for the Etruscan shrew on one side and the raccoon dog on the other. The ranks of these two areas using OC^3 are 591 and 294 out of the 2 183, respectively. There are also top-ranked areas that are explained by two groups of animals which habitat intersects in multiple areas (of course including the area that has received this score).

6.7 Discussion

The experiments show that although the state of the art in anomaly detection is not able to identify the newly defined class 2 anomalies, we can identify them using our new UPC score.

[5]Full dataset [65] via Societas Europaea Mammalogica: `http://www.european-mammals.org`.

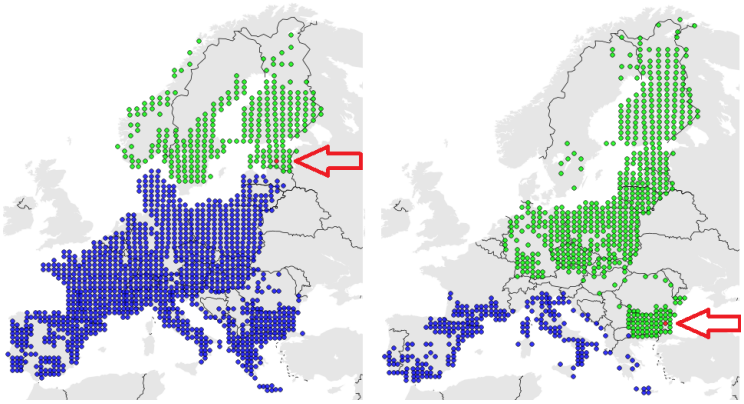


Figure 6.7: **UPC in action; top-ranked anomalies on the *Mammals* dataset.** The explanations for these highly ranked areas are as follows. On the left we see that the habitat of the beech marten (blue) only intersects with that of the moose, European hedgehog and mountain hare (green) at the (red) area pointed to by the arrow. On the right we see the habitat of the Etruscan shrew (blue) only intersects with that of the raccoon dog (green) at the (red) area pointed to by the arrow.

We demonstrated that a naive baseline approach using closed frequent items as input set quickly becomes infeasible when the number of patterns present in the data grows. Using a SLIM pattern set to compute our UPC score, however, we attain similar results in a fraction of the time. We showed the statistical power of our method which scores transactions containing planted class 2 anomalies significantly higher than 'normal' transactions. Moreover, both on transaction and categorical synthetic data we showed that UPC always ranked the planted anomaly at the top.

From our experiments on real world datasets we find that the class 2 anomalies do actually exist and can provide useful insights. That is, because next to identifying interesting transactions the UPC score also readily explains which co-occurrence of patterns is responsible for the transaction's anomaly score. For example, in the *Adult* dataset we found a very unexpected individual who is described as being a female husband. Further we showed how a UPC ranking can be used to study the significance of identified anomalies using a bootstrap approach. For example, in the *Zoo* dataset we found that the platypus, which is special because it is the only oviparous mammal, has a less than 8% chance on being 'normal' given the data. Each of these class 2 anomalies were not identified, i.e. ranked low, using OC^3 or COMPREX.

6.8 Conclusion

The recognition of anomalies provides useful application-specific insights [2]. More specifically, the field of anomaly detection focusses on the identification of data that significantly

differ from the rest of the dataset. There are many reasons for anomalies – ranging from errors to outliers to simply highly unexpected data points – however, whatever the reason, the anomalies should be brought to the attention of the data miner.

There are also many ways in which a data point (or a subset of the data) can differ from the rest of the data set. That is, there are many types of anomalies [2]. In this chapter we introduced a new class of anomalies which consist of unexpected co-occurrences of patterns. In a world where the vast part of the population drinks either soft drink $\mathbb{C}$ or soft drink $\mathbb{P}$ but not both, it is surprising to find someone who apparently drinks both.

We introduced the UPC score which intuitively scores a transactions based on its most unexpected co-occurrence of patterns. Next we introduced an algorithm that discovers and ranks transactions with a high UPC score. Moreover, it does so efficiently by relying on a small set of characteristic patterns [79] rather than on the full set of low support patterns (which would make an algorithmic approach intractable quickly). Finally we introduced a statistical test to decide whether or not an anomaly is significantly anomalous.

We tested our methods firstly on synthetic data. These experiments show that we are able to reliably discover the anomalous patterns we planted in a wide range of different settings and circumstances. Moreover, these experiments show that the anomalies identified by state of the art methods for anomaly detection are of a different class and that these methods are not able to identify planted anomalies.

That we can discover a new class of anomalies does not make them into an interesting class of anomalies. To illustrate that our anomalies indeed provide interesting and useful information we also did experiments on four real world data sets, viz., *Adult*, *Zoo*, *ICDM Abstracts*, and *Mammals*. In all cases the identified transactions with a high UPC score were truly anomalous. None of these examples were discovered using the state of the art anomaly detection algorithms

In some cases – such as on the *Adult* data set in which we discovered a female husband – the identified anomalies are very likely errors. In other cases – such as on the *Zoo* data set where we discovered the platypus – the discovered anomalies are not an error but simply a highly surprising combination of patterns: whereas laying eggs is quite normal, so is being a mammal, but being an egg-laying mammal is truly special.

Whatever the reason, these anomalies provide useful information to the analyst; whether they point to errors that probably should be corrected, such as female husbands, or to genuinely new information, such as the existence of egg-laying mammals.

While we tested our approach against state of the art algorithms this does not mean that we claim that our methods are better than existing methods. Rather, the experiments were performed to show that our methods are *complementary* to those methods. When looking for anomalies, one should not use one method, but many.

CHAPTER 7

Detection and Explanation of Anomalies in Multivariate Event Sequences

Anomaly detection is an important data mining task as the occurrence of (very) low probability events may provide valuable information [2]. For example in equipment monitoring such a (sequence of) event(s) may signal impeding equipment failure. In this example, equipment is often monitored by multiple sensors leading to multiple aligned sequences of data, i.e., a multivariate data stream. Hence, the importance of anomaly detection in such streams.

In this chapter we use multivariate patterns to identify and explain anomalies in multivariate data streams. We define two classes of anomalies for this type of data and we show how to identify them. We present extensive tests on synthetic data as well as results for real world problems such as the prediction of failing equipment. For the latter example, the proposed techniques are already successfully applied in business.[1]

[1] This work is under review as:
R. Bertens, J. Vreeken, and A. Siebes. Detection and Explanation of Anomalies in Multivariate Event Sequences.

7.1 Introduction

When industrial equipment fails it leads to a loss of money. Both as a result of high ad hoc maintenance costs and possible downtime that might lead to missed income. To prevent this loss many businesses perform scheduled service checks to ensure that their equipment stays in good health. The difficulty with this approach is that ideal service intervals are hard to determine. Even two identical machines might require different maintenance, e.g. caused by different workloads. As a result, the business is likely to face either unnecessary maintenance or occasional equipment failures, both leading to a waste of money.

With smart condition monitoring equipment is only serviced when necessary. When we use real-time data to predict the ideal moment for maintenance we call it smart condition monitoring. As this task is not straightforward, in this chapter we propose a framework in which multiple sensors (aligned sequences of data) can be considered simultaneously to automatically identify deviations in the behaviour of equipment. In other words, we perform anomaly detection on multivariate event sequences. These anomalies indicate a change in the behaviour of the equipment and, thus, may indicate current or future equipment failure. By studying multiple sensors simultaneously, we are able to benefit from any present correlation that indicates a possible failure. As a hypothetical example consider an engine for which we continuously measure its temperature and fuel consumption. Both temperature and fuel consumption might differ greatly even for healthy engines. A high temperature together with a very low fuel consumption, however, might indicate that the engine is not functioning properly and needs maintenance. Methods that can only analyse each sensor separately might not be able to detect such anomalies.

The prediction of equipment failure is but one example use case of our approach to anomaly detection. More in general, we consider multivariate event based data streams, including, e.g., discretised time series data. Following [15], we first use the DITTO algorithm to compute a pattern based summary of the data; it is based on a rich pattern language, i.e., patterns may span multiple aligned sequences (attributes) in which no order between these attributes is assumed and occurrences may contain gaps. The resulting code table not only presents a summary of the data, but also puts a probability on the (co-)occurrence of a pattern – sequence of multivariate events – in the event stream. Building on the theory for anomaly detection in transaction data from Chapter 6, we define two classes of anomalies and show how using the code table computed by DITTO can be used to identify the occurrences of these two classes of anomalies in multivariate data streams.

As an aside, note that we do *not* claim that our methods can discover *all* anomalies in multivariate data streams. After all, given a specific context any event could be an anomaly. We do claim that the anomalies we discover are worthwhile to investigate by a domain expert. To substantiate this claim we report on experiments on real world data. For example, in the field of predictive maintenance, we show that we can use our methods to predict a failure in a railway switch about 10 days ahead of time.

Besides answering the question when equipment may fail, our method also explains why we think it will fail. This explanation is given by pinpointing the unlikely (co-)occurrence of patterns in the data stream. In fact, it may even give insight into the type of maintenance that

is required. Consider, e.g., a small sequence of data that contains rare events that (almost) only occur when a specific part of the monitored equipment is about to break down, then it may be wise to service that specific part. The full integration of our anomaly scores and the classification of (possible) subsequent failures is, however, still work in progress and not further reported upon in this chapter.

The remainder of this chapter is organised as follows. We start by introducing preliminaries in Section 7.2. In Section 7.3 we define the different classes of anomalies in complex event sequences. We discuss related work in Section 7.4, and empirically evaluate our scores in Section 7.5. We round up with discussion and conclusions in Sections 7.6 and 7.7, respectively.

7.2 Preliminaries

Notation

Following Chapter 5, we consider datasets D of $|D|$ multivariate event sequences $S \in D$, all over attributes A. We assume that the set of attributes is indexed, such that A_i refers to the i^{th} attribute – the i^{th} data stream – of D.

A multivariate, or *complex*, event sequence S is a bag of $|A|$ univariate event sequences, $S = \{S^1, \ldots, S^{|A|}\}$. An event sequence $S^i \in \Omega_i^n$ is simply a sequence of n events drawn from discrete domain Ω_i, which we will refer to as its *alphabet*. An event is hence simply an attribute–value pair. That is, the j^{th} event of S^i corresponds to the value of attribute A_i at time j.

By $||S||$ we indicate the number of events in a multivariate sequence S, and by $t(S)$ the length of the sequence, i.e. the number of time steps. We will refer to the set of events at a single time step j as a *multi-event*, writing $S[j]$ for the j^{th} multi-event of S.

Patterns are partial orders of multi-events, i.e. sequences of multi-events, allowing for gaps between time steps in their occurrences. By $t(X)$ we denote the *length* of a pattern X, i.e. the number of time steps for which a pattern X defines a value. In addition, we write $||X||$ to denote the *size* of a pattern X, the total number of values it defines, i.e. $\sum_{X[i] \in X} \sum_{x \in X[i]} 1$.

A pattern X is *present* in a sequence $S \in D$ of the data when there exists an interval $[t_{start}, \cdots, t_{end}]$ in which all multi-events $X[i] \in X$ are contained in the order specified by X, allowing for gaps in time. That is, $\forall_{X[i] \in X} \exists_{j \in [start,end]} X[i] \subseteq S[j]$, and $k \geq j + 1$ for $X[i] \subseteq S[j]$ and $X[i+1] \subseteq S[k]$. A singleton pattern is a single event $e \in \Omega$ and $\mathcal{P}$ is the set of all non-singleton patterns. A minimal window for a pattern is an interval in the data in which a pattern occurs which cannot be shortened while still containing the whole pattern [86]. In the remainder of this chapter when we use the term pattern occurrence we always expect it to be a minimal window. Further, we use $occs(X, S)$ for the disjoint set of all occurrences of X in S.

DITTO (Chapter 5) operates on categorical data. To find patterns in continuous real-valued time series we first have to discretise it. In Chapter 2 we discussed SAX, which is a celebrated approach for doing so, and we will use it in our experiments – though we note that any discretisation scheme can be employed.

7.3 Anomalies in Complex Event Sequences

Opposed to Chapter 6, which discusses anomalies in transaction data, we do not consider the rather trivial class 0 anomalies in this chapter. Instead, we focus on the more interesting class 1 and class 2 anomalies.

Class 1 Anomalies

For smart condition monitoring the anomalies we are interested in are those cases where the monitored system starts to behave differently from expected. In compression terms this means that our compression ratio on the multivariate data stream starts to get worse. So, following the approach of [78] we can use DITTO to identify such anomalies. A first intuitive approach is to rank all sequences according to their encoded length, given the code table on the whole dataset, proportional to their length in time steps.

Definition 10. *The anomaly score for a single sequence of multi-events S can be computed by dividing the encoded length of the sequence given the code table on the whole dataset CT by the length of the sequence $t(S)$.*

$$score_1(S) = \frac{L(S \mid CT)}{t(S)} \quad .$$

However, often the anomalies need to be discovered in long data streams that for the largest part reflect normal behaviour. Therefore we also propose a sliding window based approach, which computes a score for each window of the data. Using windows has two potential drawbacks, both caused by the fact that a window is typically small compared to the complete sequence.

The first potential problem is that because windows are typically small a code table computed on a window only will hardly be representative for the whole sequence. We mitigate this problem by computing the encoded length of a window using the code table computed on the whole data.

The second potential problem is that the cover of the sequence depends very much on the arbitrary starting point of the data, i.e., the moment one starts measuring. While for large sequences this does not matter too much – detrimental effects of a bad starting point are averaged out – it could have a large effect on the encoding of a small sequence. To mitigate this problem we compute covers starting from all possible starting points; starting from the first time step, the second time step, the third, etc. We stop, of course, when the cover we compute is identical to one we computed before – in practice this does not take more steps than twice the length of the longest pattern. To compute the encoded length of a window we take the minimum length over all these covers.

Another issue with the use of windows is that a cover may involve patterns that do not fit completely into the window. If we compute the encoded length of a window, such patterns should, of course, only be counted partially. That is, their contribution to the total size should

be proportional to their part that falls within the window. To compute this proportion we use the standard encoding ST.

Definition 11. *The anomaly score for a window w is the sum of the pattern code lengths for all occurrences that intersect the window. These pattern code lengths are weighted proportionally to the part of the ST code length of the events that fall within the window.*

$$score_1(w) = L(w \mid CT) = \sum_{X \in CT} \sum_{o \in occs(X,S)} \begin{cases} L(code_p(X \mid CT)) \times P(X, o, w) & \text{if } o \cap w \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Here $P(X, o, w)$ represents the part of pattern X for which its occurrence o falls within window w with respect to the ST code lengths of the events in X, defined as follows

$$P(X, o, w) = \frac{\sum_{\{e \in X | e \in o, e \in w\}} L(code_p(e \mid ST))}{\sum_{\{e \in X\}} L(code_p(e \mid ST))} \quad .$$

For ease of interpretation we consider the univariate example from Figure 7.1 where we show a sequence S which is covered using a code table comprising the patterns X, Y, and the singleton events. Now we can compute the anomaly score for the first window w, which spans the first 8 time steps, as follows.

$$\begin{aligned} score_1(w) = {} & L(code_p(X \mid CT)) + L(code_p(b \mid CT)) + \\ & L(code_p(Y \mid CT)) + L(code_p(c \mid CT)) + \\ & L(code_p(Y \mid CT)) \times \frac{L(code_p(a \mid ST)}{L(code_p(Y \mid ST)} \end{aligned}$$

Algorithm

To build a ranking based on window or sequence based class 1 anomalies we use Algorithm 16. It simply scores all windows or sequences based on our definitions and returns the ranked results.

New data

The window based approach to identify anomalies using the encoded length of the data can also be applied in a streaming setting. In fact, this is exactly what is needed for our smart

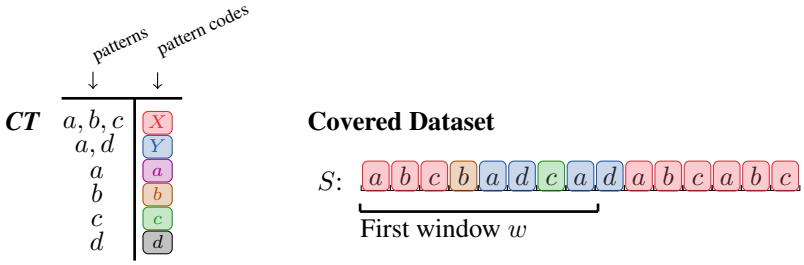


Figure 7.1: **Example** for class 1 anomaly scores on windows.

Algorithm 16 Class 1 Anomaly Detection

Input: A dataset D and minimum support min_sup
Output: A ranking of sequences or windows

1: $CT \leftarrow$ DITTO(D, min_sup)
2: $ranking \leftarrow \emptyset$ // Ordered on score descendingly
3: **for** $S \in D$ or $w \in D$ **do**
4: $\quad score \leftarrow score_1(S)$ or $score \leftarrow score_1(w)$
5: $\quad ranking \leftarrow ranking \cup (score, S)$
6: **return** ranking

condition monitoring use-case. Hence, we give a brief description on how to detect anomalous behaviour as soon as it starts to happen.

Firstly, we compute the code table on all data. For each new window of data that arrives we compute its score and compare it with the scores for all previous windows.

There is one problem, however, and that is that we do not know how the sequence will continue. This is a problem for (possible) patterns in the cover that do not fit into the window. For example, consider the sequence of Figure 7.2, which is similar to the sequence from Figure 7.1, but only the first window of data is known. Given the code table CT from Figure 7.1 the first 7 time steps of the window will be covered exactly the same regardless of the data that follows after time step 8. The event a in time step 8, however, could be covered by X if events b and c were to follow in the next time steps, or it could be covered by Y (as is the case Figure 7.1), or it could be covered by the singleton a.

Since we do not know what the future will be, we compute the *expected* anomaly score rather than the true one. We can compute this expectation because the pattern code lengths in CT are based on the probability that that pattern occurs in a cover. Hence we can account for each possible future proportional to the probability it will occur. Given that such possible covers may include events that are outside our window, we have to weigh them – as above – for their addition to the total encoded window size. This leads to the following score for the example window from Figure 7.2.

$$
\begin{aligned}
score_1(w) = {} & L(code_p(X \mid CT)) + L(code_p(b \mid CT)) + \\
& L(code_p(Y \mid CT)) + L(code_p(c \mid CT)) + \\
& sc(X, Z) \times L(code_p(X \mid CT)) \times \frac{L(code_p(a \mid ST)}{L(code_p(X \mid ST)} + \\
& sc(Y, Z) \times L(code_p(Y \mid CT)) \times \frac{L(code_p(a \mid ST)}{L(code_p(Y \mid ST)} + \\
& sc(a, Z) \times L(code_p(a \mid CT)) \quad ,
\end{aligned}
$$

where $Z = \{X, Y, a\}$ contains a pattern corresponding to each possible scenario for a single event, the function sc is defined as

$$
sc(X, Z) = \frac{2^{-L(code_p(X|CT))}}{\sum_{Y \in Z} 2^{-L(code_p(Y|CT))}} \quad .
$$

That is, $sc(X, Z)$ describes the probability on the scenario where pattern X is used to cover the event.

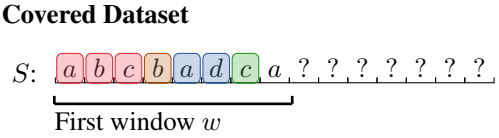


Figure 7.2: **Example** for class 1 anomalies on streaming data.

Class 2 Anomalies

Our class 2 anomalies are unlikely co-occurrences of two patterns in a sequence, i.e., we did not expect X and Y to occur close to each other. For example, in the smart condition monitoring use-case, first the power-consumption in the monitored system goes down followed by a rise in the temperature. This may be totally normal (the cooling system shut down) or indicate something harmful like a fire; note that these two are easily distinguished by the absolute temperature.

Note that both X and Y may reflect completely standard behaviour, i.e., a window containing both X and Y compresses well. Hence, we need a second anomaly score to discover such unexpected co-occurrences.

The score is based on the class 2 anomaly score from Chapter 6 for transactions and is potentially a very costly score to compute since it requires access to the support of all (very)

low support itemsets. The crucial observation in [16] is, however, that it is enough to consider only the itemsets in the code table CT. The intuition is that MDL gives us characteristic patterns and, thus, anomalies will (also) be anomalous with regard to these characteristic patterns only.

To translate this score to our multivariate sequence setting, we again use windows. The reason is simply that only the unexpected co-occurrence of patterns close to each other can be reasonably deemed an anomaly.

Definition 12. *The anomaly score of a pair of patterns X and Y is given by*

$$score_2(t) = \max_{\{X,Y \in PS | X,Y \subseteq t\}} -\log P(XY) + \log\big(P(X) \times P(Y)\big) \ .$$

Where $P(X)$ is given by the relative number of windows w in which X occurs at least once, that is we have

$$P(X) = \frac{|\{w \in windows \mid X \subseteq w\}|}{|windows|}$$

and, $P(X, Y)$ is defined similarly for windows containing both X and Y.

Note that we compute the probabilities directly from the windows rather than from the code lengths in CT. The reason is that the probability that X occurs in a window w depends as much on the size of W as it does on the number of times X (and its super patterns!) occur in the cover of D. While in theory one could correct for this, it is far easier to simply count the number of windows in which X occurs.

Moreover, note that we do not take into account whether or not X occurs multiple times in w. The reason is that this is already done indirectly, because multiple occurrences of X in one window w imply that X will occur in more windows than a single occurrence of X in w would entail.

Algorithm

To rank pattern co-occurrences based on their class 2 anomaly score we use Algorithm 17. It scores all pattern combinations from the pattern set returned by DITTO, excluding singletons with a support below min_sup and patterns that are longer than the specified window length.

Window size

As usual in sequence mining, both our scores, $score_1$ and $score_2$, have a parameter, viz., the window size. The window size not only determines the anomalies we can discover, it also influences the actual score an anomaly gets. So, what window size should one choose?

Again, as usual in sequence mining there is no definite answer to this question. It mostly boils down to: it depends on what you want to discover. For $score_1$, a too big window size will average out "small" anomalies that would get caught by a smaller window size. On the other hand, a small window size may signal too many potential anomalies, e.g., all local maxima

Algorithm 17 Class 2 Anomaly Detection

Input: A dataset D, minimum support min_sup, and window length $t(w)$
Output: A ranking of anomalous co-occurrences

1: $CT \leftarrow$ DITTO(D, min_sup)
2: $PS \leftarrow$ first column of CT
3: $PS \leftarrow \{p \in PS \mid support(p) \geq min\_sup$ **and** $t(p) \leq t(w)\}$ // Prune singletons below min_sup and patterns longer than $t(w)$
4: $ranking \leftarrow \emptyset$ // Ordered on score descendingly
5: **for** $X \in PS$ **and** $Y \in PS$ **do**
6: **if** $support(X, Y) = 0$ **or** $X = Y$ **then continue**
7: **if** $\{Z \in CT \mid X \subseteq Z, Y \subseteq Z\} \neq \emptyset$ **then continue**
8: $score = -\log(P(X, Y)) + \log(P(X) \times P(Y))$
9: **if** $score > 0$ **then**
10: $ranking \leftarrow ranking \cup (score, (X, Y))$
11: **return** ranking

and minima in a time series. Next to tests with different window sizes, application domain expertise seems the only answer to the question.

For $score_2$, the window size defines what we find a co-occurrence of patterns X and Y. A window should at least allow for non-interleaving occurrences of X and Y to make sense. Given that occurrences may have gaps, 4 times the number of time steps of the "largest" pattern (in terms of time steps) in CT seems a reasonable lower bound. Again, tests with different window sizes as well as application domain expertise should provide the ultimate answer.

7.4 Related Work

Anomaly detection is an active and large field in the data mining literature, far too large to cover here. The best recent overview is undoubtedly given by [2]. For a good overview of the closely related field of novelty detection, see [73].

To the best of our knowledge there is no work on anomaly detection in multivariate data streams. There is research in the univariate case, but that does not translate straightforwardly to the multivariate case. After all, correlations between the different variables get lost by projections to single variables.

There is a lot of research on anomaly detection in multivariate time series; see [36] and [24] for an overview. Techniques used range, e.g., from projection pursuit based methods ([32] and [109]), to kernel-based methods ([5] and [26]), to independent component based methods ([10]). There is, however, a crucial difference between our approach and these time series approaches.

The time-series approaches focus on the *values* the variables on the multivariate time series take. They aim to identify data points that are outside the expected. While $score_1$ may catch such anomalies if they cause bad compression, it is not its aim. Rather, both $score_1$ and $score_2$

aim to discover anomalous *behaviour*; the values the variables take may be completely normal. Hence, our scores do not compete with these time series approaches, they identify a different type of anomaly and could be used in parallel.

Finally, our work builds on [15], [16], and [78]. It differs from them by detecting anomalies in multivariate event sequences, a topic none of these papers is concerned with.

7.5 Experiments

Our implementation is made available for research purposes, together with the code to generate and perform experiments on synthetic data.[2]

We performed a wide range of synthetic experiments to test how well, and under what conditions, our methods can discover planted anomalies. Moreover, we performed experiments on real world data to illustrate the usefulness of our two classes of anomalies. For the latter we always set the minimum support threshold as low as possible, unless domain knowledge suggests otherwise.

In our report on these experiments we do not compare with other methods. As discussed in Section 7.4 related methods in multivariate time series anomaly detection focus on different types of anomalies. The anomalies we detect are meant to complement those found by other methods, not to surpass them.

Synthetic Experiments

Class 1 Anomalies

To study the robustness of the class 1 anomaly score on windows of data we performed the following experiment. We generated datasets containing 10 000 multi-events over 3 attributes, each with an alphabet of 25. In these datasets we inserted 10 patterns (randomly generated) with a size of 5 events and a support of 300. Note that this makes that 50% of the events in the dataset is part of one of these patterns. Of course, these patterns showed up in the CT computed by DITTO.

Only 1 window of length 32 was kept clean of patterns, consisting of random events only. This is the test window. With a window length of 32 we have a total of 9 969 different windows which we all score using our class 1 anomaly score. The average time that DITTO needed was 8 seconds and the anomaly scoring was completed within a single second. The test window was ranked first (highest anomaly score) in each of the 10 repeated runs that we performed. This result is visualised with the most left red dot in both graphs of Figure 7.3a. The bottom plot shows the average anomaly score of the test window over multiple runs and the top plot shows where it ranks compared to all other windows. As the most left red dot in these plots corresponds to a test window without any patterns we assign it a regularity of 0, as also indicated on the x-axis. To study how our score performs when the test window looks more like all other data we varied the regularity from 0 to 1. That is, from completely random

[2] Code – `http://eda.mmci.uni-saarland.de/raar/`

to containing the same amount of patterns as the rest of the data. Figure 7.3a shows that we can easily identify the test window until it is about 80% similar to the rest of the data.

We repeated the same experiment for a dataset comprising 1 test and 500 normal sequences, all of length 50, using the class 1 anomaly score for sequences of data. Moreover, we planted 10 random patterns of size 5 with a support of 750 in the normal sequences and we varied the amount of patterns in the test sequence. The average runtime of DITTO was 74 seconds and the anomaly scoring finished in about 7 seconds. The results are very similar to the window based setting and are plotted in Figure 7.3b.

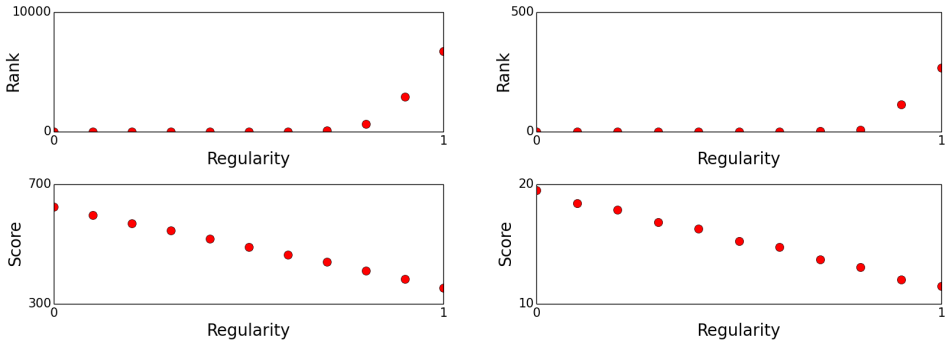


Figure 7.3: The rank and anomaly score of the test **window** (a) or **sequence** (b) while varying its contents from random (regularity = 0) to similar to the rest of the data (regularity = 1). The plots show averages over 10 runs.

Class 2 Anomalies

For class 2 anomalies we generated synthetic data based on randomly generated patterns as before. For each anomaly, we planted 2 new, different (randomly generated) generator patterns both in another half of the data. Thereafter, we planted both generator patterns once again but together in an additional sequence; this is the anomaly.

Ideally, each anomaly, a co-occurrence of two corresponding generator patterns, gets the highest anomaly score considering all co-occurrences of patterns in the data. When the data contains 5 anomalies and thus 10 generator patterns, we expect them to be ranked as the top 5. In the 9^{th} and 10^{th} row of Table 7.1 we find that the average rank of the 5 anomalies is 2. This indicates that the 5 planted anomalies were ranked at the top five positions 0, 1, 2, 3 and 4. Table 7.1 further shows that the anomalies are always ranked at the top regardless of the number of anomalies, the number of attributes, the alphabet size per attribute, the number of planted patterns and the support of the anomaly generators.

Note that in Table 7.1 *support* indicates the percentage of events that are covered by all occurrences of the pattern. For example, when we have 100 multi-events over 3 attributes, and a pattern X with size $||X|| = 5$ and a *support* of 10%, this means that X occurs $\frac{0.10 \times 3 \times 100}{5} = 6$ times. That is, its 6 occurrences span $6 \times 5 = 30$ events of the data. For an experiment with 2 generator patterns with a support of 10% and 40 random patterns with a support of 1% we thus have a dataset where 60% of the data is covered with patterns.

Table 7.1: The class 2 anomaly score always ranks the planted anomalies at the top regardless of the number of anomalies, the number of attributes, the alphabet size per attribute, the number of planted patterns and the support of the anomaly generators. Ranks and times are averages over 20 runs.

Data			**Planted Generators**			**Planted Patterns**			**Ranking**		
										Time (sec)	
$t(D)$	$\|A\|$	$\|\Omega_i\|$	$\|\mathcal{P}\|$	$\|\|X\|\|$	*support*	$\|\mathcal{P}\|$	$\|\|X\|\|$	*support*	rank anomaly	DITTO	Anomaly scoring
10 500	5	25	2	5	1%	40	5	1%	0 of 3 527	716	235
10 500	5	25	2	5	2%	40	5	1%	0 of 3 449	812	246
10 500	5	25	2	5	4%	40	5	1%	0 of 3 789	1 251	243
10 500	5	25	2	5	6%	40	5	1%	0 of 4 301	2 065	234
10 500	5	25	2	5	8%	40	5	1%	0 of 4 659	3 301	235
11 500	3	25	6	5	1%	40	5	1%	1 of 2 436	785	124
11 500	3	25	6	5	2%	40	5	1%	1 of 2 406	845	123
11 500	3	25	6	5	4%	40	5	1%	1 of 2 842	1481	154
12 500	5	25	10	5	1%	20	5	1%	2 of 4 584	472	336
12 500	5	25	10	5	1%	40	5	1%	2 of 3 772	1 488	325
10 500	10	25	2	5	1%	10	5	1%	0 of 6 640	159	934
10 500	10	25	2	5	2%	10	5	1%	0 of 6 733	160	912
10 500	10	25	2	5	1%	40	5	1%	0 of 7 470	1 121	968
10 500	5	50	2	5	1%	40	5	1%	0 of 13 929	497	296
10 500	5	50	2	5	2%	40	5	1%	0 of 13 974	407	282
10 500	3	100	2	5	1%	40	5	1%	0 of 20 677	187	100
10 500	3	100	2	5	2%	40	5	1%	0 of 19 209	143	94

Real World Experiments

To illustrate the ability of our anomaly scores to identify anomalies in multivariate event sequences we performed experiments on real world datasets.

Smart Condition Monitoring

First we show how our anomaly scores can be used for our example use-case, viz., smart condition monitoring. In our experiments, conducted in cooperation with Semiotic Labs,[3]

[3]Semiotic Labs – `www.semioticlabs.com`

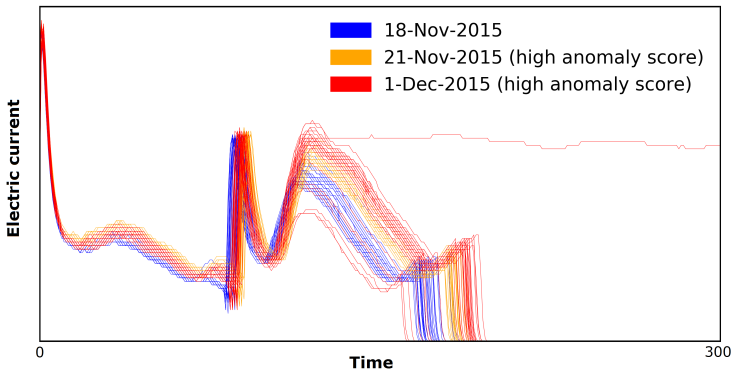


Figure 7.4: The electric current during all switch moments on 3 days. Although most plots look fairly similar, the ones on the 21^{th} of November and the 1^{st} of December receive much higher anomaly scores.

we used our anomaly scores to identify anomalies that may indicate that a railway switch is going to fail. Such a switch provides us with data about the electric current that flows through the electric motor for the entire duration of each switch moment. See Figure 7.4 for example graphs of these currents. We collected data over a period of 2 years. For each switch moment we recorded the area under the curve and the duration. This results in a dataset of 10 864 multi-events over 2 attributes. As the switch moves in 2 directions we have 2 separate datasets; one for each direction. We computed the class 1 anomaly scores for windows of length 8, which are plotted over time together with logged failures in Figure 7.5. Note that the maintenance log is not perfect and is likely to be incomplete. This means that we do not know what happened at the moment of all the peaks in the anomaly scores. However, in Figure 7.5 we clearly see a peak right before the failure. This information could have been used as an early warning to prevent this failure. Note that the anomaly is only visible in one of the two directions, which supplies the business with some more information on what might be wrong. To show that identifying a failure before it happens is not trivial, in Figure 7.4 we show (fairly similar) plots of the electric current for both 'normal' and very high anomaly scores. In Figure 7.6 we see that the plots for the 18^{th} of November correspond to 'normal' anomaly scores and the plots for the 21^{th} of November and the 1^{st} of December (day of the failure) correspond to very high anomaly scores. Moreover, using our anomaly scores we are able to identify this failure about 10 days ahead of time.

Many more such examples were found and are perceived as valuable by the business. The next (current) step is to investigate, together with domain experts, which patterns cause the high anomaly scores. Moreover, we will examine whether these patterns provide useful information about the type of failure.

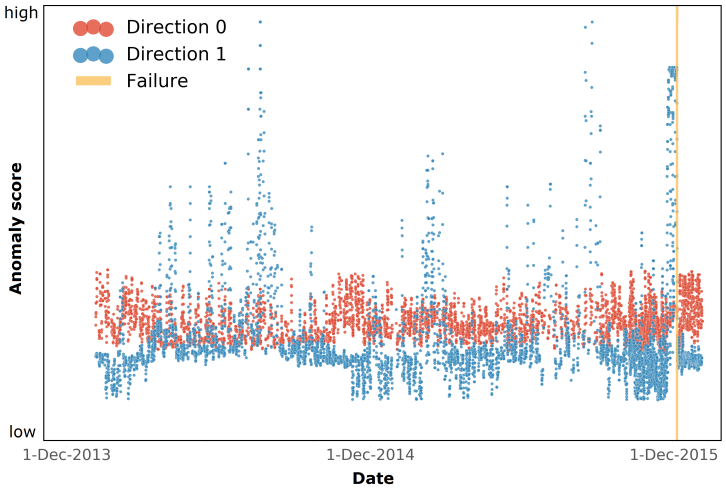


Figure 7.5: For both directions in which the switch moves (red and blue) we plot the class 1 anomaly scores (window length 8) over 2 years together with the failure (vertical bar).

Moby Dick

Our second dataset consists of the first chapter of the book Moby Dick[4] by Herman Melville. The dataset comprises 2 attributes: the original text and the part-of-speech tags[5,6] corresponding to each word in the text. For example,

attribute 1:	VB	PRP	NNP
attribute 2:	Call	me	Ishmael

for which we will use the following notation, in which each time step is enclosed by curly brackets and the symbols for different attributes within a time step are divided by a comma: {Call, VB}{me, PRP}{Ishmael, NNP}. The part-of-speech tags in this example are a verb (base form), a personal pronoun and a proper noun (singular), respectively.

Further, for this dataset each sentence is regarded as a sequence, which avoids the problem of finding spurious patterns spanning multiple sentences. In total the data comprises 4 496 events in 103 sequences over 2 attributes with alphabet sizes 31 and 856 for the tags and the text, respectively.

We use Ditto (Chapter 5) to compute a summary of the data. With a minimum support of 5 it takes 102 seconds to find a pattern set containing 79 non-singletons.

[4]Moby Dick – `www.gutenberg.org`
[5]Part-of-speech tags – `http://nlp.stanford.edu/software/tagger.shtml`
[6]Part-of-speech tags – `https://gate.ac.uk/wiki/twitter-postagger.html`

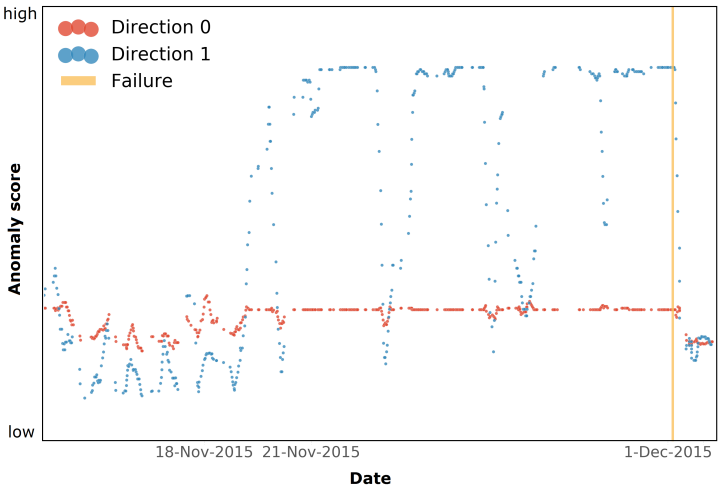


Figure 7.6: An enlargement of Figure 7.5, which shows 'normal' anomaly scores on the 18^{th} of November and very high anomaly scores on the 21^{th} of November and the 1^{st} of December.

Using the class 2 score we compute a ranking for all combinations, between the 966 patterns and singletons from the DITTO pattern set, that occur within a window of length 4. This takes only 1 second. In Table 7.2 we give an overview of two top ranked pattern combinations. Both are mistakes made by the tagger. Note that mistakes are a common reason for anomalies and hence it is good that they are discovered. Moreover, note that our class 1 anomaly scores do not rank these mistakes very high because all involved patterns are very frequent.

We performed an extra sanity check and manually added two generator patterns of size 2, {TAG1, word1} and {TAG2, word2}, with a support of 100, making them the 9^{th} to 12^{th} most frequent singletons. Further we added them once together in the same sequence (the anomaly). This single co-occurrence of these 2 generator patterns is correctly ranked first using our class 2 score.

Cycling

The last real world dataset comprises data recorded using multiple sensors during cycling. We consider speed and heart rate simultaneously. To be able to find interesting patterns we first pre-process the data from each sensor by transforming the absolute values to relative values (by replacing each value by the difference of its successor and its own value) and by discretising the resulting values into 8 intervals using SAX. We transform the data from absolute to relative values, because we are not interested in the actual speeds or heart rates, but by how much they are increasing or decreasing. The discretisation is of course necessary to construct a valid

Table 7.2: Top ranked examples of class 2 anomalies in chapter 1 of **Moby Dick**. The used tags are determiner (DT), adjective (JJ), noun singular/plural (NN/NNS), possessive pronoun (PRP$), and 3rd person singular present verb (VBZ).

Pattern X	Pattern Y	Fragment	Explanation
{DT, the}{NN}	{PRP$}	the (commonalty) lead their	'lead' is verb not noun
{JJ}{NNS}	{VBZ}	Patagonian sights (and) sounds	'sounds' is noun not verb

alphabet for each sensor, but also because we are only interested in the increase or decrease of a sensor's values at a course granularity.

We searched for anomalies on a training session of 50 km, corresponding to a dataset containing 2 844 events over 2 attributes. With a minimum support of 10 it took DITTO 17 seconds to find a summary containing 10 non-singleton patterns with a compression gain of over 20% compared to the set of singletons. Within 1 second we find 73 co-occurrences, within windows of length 4, containing the following top ranked results. We find a co-occurrence of a pattern of 4 time steps with constant speed together with a pattern describing a highly increasing heart rate. This co-occurrence has a high anomaly score because normally the heart rate is constant when the speed is constant. However, because the current speed was very high this was a special case (anomaly). Another highly ranked co-occurrence, of a pattern describing an increasing heart rate and another pattern describing a very highly decreasing heart rate, coincided with a fall in dull sand.

7.6 Discussion

The synthetic experiments show that our anomaly scores ably identify the targeted types of anomalies. Our class 1 scores successfully identify those parts of the data containing less or other patterns than the rest of the data. Further, our class 2 score shows high performance in identifying unexpected co-occurrences under a wide range of settings. Again, we want to emphasise that we do not claim to exhaustively identify all types of anomalies with a single method.

To verify the use of our scores in practice we also performed experiments on real world datasets. We showed that in the field of smart condition monitoring our scores can be very useful to predict failures ahead of time. Next to this early warning capability, our method can also supply additional information about the type of failure after further studying the patterns that comprise the anomaly. Since this is not an automated process and differs for each domain, we leave this step to domain experts. Future work would be to combine the domain expert's knowledge with our algorithms to classify or cluster anomalies.

Further support for the descriptiveness of our approach to detect anomalies can be found in the Moby Dick dataset, see Table 7.2. Because our score exactly highlights the patterns

responsible for the high anomaly score we discovered that these anomalies were caused by mistakes made by the tagger.

7.7 Conclusion

Anomaly detection is an important data mining task as the occurrence of (very) low probability events may provide valuable information [2]. A case in point is smart condition monitoring, where one tries to predict the optimal time-point to service monitored equipment. If equipment failures are preceded by unexpected behaviour – as measured by monitoring sensors – of the equipment, the detection of such anomalies may help in predicting the imminent failure.

Since systems are usually monitored by multiple sensors, such as temperature and power-usage, anomaly detection in multivariate event sequences is required. After all, anomaly detection for each sensor separately will miss correlations between the sensor readings and, thus, miss some anomalies. Consider for example the case where we have both the above mentioned temperature and power-usage sensors. It may be completely normal for the temperature to go up and for the power-usage to go down. However, it may be anomalous if the temperature goes down while the power-usage goes down. This may, e.g., signal a failure in the cooling system.

In this chapter we introduce two window based anomaly scores, prosaically called $score_1$ and $score_2$, to detect anomalies in multivariate event sequences, as well as two algorithms to do the actual discovery. The scores and the algorithms are based on summaries of the data as produced by the DITTO algorithm [15]. These summaries consist of code tables, comprising a set of multivariate patterns that collectively describe the data well.

With experiments we show firstly that our methods are able to discover the anomalies they were devised for. More specifically we show on synthetic data that the anomalies we plant are discovered easily even under rather adverse conditions; i.e., when the anomalies are not all that different from the rest of the data.

Secondly, using real world data we show that the anomalies we discover are useful. On railway switch data we discover anomalies that precede failure of that switch by up to 10 days. Further investigation of the use of these patterns for condition based maintenance is currently under way. On a tagger annotated version of the celebrated novel Moby Dick we show how the anomaly scores are able to pinpoint mistakes made by the tagger. Finally, on data from a cycling trip our methods were able to discover the fall from a bike as an anomaly.

CHAPTER 8

Conclusions

Before we conclude this thesis let us first recall our motivation and goals. In the Introduction we stated that data is collected everywhere, but remains useless until information can be extracted from it. One of the key challenges in data mining is to gain insight in this data through the identification of interesting patterns.

Building on works in sequence mining (Chapter 2) and the successful application of MDL in data summarisation (Chapter 3) the first goal of this thesis is to extend the state-of-the-art in summarisation to multivariate sequential data. To this end, we posed the following research problem:

How to summarise multivariate sequential data in terms of easily understandable patterns that are able to capture multivariate structure.

That is, we want to gain insight in this multivariate data by presenting a domain expert with only a small set of patterns that together succinctly describe the whole dataset. Moreover, these patterns must be able to capture multivariate structure.

From literature we know that the resulting summaries can be used for other tasks on top of providing a first insight into the data. We can, for example, use them to characterise the difference [95], identify the components [52], preserve privacy [96] or detect anomalies [78] in a dataset. Besides summarisation, in this thesis, we also focused on on anomaly detection using these summaries. This led to the second research problem as follows:

How to identify and characterise unexpected behaviour in multivariate sequential datasets.

That is, we want to devise methods to automatically detect and, as a bonus, explain anomalous data.

In addressing these research problems the main contributions of this thesis can be summarised as follows:

- We showed how MDL can be employed to characterise a seismogram by a small set of characteristic patterns. The resulting code sets provide insight into the data and can also be used to compare different seismograms. That is, given a set of seismograms for which we know what events (earthquake, passing truck, etc.) happened we can identify the events in a new seismogram. Further, a code set computed for a cluster of seismograms can be used to generate a synthetic seismogram that shows what all these seismograms have in common.

- We extended the state-of-the-art in summarising sequence data to multivariate event sequences. We used a very rich pattern language that is able to capture multivariate structure. Further, the pattern language allows for gaps in both directions (time and attributes) and we allow patterns to interleave when describing the data. We introduced the heuristic DITTO algorithm that efficiently mines a small set of patterns that gives a succinct description of the dataset, possibly containing multivariate patterns. The performance and quality of the summaries that DITTO produces is validated by a wide range of experiments both on synthetic and real world datasets.

- We extended the field of anomaly detection by introducing a new class of anomalies in transaction data. We showed that these unexpected co-occurrences of patterns are indeed anomalous and from the experimental section we also find that they can provide useful insight in real world scenarios. Further, we introduced an algorithm that efficiently discovers and ranks transactions based on these anomalies, improving enormously over a naive approach.

- Finally, we presented two window based anomaly scores and algorithms to identify these anomalies in multivariate event sequence data. With experiments we show firstly that our methods are able to discover the anomalies they were devised for. Secondly, using real world data we show that the anomalies we discover are useful. As a result of our pattern based approach our algorithms can automatically provide additional insight to explain an anomaly by the patterns that it comprises.

Looking at these contributions we can conclude that the MDL principle can be successfully employed in the domain of multivariate sequential data. Both for summarisation and anomaly detection successful algorithms have been introduced.

Experimental results on real world datasets, such as a text in different languages, show that we are able to capture true multivariate structure, for example in the form of patterns that describe translations between the different languages. Further, our experiments show that as a result of DITTO's rich pattern language we attain much higher compression ratios, using a comparable number of patterns, compared to the state-of-the-art in summarising univariate sequential data.

Among many of the convincing results for our experiments in anomaly detection we identified mistakes made by a tagger for text data. In this example the dataset contained two aligned sequences, one containing the words of the first chapter of the book Moby Dick, and the second containing tags describing the function for each word. The mistakes made by the

tagger became clear as they were caused by unexpected combinations of patterns, exactly the new class of anomalies that we introduced in this thesis. Note that these mistakes consist of very regular data (frequent patterns) and are therefore not easily identified by previous approaches.

For multivariate sequential data a very successful result was found on a dataset comprising two features extracted from a sensor that monitors the electric current that flows through an electric motor that moves a railway switch. In this data we identified unexpected behaviour for about 10 days up to the time of a failure of the electric motor. This information could have been used as an early warning to service this motor in time thereby preventing costly downtime.

In addition to our results we would like to note that an anomaly may occur as a result of an error, it can be an outlier, or it can be a highly unexpected data point. In any case, it can lead to insight after being presented to a domain expert. As described in the earlier chapters, there is not a single score to identify all anomalies, thus we must always consider all available approaches to get the most complete view on the data.

We started with summarisation and ended with anomaly detection, but what about the future? There are of course ample opportunities to continue the presented work. It would, for example, be nice to be able to work on continuous data directly without the need to discretise it. The extension to this domain, however, is not apparent. More feasible improvements include allowing for noise in patterns. That is, when the pattern abc may also cover the instances abd or abe with a penalty based on the amount of noise. As a result we might be able to attain even better compression ratios and thus expose even more information. The challenge is again to control the search space that grows even larger as a result of the extra choices in covering the data. Finally, further improvements may result from the use of prequential codes [35] in describing a dataset and the parallelisation of the cover and candidate generation phases of DITTO. Exploring these possibilities may lead to even more insight in information.

Bibliography

1. A. Achar, S. Laxman, R. Viswanathan, and P. Sastry. Discovering injective episodes with general partial orders. *Data Mining and Knowledge Discovery*, 25(1):67–108, 2012.
2. C. C. Aggarwal, editor. *Outlier Analysis*. Springer, 2013.
3. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.
4. R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, pages 3–14. IEEE Computer Society, 1995.
5. T. Ahmed, M. Coates, and A. Lakhina. Multivariate online anomaly detection using kernel recursive least squares. In *INFOCOM*, pages 625–633, 2007.
6. H. Akaike. Information theory as an extension of the maximum likelihood principle. In *Second International Symposium on Information Theory*, 1973.
7. K. Aki and P. G. Richards. *Quantitative seismology*, volume 1. 2002.
8. L. Akoglu, H. Tong, J. Vreeken, and C. Faloutsos. COMPREX: Compression based anomaly detection. In *CIKM*. ACM, 2012.
9. J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *KDD*, pages 429–435. ACM, 2002.
10. R. Baragona and F. Battaglia. Outliers detection in multivariate time series by independent component analysis. *Neural Comp.*, 19(7):1962–1984, 2007.
11. R. Bathoorn, A. Koopman, and A. Siebes. Reducing the frequent pattern set. In *ICDM-Workshop*, pages 55–59, 2006.
12. R. Bayardo. Efficiently mining long patterns from databases. In *SIGMOD*, pages 85–93, 1998.
13. R. Bertens and A. Siebes. Characterising seismic data. In *SDM*, pages 884–892. SIAM, 2014.
14. R. Bertens and A. Siebes. Characterising seismic data. Technical Report UU-CS-2014-002, Department of Information and Computing Sciences, Utrecht University, 2014.

15. R. Bertens, J. Vreeken, and A. Siebes. Keeping it short and simple: Summarising complex event sequences with multivariate patterns. In *KDD*, pages 735–744. ACM, 2016.

16. R. Bertens, J. Vreeken, and A. Siebes. Efficiently discovering unexpected pattern-co-occurrences. In *SDM*. SIAM, 2017.

17. M. Boley, C. Lucchese, D. Paurat, and T. Gärtner. Direct local pattern sampling by efficient two-step random procedures. In *KDD*, pages 582–590. ACM, 2011.

18. S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *SIGMOD*, pages 265–276. ACM, 1997.

19. B. Bringmann and A. Zimmermann. The chosen few: On identifying valuable patterns. In *ICDM*, pages 63–72, 2007.

20. K. Budhathoki and J. Vreeken. The difference and the norm – characterising similarities and differences between databases. In *ECML PKDD*, pages 206–223. Springer, 2015.

21. T. Calders and B. Goethals. Non-derivable itemset mining. *Data Mining and Knowledge Discovery*, 14(1):171–206, 2007.

22. A. C. Cameron, J. B. Gelbach, and D. L. Miller. Bootstrap-based improvements for inference with clustered errors. *The Review of Economics and Statistics*, 90:414–427, 2008.

23. G. J. Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM (JACM)*, 13(4):547–569, 1966.

24. V. Chandola. *Anomaly Detection for Symbolic Sequences and Time Series Data*. PhD thesis, University of Minnesota, September 2009.

25. Y.-C. Chen, J.-C. Jiang, W.-C. Peng, and S.-Y. Lee. An efficient algorithm for mining time interval-based patterns in large database. In *CIKM*, pages 49–58. ACM, 2010.

26. H. Cheng, P.-N. Tan, C. Potter, and S. A. Klooster. Detection and characterization of anomalies in multivariate time series. In *SDM*, pages 413–424. SIAM, 2009.

27. B. Chiu, E. Keogh, and S. Lonardi. Probabilistic discovery of time series motifs. In *KDD*, pages 493–498. ACM, 2003.

28. T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience New York, 2006.

29. B. Crémilleux and J.-F. Boulicaut. Simplest rules characterizing classes generated by δ-free sets. In *KBSAAI*, pages 33–46, 2002.

30. I. Daubechies et al. *Ten lectures on wavelets*, volume 61. SIAM, 1992.

31. T. De Bie. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. *Data Mining and Knowledge Discovery*, 23(3):407–446, 2011.

32. P. Galeano, D. Peña, and R. S. Tsay. Outlier detection in multivariate time series by projection pursuit. *JASA*, 101(474):654–669, 2006.

33. F. Geerts, B. Goethals, and T. Mielikäinen. Tiling databases. In *International Conference on Discovery Science*, pages 278–289. Springer, 2004.
34. G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford university press, 2001.
35. P. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.
36. M. Gupta, J. Gao, C. Aggarwal, and J. Han. Outlier detection for temporal data: A survey. *IEEE TKDE*, 26(9):2250–2267, Sept 2014.
37. J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.
38. J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *KDD*, pages 355–359. ACM, 2000.
39. M. A. Hasan and M. Zaki. Musk: Uniform sampling of k maximal patterns. In *SDM*, pages 650–661. SIAM, 2009.
40. Y. He, J. Wang, and L. Zhou. Efficient incremental mining of frequent sequence generators. In *DASFAA*, pages 168–182, 2011.
41. Z. He, X. Xu, J. Z. Huang, and S. Deng. Fp-outlier: Frequent pattern based outlier detection. *Computer Science and Information Systems*, 2(1):103–118, 2005.
42. H. Heikinheimo, J. Vreeken, A. Siebes, and H. Mannila. Low-entropy set selection. In *SDM*, pages 569–580, 2009.
43. E. T. Jaynes. On the rationale of maximum-entropy methods. *Proceedings of the IEEE*, 70(9):939–952, 1982.
44. A. J. Knobbe and E. K. Ho. Pattern teams. In *PKDD*, pages 577–584. Springer, 2006.
45. Y. S. Koh and S. D. Ravana. Unsupervised rare pattern mining: A survey. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10(4):45, 2016.
46. A. Kolmogorov. Three approaches to the quantitative definition of information. *Problemy Peredachi Informatsii*, 1(1):3–11, 1965.
47. P. Kontkanen and P. Myllymäki. MDL histogram density estimation. In *AISTATS*, 2007.
48. H. T. Lam, F. Mörchen, D. Fradkin, and T. Calders. Mining compressing sequential patterns. In *SDM*, pages 319–330, 2012.
49. H. T. Lam, F. Mörchen, D. Fradkin, and T. Calders. Mining compressing sequential patterns. *Statistical Analysis and Data Mining*, 7(1):34–52, 2014.
50. N. Lavrač, P. Flach, and B. Zupan. Rule evaluation measures: A unifying view. In *International Conference on Inductive Logic Programming (ILP)*, pages 174–185, 1999.
51. S. Laxman, P. Sastry, and K. Unnikrishnan. Discovering frequent episodes and learning hidden markov models: A formal connection. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1505–1517, 2005.
52. M. van Leeuwen, J. Vreeken, and A. Siebes. Identifying the components. *Data Mining and Knowledge Discovery*, 19(2):173–292, 2009.

53. M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, 1993.

54. T. W. Liao. Clustering of time series data - a survey. *Pattern Recognition*, 38(11):1857–1874, 2005.

55. J. Lin, E. Keogh, P. Patel, and S. Lonardi. Finding motifs in time series. In *Proceedings of the KDD Workshop on Temporal Data Mining*, pages 53–68, 2002.

56. J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing sax: A novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

57. M. Mampaey, J. Vreeken, and N. Tatti. Summarizing data succinctly with the most informative itemsets. *ACM TKDD*, 6:1–44, 2012.

58. H. Mannila and C. Meek. Global partial orders from sequential data. In *KDD*, pages 161–168, 2000.

59. H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *KDD*, pages 146–151, 1996.

60. H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *KDD*, pages 181–192, 1994.

61. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences extended abstract. In *KDD*, 1995.

62. 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.

63. M. Markou and S. Singh. Novelty detection: a review. *Signal processing*, 83(12), 2003.

64. D. Minnen, C. Isbell, I. Essa, and T. Starner. Detecting subdimensional motifs: An efficient algorithm for generalized multivariate pattern discovery. In *ICDM*, pages 601–606. IEEE, 2007.

65. T. Mitchell-Jones. Societas europaea mammalogica. http://www.european-mammals.org, 1999.

66. F. Mörchen and A. Ultsch. Efficient mining of understandable patterns from multivariate interval time series. *Data Mining and Knowledge Discovery*, 15(2):181–215, 2007.

67. K. Narita and H. Kitagawa. Outlier detection for transaction databases using association rules. In *The Ninth International Conference on Web-Age Information Management*, pages 373–380, July 2008.

68. T. Oates and P. R. Cohen. Searching for structure in multiple streams of data. In *ICML*, pages 346–354. Morgan Kaufmann, 1996.

69. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT*, pages 398–416. ACM, 1999.

70. J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 17th International Conference on Data Engineering*, pages 215–224, 2001.

71. J. Pei, J. Han, B. Mortazaviasl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE TKDE*, 16(11):1424–1440, 2004.

72. G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. Frawley, editors, *Knowledge Discovery in Databases*. AAAI/MIT Press, 1991.

73. M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, 2014.

74. C. Riggelsen, M. Ohrnberger, and F. Scherbaum. Dynamic bayesian networks for real-time classification of seismic signals. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 565–572. Springer, 2007.

75. J. Rissanen. Modeling by shortest data description. *Automatica*, 14(1):465–471, 1978.

76. J. Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of statistics*, 11(2):416–431, 1983.

77. A. Siebes, J. Vreeken, and M. van Leeuwen. Item sets that compress. In *SDM*, pages 393–404. SIAM, 2006.

78. K. Smets and J. Vreeken. The odd one out: Identifying and characterising anomalies. In *SDM*, pages 804–815. SIAM, 2011.

79. K. Smets and J. Vreeken. SLIM: Directly mining descriptive patterns. In *SDM*, pages 236–247. SIAM, 2012.

80. R. J. Solomonoff. A formal theory of inductive inference, part 1 and part 2. *Information and Control*, 7:1–22, 224–254, 1964.

81. X. Song, M. Wu, C. Jermaine, and S. Ranka. Conditional anomaly detection. *TKDE*, 19(5):631–645, 2007.

82. R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *International Conference on Extending Database Technology*, pages 1–17. Springer, 1996.

83. Z. R. Struzik and A. Siebes. The haar wavelet transform in the time series similarity paradigm. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 12–22. Springer, 1999.

84. P. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *KDD*, pages 32–41. ACM, 2002.

85. Y. Tanaka, K. Iwamoto, and K. Uehara. Discovery of time-series motif from multi-dimensional data based on mdl principle. *Machine Learning*, 58(2–3):269–300, 2005.

86. N. Tatti. Significance of episodes based on minimal windows. In *ICDM*, pages 513–522, 2009.

87. N. Tatti and B. Cule. Mining closed episodes with simultaneous events. In *KDD*, pages 1172–1180, 2011.

88. N. Tatti and B. Cule. Mining closed strict episodes. *Data Mining and Knowledge Discovery*, 2011.

89. N. Tatti and J. Vreeken. Finding good itemsets by packing data. In *ICDM*, pages 588–597, 2008.

90. N. Tatti and J. Vreeken. The long and the short of it: Summarizing event sequences with serial episodes. In *KDD*, pages 462–470. ACM, 2012.

91. P. Tzvetkov, X. Yan, and J. Han. Tsp: Mining top-k closed sequential patterns. In *ICDM*, pages 347–354, 2003.

92. U. Vespier, A. J. Knobbe, S. Nijssen, and J. Vanschoren. MDL-based analysis of time series at multiple time-scales. In *ECML PKDD*, pages 371–386. Springer, 2012.

93. U. Vespier, S. Nijssen, and A. Knobbe. Mining characteristic multi-scale motifs in sensor-based time series. In *CIKM*, pages 2393–2398. ACM, 2013.

94. J. Vreeken. Causal inference by direction of information. In *SDM*. SIAM, 2015.

95. J. Vreeken, M. van Leeuwen, and A. Siebes. Characterising the difference. In *KDD*, pages 765–774, 2007.

96. J. Vreeken, M. van Leeuwen, and A. Siebes. Preserving privacy through data generation. In *ICDM*, pages 685–690. IEEE, 2007.

97. J. Vreeken, M. van Leeuwen, and A. Siebes. KRIMP: Mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

98. C. S. Wallace and D. M. Boulton. An information measure for classification. *The Computer Journal*, 11(2):185–194, 1968.

99. C. Wang and S. Parthasarathy. Summarizing itemset patterns using probabilistic models. In *KDD*, pages 730–735, 2006.

100. J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In *ICDE*, pages 79–90. IEEE, 2004.

101. G. Webb and J. Vreeken. Efficient discovery of the most interesting associations. *ACM TKDD*, 8(3):1–31, 2014.

102. C.-W. Wu, Y.-F. Lin, P. S. Yu, and V. S. Tseng. Mining high utility episodes in complex event sequences. In *KDD*, pages 536–544. ACM, 2013.

103. Y. Xiang, R. Jin, D. Fuhry, and F. F. Dragan. Succinct summarization of transactional databases: an overlapped hyperrectangle scheme. In *KDD*, pages 758–766, 2008.

104. X. Yan, H. Cheng, J. Han, and D. Xin. Summarizing itemset patterns: a profile-based approach. In *KDD*, pages 314–323, 2005.

105. X. Yan, J. Han, and R. Afshar. Clospan: Mining closed sequential patterns in large datasets. In *SDM*, pages 166–177, 2003.

106. B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. VLDB, 2000.

107. J. Zakaria, A. Mueen, and E. Keogh. Clustering time series using unsupervised-shapelets. In *ICDM*, pages 785–794. IEEE, 2012.

108. M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60, 2001.

109. J. Zhang, Q. Gao, and H. Wang. Spot: A system for detecting projected outliers from high-dimensional data streams. In *ICDE*, pages 1628–1631. IEEE, 2008.

Samenvatting

Sinds de opslag van data erg goedkoop is verzamelen bedrijven en instellingen enorm veel data, vaak nog zonder te weten wat ze ermee aan moeten. Het onderzoeksveld van gegevensanalyse, ook wel data mining genoemd, houdt zich bezig met het verkrijgen van informatie of inzicht vanuit al deze verzamelde data. Dit inzicht kan bijvoorbeeld gebruikt worden voor het beter voorspellen van natuurrampen, de optimalisatie van allerlei bedrijfsprocessen of het identificeren van fraude.

Een groot deel van de data die verzameld wordt is sequentiële data. Voorbeelden hiervan zijn: reeksen van uitgevoerde muisklikken of bezochte webpagina's op het internet, lijsten van gebeurtenissen afgegeven door alarmsystemen, teksten die bestaan uit een opeenvolging van woorden, en sensoren die van alles meten over een bepaalde tijdspanne. Om te leren van deze data bestuderen we de patronen die zich hierin bevinden. Zo kan bijvoorbeeld een patroon, dat een veel bezochte reeks van webpagina's beschrijft, inzicht bieden in hoe gebruikers door een website navigeren en waarom sommige pagina's vaker bekeken worden dan anderen.

Een bekend groot probleem in dit vakgebied is de enorme hoeveelheid patronen die elke dataset bevat. Vele oplossingen zijn bedacht om de omvang van de verzameling mogelijke patronen terug te dringen naar werkbare proporties. Dit kan door zoveel mogelijk redundantie uit de verzameling te verwijderen. Deze methodes leveren helaas nog steeds een veel te grote verzameling aan patronen. Als reactie hierop is er een andere aanpak ontwikkeld. In plaats van het zoeken naar een kleine selectie uit de verzameling van alle patronen, doen we een stap terug en richten we ons weer op de dataset zelf. Dat wil zeggen, we zoeken naar een kleine verzameling patronen die de gehele dataset goed samenvat. Deze samenvatting is dus een beschrijving van de dataset in plaats van een beschrijving van de totale set patronen in de dataset. Deze aanpak is gebaseerd op het zogenaamde Minimum Description Length (MDL) principe en zoekt naar de verzameling patronen die de beste compressie van de data geeft.

In dit proefschrift passen we het MDL principe toe op multivariate datareeksen. Dat wil zeggen, meerdere parallelle datareeksen. In deze data bevinden zich potentieel nog veel meer patronen, zoals patronen die een correlatie tussen de verschillende reeksen beschrijven. Denk aan een dataset waarbij twee sensoren op elk moment de snelheid en hoogte van een hardloper bijhouden. Een multivariaat patroon kan een correlatie tussen de sensoren beschrijven, bijvoorbeeld de correlatie tussen een toenemende snelheid en gelijktijdig afnemende hoogte van het gelopen traject. We definiëren hoe deze patronen eruit kunnen zien en introduceren een algoritme dat efficiënt goede samenvattingen vindt. Een samenvatting is dus simpelweg een

verzameling van patronen en kan ook multivariate patronen bevatten.

Naast het inzicht dat de patronen in de gevonden samenvattingen bieden kunnen deze samenvattingen ook gebruikt worden voor andere doeleinden in data mining. Voorbeelden hiervan zijn het clusteren van gelijksoortige data, het genereren van data om privacygevoelige gegevens te bewaken, en het vinden van afwijkingen (ook wel anomalieën) in de data. We richten ons verder op het gebruik van deze samenvattingen voor het vinden van afwijkingen in de data, vandaar dat de ondertitel van dit proefschrift 'van samenvatting naar afwijking' luidt.

Het onderzoeksveld genaamd anomalie detectie houdt zicht bezig met het identificeren van datapunten die significant afwijken van de rest van de data — zo anders dat het erop lijkt dat ze door een ander mechanisme gegenereerd zijn. Een anomalie kan voorkomen in de data als gevolg van een fout, het kan een extreme waarde zijn of het is een erg onverwacht datapunt.

In het tweede deel van dit proefschrift definiëren we een nieuw soort anomalie, namelijk het onverwacht voorkomen van een combinatie van patronen in de data. Ook introduceren we een algoritme om deze anomalieën efficiënt te identificeren. Neem als voorbeeld de volgende twee patronen: het drinken van Coca Cola en het drinken van Pepsi Cola. Verder weten we dat beide cola's veel gedronken worden, maar dat zo goed als iedereen een sterke merkvoorkeur heeft. Dat wil zeggen dat bijna niemand beide cola's drinkt. Komen we toch zo iemand tegen, dan is dit erg onverwacht. Niet door het voorkomen van de individuele patronen, maar juist door de onverwachte combinatie ervan. Zo'n onverwachte combinatie is dus per definitie een anomalie.

We concluderen in dit proefschrift dat het MDL principe succesvol toegepast kan worden voor zowel het samenvatten van multivariate datareeksen als het identificeren van anomalieën in deze data.

Dankwoord

Het dankwoord houd ik graag simpel. Daarom zal ik slechts de personen benoemen die de grootste invloed op mijn promotie hebben gehad. Daarnaast, zonder specifieke namen te noemen, iedereen die mijn promotietijd leuker, beter of interessanter heeft gemaakt: bedankt!

Ik dank Nicola en Kim van de ING voor de kans die ze me gaven om ervaring in het bedrijfsleven op te doen en Simon en Gerben van Semiotic Labs voor de mogelijkheid om mijn onderzoek op echte data toe te passen.

Voor het aanwakkeren van mijn interesse in het doen van onderzoek bedank ik Linda. Haar benaderbaarheid en open karakter hebben me overgehaald om bij haar als onderzoeker te beginnen. Ik heb daar goede herinneringen aan overgehouden.

Jilles heeft zich als tutor al sinds mijn bachelor met mijn toekomst bemoeid. Daarna hebben we contact gehouden en zowel privé als werkgerelateerd veel waardevolle discussies gehad. Bedankt voor de mogelijkheid die je me gegeven hebt om in Saarbrücken samen onderzoek te doen. Ik heb veel geleerd maar ook erg genoten van mijn tijd in Duitsland. Je positieve energie en enthousiasme hebben zowel mijn onderzoek als mijn ontwikkeling veel goed gedaan. Ik had me geen betere copromotor kunnen wensen.

De volgende die ik bedank is Arno. Hij zal zelf waarschijnlijk vinden dat zijn bijdrage in zijn rol als promotor vanzelfsprekend is, maar de manier waarop hij deze levert benoem ik hier graag. Jouw stijl is volgens mij goed samen te vatten met de volgende woorden: met vrijheid komt verantwoordelijkheid. Een goede begeleider geeft je niet altijd de antwoorden, maar laat je die zelf vinden. Arno, bedankt voor je inzicht en je vertrouwen. Je persoonlijkheid maakt de samenwerking met jou plezierig. Ik ben erg blij dat ik je als promotor heb getroffen.

Karien, met het ontwerp dat de omslag van dit boekje siert heb je een directe bijdrage aan dit proefschrift geleverd. Besef dat je als partner ook op een andere manier invloed hebt. Al is de inhoud van dit proefschrift voor jou vooral codetaal, al onze gesprekken en jouw kritische vragen hebben mij geholpen. Omdat dit niet direct van een x aantal pagina's is af te lezen ben ik blij dat de omslag hiervoor symbool staat.

Tot slot bedank ik in het bijzonder mijn ouders. Ze steunen me altijd in alle mogelijke vormen zodat ik kan doen wat ik wil. Ik heb dit soms als vanzelfsprekend ervaren en voor lief genomen, maar weet dat ik heel erg gelukkig met jullie ben. Jullie zijn de beste!

Curriculum Vitae

Roel Bertens was born in Goirle on the 25^{th} of September in 1986. He obtained his B.Sc. degree in Computer Science at Utrecht University in 2009 after he finished his final semester at the University of Florida. In 2011 he also obtained his M.Sc. degree at Utrecht University, whereafter he continued his final research project on Bayesian networks in the Decision Support Systems group headed by professor Linda van der Gaag.

In 2012 Roel started his Ph.D. research in the Algorithmic Data Analysis group under the supervision of professor Arno Siebes. The foundation of the joint work with Jilles Vreeken was laid in July 2014 when Roel visited the Exploratory Data Analysis group at the Cluster of Excellence on Multimodal Computing and Interaction in Saarbrücken.

In 2016 Roel started a six month internship at the ING bank and a three month project at Semiotic Labs while finishing his Ph.D. thesis.

SIKS Dissertation Series

2011 01 Botond Cseke (RUN), Variational Algorithms for Bayesian Inference in Latent Gaussian Models
02 Nick Tinnemeier (UU), Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language
03 Jan Martijn van der Werf (TUE), Compositional Design and Verification of Component-Based Information Systems
04 Hado van Hasselt (UU), Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference
05 Bas van der Raadt (VU), Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.
06 Yiwen Wang (TUE), Semantically-Enhanced Recommendations in Cultural Heritage
07 Yujia Cao (UT), Multimodal Information Presentation for High Load Human Computer Interaction
08 Nieske Vergunst (UU), BDI-based Generation of Robust Task-Oriented Dialogues
09 Tim de Jong (OU), Contextualised Mobile Media for Learning
10 Bart Bogaert (UvT), Cloud Content Contention
11 Dhaval Vyas (UT), Designing for Awareness: An Experience-focused HCI Perspective
12 Carmen Bratosin (TUE), Grid Architecture for Distributed Process Mining
13 Xiaoyu Mao (UvT), Airport under Control. Multiagent Scheduling for Airport Ground Handling
14 Milan Lovric (EUR), Behavioral Finance and Agent-Based Artificial Markets
15 Marijn Koolen (UvA), The Meaning of Structure: the Value of Link Evidence for Information Retrieval
16 Maarten Schadd (UM), Selective Search in Games of Different Complexity
17 Jiyin He (UVA), Exploring Topic Structure: Coherence, Diversity and Relatedness
18 Mark Ponsen (UM), Strategic Decision-Making in complex games
19 Ellen Rusman (OU), The Mind's Eye on Personal Profiles
20 Qing Gu (VU), Guiding service-oriented software engineering - A view-based approach
21 Linda Terlouw (TUD), Modularization and Specification of Service-Oriented Systems
22 Junte Zhang (UVA), System Evaluation of Archival Description and Access
23 Wouter Weerkamp (UVA), Finding People and their Utterances in Social Media
24 Herwin van Welbergen (UT), Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior
25 Syed Waqar ul Qounain Jaffry (VU), Analysis and Validation of Models for Trust Dynamics
26 Matthijs Aart Pontier (VU), Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots
27 Aniel Bhulai (VU), Dynamic website optimization through autonomous management of design patterns
28 Rianne Kaptein (UVA), Effective Focused Retrieval by Exploiting Query Context and Document Structure
29 Faisal Kamiran (TUE), Discrimination-aware Classification
30 Egon van den Broek (UT), Affective Signal Processing (ASP): Unraveling the mystery of emotions
31 Ludo Waltman (EUR), Computational and Game-Theoretic Approaches for Modeling Bounded Rationality
32 Nees-Jan van Eck (EUR), Methodological Advances in Bibliometric Mapping of Science
33 Tom van der Weide (UU), Arguing to Motivate Decisions
34 Paolo Turrini (UU), Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations
35 Maaike Harbers (UU), Explaining Agent Behavior in Virtual Training
36 Erik van der Spek (UU), Experiments in serious game design: a cognitive approach
37 Adriana Burlutiu (RUN), Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference
38 Nyree Lemmens (UM), Bee-inspired Distributed Optimization
39 Joost Westra (UU), Organizing Adaptation using Agents in Serious Games
40 Viktor Clerc (VU), Architectural Knowledge Management in Global Software Development
41 Luan Ibraimi (UT), Cryptographically Enforced Distributed Data Access Control
42 Michal Sindlar (UU), Explaining Behavior through Mental State Attribution
43 Henk van der Schuur (UU), Process Improvement through Software Operation Knowledge
44 Boris Reuderink (UT), Robust Brain-Computer Interfaces
45 Herman Stehouwer (UvT), Statistical Language Models for Alternative Sequence Selection
46 Beibei Hu (TUD), Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work
47 Azizi Bin Ab Aziz (VU), Exploring Computational Models for Intelligent Support of Persons with Depression
48 Mark Ter Maat (UT), Response Selection and Turn-taking for a Sensitive Artificial Listening Agent
49 Andreea Niculescu (UT), Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

2012 01 Terry Kakeeto (UvT), Relationship Marketing for SMEs in Uganda

02 Muhammad Umair (VU), Adaptivity, emotion, and Rationality in Human and Ambient Agent Models
03 Adam Vanya (VU), Supporting Architecture Evolution by Mining Software Repositories
04 Jurriaan Souer (UU), Development of Content Management System-based Web Applications
05 Marijn Plomp (UU), Maturing Interorganisational Information Systems
06 Wolfgang Reinhardt (OU), Awareness Support for Knowledge Workers in Research Networks
07 Rianne van Lambalgen (VU), When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions
08 Gerben de Vries (UVA), Kernel Methods for Vessel Trajectories
09 Ricardo Neisse (UT), Trust and Privacy Management Support for Context-Aware Service Platforms
10 David Smits (TUE), Towards a Generic Distributed Adaptive Hypermedia Environment
11 J.C.B. Rantham Prabhakara (TUE), Process Mining in the Large: Preprocessing, Discovery, and Diagnostics
12 Kees van der Sluijs (TUE), Model Driven Design and Data Integration in Semantic Web Information Systems
13 Suleman Shahid (UvT), Fun and Face: Exploring non-verbal expressions of emotion during playful interactions
14 Evgeny Knutov (TUE), Generic Adaptation Framework for Unifying Adaptive Web-based Systems
15 Natalie van der Wal (VU), Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.
16 Fiemke Both (VU), Helping people by understanding them - Ambient Agents supporting task execution and depression treatment
17 Amal Elgammal (UvT), Towards a Comprehensive Framework for Business Process Compliance
18 Eltjo Poort (VU), Improving Solution Architecting Practices
19 Helen Schonenberg (TUE), What's Next? Operational Support for Business Process Execution
20 Ali Bahramisharif (RUN), Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing
21 Roberto Cornacchia (TUD), Querying Sparse Matrices for Information Retrieval
22 Thijs Vis (UvT), Intelligence, politie en veiligheidsdienst: verenigbare grootheden?
23 Christian Muehl (UT), Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction
24 Laurens van der Werff (UT), Evaluation of Noisy Transcripts for Spoken Document Retrieval
25 Silja Eckartz (UT), Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application
26 Emile de Maat (UVA), Making Sense of Legal Text
27 Hayrettin Gurkok (UT), Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games
28 Nancy Pascall (UvT), Engendering Technology Empowering Women
29 Almer Tigelaar (UT), Peer-to-Peer Information Retrieval
30 Alina Pommeranz (TUD), Designing Human-Centered Systems for Reflective Decision Making
31 Emily Bagarukayo (RUN), A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure
32 Wietske Visser (TUD), Qualitative multi-criteria preference representation and reasoning
33 Rory Sie (OUN), Coalitions in Cooperation Networks (COCOON)
34 Pavol Jancura (RUN), Evolutionary analysis in PPI networks and applications
35 Evert Haasdijk (VU), Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics
36 Denis Ssebugwawo (RUN), Analysis and Evaluation of Collaborative Modeling Processes
37 Agnes Nakakawa (RUN), A Collaboration Process for Enterprise Architecture Creation
38 Selmar Smit (VU), Parameter Tuning and Scientific Testing in Evolutionary Algorithms
39 Hassan Fatemi (UT), Risk-aware design of value and coordination networks
40 Agus Gunawan (UvT), Information Access for SMEs in Indonesia
41 Sebastian Kelle (OU), Game Design Patterns for Learning
42 Dominique Verpoorten (OU), Reflection Amplifiers in self-regulated Learning
43 Withdrawn
44 Anna Tordai (VU), On Combining Alignment Techniques
45 Benedikt Kratz (UvT), A Model and Language for Business-aware Transactions
46 Simon Carter (UVA), Exploration and Exploitation of Multilingual Data for Statistical Machine Translation
47 Manos Tsagkias (UVA), Mining Social Media: Tracking Content and Predicting Behavior
48 Jorn Bakker (TUE), Handling Abrupt Changes in Evolving Time-series Data
49 Michael Kaisers (UM), Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions
50 Steven van Kervel (TUD), Ontologogy driven Enterprise Information Systems Engineering
51 Jeroen de Jong (TUD), Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

---

2013 01 Viorel Milea (EUR), News Analytics for Financial Decision Support
02 Erietta Liarou (CWI), MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing
03 Szymon Klarman (VU), Reasoning with Contexts in Description Logics
04 Chetan Yadati (TUD), Coordinating autonomous planning and scheduling
05 Dulce Pumareja (UT), Groupware Requirements Evolutions Patterns
06 Romulo Goncalves (CWI), The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience
07 Giel van Lankveld (UvT), Quantifying Individual Player Differences
08 Robbert-Jan Merk (VU), Making enemies: cognitive modeling for opponent agents in fighter pilot simulators
09 Fabio Gori (RUN), Metagenomic Data Analysis: Computational Methods and Applications
10 Jeewanie Jayasinghe Arachchige (UvT), A Unified Modeling Framework for Service Design.
11 Evangelos Pournaras (TUD), Multi-level Reconfigurable Self-organization in Overlay Services
12 Marian Razavian (VU), Knowledge-driven Migration to Services
13 Mohammad Safiri (UT), Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly
14 Jafar Tanha (UVA), Ensemble Approaches to Semi-Supervised Learning Learning
15 Daniel Hennes (UM), Multiagent Learning - Dynamic Games and Applications
16 Eric Kok (UU), Exploring the practical benefits of argumentation in multi-agent deliberation
17 Koen Kok (VU), The PowerMatcher: Smart Coordination for the Smart Electricity Grid

18 Jeroen Janssens (UvT), Outlier Selection and One-Class Classification
19 Renze Steenhuizen (TUD), Coordinated Multi-Agent Planning and Scheduling
20 Katja Hofmann (UvA), Fast and Reliable Online Learning to Rank for Information Retrieval
21 Sander Wubben (UvT), Text-to-text generation by monolingual machine translation
22 Tom Claassen (RUN), Causal Discovery and Logic
23 Patricio de Alencar Silva (UvT), Value Activity Monitoring
24 Haitham Bou Ammar (UM), Automated Transfer in Reinforcement Learning
25 Agnieszka Anna Latoszek-Berendsen (UM), Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System
26 Alireza Zarghami (UT), Architectural Support for Dynamic Homecare Service Provisioning
27 Mohammad Huq (UT), Inference-based Framework Managing Data Provenance
28 Frans van der Sluis (UT), When Complexity becomes Interesting: An Inquiry into the Information eXperience
29 Iwan de Kok (UT), Listening Heads
30 Joyce Nakatumba (TUE), Resource-Aware Business Process Management: Analysis and Support
31 Dinh Khoa Nguyen (UvT), Blueprint Model and Language for Engineering Cloud Applications
32 Kamakshi Rajagopal (OUN), Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development
33 Qi Gao (TUD), User Modeling and Personalization in the Microblogging Sphere
34 Kien Tjin-Kam-Jet (UT), Distributed Deep Web Search
35 Abdallah El Ali (UvA), Minimal Mobile Human Computer Interaction
36 Than Lam Hoang (TUe), Pattern Mining in Data Streams
37 Dirk Börner (OUN), Ambient Learning Displays
38 Eelco den Heijer (VU), Autonomous Evolutionary Art
39 Joop de Jong (TUD), A Method for Enterprise Ontology based Design of Enterprise Information Systems
40 Pim Nijssen (UM), Monte-Carlo Tree Search for Multi-Player Games
41 Jochem Liem (UVA), Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning
42 Léon Planken (TUD), Algorithms for Simple Temporal Reasoning
43 Marc Bron (UVA), Exploration and Contextualization through Interaction and Concepts

2014 01 Nicola Barile (UU), Studies in Learning Monotone Models from Data
02 Fiona Tuliyano (RUN), Combining System Dynamics with a Domain Modeling Method
03 Sergio Raul Duarte Torres (UT), Information Retrieval for Children: Search Behavior and Solutions
04 Hanna Jochmann-Mannak (UT), Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation
05 Jurriaan van Reijsen (UU), Knowledge Perspectives on Advancing Dynamic Capability
06 Damian Tamburri (VU), Supporting Networked Software Development
07 Arya Adriansyah (TUE), Aligning Observed and Modeled Behavior
08 Samur Araujo (TUD), Data Integration over Distributed and Heterogeneous Data Endpoints
09 Philip Jackson (UvT), Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language
10 Ivan Salvador Razo Zapata (VU), Service Value Networks
11 Janneke van der Zwaan (TUD), An Empathic Virtual Buddy for Social Support
12 Willem van Willigen (VU), Look Ma, No Hands: Aspects of Autonomous Vehicle Control
13 Arlette van Wissen (VU), Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains
14 Yangyang Shi (TUD), Language Models With Meta-information
15 Natalya Mogles (VU), Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare
16 Krystyna Milian (VU), Supporting trial recruitment and design by automatically interpreting eligibility criteria
17 Kathrin Dentler (VU), Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability
18 Mattijs Ghijsen (UVA), Methods and Models for the Design and Study of Dynamic Agent Organizations
19 Vinicius Ramos (TUE), Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support
20 Mena Habib (UT), Named Entity Extraction and Disambiguation for Informal Text: The Missing Link
21 Kassidy Clark (TUD), Negotiation and Monitoring in Open Environments
22 Marieke Peeters (UU), Personalized Educational Games - Developing agent-supported scenario-based training
23 Eleftherios Sidirourgos (UvA/CWI), Space Efficient Indexes for the Big Data Era
24 Davide Ceolin (VU), Trusting Semi-structured Web Data
25 Martijn Lappenschaar (RUN), New network models for the analysis of disease interaction
26 Tim Baarslag (TUD), What to Bid and When to Stop
27 Rui Jorge Almeida (EUR), Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty
28 Anna Chmielowiec (VU), Decentralized k-Clique Matching
29 Jaap Kabbedijk (UU), Variability in Multi-Tenant Enterprise Software
30 Peter de Cock (UvT), Anticipating Criminal Behaviour
31 Leo van Moergestel (UU), Agent Technology in Agile Multiparallel Manufacturing and Product Support
32 Naser Ayat (UvA), On Entity Resolution in Probabilistic Data
33 Tesfa Tegegne (RUN), Service Discovery in eHealth
34 Christina Manteli (VU), The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.
35 Joost van Ooijen (UU), Cognitive Agents in Virtual Worlds: A Middleware Design Approach
36 Joos Buijs (TUE), Flexible Evolutionary Algorithms for Mining Structured Process Models
37 Maral Dadvar (UT), Experts and Machines United Against Cyberbullying
38 Danny Plass-Oude Bos (UT), Making brain-computer interfaces better: improving usability through post-processing.
39 Jasmina Maric (UvT), Web Communities, Immigration, and Social Capital
40 Walter Omona (RUN), A Framework for Knowledge Management Using ICT in Higher Education
41 Frederic Hogenboom (EUR), Automated Detection of Financial Events in News Text
42 Carsten Eijckhof (CWI/TUD), Contextual Multidimensional Relevance Models

43 Kevin Vlaanderen (UU), Supporting Process Improvement using Method Increments
44 Paulien Meesters (UvT), Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.
45 Birgit Schmitz (OUN), Mobile Games for Learning: A Pattern-Based Approach
46 Ke Tao (TUD), Social Web Data Analytics: Relevance, Redundancy, Diversity
47 Shangsong Liang (UVA), Fusion and Diversification in Information Retrieval

2015 01 Niels Netten (UvA), Machine Learning for Relevance of Information in Crisis Response
02 Faiza Bukhsh (UvT), Smart auditing: Innovative Compliance Checking in Customs Controls
03 Twan van Laarhoven (RUN), Machine learning for network data
04 Howard Spoelstra (OUN), Collaborations in Open Learning Environments
05 Christoph Bösch (UT), Cryptographically Enforced Search Pattern Hiding
06 Farideh Heidari (TUD), Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes
07 Maria-Hendrike Peetz (UvA), Time-Aware Online Reputation Analysis
08 Jie Jiang (TUD), Organizational Compliance: An agent-based model for designing and evaluating organizational interactions
09 Randy Klaassen (UT), HCI Perspectives on Behavior Change Support Systems
10 Henry Hermans (OUN), OpenU: design of an integrated system to support lifelong learning
11 Yongming Luo (TUE), Designing algorithms for big graph datasets: A study of computing bisimulation and joins
12 Julie M. Birkholz (VU), Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks
13 Giuseppe Procaccianti (VU), Energy-Efficient Software
14 Bart van Straalen (UT), A cognitive approach to modeling bad news conversations
15 Klaas Andries de Graaf (VU), Ontology-based Software Architecture Documentation
16 Changyun Wei (UT), Cognitive Coordination for Cooperative Multi-Robot Teamwork
17 André van Cleeff (UT), Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs
18 Holger Pirk (CWI), Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories
19 Bernardo Tabuenca (OUN), Ubiquitous Technology for Lifelong Learners
20 Lois Vanhée (UU), Using Culture and Values to Support Flexible Coordination
21 Sibren Fetter (OUN), Using Peer-Support to Expand and Stabilize Online Learning
22 Zhemin Zhu (UT), Co-occurrence Rate Networks
23 Luit Gazendam (VU), Cataloguer Support in Cultural Heritage
24 Richard Berendsen (UVA), Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation
25 Steven Woudenberg (UU), Bayesian Tools for Early Disease Detection
26 Alexander Hogenboom (EUR), Sentiment Analysis of Text Guided by Semantics and Structure
27 Sándor Héman (CWI), Updating compressed colomn stores
28 Janet Bagorogoza (TiU), Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO
29 Hendrik Baier (UM), Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains
30 Kiavash Bahreini (OU), Real-time Multimodal Emotion Recognition in E-Learning
31 Yakup Koç (TUD), On the robustness of Power Grids
32 Jerome Gard (UL), Corporate Venture Management in SMEs
33 Frederik Schadd (TUD), Ontology Mapping with Auxiliary Resources
34 Victor de Graaf (UT), Gesocial Recommender Systems
35 Jungxao Xu (TUD), Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction

2016 01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
04 Laurens Rietveld (VU), Publishing and Consuming Linked Data
05 Evgeny Sherkhonov (UVA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
07 Jeroen de Man (VU), Measuring and modeling negative emotions for virtual training
08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
09 Archana Nottamkandath (VU), Trusting Crowdsourced Information on Cultural Artefacts
10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
11 Anne Schuth (UVA), Search Engines that Learn from Their Users
12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
13 Nana Baah Gyan (VU), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
16 Guangliang Li (UVA), Socially Intelligent Autonomous Agents that Learn from Human Reward
17 Berend Weel (VU), Towards Embodied Evolution of Robot Organisms
18 Albert Meroño Peñuela (VU), Refining Statistical Data on the Web
19 Julia Efremova (Tu/e), Mining Social Structures from Genealogical Data
20 Daan Odijk (UVA), Context & Semantics in News & Web Search
21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
22 Grace Lewis (VU), Software Architecture Strategies for Cyber-Foraging Systems
23 Fei Cai (UVA), Query Auto Completion in Information Retrieval
24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
26 Dilhan Thilakarathne (VU), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning

30 Ruud Mattheij (UvT), The Eyes Have It
31 Mohammad Khelghati (UT), Deep web content monitoring
32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
33 Peter Bloem (UVA), Single Sample Statistics, exercises in learning from just one example
34 Dennis Schunselaar (TUE), Configurable Process Trees: Elicitation, Analysis, and Enactment
35 Zhaochun Ren (UVA), Monitoring Social Media: Summarization, Classification and Recommendation
36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
40 Christian Detweiler (TUD), Accounting for Values in Design
41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
42 Spyros Martzoukos (UVA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
44 Thibault Sellam (UVA), Automatic Assistants for Database Exploration
45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
46 Jorge Gallego Perez (UT), Robots to Make you Happy
47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
48 Tanja Buttler (TUD), Collecting Lessons Learned
49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
50 Yan Wang (UVT), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

2017 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime
02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
05 Mahdieh Shadi (UVA), Collaboration Behavior
06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly