Universiteit Leiden
Opleiding Informatica

Mining and visualizing sequential healthcare data

Name: Floris Kleyn

Date: 05/09/2016

1st supervisor: Dr. Wojtek Kowalczyk
2nd supervisor: Prof. Dr. Thomas H.W. Bäck

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Mining and visualizing sequential healthcare data

Floris Kleyn

September 5, 2016

Contents

Chapter 1

Introduction

Sequence mining techniques have been successfully applied in many areas such as speech recognition, web mining, bioinformatics and many more. Over the years, many algorithms have been developed to solve problems in those specific domains.

Sequential data is generated by all kinds of processes. Records in a sequence database represent sequences of ordered events. Take for example a database containing purchase transactions. A transaction consists of the set of items that were bought together by a certain customer at a specific time. In this case, a sequence consists of all the transactions of a customer ordered by date of purchase. The order of events in a sequence is not necessarily defined by time. Consider the order of nucleotides in strands of DNA. The order of nucleotides is simply the order of their occurrence in the DNA. Other examples of sequential data are datasets about browsing behaviour on a web site and treatments of patients in hospitals.

Today, the healthcare industry produces large volumes of data. A lot of this data is sequential in nature: patients undergo blood tests, receive medical tests, are prescribed drugs, etc. All these events occur in a certain order and form a patient's *healthcare trajectory*. Mining these sequences can result in valuable information, provided that appropriate methods are applied and high quality data is available. Acquired information might benefit patients, hospitals and insurance companies. It could result in, for instance, better diagnosis of patients, improved effectiveness of hospitals or new insights in how care is provided to patients.

We examined how similarity between trajectories could be defined, and how we could cluster them based on the similarity of trajectories. We incorporated domain knowledge about specific medical events in our similarity measure. With a good similarity measure and proper clustering, this allows identification of groups of patients that received a similar treatment. We also examined visualization of sequences, such that information can be communicated clearly to people who are not data mining experts.

Domain knowledge was crucial for our task. Input and feedback from domain

experts was necessary to create a meaningful similarity measure, such that sequences that are considered similar by our metric correspond to patients who are considered similar by domain experts. One of our challenges was to translate domain knowledge into an adequate similarity measure and good visualization techniques.

The objective of this thesis was to adapt sequence mining techniques and test their usefulness when applied to medical data. We focused on patients in The Netherlands who were diagnosed with chest pain and applied sequence mining techniques to their healthcare trajectories. Trajectories consisted out of activities related to the diagnosis. Records of chest pain patients that belonged to other diagnoses were removed. We created a chronological sequence of events for every patient, where every event is a single care product.

This thesis was made possible with help from Zorginstituut Nederland (ZIN), a Dutch governmental organisation tasked with several activities [44]. ZIN monitors the quality of the Dutch healthcare system. It also makes sure the system stays affordable, for example, by advising the Ministry of Health, Welfare and Sport which care should be covered by the mandatory health insurance policy every Dutch adult citizen should have. Another task of ZIN is to guarantee that healthcare stays accessible to everyone. For instance, it provides arrangements for defaulters and for the uninsured. To support its activities, it has access to large volumes of healthcare data. It has data about all care provided by hospitals in the Netherlands. This gave us a unique opportunity to research how to apply sequence mining techniques to healthcare data.

This thesis is organized as follows: We start with some formal definitions and notations about sequences in Chapter 2 and discuss relevant literature and provide examples of common algorithms used in sequence mining in Chapter 3. Chapter 4 describes our contributions. We present the developed similarity measure, describe the clustering algorithms we used and present our visualization technique. In Chapter 5 we provide a description of the dataset we used and describe the experiments we conducted. The results are presented in Chapter 6. Chapter 7 contains our discussion, conclusions and directions for future work.

Chapter 2

Preliminaries

Throughout this thesis we will be working with sequences and sets. In this chapter we will provide definitions and notations that we will use throughout the thesis.

2.1 Notations and Definitions

We start with a definition of the components of a sequence. Sequences consist of *transactions* and each transaction consists of one or more *items*. Let $I = \{i_1, i_2, \ldots, i_k\}$ be the set of items. The order and frequency of items in a set are irrelevant, although for readability we will often use an alphabetic ordering when specifying sets. A sequence $S = \langle t_1, t_2, \ldots, t_n \rangle$ is an ordered list of transactions where t_i is a transaction and $t_i \subseteq I$ for $1 \leq i \leq n$. To refer to the ith transaction in S we also write $S_{(i)}$. In other words $t_i = S_{(i)}$. In case a transaction consists of only a single item we will often omit the braces. So the sequence $\langle \{a, b\}, \{c\}, \{d\} \rangle$ is identical to sequence $\langle \{a, b\}, c, d \rangle$.

A sequence $S' = \langle t'_1, t'_2, \ldots, t'_m \rangle$ is a subsequence of $S = \langle t_1, t_2, \ldots, t_n \rangle$, denoted $S' \preceq S$, if there exist integers $1 \leq j_1 < j_2 \cdots < j_m \leq n$ such that $t'_1 \subseteq t_{j_1}, t'_2 \subseteq t_{j_2}, \ldots, t'_m \subseteq t_{j_m}$. Consider sequence $S = \langle \{a, b\}, c, \{d, e, f\}, g \rangle$. Sequence $S' = \langle a, \{d, f\}, g \rangle$ is a subsequence of S, for integers $j_1 = 1, j_2 = 3$ and $j_3 = 4$. The number of transactions in S (i.e., its length) is denoted as $|S|$. We will usually use D to denote a set of sequences.

Let D be a set of sequences, a sequence S is called *frequent* with respect to D if it meets a user defined threshold value, called *minimum support* threshold. The support of a subsequence S is the proportion of all sequences in D that contain S. A *maximal* frequent sequence is a frequent sequence for which there exists no super sequence in D that is also frequent. A frequent sequence is *closed* if there exists no super sequence in D with the same support. Maximal frequent sequences are also closed, since extending it will results in a sequence that is not frequent and thus the super sequence must have, by definition, lower support.

Sequence mining or sequential pattern mining is a field of data mining, often described as a field tasked with discovering frequent (sub)sequences in data. According to [26] "Sequential pattern mining discovers frequent subsequences as patterns in a sequence database." However, we will make a distinction between sequential pattern mining and frequent sequence mining. When referring to sequential pattern mining or sequence mining, we will mean the larger field of data mining in sequential data. If we refer to the task of discovering frequent sequences in data, we will refer to it as frequent sequence mining.

Throughout the thesis, we will often refer to *events* instead of items, although the term we use depends on the context. Take for example the area of market basket analysis. In this case, a sequence would be an ordered set of transactions where every transaction contains the items that were bought together. In case of medical treatment data, it would make sense to refer to items as (medical) events. Referring to the set of events a patient received on the same day as a transaction is also a bit peculiar. Therefore we will often refer to a transaction as an *event set*, although we use it interchangeably with event. Based on the context it should be clear if we either mean a single event or an event set, although a set can of course only contain a single event. In the context of healthcare we will sometimes refer to sequences as trajectories or paths.

2.2 Constraints

It is possible to define constraints that (frequent) sequences should satisfy. This can limit the number of results returned by a pattern mining algorithm. The following seven types of constraints are given in [37]:

- *Item constraint*: a constraint that specifies one or more items that should or should not occur in mined patterns.
- *Length constraint*: a constraint that constrains the length of mined patterns. Examples are constraints on the number of transactions or number of distinct transactions in a sequence.
- *Super-pattern constraint*: a constraint that specifies which sub-pattern of transactions should be present in the mined patterns.
- *Aggregate constraint*: a constraint based on some aggregate of items in a pattern. An example would be a constraint on the total value of the pattern.
- *Regular expression constraint*: a constraint defined by regular expression operators (e.g., disjunction and union).
- *Duration constraint*: a constraint on the duration of a pattern. This type of constraint specifies how much time is allowed to pass between the first and last item in the pattern.
- *Gap constraint*: a constraint on the time interval between adjacent transactions.

Chapter 3

Related work

In this chapter we will discuss related work in different areas of sequence mining and provide examples of existing methods. We will discuss frequent sequence mining, in particular frequent sequence mining applied in the healthcare domain. We will discuss sequence similarity, clustering of sequences and visualization of health care trajectories.

3.1 Frequent sequence mining

The task of frequent sequence mining is finding temporal patterns in data. The foundations of frequent sequence mining were laid by Agrawal and Srikant [2] in 1995. They developed AprioriAll, a frequent sequence mining algorithm based on Apriori [1, 3], which is an algorithm for mining frequent itemsets. Frequent itemset mining is related to frequent sequence mining. Frequent itemset mining focuses on *intra-transaction* patterns (e.g., patterns inside a single shopping basket). No order is present between items within a transaction. Frequent sequence mining searches for *inter-transaction* patterns (e.g., patterns between different shopping baskets of the same customer), where there is a certain order: the order in which the transactions took place.

AprioriAll is a *breadth-first search* algorithm: in iteration k all sequences of length k are constructed. It is based on the Apriori principle: subsequences of frequent sequences are also frequent. Conversely, supersequences of infrequent sequences are also infrequent. The search space is reduced by pruning candidate sequences during mining, since a pattern does not have to be extended as soon as it does not meet minimum support. Still, it will often generate a lot of candidate sequences in the early stages of mining, of which many might not even be present in the data.

The discrepancy between the number of generated candidate sequences and the number of frequent sequences is caused by the *generate-and-test* feature of AprioriAll. A pattern is increased incrementally, one event at a time, and tested if it still meets the minimum support threshold. Since it generates all candidate sequences of length k based on frequent sequences of length $k-1$, it

u_id	Item	Timestamp
1	a	3
1	b	8
1	c	12
2	a	20
2	c	25
3	a	1
3	b	2
3	c	4
3	d	7
4	{a,b}	10
4	c	15
5	e	3
5	{a,b}	8
5	c	11
6	e	2
6	c	21
7	{c,e}	12

(a) Sorted transaction dataset.

u_id	Sequence
1	$\langle a, b, c \rangle$
2	$\langle a, c \rangle$
3	$\langle a, b, c, d \rangle$
4	$\langle \{a, b\}, c \rangle$
5	$\langle e, \{a, b\}, c \rangle$
6	$\langle e, c \rangle$
7	$\langle \{c, e\} \rangle$

(b) Sequence dataset.

Sequence	Support	Mapping
$\langle a \rangle$	$^5/_7$	1
$\langle b \rangle$	$^4/_7$	2
$\langle c \rangle$	1	3
$\langle d \rangle$	$^1/_7$	-
$\langle e \rangle$	$^3/_7$	4
$\langle \{a, b\} \rangle$	$^2/_7$	5
$\langle \{c, e\} \rangle$	$^1/_7$	-

(c) length-1 sequences.

Table 3.1: Transforming our example transaction dataset into sequences.

could happen that generated sequences do not occur in the data. The memory requirements are also high and *multiple scans of the data* are required to find all frequent sequences. Other algorithms were developed after AprioriAll that share its characteristics and downsides. These are *Apriori-based* algorithms. Examples are GSP [49], SPIRIT [10], SPAM [36] and SPADE [60].

Other groups of frequent sequence mining algorithms exist, but first we will give an example of frequent sequence mining based on the AprioriAll algorithm. The transaction data of this example is listed in Table 3.1a, which is already sorted on user id (u_id) and timestamp. There are seven users with a total of seventeen transactions present in our dataset. Each transaction consists of items from itemset $I = \{a, b, c, d, e\}$

The first step is to construct sequences from our dataset of individual transactions. The order of transactions in the sequences is determined by their timestamps. The sequences are listed in Table 3.1b.

Our goal is to find all subsequences that are frequent. In our example, we will consider subsequences frequent if they occur at least twice in the dataset. Therefore, the minimum support equals $^2/_7$. The first step is to find all frequent itemsets and count their support. All frequent sequences of length 1 are obtained automatically during this step, since these are identical to itemsets. The result is shown in Table 3.1c. There are two sequences of length 1 below our support threshold: $\langle d \rangle$ and $\langle \{c, e\} \rangle$. Removing these sequences will give us the set of frequent sequences of length 1.

Every frequent itemset (i.e., frequent sequence of length 1) is mapped to a single integer. This allows to test sets for equality in $O(1)$. The mapping from itemsets

u_id	Sequence	Transformed sequence	Mapped sequence
1	$\langle a, b, c\rangle$	$\langle\{a\}, \{b\}, \{c\}\rangle$	$\langle\{1\}, \{2\}, \{3\}\rangle$
2	$\langle a, c\rangle$	$\langle\{a\}, \{c\}\rangle$	$\langle\{1\}, \{3\}\rangle$
3	$\langle a, b, c, d\rangle$	$\langle\{a\}, \{b\}, \{c\}\rangle$	$\langle\{1\}, \{2\}, \{3\}\rangle$
4	$\langle\{a, b\}, c\rangle$	$\langle\{a, b, \{a, b\}\}, \{c\}\rangle$	$\langle\{1, 2, 5\}, \{3\}\rangle$
5	$\langle e, \{a, b\}, c\rangle$	$\langle\{e\}, \{a, b, \{a, b\}\}, \{c\}\rangle$	$\langle\{4\}, \{1, 2, 5\}, \{3\}\rangle$
6	$\langle e, c\rangle$	$\langle\{e\}, \{c\}\rangle$	$\langle\{4\}, \{3\}\rangle$
7	$\langle\{c, e\}\rangle$	$\langle\{c, e\}\rangle$	$\langle\{3, 4\}\rangle$

Table 3.2: Sequence transformation. The transformed and mapped sequences consist of sets of frequent itemsets.

to integers is also shown in Table 3.1c. The next step is to transform sequences into sequences of frequent itemsets, and map them into sequences of sets of integers. This process is shown in Table 3.2. Note that during transformation all nonfrequent itemsets are removed from the sequences (e.g., itemset d from u_id 3) and that all subsets of frequent itemsets are also present in the transformed sequences (e.g., transaction $\{a, b\}$ in the sequence of u_id 4 is transformed into $\{a, b, \{a, b\}\}$[1]) since subsequences of frequent sequences are also frequent. For readability we will be working with the transformed sequences and not with the mapped sequences for the remainder of the example.

Next, the AprioriAll algorithm will generate candidate sequences of length 2, by joining the frequent sequences of length 1 with each other. When generating sequences of length k, the prefix of length k-1 should be identical for the join. For length-2 candidate sequences this means all combinations of frequent sequences of length 1 are generated. All candidate length-2 sequences are listed in Table 3.3a.

Thanks to the Apriori principle we did not have to generate sequences of length 2 that contained non-frequent itemsets, for example, sequences containing itemset d. However, only eight of the twenty-five candidate sequences listed in Table 3.3a are actually present in the data. Only five of them meet minimum support, shown in bold in Table 3.3a. Our small example dataset had only five different items. If there are a lot more distinct items (and itemsets containing multiple items), many candidate sequences would be generated, of which only a fraction is present in the data and even fewer would be frequent.

Now that we have frequent sequences of length 2, we can generate candidate sequences of length 3. These are shown in Table 3.3b and were obtained by joining $\langle a, b\rangle$ with $\langle a, c\rangle$ since these are the only sequences that share a length-1 prefix (i.e., a). No other candidates of length 3 can be frequent. We can prune sequence $\langle a, c, b\rangle$ since subsequence $\langle c, b\rangle$ is not frequent and thus neither is one of its supersets. Therefore, it is not required to count support of this sequence.

Since there are no candidate sequences of length 3 that share the same prefix, no sequence of length 4 can be frequent and the algorithm will terminate. We have found eleven frequent subsequences, listed in Table 3.4.

[1] actually, since it is transformed into a set of frequent itemsets the notation should be $\{\{a\}, \{b\}, \{a, b\}\}$, but we omitted braces for sets containing a single item.

Cand. seq.	Support
$\langle a, a \rangle$	0
$\boldsymbol{\langle a, b \rangle}$	$\boldsymbol{^2/_7}$
$\boldsymbol{\langle a, c \rangle}$	$\boldsymbol{^4/_7}$
$\langle a, e \rangle$	0
$\langle a, \{a, b\} \rangle$	0
$\langle b, a \rangle$	0
$\langle b, b \rangle$	0
$\boldsymbol{\langle b, c \rangle}$	$\boldsymbol{^4/_7}$
$\langle b, e \rangle$	0
$\langle b, \{a, b\} \rangle$	0
$\langle c, a \rangle$	0
$\langle c, b \rangle$	0
$\langle c, c \rangle$	0
$\langle c, e \rangle$	0
$\langle c, \{a, b\} \rangle$	0
$\langle e, a \rangle$	$^1/_7$
$\langle e, b \rangle$	$^1/_7$
$\boldsymbol{\langle e, c \rangle}$	$\boldsymbol{^3/_7}$
$\langle e, e \rangle$	0
$\langle e, \{a, b\} \rangle$	$^1/_7$
$\langle \{a, b\}, a \rangle$	0
$\langle \{a, b\}, b \rangle$	0
$\boldsymbol{\langle \{a, b\}, c \rangle}$	$\boldsymbol{^2/_7}$
$\langle \{a, b\}, e \rangle$	0
$\langle \{a, b\}, \{a, b\} \rangle$	0

(a) Candidate sequences of length 2.

cand. gen.	support
$\langle a, b, c \rangle$	$^2/_7$
$\langle a, c, b \rangle$	-

(b) Candidate sequences of length 3.

Table 3.3: Candidate sequences.

AprioriAll also removes all subsequences of frequent sequences, keeping only sequences that are *maximal*. Applying this step will result in the sequences shown in bold in Table 3.4.

We already discussed *Apriori*-based algorithms. Other types of algorithms are *pattern growth* based and *early pruning* based. A detailed taxonomy of frequent sequence mining algorithms is given in [26], but a brief overview is given below.

The problem of generate-and-test is illustrated in Table 3.3a. There are many generated sequences listed that did not occur in the data. To overcome the problems of generate-and-test, pattern growth algorithms were developed. The main idea was to avoid the candidate generation phase. Instead, a new representation of the data is created to allow search space partitioning. Each search space is solved recursively. Examples are FreeSpan [14], PrefixSpan [38] and FS-Miner [7]. FS-miner, which is based on FP-miner [13] and only suitable for mining of contiguous subsequences, converts the data into a tree. Only a single scan of the data is required to construct the tree. After the tree is constructed it can be mined efficiently for frequent sequences.

Freq. seq.	Support
$\langle a \rangle$	$^5/_7$
$\langle b \rangle$	$^4/_7$
$\langle c \rangle$	1
$\langle e \rangle$	$^3/_7$
$\langle \{a, b\} \rangle$	$^2/_7$
$\langle a, b \rangle$	$^2/_7$
$\langle a, c \rangle$	$^4/_7$
$\langle b, c \rangle$	$^4/_7$
$\boldsymbol{\langle e, c \rangle}$	$\boldsymbol{^2/_7}$
$\boldsymbol{\langle \{a, b\}, c \rangle}$	$\boldsymbol{^2/_7}$
$\boldsymbol{\langle a, b, c \rangle}$	$\boldsymbol{^2/_7}$

Table 3.4: Frequent sequences and their support. Bold sequences are maximal.

Early-pruning algorithms are pattern growth algorithms that try to prune candidate sequences as early as possible and limit counting support of candidate sequences as much as possible. Examples are HSVM [47] and LAPIN [59].

3.1.1 Frequent sequence mining applied to healthcare

The healthcare sector produces lots of data, so it is no surprise that data mining gets a lot of attention. Fewer attention has been given to sequence mining in healthcare data. One cause is the limited access to healthcare data since healthcare providers do not share data for several reasons: privacy of patients, competition between hospitals and a variety of registration systems. We will present an overview of frequent sequence mining applied to healthcare data.

The work presented in [4] is similar to ours. Their goal was to identify frequent medical sequences, and discover deviations of such sequences in treatment data of diabetic patients. Sequences were created with only a unique patient identifier, activity name and activity date. They mined for frequent closed sequences and for frequent itemsets (i.e., sets of activities performed on the same day). They used BIDE [52] to mine for frequent closed sequences. Domain knowledge was used in the preprocessing phase to be able to limit data analysis to patients that received a certain activity (e.g., the most expensive one) or to paths of certain length (e.g., to patients receiving many activities in a year). Mined pathways were classified as corresponding to guidelines, as an alternative or new pathway, or as erroneous.

Similar data to the data we had access to was used for process mining [29]. The data consisted of billing information from a Dutch hospital about gynecologic oncology patients. No order was known in cases where multiple events took place on the same day, since only the billing date was available. However, since the authors were interested in discovering patterns between departments, they did not look at sequences of events. Instead, they mapped events to the corresponding department and removed repetitions of events that occurred after mapping.

Process mining of healthcare data was also performed by Huang et al. [17].

The data had a similar level of detail as ours. Clinical events were treated as point-based events. Events were represented with two points, a start and end point, in case of interval-based events. To investigate if care was given according to medical guidelines, they took the time between events into account during mining for frequent closed sequences. By quantifying the time between events in a sequence, they showed that it is possible to check if the time intervals present in the data correspond to the time intervals that should occur according to guidelines.

In other research more information of patients is used, such as drug prescriptions, test results (e.g., results from blood and urine tests) or outcome of treatments. We did not have access to such data, but we will discuss some of it below.

The health condition of HIV patients was described with a small set of variables by Ramirez et al. [42]. Every record consisted of event date, event type (e.g., hospital visit, emergency room visit), health status, recovery time and specific laboratory results and drug prescriptions. Laboratory results were normalized and discretized. Drug prescription period was taken into account as well. If a new event occurred during the prescription period of a drug, the drug would be part of that event. Time windows were used to determine if events should be considered to have occurred simultaneously. This way, lab results that occurred just before a clinical visit and drugs prescribed just after the clinical visit were grouped together with the actual clinical visit to form an event set. They started with the GSP algorithm, but switched to their own system, TEMPADIS, because GSP was too slow to mine frequent sequences. Partial matches were used to allow some small differences between lab results and still consider records identical. With help of domain experts they discovered some interesting sequential patterns.

Lab results were also used by Ohsaki et al. [34]. A system was built to discover rules in data from chronic hepatitis patients. Time series mining was used to discover rules and predict future values in blood and urine test results.

Treatment outcomes were combined with sequences [23, 39, 40] to classify mined frequent sequences. An insight was obtained about which patterns (i.e., treatment of a disease) resulted in positive (e.g., alive, no rehospitalization) or negative outcomes. The data they used "consist of events including diagnoses, medication orders, laboratory reports and vital statistics for a given patient" [23]. They also addressed the problem of simultaneous events (i.e., occurring on the same day), which can result in pattern explosion in situations where sequences have many simultaneous events, since all subsequences of a frequent sequence are also frequent. They developed a method to reduce the number of simultaneous events. First, they mined for frequent itemsets in all sets of simultaneous events. A frequent itemset is then treated as a single super event. Second, break down sets of simultaneous events into super events and regular events using a two-way sorting technique, reducing the number of simultaneous events. They adapted SPAM to add time window capabilities and perform outcome analysis.

A combination of medical event data and drug prescription data was used by Norén et al. [33]. Instead of being interested in the sequential order of events,

as in most literature about frequent sequence mining, they were interested in the relative time between events, especially time between drug prescription and certain medical events. Temporal patterns between drugs and medical events were discovered, such as side effects of certain drugs.

3.2 Sequence similarity

In order to discover groups of similar sequences, we need to determine the similarity between sequences. Alternatively, we can define a distance measure. The rationale is that the smaller the distance between a pair of sequences, the higher their similarity.

To determine the distance between two sequences, a distance function, or *metric*, is required. A metric should have the following properties:

1. $dist(x, y) \geq 0$ (non-negativity)
2. $dist(x, y) = 0 \Leftrightarrow x = y$ (identity)
3. $dist(x, y) = dist(y, x)$ (symmetry)
4. $dist(x, z) \leq dist(x, y) + dist(y, z)$ (triangle inequality)

A natural approach is to define the distance between two sequences as the minimum cost to transform one sequence into the other using a small set of operators, where every operator has a certain cost. This approach was first applied to strings, so we will first discuss how to apply it on strings and then make the transition to sequences.

Levenshtein distance [25], named after Vladimir Levenshtein who invented it in 1966, is probably the best known string metric. It is also known as *edit distance*, although edit distance is in fact a larger family of distance measures of which Levenshtein distance is just a common variant. The Levenshtein distance between two strings is measured by the minimum number of insertions, deletions and substitutions required to transform one string into the other.

Take for example strings 'hospitals', 'hospitable' and 'hospice'. If the cost of every operation is 1, then we have the following alignments with minimum cost, although other alignments with equal cost are possible:

```
hospita_ls     hospic___e     hospitals
hospitable     hospitable     hospice__
```

The cost of transforming `hospitals` into `hospitable` is 2: inserting `b` and substituting `s` for `e`. The cost is 4 to transform `hospice` into `hospitable`: 1 substitution and 3 insertions. Transforming `hospitals` into `hospice` also has cost 4: it requires 2 substitutions and 2 deletions. Note that the distance is symmetric: transforming `hospitals` into `hospice` has the same cost as transforming `hospice` into `hospitals` since deletion and insertion have equal cost.

More formally, we can write down the cost of transforming one string into another using Levenshtein distance with the following (recursive) formula. Con-

sider strings a and b of length $|a|$ respectively $|b|$, then the Levenshtein distance is given by $lev_{a,b}(|a|, |b|)$ where:

$$lev_{a,b}(i,j) = \begin{cases} min \begin{cases} lev_{a,b}(i-1,j-1) + \begin{cases} 0 & \text{if } a_i = b_j \\ 1 & \text{if } a_i \neq b_j \end{cases} \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j) + 1 \end{cases} & \text{if } min(i,j) \neq 0 \\ max(i,j) & \text{otherwise} \end{cases}$$

and where $lev_{a,b}(i,j)$ is the distance between the first i characters of a and the first j characters of b and $0 \leq i \leq |a|$ and $0 \leq j \leq |b|$.

In cases where a (not necessarily non-empty) prefix of one string is compared with the empty string, the cost is simply the length of the prefix[2] since deleting all characters of the prefix will result in the empty string. This corresponds to the *max*-case. The *min*-case has three parts, corresponding to a match/mismatch, an insertion and a deletion.

The distance between two strings can be calculated using dynamic programming. An algorithm to achieve this is the Wagner-Fischer algorithm [51], which uses a matrix to store intermediate results. Every cell of the matrix corresponds to an alignment of the first i characters of the first string and the first j characters of the second string where $i, j \geq 0$. The first row and column are initialized with the cost of aligning those strings to the empty string. After initialization, the matrix can be filled in a row wise or column wise fashion. The final distance is stored in the bottom right cell. The time complexity of the algorithm is $O(|a| \cdot |b|)$ and the space complexity is also $O(|a| \cdot |b|)$.

As an example, take the matrix containing the minimum costs to transform all possible prefixes of 'hospice' into all possible prefixes of 'hospitable', shown in Table 3.5. The cost of transforming the complete strings into each other is 4, shown in the bottom right cell of the matrix. If required, the exact alignment can also be reconstructed using the filled matrix. It can be reconstructed by iteratively inspecting the three neighboring cells: left, diagonal up, and up, corresponding to respectively deletion, match/mismatch and insertion. By inspecting these cells, starting in the bottom right cell, we can determine which operations were used during matrix construction.

We will give an example of reconstructing the alignment based on Table 3.5. We start the reconstruction in the bottom right cell. We can not move to the left since it will increase the cost. According to our recurrence relation the cost should be decremented by 1 (cost of a deletion) if we would move the the left. The cost stays the same along the diagonal. This would correspond to a match and since we indeed have a match (both the column and the row contain 'e'), this move would be allowed. Additionally, we could check if we could also move up. This is not the case since the cost stays the same although moving up corresponds to an insertion. We move up along the diagonal and check the

[2]Only if indels have cost 1. Otherwise the cost of the prefix is determined by the sum of indel costs.

	λ	h	o	s	p	i	c	e
λ	**0**	1	2	3	4	5	6	7
h	1	**0**	1	2	3	4	5	6
o	2	1	**0**	1	2	3	4	5
s	3	2	1	**0**	1	2	3	4
p	4	3	2	1	**0**	1	2	3
i	5	4	3	2	1	**0**	1	2
t	6	5	4	3	2	**1**	*1*	2
a	7	6	5	4	3	**2**	*2*	2
b	8	7	6	5	4	**3**	*3*	3
l	9	8	7	6	5	4	**4**	4
e	10	9	8	7	6	5	5	**4**

Table 3.5: Alignment of two strings using Wagner-Fischer algorithm.

neighboring cells. There are two valid moves, either the diagonal (mismatch) or up (insertion). Both moves will result in an alignment with minimum cost. For this example we will take the diagonal. This approach is continued until the top left cell is reached. The alignment corresponding to our example is equal to:

```
hospi___ce
hospitable
```

This alignment is shown in bold in Table 3.5. Italic numbers correspond to cells belonging to different alignments that also have minimum cost.

The Wagner-Fischer algorithm fills a complete matrix, where it is also possible to reconstruct the alignment. However, we are not interested in the exact alignment of two sequences but only in the cost of such an alignment. Hirschberg's algorithm [16] is a modification of the Wagner-Fischer algorithm which reduces the space complexity to $O(\min(|a|, |b|)$ since it only requires two vectors instead of a matrix to store intermediate results. This modification is possible since only the previous and current row of the matrix are required to calculate the distance. The time complexity is still $O(|a| \cdot |b|)$, but a slight improvement in performance is possible since less memory has to be allocated every time a distance is calculated.

The problem of string similarity can be translated easily to sequence similarity, where the length of a sequence corresponds to the length of a string and every transaction corresponds to a character. Calculating sequence similarity using dynamic programming was proposed by Mannila et al. [28], although they also took the time between events into account: They described an event as a pair (e, t), where e is an event type and t the time the event took place. By taking

into account the time an event took place, it becomes a different problem than pure similarity between sequences of events instead of strings. They used an edit distance in combination with the following possible transformations: $indel(e, t)$, which *in*serts or *del*etes event e at time t and $move(e, t, t')$, which moves event e at time t to time t'.

The cost of inserting or deleting a specific event was proportional to the frequency of that event in a long reference sequence. As a result, the *indel* cost of a more common event is lower than a rarer event. The cost of a move was proportional to the length of the move, such that it is cheaper to move an event over a short period of time than over a longer period of time.

3.3 Clustering sequences

The goal of clustering is to group the data into different clusters where the *intra-cluster* similarity between objects is as high as possible and the *inter-cluster* similarity as low as possible. It is an unsupervised learning process. By clustering sequences, we can create groups of similar sequences and search for intra-cluster characteristics. We provide an overview of sequence clustering of healthcare data.

Sequences were first clustered based on their similarity and then an approximate sequence was generated for every cluster by an algorithm by Kum et al. [22] to discover patterns that are (approximately) present in many sequences. First the edit distance was used to calculate the similarity and then they applied their density-based "uniform kernel k-Nearest Neighbor" (k-NN) algorithm to cluster the sequences: each sequence is initially in its own cluster and clusters are merged based on the density of surrounding clusters. Approximate sequences were obtained by first applying multiple sequence alignment on the sequences of every cluster, and then selecting transactions that were shared among many sequences. The edit distance was normalized by dividing the distance between two sequences by the length of the longest of the two sequences. To extend their method to sequences of sets, they used Sørensen-Dice coefficient [48, 6] to calculate the cost of a substitution. The cost of inserting or deleting a set was defined as the cost to substitute it by the empty set, which is 1. Inserting any set of arbitrary length has therefore the same cost: in case of sequences $S_1 = \langle \{a, b\}, c, d \rangle, S_2 = \langle a, c, d \rangle, S_3 = \langle c, d \rangle$ then $dist(S_1, S_3) = dist(S_2, S_3)$ since in both cases the first set ($\{a, b\}$ and (a) respectively) is substituted for the empty set.

Patient sequences consisting of medical events were clustered with DBScan by Lakshmanan et al. [23] using a string distance metric. First, event sequences were created; second, events were mapped to single Unicode characters; third, Levenshtein distance between strings was calculated. No further details are given about their metric. They used clustering to remove outliers. To deal with events occurring on the same day (and with no more detailed timestamp available), they decreased the number of simultaneous events by creating super events of frequent simultaneous event sets, as already discussed in Chapter 3.1.1. Their approach only reduced the problem instead of eliminating it: it still re-

quired assumptions about the order of events in case these events are not in a single frequent super event. No further detail was provided about how their metric dealt with super events, e.g., it is unclear if edit operations concerning super events had same costs as normal events.

Web page browsing sessions were clustered by Wang and Zaïane [54] using sequence alignment. The alignment score between web pages was calculated with a custom scoring function, where the scoring depends on the similarity between the web page URLs. The higher the total alignment score of two sessions, the more those sessions are considered to be similar. Several clustering algorithms were applied (ROCK[12], TURN[9] and CHAMELEON[19]).

3.4 Visualization

An important aspect of sequential pattern mining is visualization. We focused on visualization of sequences in the field of healthcare. Recently, there has been a lot of research on visualization of medical data.

LifeLines [41] is one of the older visualization methods for visualizing sequences. It was only capable of visualising a single person on (multiple) horizontal timelines. The work was extended in LifeLines 2 [53], which included the option of a relative timeline view: align patient data to important medical events.

LifeFlow's [58] visualization provides an overview of all event sequences by aggregating records that have the same event sequence. Its visualization also takes into account time between events. To create a LifeFlow display, event sequences are first aggregated into a prefix tree. Every node in the tree is then visualized by a colored bar, corresponding to a color that is associated with a particular event. The bar size depends on the number of records that share the corresponding prefix. The distance between nodes at adjacent levels depends on the mean time between events. Time is displayed at the horizontal axis and the number of records is displayed on the vertical axis.

EventFlow [31] is an adaptation of LifeFlow that could not only visualize point events, but also allowed visualization of interval events (e.g., hospital bed days, drug prescription period). It was extended with capabilities to simplify the visualization [32] since results turned out to be too complex for certain datasets. By providing the user with filtering capabilities (e.g., by record, category, time, attribute) and transformation-based simplifications (e.g., interval merging in case of small gaps or small overlap, category merging in case of similarity between events) the resulting visualizations were improved.

VISITORS (VISualizatIon of Time-Oriented RecordS) [21] is also able to visualize records of multiple patients. It is an extension of the KNAVE-II system [46] which only allowed single patient visualization.

Events were aggregated in Outflow [57], where it was assumed that events are cumulative. For example, if a patient first had symptom a, then b and finally c it was treated as a the following sequence $\langle\{a\}, \{a, b\}, \{a, b, c\}\rangle$ instead of $\langle a, b, c\rangle$. So in an Outflow graph, every node contained the set of events that led to the current state. This graph was annotated with some statistics such as average

time, average outcome (e.g., did the patient survive or die) and number of patients. This graph was then used to create the Outflow visualization, which is essentially a Sankey [43] diagram. Colors of nodes and edges show the outcome (shaded from green to red) of the event and their height represents frequency. The distance between adjacent nodes represents the average time between transitions, and its horizontal position corresponds to the position in the sequence. The visualization can be used interactively to explore the data.

They also discuss an "alignment point", although it is basically a way of filtering the data. Sequences $S_1 = \langle a, b, c, d \rangle$ and $S_2 = \langle c, b, a, e \rangle$ would be transformed into $S'_1 = \langle a, \{a, b\}, \{a, b, c\}, \{a, b, c, d\} \rangle$ and $S'_2 = \langle c, \{b, c\}, \{a, b, c\}, \{a, b, c, e\}$. Now sequences S'_1 and S'_2 have an event set in common (t_3), which can be used to 'align' the sequences on. This simply means that the (unaggregated) prefixes of sequences that have been aligned, contain the same events, either in the same or different order.

Frequence [39] is similar to Outflow: It is a user interface for mining and visualizing frequent sequences and uses Sankey diagrams to visualize sequences. The main difference is that it does not aggregate data but instead showed all paths. Unlike Outflow, events were not considered to be cumulative. Care Pathway Explorer [40] was an extension and adaptation of Frequence to make it more suitable for medical data.

Chapter 4

Methods

In this chapter we will describe the contributions we made to the area of sequence mining. First, we discuss the adaptations we made to Levenshtein distance to create our edit distance metric for sequences, which is suitable for comparing the similarity of (event) sequences instead of character strings, and how domain knowledge can be incorporated into the new metric. Second, we will discuss the clustering of sequences based on the proposed metric. Third, we describe the frequent sequence mining we did before visualizing the data. We conclude with describing a new way to visualize sequences based on Sankey diagrams.

4.1 Measuring sequence similarity

In order to calculate the similarity between sequences, we need to address a couple of points: We propose a method to define costs that reflect the sequence similarity. We discuss how we treated simultaneous events. Defining costs can be difficult in case a lot of costs have to be defined, so we propose a method to make it easier for a domain expert. In the following sections we will address these issues, as well as an algorithm to calculate the distances between sequences

4.1.1 Costs of operations

Levenshtein distance has unit cost for its three edit operations: The distance between two strings is equal to the minimum number of insertions, deletions and substitutions that are required to transform one string into another, i.e., all operations have a cost equal to 1. According to [30] a variant of this metric where no substitutions are allowed was also proposed by Vladimir Levenshtein. This corresponds to using a substitution cost of 2, since it then equals the cost of an insertion plus a deletion. Other costs are also possible, resulting in Weighted Levenshtein Distance (WLD) [35] where every operation is assigned its own non-negative weight. Although different weights (i.e., costs) can be chosen, two constraints are required. First, the same cost should be used for insertions and

deletions since a metric should be symmetric. As a consequence, insertions and deletions can be treated as a single operation: an *indel*. Second, the cost of a substitution should not exceed the cost of an insertions plus a deletion, otherwise substitutions will never happen since it is cheaper to perform a deletion plus an insertion.

Besides assigning different costs to different operations, more complex weighting schemes can also be used, such as confusion matrices [30] or using costs proportional to the number of occurrences of an event [28], resulting in a lower cost to insert a common event than a rare event. We wanted the costs to reflect the similarity of events. To determine how similar events are, domain knowledge about the events is required. We made a framework where, with help from domain experts, event similarity could be quantified into costs such that the cost to transform one sequence into another reflects the similarity between the two sequences as determined by domain experts.

To achieve this, we made the following modifications to the edit distance metric: We used a vector of length n, where n is the number of unique events in the dataset, to hold the indel cost of every event. We also used an $n \times n$ substitution matrix. This matrix has to be symmetric and contains the costs to substitute one event by another. These kind of matrices are also used in bioinformatics. Different substitution matrices have been developed containing costs for replacing amino acids in protein. Two well known families of matrices are PAM [5] and BLOSUM [15] matrices, where costs depend on the likelihood that one amino acids mutates, in one or more steps, into another. The costs in our matrix should reflect the similarity between every pair of events.

One problem that might arise when choosing substitution costs is that the final substitution matrix will not satisfy the triangle inequality:

$$dist(x, z) \leq dist(x, y) + dist(y, z)$$

	a	b	c
a	0	3	1
b	3	0	1
c	1	1	0

Table 4.1: Substitution matrix.

Take for example sequences $S_1 = \langle a \rangle$, $S_2 = \langle b \rangle$ and $S_3 = \langle c \rangle$. The substitution matrix is listed in Table 4.1. According to the triangle inequality the following should hold: $dist(S_1, S_2) \leq dist(S_1, S_3) + dist(S_3, S_2)$. However, based on our substitution matrix this means that $3 \leq 2$.

One cause might be that we made poor choices when picking costs for our matrix. According to the substitution matrix, the similarity between events a and c is the same as the similarity between events b and c, while events a and b are defined as very dissimilar. So care has to be taken when choosing costs.

A simple solution to create a substitution matrix that will always satisfy the triangle inequality, is to set the substitution cost equal to the cost to delete one

event and insert the other. Continuing from the previous example would result in:

$$del(a) + ins(b) \leq del(a) + ins(c) + del(c) + ins(b)$$

It is easy to see that this will hold for all non-negative values of inserting and deleting event c. It is also possible to use a fixed ratio of the insertion and deletion costs, for example, set every substitution to half the insertion plus deletion cost which results in equal costs for insertions, deletions and substitutions.

4.1.2 Event groups

Defining a substitution cost for every event pair can be a cumbersome task. In many cases there will not be a clear definition of similarity between events, so all costs have to be determined together with a domain expert. Sometimes it might be more convenient to specify a couple of groups, where the rationale is that events belonging to the same group are more similar than events belonging to different groups. Now a distinction can be made between *intragroup* and *intergroup* substitution costs, reducing the number of costs that have to be defined. The grouping is implicitly removed if each event is assigned its own group.

In order to fill the substitution matrix, we can first create a $p \times p$ matrix, where p is the number of groups, the off-diagonal values contain the intergroup costs and the diagonal contains the intragroup costs. We can fill the substitution matrix based on this smaller matrix. Of course, the intragroup cost of two identical events should still be zero (i.e., the diagonal of the substitution matrix should be zero).

Instead of having to define $\frac{1}{2}(n^2 - n)$ values, we need only $\frac{1}{2}(p^2 + p)$ values[1]. This can be a huge advantage for a domain expert since less values have to be determined: In case a dataset has ten distinct events, 45 values have to be determined. If it would be possible to categorize those ten events in four groups, only ten values have to be determined. This can make the task of defining costs much more manageable.

4.1.3 Simultaneous events

One problem that arises when applying a string metric to sequences is the occurrence of simultaneous events (e.g., events occurring on the same day where no more detailed timestamp is available), since it is only capable of single edit operations. Costs have to be defined to insert, delete and substitute itemsets instead of items, because in many cases it would not make sense to assign the same cost to every itemset, especially when they are different in size. To illustrate the problem, take the following three itemsets: $I_1 = \{a\}, I_2 = \{b\}$ and $I_3 = \{b, c, d\}$. If we would use the standard costs of Levenshtein distance, substituting itemset I_1 with itemset I_2 would have the same cost as substituting itemset I_1 with

[1] Note the sign difference since in the first case the diagonal is 0 and in the second case the diagonal contains intragroup substitution costs.

itemset I_3, although the latter is clearly less similar than I_2 with respect to I_1.

There are several possibilities to deal with itemsets. We could define some artificial ordering of the events, as in [23], and then treat a sorted itemset as a subsequence of the entire sequence. For example, we could transform an itemset into an alphabetically sorted subsequence. This ensures that every itemset is treated in the same way. Consider sequence $S = \langle b, \{c, a\}, \{a, g, d, f\}, c\rangle$. It would be transformed into $S' = \langle b, a, c, a, d, f, g, c\rangle$.

An alternative is to keep the ordering as present in the data and assume that it equals the order in which the events took place. In that case S would be transformed into $S' = \langle b, c, a, a, g, d, f, c\rangle$.

A third option, that might be feasible in some situations, is to reconstruct the order the events (probably) took place, and again treat them as a sequence.

However, for all these approaches assumptions have to be made about the order the events took place. We think it would be better to minimize the number of assumptions and directly deal with itemsets. Of course, some methods exists to calculate similarity between sets, such as Jaccard Index [18] and the already mentioned Sørensen-Dice coefficient [48, 6]. However, these methods are not suitable for our task. The set similarity should not be expressed as a value based on how many elements they have in common, but should take the costs into account to transform one itemset into the other in the same way we already defined it for items. Another issue is that these methods normalize the similarity value between 0 and 1. We want to apply normalization only on the entire sequence, not on individual transactions.

We wanted the cost to reflect the cost to transform one itemset into the other (in case of an indel the cost to transform it into the empty set). To achieve this, we propose the following: The indel cost of an itemset should be equal to the cost of inserting (or deleting) all events in the set, and the substitution cost of an itemset should be equal to the minimum cost (defined by edit operations) to transform one itemset into the other. To accomplish this, we treated every itemset as a "super event" and changed our substitution matrix to hold all "super events" that occurred in the data instead of all events.

A problem that arose is that it matters which event pairs are substituted, and which are inserted or deleted. Consider itemsets $I_1 = \{a, d\}$ and $I_2 = \{c\}$. To transform I_1 into I_2 we could perform either $sub(a, c) + indel(d)$ or $sub(d, c) + indel(a)$. Therefore, the minimum transformation cost has to be determined.

To calculate the minimum cost, the number of insertions, intragroup substitutions and intergroup substitutions have to be determined. Items that occur in both itemsets are removed by taking the relative complement, because those do not need to be substituted. Take itemsets A and B. Then $A' = A \setminus B$ and $B' = B \setminus A$ such that $A' \cup B' = A \triangle B$ (symmetric difference). The number of substitutions is equal to the set with lowest cardinality: $min(|A'|, |B'|)$. The number of indels equals the difference in cardinality: $max(|A'|, |B'|) - min(|A'|, |B'|)$.

Based on the number of substitutions, the number of intragroup substitutions

can be determined. The number of intragroup substitutions should be maximal since these are cheaper than intergroup substitutions. In order to obtain the intragroup and intergroup substitutions, the frequency of every event group is calculated for sets A' and B'. The lowest frequency of every group corresponds to the number of intragroup substitutions that can be performed for that group. The remaining substitutions are intergroup substitutions.

It is trivial to calculate the total cost of intragroup substitutions: it is just a summation of substitution costs. Determining the costs of intergroup substitutions is harder since these should be minimized.

Consider itemsets A' and B' again, and the number of intergroup substitutions s and indels d. Let A'' and B'' be sets where all events have been removed for which an intragroup substitution should be performed. Then the cost that has to be minimized equals:

$$min\left(\sum_{i=1}^{s} sub(x_i, y_i) + \sum_{j=1}^{d} indel(z_j)\right)$$

where $x_i \in A'' \wedge y_i \in B'' \wedge (z_j \in A'' \vee z_j \in B''$. Every element of A'' and B'' should be used exactly once, as an item can not be part of a substitution (or insertion/deletion) multiple times. In other words, the union of all x_i's, y_i's and z_j's is equal to $A'' \cup B''$. To obtain the total event set substitution cost, the cost of intragroup substitutions has to be added.

Let us look at an example to illustrate the entire process. Say we have itemsets $A = \{a, b, x, y\}$ and $B = \{a, c, d, w, z\}$ and four groups $G_1 = \{a, b, c, d\}$, $G_2 = \{x\}$, $G_3 = \{y\}$, $G_4 = \{w, z\}$. The first step is to remove common elements: $A' = \{b, x, y\}$ and $B' = \{c, d, w, z\}$. The number of substitutions equals $min(|A'|, |B'|) = 3$ and the number of indels equals $max(|A'|, |B'|) - min(|A'|, |B'|) = 4 - 3 = 1$.

The next step is to calculate for both itemsets the frequency of every group. In A' groups G_1, G_2, G_3 occur once. In B' groups G_1 and G_4 both occur twice. The minimum frequency of groups G_1, G_2, G_3 and G_4 is respectively 1, 0, 0 and 0. Therefore, one intragroup substitution should be performed: $sub(b, c)$ or $sub(b, d)$, which have equal cost since it is an intragroup substitution. We pick $sub(b, c)$. The remaining substitutions, in this case two, are intergroup substitutions. After removing the items that were used in the intragroup substitution we have sets $A'' = \{x, y\}$ and $B'' = \{d, w, z\}$. The cost to transform A into B equals:

$$sub(b, c) + min\begin{cases} \mathbf{sub(x, d)} + \mathbf{sub(y, w)} + \mathbf{indel(z)} \\ \mathbf{sub(x, w)} + \mathbf{sub(y, d)} + \mathbf{indel(z)} \\ \mathbf{sub(x, z)} + \mathbf{sub(y, w)} + \mathbf{indel(d)} \\ sub(x, d) + sub(y, z) + indel(w) \\ sub(x, w) + sub(y, z) + indel(d) \\ sub(x, z) + sub(y, d) + indel(w) \end{cases}$$

Since w and z are elements of the same group it is sufficient to calculate the costs marked in bold.

4.1.4 Distance calculation

With our metric in place, we were able to calculate distances between sequences. We used Hirschberg's algorithm [16] and modified it to calculate the proposed metric. In Algorithm 1 the algorithm to calculate the distance between sequences is listed. Vectors $v0$ and $v1$ are used to store the cost of the alignment.

To make the edit distances comparable between sequences of various lengths, we normalized the results by dividing the distance by the maximum of indel costs of the two sequences. However, distances will not satisfy the triangle inequality after normalization. We conducted experiments to assess the impact of normalization.

Only a triangular distance matrix has to be calculated. To take advantage of datasets where there are many duplicate sequences, we only calculated the distances between unique sequences, and used these to fill the complete distance matrix. This can greatly reduce the running time of the algorithm.

4.2 Clustering

We used the distance matrix as input to cluster sequences. It is not possible to apply k-means [27] on the data since there is no clear concept of a *mean* because we only have distances between sequences and not its position in an Euclidean space. We chose an agglomerative hierarchical clustering method, which means it is bottom-up, starting with every sequence in its own cluster, and merging the most similar cluster pairs until there is only a single cluster left. To determine which clusters should be merged we used Ward's method [55]. It minimizes the total within-cluster variance. A Lance-Williams algorithm [24] was used in combination with Ward's method [56] to create a hierarchical clustering, although we also experimented with other linkage criteria.

We also experimented with k-medoids [20] clustering using the partitioning around medoids (PAM) algorithm.

4.3 Frequent Sequences

An important question is what care patients received until or around a certain event. To provide this insight we wanted to visualize as much treatment data of the patient population as possible, without making the visualization too complex. If we would mine for all frequent sequences we could get multiple patterns per sequence, especially when sequences are long. However, we wanted only a single pattern per sequence. To achieve this, a couple of constraints were required for mining. We will discuss two groups of frequent sequences that we

Algorithm 1: Edit distance with custom costs

function Levdist $(a, b, sm, indel)$
Input : Sequences a and b, substitution cost matrix sm and indel cost vector $indel$, where sm and $indel$ are hash tables
Output: dist(a, b)
`/* Identical sequences */`
if $a = b$ **then**
  return 0
end

`/* Empty sequence */`
if $\min\big(|a|, |b|\big) = 0$ **then**
  `/* return sum of indel costs of all elements of the non-empty sequence */`
  if $|a| = 0$ **then**
    return $\sum_{n=0}^{|b|-1} indel\big[b[n]\big]$
  else
    return $\sum_{n=0}^{|a|-1} indel\big[a[n]\big]$
  end
end

`/* Initialization: cost to align first` i `events of sequence` b `to empty sequence */`
$v0[0] \leftarrow 0$
for $i \leftarrow 1 : |b|$ **do**
  $v0[i] \leftarrow \sum_{n=0}^{i-1} indel\big[b[n]\big]$
end

`/* Calculate cost to align first` i `events of sequence` a `to first` j `events of sequence` b `*/`
for $i \leftarrow 0 : |a|-1$ **do**
  $v1[0] \leftarrow \sum_{n=0}^{i} indel\big[a[n]\big])$
  for $j \leftarrow 0 : |b|-1$ **do**
    $cost \leftarrow sm\big[a[i], b[j]\big]$
    $v1[j+1] \leftarrow min \begin{cases} v1[j] + \text{indel}\big[b[j]\big] \\ v0[j+1] + \text{indel}\big[a[i]\big] \\ v0[j] + cost \end{cases}$
  end
  $v0 \leftarrow v1$
end

return $v1\big[|b|\big]$ `// last element of` $v1$ `contains total cost`

mined, used as input for two different types of visualization. The visualization based on the mined sequences is discussed in Chapter 4.4.

A couple of constraints on the mined patterns were required. We were only interested in frequent sequences that contained certain events at a specific position in the sequence, where the position is either absolute or relative. We were also only interested in contiguous sequences and we did not take subsets of event sets into account (i.e., an event set was treated as as single super event). We either mined for frequent prefixes or mined for frequent subsequences that contained (the first occurrence of) a specific event. In the latter case, multiple events or event sets could be specified. The first group of sequences consisted of prefixes, were a prefix of a sequence $S = \langle t_1, \ldots, t_n \rangle$ corresponds to $S' = \langle t_1, \ldots, t_m \rangle$ where $1 \leq m \leq n$. This definition corresponds to the definition of prefix of a string, which should also be contiguous.

The second group contained contiguous subsequences $S' \preceq_T S$ such that $S' = \langle t_j, \ldots, t_k \rangle$ where $1 \leq j \leq k \leq n$ and $\{S \in D \mid \exists S_{(i)} \in T\}$, where T is the set of events in which we are interested, D the set of sequences and i an arbitrary transaction of S. The first group of sequences can be expressed as the latter by making the set T contain all events that occur in the data. Then the selected subsequences is determined by the set $\{S \in D \mid S_{(1)} \in T\}$.

Sequence
$S_1 = \langle b, c, b, b \rangle$
$S_2 = \langle b, c, b, a \rangle$
$S_3 = \langle b, c, a, b \rangle$
$S_4 = \langle b, a, e \rangle$
$S_5 = \langle c, a, b, a \rangle$
$S_6 = \langle a, e \rangle$

Table 4.2: Sequences.

Sequence	Freq.
$F_1 = \langle b, c, b \rangle$	2
$F_2 = \langle b, c \rangle$	3
$F_3 = \langle b \rangle$	4
$F_4 = \langle c \rangle$	2

(a) Frequent prefixes.

Seq.	t_1	t_2	t_3	t_4
S_1	**b**	**c**	**b**	b
S_2	**b**	**c**	**b**	a
S_3	**b**	**c**	a	b
S_4	**b**	a	e	
S_5	**c**	a	b	a
S_6	a	e		
S_7	**c**	b	a	b

(b) Selection of sequence marked in bold.

Table 4.3: Mining frequent prefixes.

We will illustrate the patterns that we were interested in with an example. The set of sequences is listed in Table 4.2. In case of mining for frequent prefixes that occur at least twice in the data, the set of frequent sequences would be the set of sequences listed in Table 4.3a. Note that sequence $\langle b, a \rangle$ occurs three times (S_2, S_4, S_5), but only once as a prefix and therefore is not listed in Table 4.3a. Table 4.3b shows exactly what part of every sequence is selected based on the frequent sequences.

In case of mining frequent sequences with the constraint that it should contain the first occurrence of a specific event, the selection would be different. We use the same set of sequences and support of the previous example, but now a frequent sequence should contain event a. The resulting set of frequent sequences and corresponding frequencies are listed in the first two columns of Table 4.4a. Table 4.4b shows our data selection, where t_i is the first transaction that matches with our selection criterion. There are a couple of observations to make. First, sequence S_1 is missing since it does not contain event a at all. Second, although

Sequence	Freq.	
$F_5 = \langle c, a, b \rangle$	2	2
$F_6 = \langle c, b, a \rangle$	2	2
$F_7 = \langle a, b \rangle$	3	2
$F_8 = \langle a, e \rangle$	2	1
$F_9 = \langle b, a \rangle$	3	2
$F_{10} = \langle c, a \rangle$	2	2
$F_{11} = \langle a \rangle$	7	7

(a) Frequent sequences containing a.

Seq.	t_{i-3}	t_{i-2}	t_{i-1}	t_i	t_{i+1}	t_{i+2}
S_1	-	-	-	-	-	-
S_2	b	**c**	**b**	**a**		
S_3		b	**c**	**a**	**b**	
S_{4_a}			b	**a**	**e**	
S_{4_b}			**b**	**a**	e	
S_5			**c**	**a**	**b**	a
S_6				**a**	**e**	
S_7		**c**	**b**	**a**	b	

(b) Selection of sequence marked in bold.

Table 4.4: Mining frequent subsequences containing particular event.

frequent sequence F_9 occurs four times in the data it is only listed three times since in sequence S_5 there is an event a that occurred earlier in the sequence. Third, note that sequence S_4 is listed twice: it contains two frequent sequences of maximum length. Before we can visualize the result we would have to choose which one to pick, since otherwise we would either end up with two sequences for a single patient or, if we would take the union and visualize sequence $\langle b, a, e \rangle$, we would get a non-frequent sequence. A consequence is that the resulting visualization might contain infrequent sequences. If we would pick S_{4_a} then F_9 would not be frequent, and if we would pick S_{4_b} then F_8 would not be frequent. Fourth, sequence S_7 is only listed once, although it contains two frequent sequences: F_6 and F_7. We are only interested in the longest frequent sequences that is present in the sequence.

The third and fourth observation brings us to the following: although we mine for frequent sequences, it does not mean that they will be frequent in the visualization. If we would pick S_{4_b} over S_{4_a} we would end up with the frequencies listed in the last column of Table 4.4a, resulting in the support of F_8 drop below our threshold and in a decrease in support of F_9.

In order to determine which frequent sequences should be visualized and what their frequency should be, an extra pass over the dataset is required after mining. This pass is also used to determine if a frequent sequence is a proper (contiguous) subsequence and if so, determine if it is a proper prefix, a proper suffix or neither. This information was used to indicate in the visualization if a complete path is visualized or only part of it.

Since we are interested in pattern that start at a certain position, either absolute (first group of frequent sequences) or relative (second group), certain patterns that occur in the data might have high enough support, but will not be mined since they will not fulfill the constraints we imposed. Therefore, it could be helpful to also produce a set of frequent sequences where no such constraints were imposed. This set does not have to be visualized but could help with discovering patterns in the data when used in conjunction with the visualization.

4.4 Visualization of Sequences

Sankey diagrams [43] are a powerful visualization method for sequential data. They can be used as a starting point to get insight in data, or as a final step to visualize results. It is a flow diagram where the width of nodes and edges is proportional to the flow quantity. Usually, every path starts completely to the left of the diagram with the flow moving to the right through several nodes until an end node is reached.

It can be used for sequence visualization [39] by creating a node at level i of the diagram for every unique transaction t that occurs at position i in the sequence: The set of nodes for an arbitrary level of the Sankey diagram is defined as $N_i = \{t_i \mid t_i = S_{(i)} \colon S \in D\}$. Edges to nodes at level i (and coming from level $i-1$) are determined by the set of unique length-2 subsequences $E_i = \{S' \mid S' = \langle S_{(i-1)}, S_{(i)} \rangle \colon S \in D\}$. The width of every node $n \in N_i$ and edge $e \in E_i$ should be proportional to their frequency.

Sankey diagrams are used primarily for visualizing relatively short sequences with a few distinct events. A potential risk is that the whole diagram will become too cluttered in case too many transitions and events are visualized.

We extended the Sankey diagram to create a new method to visualize sequences.

4.4.1 Aligned Sankey diagrams

We combined existing visualization techniques to address situations where the main interest lies in events occurring at or around a specific point in the sequence. Sankey diagrams have already been used to visualize (medical) sequences [57, 39, 40]. Aligning sequences in a visualization has also been done [53, 31, 32]. We combined these two techniques for our visualization interface to make it easier to get insight in sequential data.

It might not always be convenient to start all sequences at the first level of the diagram. Therefore, we introduced a Sankey diagram where a sequence may start at an arbitrary level in the diagram, unlike [57] where although they discuss an alignment point, sequences still start at the first level of the diagram since it always takes n transactions to reach the state shared by all sequences. Aligning sequences is useful in situations were we are interested in the relative position of an event in a sequence instead of the absolute position. It allows us to center the Sankey diagram on a specific event. This provides insight in the events that occurred before and after that specific event. This is not clear in the traditional Sankey diagram, where the selected event can occur at many levels. This can be especially useful when the same events happen before and after the alignment point.

We filtered a dataset such that every sequence contained event F to make a fair comparison between the two methods. In Figure 4.1 the 'traditional' Sankey diagram is shown. Figure 4.2 contains the same data[2], but this time event F

[2]We applied edge filtering to simplify the diagram. Every edge now has at least a frequency of 5. Therefore, not the exact same data is visualized.

was used to align the Sankey diagram on[3]. Here it is immediately visible which events took place before and after event sets containing event F, as where as in Figure 4.1 event F occurs at all three levels of the Sankey diagram (sometimes preceded by another occurrence of event F.

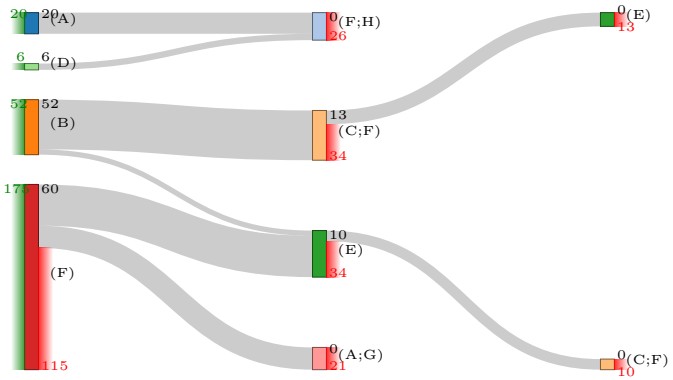


Figure 4.1: 'Traditional' Sankey diagram where sequences start at the first level of the diagram.

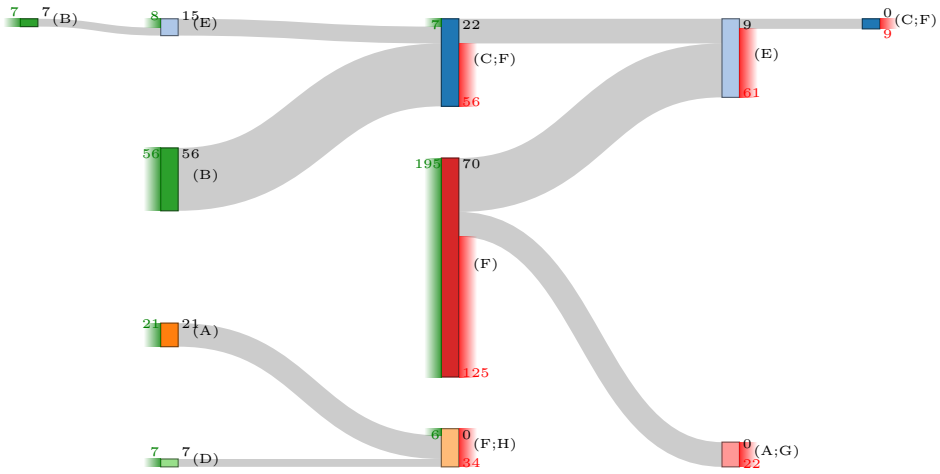


Figure 4.2: 'Aligned' Sankey diagram containing the same data as Figure 4.1 but aligned on event F.

Additionally, we can indicate if a subsequence was part of a longer sequence. Sometimes the entire sequence is visualized, but sometimes only a subsequence. In Figure 4.3 we indicate in blue the longer sequences that were cut off instead of treating them as sequences that started or terminated there, as indicated by green and red respectively.

[3]All event sets holding event F were also used to align on.

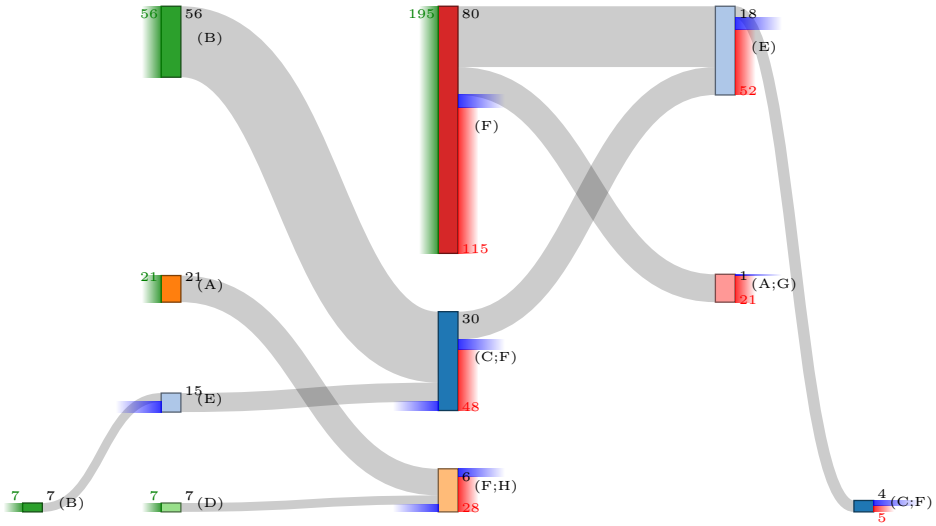


Figure 4.3: 'Aligned' Sankey diagram containing the same data as Figure 4.2 but now indicated if a sequence was terminated before it was completely visualized.

4.4.2 Edge filtering

In Chapter 4.3 we discussed frequent sequence mining as an optional preprocessing step before visualizing the data with a Sankey diagram. Using frequent sequence mining to remove uncommon subsequences can help reduce the amount of data that has to be visualized. A disadvantage was that, although we mined for patterns that met minimum support, it was not guaranteed that the resulting visualization would not contain paths with a lower frequency. Here we present an alternative method that can be used to reduce the complexity of the Sankey diagram. It can be used in conjunction with frequent sequence mining, and provides more control over the data that will be visualized.

To balance the quantity of visualized data and to reduce clutter, we made a distinction between the frequency of a sequence and the frequency of edges. The frequency of sequences in the Sankey diagram is mainly determined by frequent sequence mining. In this section we will look into the frequency of the edges between nodes.

Before data is visualized, the frequency every edge is checked, and sequences that contain an infrequent edge are pruned. Pruning infrequent edges can remove rare transitions caused by frequent sequences dropping below support, since not every frequent sequence will end up in the diagram with the same support it was mined. A second use case is to further limit the number of displayed nodes and links. A node or link will often be part of multiple unique sequences. So on average, a node or link will have a higher frequency than the minimum support threshold. There might be links (or nodes) that are only visited by very few frequent sequences, and we might want to remove them to improve the visualization.

In order to prune a Sankey diagram, we start with the transactions at t_i, where

t_i corresponds to the alignment point (in case of no alignment, this simply is t_1), and determine the frequency of all unique edges, i.e., subsequences of length 2 that start at a certain position: $\langle t_{i-1}, t_i \rangle$. If the edge frequency is below the threshold the sequence is pruned, removing all transactions to the left of t_i. Then i is decreased by one, and again the edge frequency is determined. This process is repeated until the frequency of $\langle t_1, t_2 \rangle$ is determined. Analogous, a pass to the right is performed, until the frequency of $\langle t_{n-1}, t_n \rangle$ is checked. Pruned sequences are marked, such that we can indicate in the Sankey diagram that they are not completely displayed.

We demonstrate the difference between edge filtering and frequent sequence mining with the following example. The dataset consists of the sequences displayed in Table 4.5 and the corresponding Sankey diagram containing every sequence is shown in Figure 4.4.

Sequence	Freq.
$\langle a, q, a \rangle$	2
$\langle a, q, a, e \rangle$	2
$\langle a, q, b \rangle$	2
$\langle b, q, a \rangle$	4
$\langle b, q, b \rangle$	1
$\langle b, q, b, e \rangle$	2

Table 4.5: Sequence frequency.

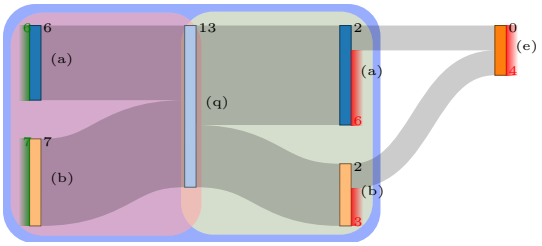


Figure 4.4: Sankey diagram.

If we set our minimum support threshold to a frequency of 5 and mine for frequent sequences, we get the Sankey diagram shaded in pink in case of mining for prefixes. In case of an alignment on q we get the Sankey diagram shaded in either pink or yellow, depending on a preference for events occurring before or after the alignment point. This is caused by the lack of frequent sequences of length 3. However, if we would not mine for frequent sequences but filter out edges with a frequency below 5, the blue shaded part would be visualized. So now every transition has at least a frequency of 5, although sequences have lower support, for example, $\langle a, q, b \rangle$ has a support of 2. But since 13 sequences pass through node q, we might be interested in this flow.

Of course, we could reduce the minimum support threshold, but we would require a minimum support threshold of 2 to visualize the blue shaded area, which would also include event e. We could remove this by setting an edge filter threshold between 3 and 5. Increasing the minimum support threshold to 3 instead of 2 would make sequence $\langle a, q, b \rangle$ infrequent, so only $\langle a, q \rangle$ or $\langle q, b \rangle$ would be visualized (in case of alignment). This might be an unwanted result since many sequences contain events a and b and they might be interesting enough to keep. Combining frequent sequences with defining an edge frequency can help to fine tune the visualization.

4.4.3 Implementation

Our Sankey diagrams were created using Javascript library D3.js[4] and a Sankey plugin[5], both created by Mike Bostock. We adapted the plugin to be able to start sequences anywhere in the Sankey diagram, and used it to built our sequence visualization interface.

We added the visualization of number of sequences that start, terminate or were cut off due to a too low frequency of a subsequence at every node. Additionally, we added some capabilities to interact with the diagram. By selecting a node, only sequences are visualized that pass through that node. When several nodes are selected, all paths that pass through at least one of those nodes are shown. Another possibility is to deselect a node, removing all sequences that pass through it. These data filtering methods can be combined to interactively explore the data, although it is recommended to perform some data filtering as a preprocessing step.

[4] www.d3js.org
[5] www.github.com/d3/d3-plugins/tree/master/sankey

Chapter 5

Experiments

In this chapter we will discuss the experiments we conducted. We ran experiments on data provided by ZIN, and discussed the results with a domain expert to improve them. We first discuss the type of data we used and the preprocessing we did. Finally, we describe how we defined the similarity between specific events. The results of the experiments are described in Chapter 6.

5.1 Data origin and selection

The dataset we used in our experiments consisted of medical records that were extracted from administrative processes. It contained care and treatment information from all Dutch hospitals and was provided to us by Zorginstituut Nederland. It was extracted from DBC-Informatiesysteem (DIS, DBC Information System). DBC stands for Diagnose Behandel Combinatie (Diagnosis Treatment Combination) and was collected by the Nederlandse Zorgautoriteit (NZa, the Dutch regulatory agency tasked with supervising the Dutch care market).

The DBC-system is used to determine the funding hospitals and medical specialists in The Netherlands should receive. A DBC contains the total treatment of a patient for a certain diagnosis. The diagnosis together with the corresponding treatment leads to a single price that care providers can claim from insurance companies, using an administrative code: the DBC-code [61]. The cost of a DBC is based on an 'average' treatment, which allows for small deviations in the treatment to better suit a patient's needs without affecting the cost of the DBC.

We only worked with a very small selection of DIS-data. For privacy reasons, it does not contain any information that could be traced back to specific individuals, and does not include outcomes of medical tests. It does contain some information about the event, such as sex and year of birth of the patient, a product code, what hospital was responsible for the medical test and date of the test.

Name	Codes	Group
cycle ergometry (cycle test)	039844, 039845	1
echocardiogram (echo)	039494, 085070	1
dobutamine stress echocardiogram (echo stress)	039495	2
dobutamine stress MRI (MRI stress)	039507, 085191	2
SPECT stress	120241, 120246	2
SPECT	120240, 120244, 120245	2
MRI	039506, 085190	3
coronary calcium scan (CT calcium)	085141, 039497	4
CT-scan (CT)	085042, 085140	4
angiography (angio)	033229, 039520, 085720	5
PCI	033231, 033232, 033233, 033234, 033238, 190342, 190343	5

Table 5.1: Names of medical tests with corresponding codes and groups.

The data we worked with consisted of records of patients diagnosed with chest pain. Two datasets were created with with help of a domain expert.

The complete set of activities that we used is listed in Table 5.1, together with the corresponding codes. Table 5.2 contains some statistics about the two experiments with chest pain data.

Description	Sequences	Unique seq.	Events	Event sets
Experiment 1	10.035	554	12	66
Experiment 2	10.035	1.633	12	70

Table 5.2: Statistics of the two datasets used in our experiments.

5.2 Data description

The domain expert created several groups of similar activities. These groups are shown in Table 5.1, although the initial grouping was slightly different: According to the domain expert MRI should be in a group of its own, but this did not give the expected result. MRIs were in the same cluster with sequences containing echos and cycle tests, although an MRI is a more severe medical test and will give you more information than can be obtained with echos and cycle tests. We decided to put it in the same group with MRI stress and the other stress tests. The new clusters were considered better by the expert.

Different costs for different event groups were used, based on input from a domain expert and adjusted during experiments. The costs were kept relatively simple. We set the intergroup substitution costs equal to the cost to delete one event (or super event) and insert the other, practically eliminating substitutions

in case of intergroup substitutions. An intragroup substitution had the same cost as the cost of a single indel of that group. This means that in case we would set all indel costs to 1, intergroup substitutions would have cost 2 and the cost of intragroup substitutions would stay 1.

However, not every medical test is as severe as the other. Additionally, some tests are cheaper, or provide more information about the health of a patient. For example, we set the indel cost of group 1 much lower, since it contains cheap medical tests which do not tell much about the health condition of a patient. We used a substitution cost of 0 for group 5, since a PCI also contains an angiography and therefore it does not matter which one a patient received.

5.3 Evaluation

We clustered sequences in order to evaluate the metric. We used a dendrogram, a "'tree-like' diagram illustrating the series of steps taken by the method in proceeding from n single member 'clusters' to a single group containing all n individuals" [8], to determine the number of clusters. We also calculated the average silhouette width [45] to decide on the number of clusters. Silhouette coefficients provide an indication how well points fit into the cluster they have been assigned to. The average value provides an indication of the overall validity of the clustering.

After assigning on the number of clusters, we wanted to analyze the clusters. However, we only have a distance matrix. To inspect how well we clustered the sequences, we used classical multidimensional scaling [50] (MDS, also known as principal coordinates analysis [11]) to project the distance matrix onto a plane. It is a generic method to reduce the data dimensionality. The aim of MDS is to place every object of a distance matrix in an n-dimensional space (in our case n=2) were distances are preserved as good as possible. We indicated to which cluster every point was assigned to. This allowed us to visualize the clusters and get an insight in the spatial relation between sequences.

A Sankey diagram was created for every cluster. These served as a method to present the clusters to the domain expert in order to discuss results.

Chapter 6

Results

As described in the previous chapter, we conducted experiments on two different datasets about patients diagnosed with chest pain. The results of the experiments on these different sets are discussed in the following two sections.

6.1 Experiment 1

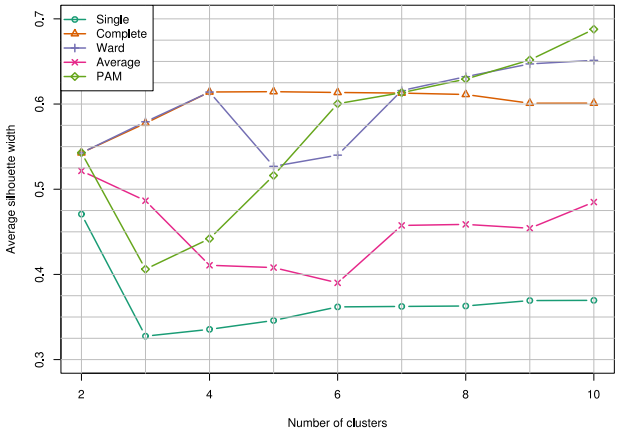


Figure 6.1: Average silhouette width for hierarchical clustering with different linkage criteria and for k-medoids.

We experimented with several linkage criteria: Single linkage, complete linkage, average linkage and Ward's criterion. We also experimented with partitioning around medoids (PAM). To compare how well each method performed, we calculated the average silhouette width for every method and for different number of clusters. The results are shown in Figure 6.1. It is interesting that in the beginning PAM produces clusters with a low average silhouette width, but

improves when the number of clusters increases. Complete linkage and Ward's criterion perform best in case of few clusters. The clustering of complete linkage and Ward's criterion is very similar for four clusters, illustrated by the confusion matrix shown in Table 6.1a where most values are on the diagonal. The corresponding dendrograms are shown in Figure 6.2.

Ward's criterion \ Complete linkage	1	2	3	4
1	7273	1	0	0
2	0	1446	0	0
3	0	2	906	0
4	0	0	0	407

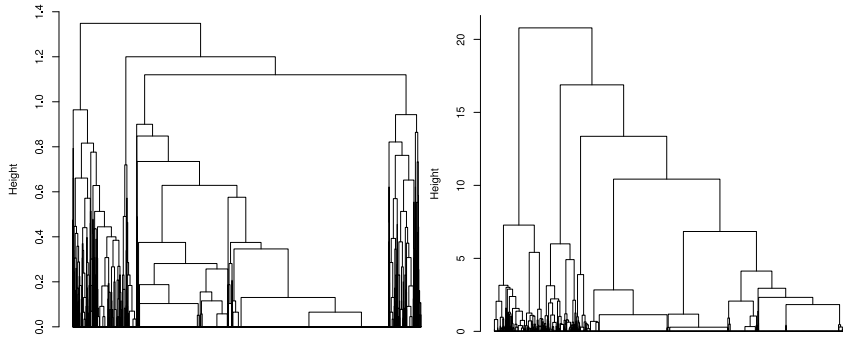
(a) Ward's criterion vs. complete linkage.

With normalization \ Without normalization	1	2	3	4
1	7274	0	0	0
2	89	1357	0	0
3	62	22	824	0
4	0	0	0	407

(b) Ward's criterion, with and without distance normalization.

Table 6.1: Confusion matrices.

Based on silhouette plots and dendrograms we picked Ward's method with four clusters to look into the impact of distance normalization. As mentioned in Chapter 4.1.4, normalization made the triangle inequality fail. To assess the impact we compared cluster solutions created with and without normalization. It turned out the differences were only minor, as shown in Table 6.1b.



(a) Complete linkage. (b) Ward's criterion.

Figure 6.2: Dendrograms for complete linkage and Ward's cluster criterion.

We applied Multidimensional Scaling [50] to project the distance matrix onto a plane, which could help to inspect how well the data is clustered. The result of MDS applied to the distance matrix is shown in Figure 6.3, where every point is colored according to the cluster it belongs to. Clusters are separated quite well, although it could be interesting to further examine the result, where subclusters seem to be present. Based on Figure 6.3 there is a group, in the bottom right

corner, containing sequences of Cluster 2 and Cluster 3. There is also a group of four points, in the top center, belonging to Cluster 3 which are far away from the other points of Cluster 4.

One cluster consisted out of patients that received predominantly either a spect or spect stress, together with some other medical tests that belonged to the same event group. Another cluster consisted out of patients that did not receive much care: mostly echos and cycle tests. A third cluster consisted out of patients that received either a CT or CT calcium. A fourth cluster contained patients who had an angiography, a CT and often a CT calcium on the same day. These clusters are visualized as Sankey diagrams and listed, together with the complete dataset, in Appendix A.1-A.5.

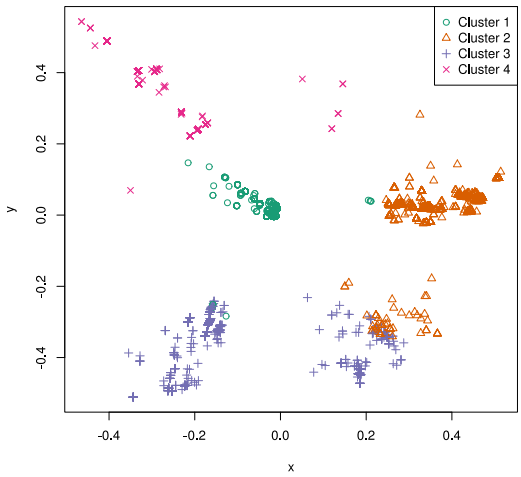


Figure 6.3: Clusters after projecting them on a plane using MDS.

We discussed these clusters with the domain expert. According to the expert the first three clusters made sense from a medical point of view. The fourth cluster was of interest since it contained patients that received multiple tests on the day of their angiography.

6.2 Experiment 2

In this experiment we applied the same clustering algorithms as in our previous experiment, resulting in the average silhouette widths shown in Figure 6.4. The average silhouette width is smaller than in our previous experiment, indicating lower quality clusters. Ward's criterion seemed to perform quite well for six clusters. Again, single linkage and average linkage did not perform well. The result of PAM improved again after a dip in silhouette width, although it stays below the initial width for two clusters. The dendrogram corresponding to Ward's criterion is shown in Figure 6.5. We chose a six cluster solution. The

corresponding Sankey diagrams, together with the complete dataset, are listed in Appendix A.6-A.12

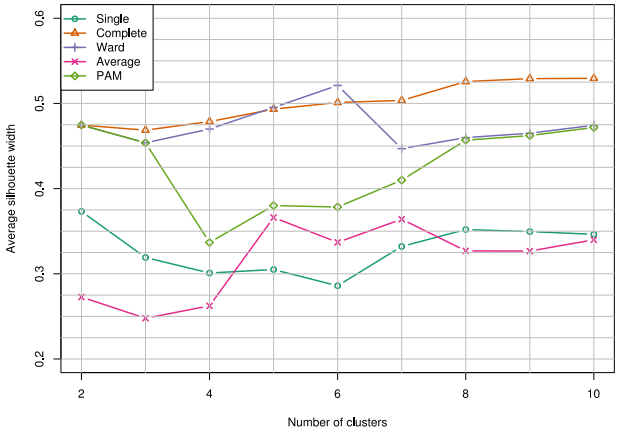


Figure 6.4: Average silhouette width for hierarchical clustering with different linkage criteria and for k-medoids.

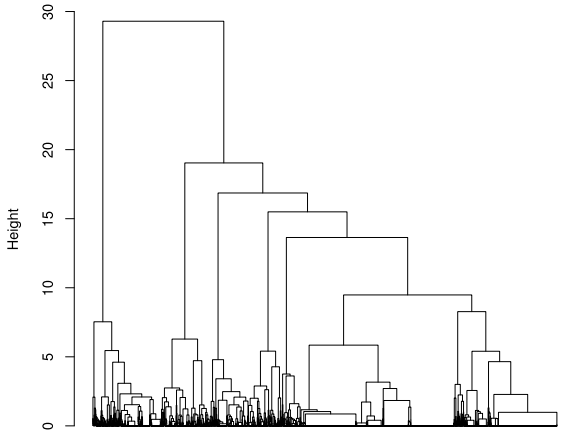


Figure 6.5: Dendrogram for Ward's criterion.

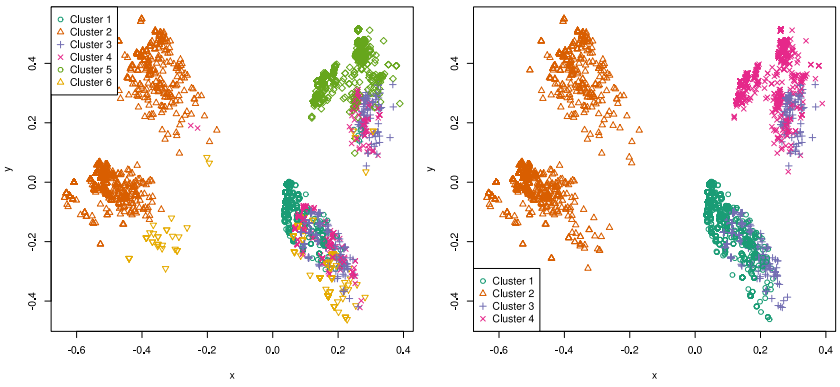
Whether or not distances are normalized does not change much to the resulting clusters, as shown by the confusion matrix listed in Table 6.2.

The result of MDS seems to indicate that there are four clusters, which are also not well represented by the produced clustering. For example, there are two clear groups of points present in Figure 6.6a consisting of points labelled as Cluster 2. The other two groups contain a mixture of points that are assigned to different clusters. Decreasing the number of clusters to four illustrates a problem of interpreting the result of MDS: Table 6.6b shows the cluster membership in case of four clusters, which do not correspond to the four groups that are clearly present: Cluster 3 seems to be mixed with sequences belonging to Cluster 1 and

With normalization \ Without normalization	1	2	3	4	5	6
1	5533	0	0	0	0	0
2	0	1476	1	0	0	0
3	0	0	804	0	1	0
4	27	0	0	683	18	0
5	5	0	0	0	1095	0
6	20	0	0	0	1	371

Table 6.2: Confusion matrix of Ward's criterion; with and without distance normalization.

Cluster 4. However, since MDS can not preserve all distances we can only conclude that clusters exist in the higher dimensional space if they are present in the lower dimensional space, not the other way around: Based on Figure 6.6a, where there are two groups consisting out of points assigned to different clusters, we can not conclude that sequences were clustered poorly. However, it could be interesting to, for example, compare the two groups of points labelled as Cluster 2 and examine the differences.



(a) Six clusters.

(b) Four clusters.

Figure 6.6: Clusters after projecting on a plane using MDS.

Chapter 7

Discussion and Conclusions

We made two contributions. We created a generalized distance metric for sequences based on a string metric. The proposed metric is suitable for datasets that contain simultaneous events and supports event groups to improve usability. We also proposed a novel method for sequence visualization based on a Sankey diagram and suggested three methods to reduce the data before visualization. We demonstrated our methods on a leading example by using data from the health care industry.

Our new metric allows incorporation of domain knowledge, which can help discovering interesting patterns in sequential data. Domain knowledge can be integrated by specifying costs to quantify event similarity. Using groups reduces the number of costs that have to be defined by a domain expert, which simplifies the task since it can be hard to determine proper costs. We took into account the occurrence of simultaneous events to better serve situations where the exact order of events is unknown due to too imprecise timestamps. By clustering sequences based on our distance measure, groups of similar sequences can be obtained.

The new sequence visualization method allows for fast insight in what events occurred before and after a user-defined critical moment in a set of sequences. This is achieved by aligning a Sankey diagram on user specified events. We created a visual interface where users can interact with the Sankey diagram. Interacting with the visualization can further help domain experts to reach new insights.

We proposed three methods to reduce the data before visualization, which can be used in conjunction with each other: Sequences can be clustered into groups of similar sequences, visualizing each group individually. Frequent sequence mining can be applied to remove uncommon sequences. Edges with low frequency can be removed to delete rare transitions which clutter the diagram. Of course, it is up to the user to decide what input is given to the Sankey diagram. We decided to only visualize contiguous sequences, but it is also possible to mine for frequent sequences under different constraints to, for example, remove infrequent intermediate transactions and display the remainder of the sequence that is frequent.

A condition that should be satisfied in order to determine sequence similarity is that it is possible to quantify event similarity, since costs have to be supplied to our metric. This could be hard to achieve and trial-and-error-based. Although event groups makes this task easier, it can still be a challenge to convert the knowledge of a domain expert to good costs.

The visual interface we created has a couple of interactive capabilities. More options could be added to better serve as a data exploration tool. Currently, there is a preprocessing phase and a visualization phase. During preprocessing users can define an alignment point and thresholds for path and edge frequency. The visualization takes place after this data reduction. A single interface to interact with the data would be more user-friendly. It could also be extended to supply some information about the sequences (e.g., sex ratios, distribution of hospitals). It is also possible to add more capabilities for interacting with the diagram, such as removing certain events and setting certain constraints.

Although we can specify insertions, deletions and (intragroup and intergroup) substitution costs to our metric, they only take events into account, not the whole sequence. An example where the sequence is taken into account to determine the cost is to link the cost of an edit operation to the presence or absence of the event (or event group) in the sequence. In case an event or event group is already present in a sequence, inserting more of these events might have a low impact on the similarity with other sequences. This could be reflected by, for example, assigning a lower insertion cost in case an event is already present. This would give more weight to the occurrence of events and less weight to the length of a sequence.

We demonstrated our methods on real life datasets. However, it is hard to determine in an objective way how well our similarity measure performed. We performed unsupervised clustering together with feedback from a domain expert, but the interpretation of results is subjective. Results based on a labelled sequential dataset could be a valuable addition, but it is hard to obtain such data.

We developed a powerful tool for analyzing and visualizing sequential data and evaluated it on healthcare related data. We hope that in the future this tool will be successfully applied to more interesting data and will help to discover new insights in data.

Bibliography

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. "Mining association rules between sets of items in large databases". In: *ACM SIGMOD Record*. Vol. 22. 2. ACM. 1993, pp. 207–216.

[2] Rakesh Agrawal and Ramakrishnan Srikant. "Mining sequential patterns". In: *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*. IEEE. 1995, pp. 3–14.

[3] Rakesh Agrawal, Ramakrishnan Srikant, et al. "Fast algorithms for mining association rules". In: *Proc. 20th int. conf. very large data bases, VLDB*. Vol. 1215. 1994, pp. 487–499.

[4] Elena Baralis et al. "Analysis of medical pathways by means of frequent closed sequences". In: *Knowledge-Based and Intelligent Information and Engineering Systems*. Springer, 2010, pp. 418–425.

[5] Margaret O Dayhoff and Robert M Schwartz. "A model of evolutionary change in proteins". In: *In Atlas of protein sequence and structure*. Citeseer. 1978.

[6] Lee R Dice. "Measures of the amount of ecologic association between species". In: *Ecology* 26.3 (1945), pp. 297–302.

[7] Maged El-Sayed, Carolina Ruiz, and Elke A Rundensteiner. "FS-Miner: efficient and incremental mining of frequent sequence patterns in web logs". In: *Proceedings of the 6th annual ACM international workshop on Web information and data management*. ACM. 2004, pp. 128–135.

[8] Brian Everitt. *Cambridge dictionary of statistics*. 4th ed. Cambridge University Press, 2010.

[9] Andrew Foss, Weinan Wang, and Osmar R Zaïane. "A non-parametric approach to web log analysis". In: *Proc. of Workshop on Web Mining in First International SIAM Conference on Data Mining*. Citeseer. 2001, pp. 41–50.

[10] Minos N Garofalakis, Rajeev Rastogi, and Kyuseok Shim. "SPIRIT: Sequential pattern mining with regular expression constraints". In: *VLDB*. Vol. 99. 1999, pp. 7–10.

[11] John C Gower. "Some distance properties of latent root and vector methods used in multivariate analysis". In: *Biometrika* 53.3-4 (1966), pp. 325–338.

[12] Saikat Guha, Rajeev Rastogi, and Kyuseok Shim. "ROCK: A robust clustering algorithm for categorical attributes". In: *Data Engineering, 1999. Proceedings., 15th International Conference on*. IEEE. 1999, pp. 512–521.

[13] Jiawei Han, Jian Pei, and Yiwen Yin. "Mining frequent patterns without candidate generation". In: *ACM SIGMOD Record*. Vol. 29. 2. ACM. 2000, pp. 1–12.

[14] Jiawei Han et al. "FreeSpan: frequent pattern-projected sequential pattern mining". In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2000, pp. 355–359.

[15] Steven Henikoff and Jorja G Henikoff. "Amino acid substitution matrices from protein blocks". In: *Proceedings of the National Academy of Sciences* 89.22 (1992), pp. 10915–10919.

[16] Daniel S. Hirschberg. "A linear space algorithm for computing maximal common subsequences". In: *Communications of the ACM* 18.6 (1975), pp. 341–343.

[17] Zhengxing Huang, Xudong Lu, and Huilong Duan. "On mining clinical pathway patterns from medical behaviors". In: *Artificial intelligence in medicine* 56.1 (2012), pp. 35–50.

[18] Paul Jaccard. *Distribution de la Flore Alpine: dans le Bassin des dranses et dans quelques régions voisines*. Rouge, 1901.

[19] George Karypis, Eui-Hong Han, and Vipin Kumar. "Chameleon: Hierarchical clustering using dynamic modeling". In: *Computer* 32.8 (1999), pp. 68–75.

[20] Leonard Kaufman and Peter Rousseeuw. *Clustering by means of medoids*. North-Holland, 1987.

[21] Denis Klimov, Yuval Shahar, and Meirav Taieb-Maimon. "Intelligent visualization and exploration of time-oriented data of multiple patients". In: *Artificial intelligence in medicine* 49.1 (2010), pp. 11–31.

[22] Hye-Chung Kum et al. "ApproxMAP: Approximate Mining of Consensus Sequential Patterns." In: *SDM*. SIAM. 2003, pp. 311–315.

[23] Geetika T Lakshmanan, Szabolcs Rozsnyai, and Fei Wang. "Investigating clinical care pathways correlated with outcomes". In: *Business process management*. Springer, 2013, pp. 323–338.

[24] Godfrey N Lance and William Thomas Williams. "A general theory of classificatory sorting strategies II. Clustering systems". In: *The computer journal* 10.3 (1967), pp. 271–277.

[25] Vladimir I Levenshtein. "Binary codes capable of correcting deletions, insertions and reversals". In: *Soviet physics doklady*. Vol. 10. 1966, pp. 707–710.

[26] Nizar R Mabroukeh and Christie I Ezeife. "A taxonomy of sequential pattern mining algorithms". In: *ACM Computing Surveys (CSUR)* 43.1 (2010), p. 3.

[27] James MacQueen et al. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297.

[28] Heikki Mannila and Pirjo Ronkainen. "Similarity of event sequences". In: *time*. IEEE. 1997, p. 136.

[29] RS Mans et al. "Application of process mining in healthcare–a case study in a dutch hospital". In: *Biomedical Engineering Systems and Technologies*. Springer, 2008, pp. 425–438.

[30] James H Martin and Daniel Jurafsky. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Pearson, 2000, pp. 107–111.

[31] Megan Monroe et al. "Exploring point and interval event patterns: Display methods and interactive visual query". In: *Human Computer Interaction Lab, University of Maryland* (2012).

[32] Megan Monroe et al. "Temporal event sequence simplification". In: *Visualization and Computer Graphics, IEEE Transactions on* 19.12 (2013), pp. 2227–2236.

[33] G Niklas Norén et al. "Temporal pattern discovery in longitudinal electronic patient records". In: *Data Mining and Knowledge Discovery* 20.3 (2010), pp. 361–387.

[34] Miho Ohsaki et al. "A rule discovery support system for sequential medical data, in the case study of a chronic hepatitis dataset". In: *Workshop Notes of the International Workshop on Active Mining, at IEEE International Conference on Data Mining*. 2002.

[35] Teruo Okuda, Eiichi Tanaka, and Tamotsu Kasai. "A Method for the Correction of Garbled Words Based on the Levenshtein Metric". In: *IEEE Transactions on Computers* 25.2 (1976), pp. 172–178.

[36] Srinivasan Parthasarathy et al. "Incremental and interactive sequence mining". In: *Proceedings of the eighth international conference on Information and knowledge management*. ACM. 1999, pp. 251–258.

[37] Jian Pei, Jiawei Han, and Wei Wang. "Constraint-based sequential pattern mining: the pattern-growth methods". In: *Journal of Intelligent Information Systems* 28.2 (2007), pp. 133–160.

[38] Jian Pei et al. "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth". In: *icccn*. IEEE. 2001, p. 0215.

[39] Adam Perer and Fei Wang. "Frequence: Interactive mining and visualization of temporal frequent event sequences". In: *Proceedings of the 19th international conference on Intelligent User Interfaces*. ACM. 2014, pp. 153–162.

[40] Adam Perer, Fei Wang, and Jianying Hu. "Mining and exploring care pathways from electronic medical records with visual analytics". In: *Journal of biomedical informatics* 56 (2015), pp. 369–378.

[41] Catherine Plaisant et al. "LifeLines: visualizing personal histories". In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 1996, pp. 221–227.

[42] Jorge CG Ramirez et al. "An event set approach to sequence discovery in medical data." In: *Intell. Data Anal.* 4.6 (2000), pp. 513–530.

[43] Patrick Riehmann, Manfred Hanfler, and Bernd Froehlich. "Interactive sankey diagrams". In: *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*. IEEE. 2005, pp. 233–240.

[44] Rijksoverheid. *Zorginstituut Nederland*. URL: `www.rijksoverheid.nl/contact/contactgids/zorginstituut-nederland` (visited on 06/02/2016).

[45] Peter J Rousseeuw. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis". In: *Journal of computational and applied mathematics* 20 (1987), pp. 53–65.

[46] Yuval Shahar et al. “Distributed, intelligent, interactive visualization and exploration of time-oriented clinical data and their abstractions”. In: *Artificial intelligence in medicine* 38.2 (2006), pp. 115–135.

[47] Shijie Song, Huaping Hu, and Shiyao Jin. “HVSM: a new sequential pattern mining algorithm using bitmap representation”. In: *Advanced data mining and applications*. Springer, 2005, pp. 455–463.

[48] Thorvald Sørensen. “{A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons}”. In: *Biol. Skr.* 5 (1948), pp. 1–34.

[49] Ramakrishnan Srikant and Rakesh Agrawal. *Mining sequential patterns: Generalizations and performance improvements*. Springer, 1996.

[50] Warren S Torgerson. “Multidimensional scaling: I. Theory and method”. In: *Psychometrika* 17.4 (1952), pp. 401–419.

[51] Robert A Wagner and Michael J Fischer. “The string-to-string correction problem”. In: *Journal of the ACM (JACM)* 21.1 (1974), pp. 168–173.

[52] Jianyong Wang and Jiawei Han. “BIDE: Efficient mining of frequent closed sequences”. In: *Data Engineering, 2004. Proceedings. 20th International Conference on*. IEEE, pp. 79–90.

[53] Taowei David Wang et al. “Aligning temporal data by sentinel events: discovering patterns in electronic health records”. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 2008, pp. 457–466.

[54] Weinan Wang and Osmar R Zaïane. “Clustering web sessions by sequence alignment”. In: *Database and Expert Systems Applications, 2002. Proceedings. 13th International Workshop on*. IEEE. 2002, pp. 394–398.

[55] Joe H Ward Jr. “Hierarchical grouping to optimize an objective function”. In: *Journal of the American statistical association* 58.301 (1963), pp. 236–244.

[56] David Wishart. “An algorithm for hierarchical classifications”. In: *Biometrics* (1969), pp. 165–170.

[57] Krist Wongsuphasawat and David Gotz. “Exploring flow, factors, and outcomes of temporal event sequences with the outflow visualization”. In: *Visualization and Computer Graphics, IEEE Transactions on* 18.12 (2012), pp. 2659–2668.

[58] Krist Wongsuphasawat et al. “LifeFlow: visualizing an overview of event sequences”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2011, pp. 1747–1756.

[59] Zhenglu Yang, Yitong Wang, and Masaru Kitsuregawa. “LAPIN: effective sequential pattern mining algorithms by last position induction for dense databases”. In: *Advances in Databases: Concepts, Systems and Applications*. Springer, 2007, pp. 1020–1023.

[60] Mohammed J Zaki. “SPADE: An efficient algorithm for mining frequent sequences”. In: *Machine learning* 42.1-2 (2001), pp. 31–60.

[61] Nederlandse Zorgautoriteit. *Dbc-systematiek*. 2016. URL: `https://www.nza.nl/zorgonderwerpen/zorgonderwerpen/ziekenhuiszorg/veelgesteldevragen/dbc-systematiek/` (visited on 04/14/2016).

Appendix A

Sankey diagrams

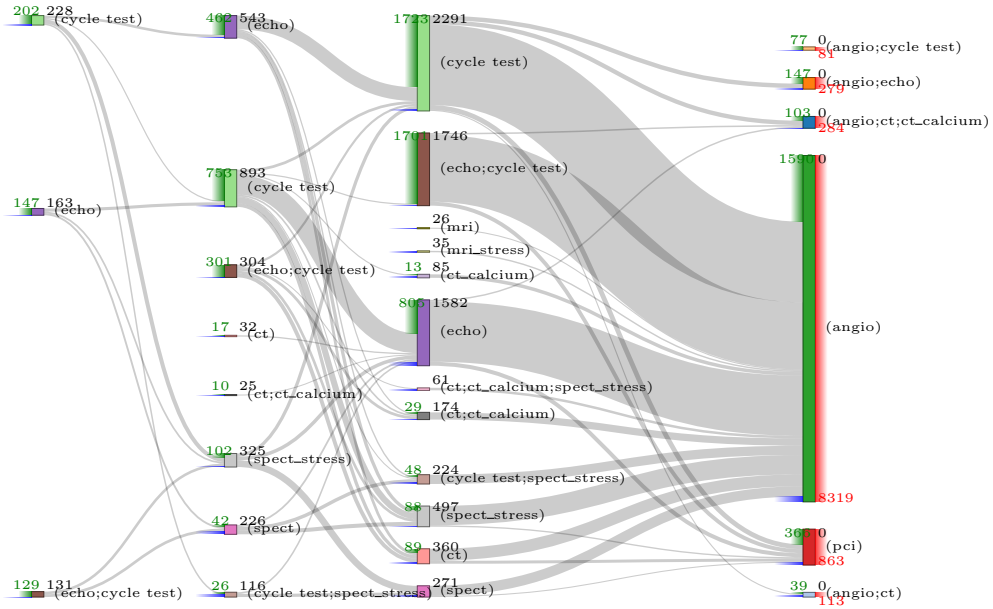


Figure A.1: The entire dataset of Experiment 1.

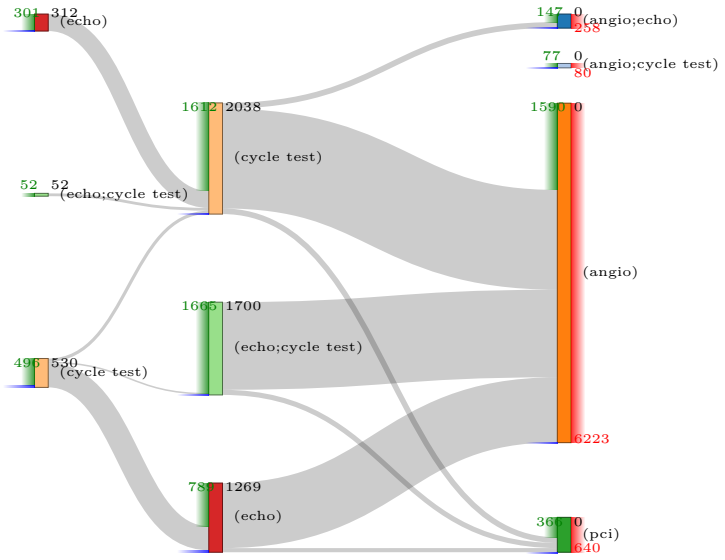


Figure A.2: Cluster 1 of Experiment 1.

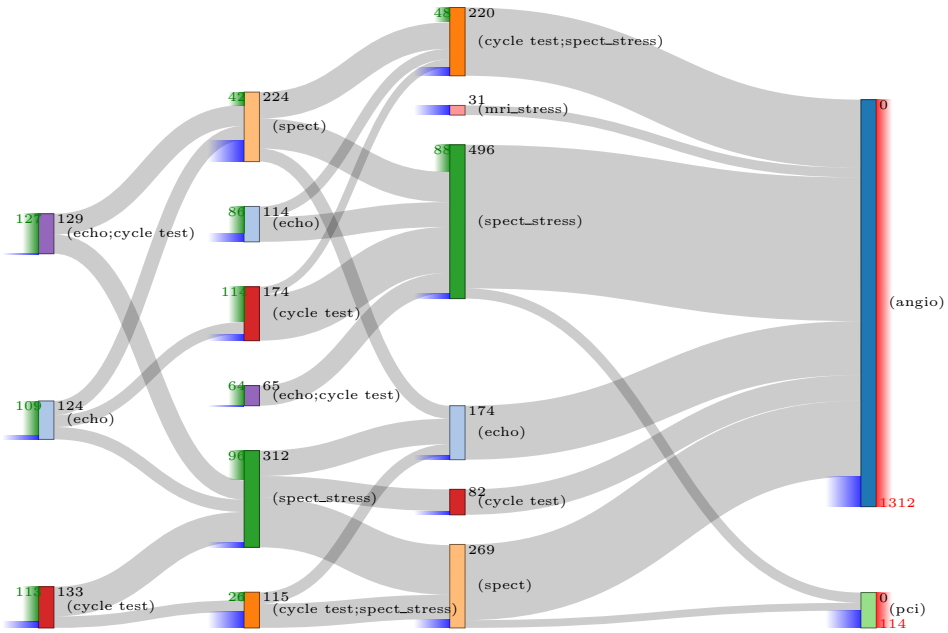


Figure A.3: Cluster 2 of Experiment 1.

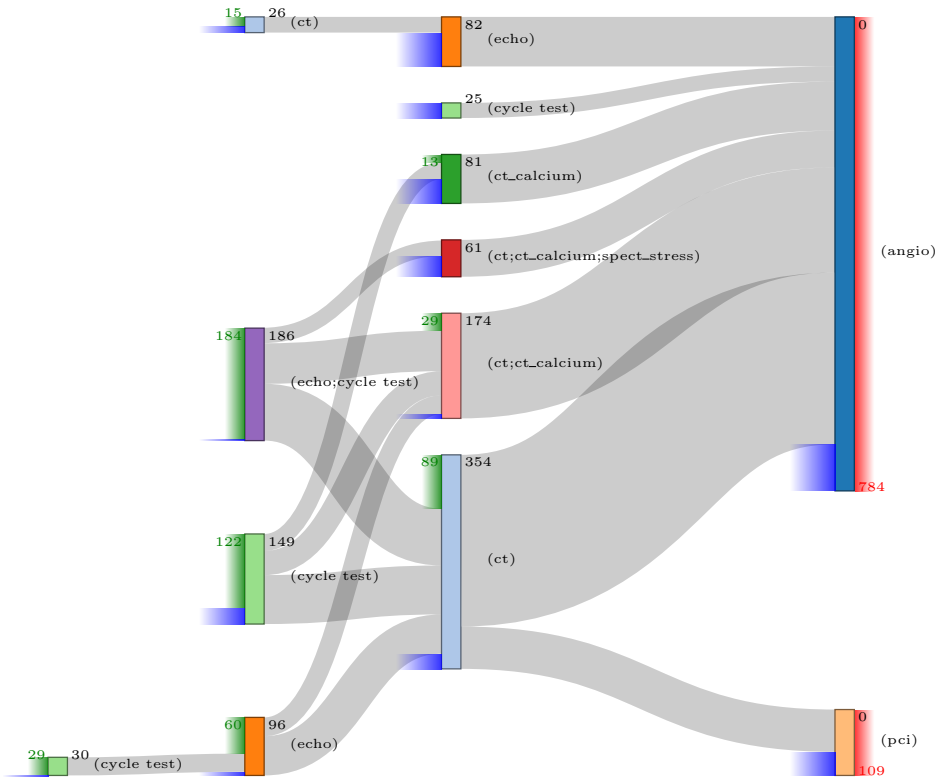


Figure A.4: Cluster 3 of Experiment 1.

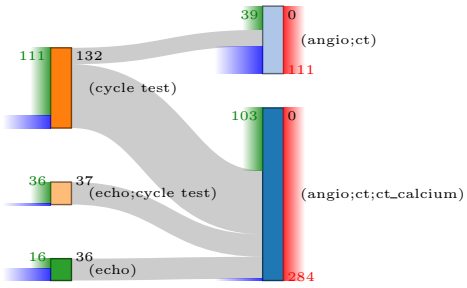


Figure A.5: Cluster 4 of Experiment 1.

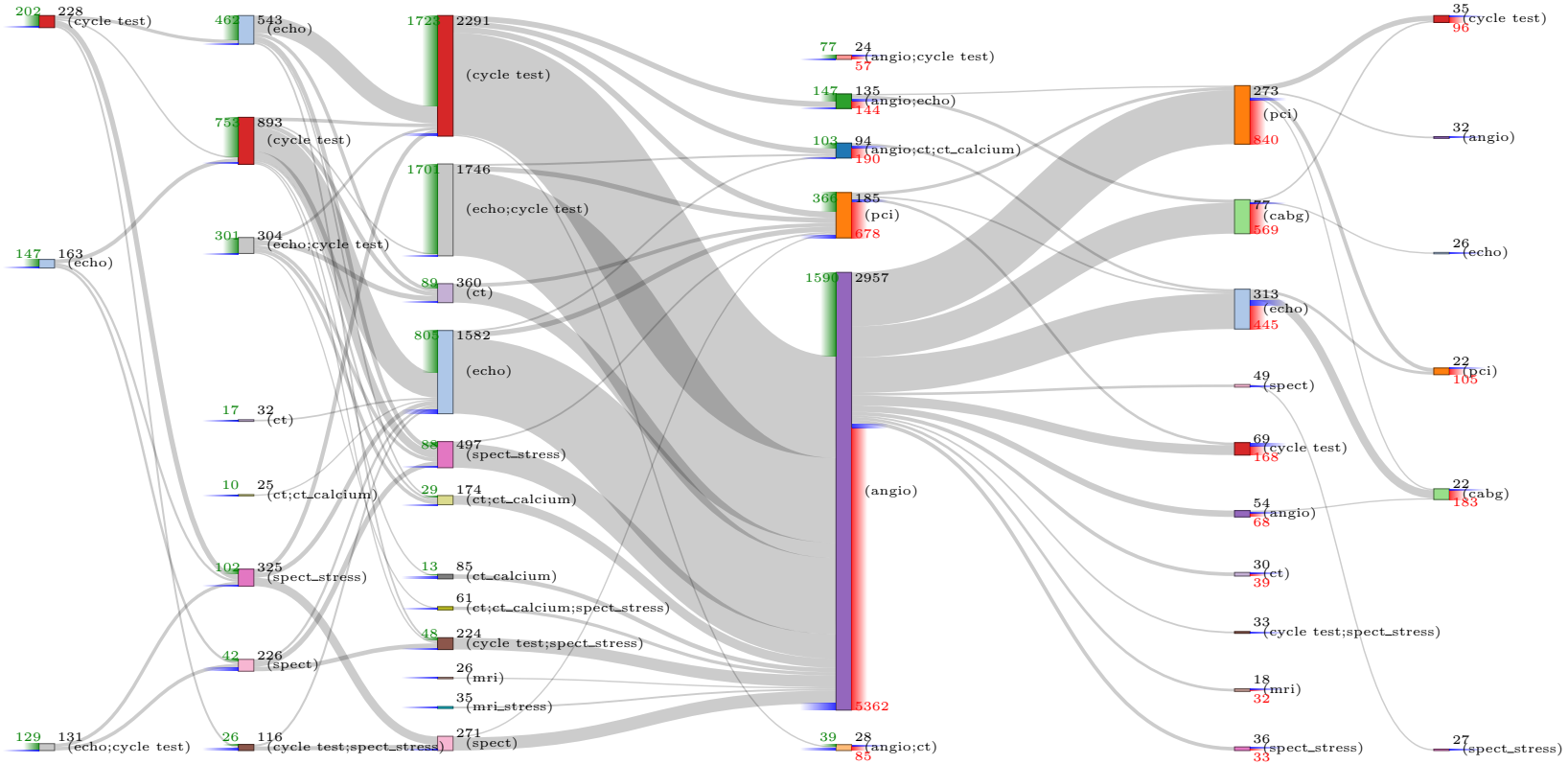


Figure A.6: The entire dataset of Experiment 2.

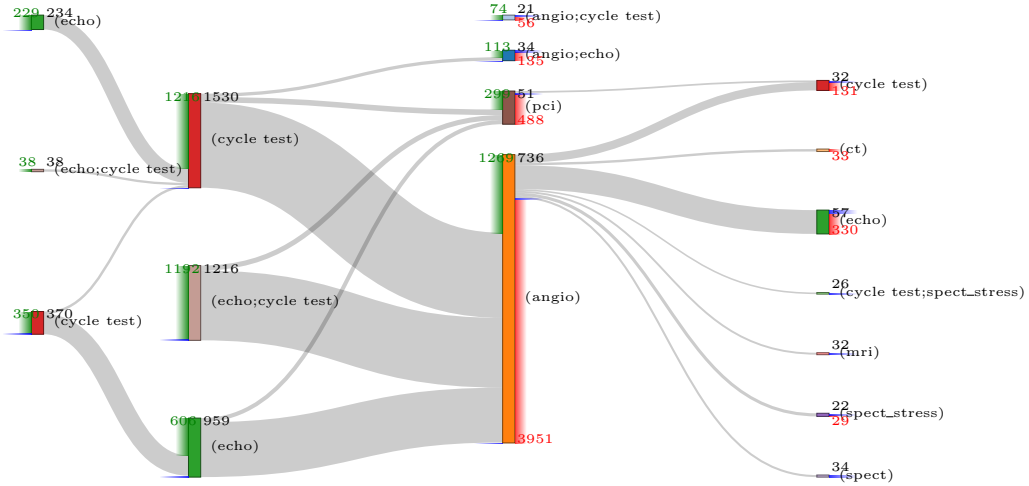


Figure A.7: Cluster 1 of Experiment 2.

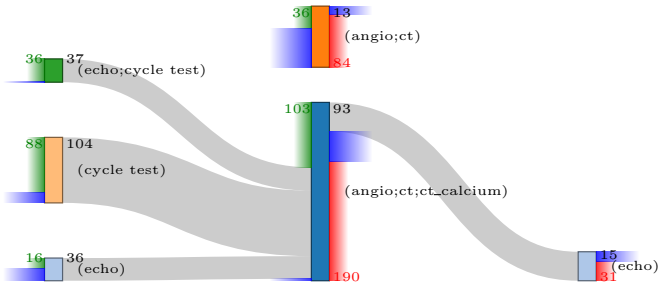


Figure A.8: Cluster 2 of Experiment 2.

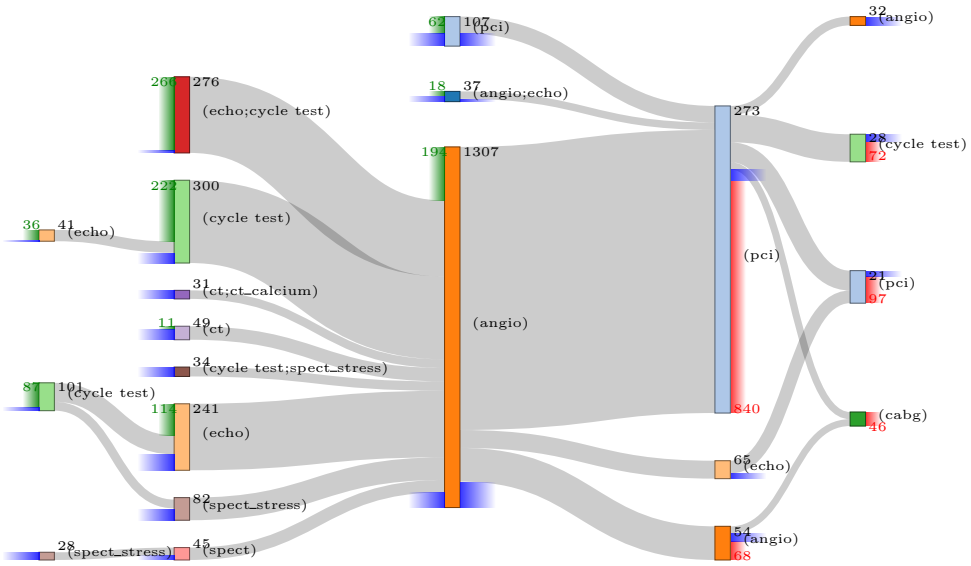


Figure A.9: Cluster 3 of Experiment 2.

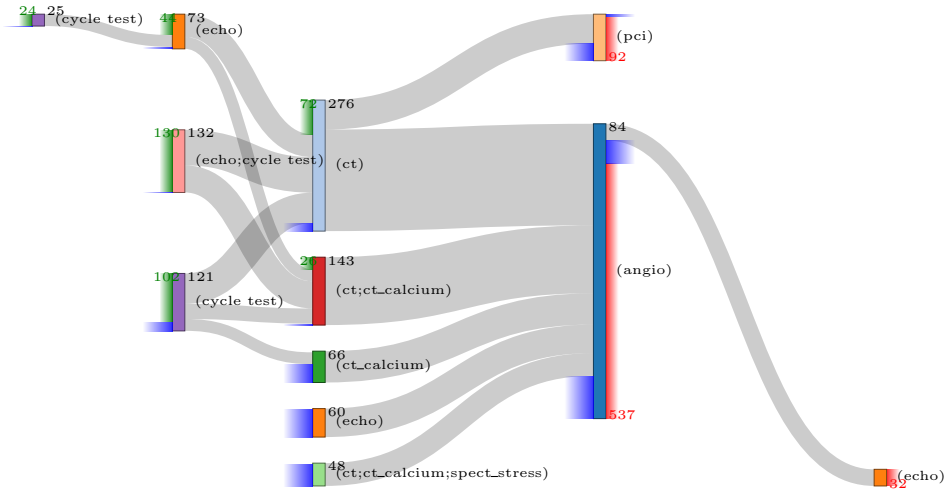


Figure A.10: Cluster 4 of Experiment 2.

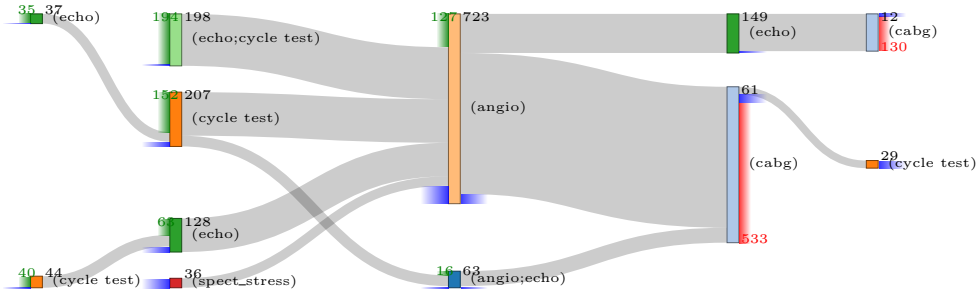


Figure A.11: Cluster 5 of Experiment 2.

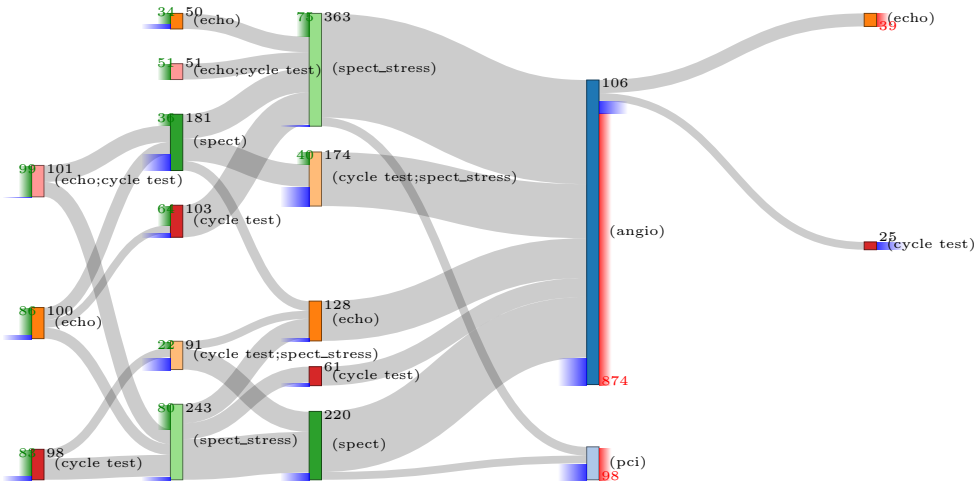


Figure A.12: Cluster 6 of Experiment 2.