Customer journey identification through temporal patterns and Markov clustering

Cordewener, M.H.H.

Award date:
2016

Link to publication
Customer journey identification through temporal patterns and Markov clustering

Master’s Thesis

M.H.H. Cordewener

Supervisors:
- dr. ir. J.C.A.M. Buijs
- ir. B.F.A. Hompes
- dr. ir. P.M.E. Van Gorp
- T. van der Steen MSc.

Eindhoven, July 2016
Abstract

Companies want insights in the behavior of their customers across multiple contact channels. Current techniques for identifying customer journeys are based on interviews and manually creating the model. Due to this limitation, not all journeys can be analyzed. Customer Relationship Management Systems store all kinds of logs when a customer contacts a company. As this data is too large to analyze by hand, a technique based on data is needed. Process discovery algorithms obtain process models from event logs. Customers can have multiple journeys in parallel. As a result, applying process discovery algorithms directly on such a set, incorrect insights are obtained. Hence, a tool is needed that is able to discover customer journeys from a dataset. However, such a technique does not exist yet.

In this thesis, a new technique is proposed for identifying customer journeys from a large dataset. The input for such a technique is an event log in which each customer is represented as a single trace, i.e. each trace contains all events of a single customer. In this thesis, an event is a moment on which the customer contacted the company. With the output of the technique, an analyst should be able to obtain and analyze customer journeys.

To show the practical application of the developed technique, it is implemented as a plug-in for ProM. Using this ProM plug-in, a case study has been conducted using data from CZ, a health insurance company in the Netherlands. With this case study, it is shown that the technique is able to distinguish the different customer journeys for each customer.

Keywords: customer journey analyses, process mining
Preface

This thesis is the result of my master graduation project and concludes my Business Information Systems study at Eindhoven University of Technology (TU/e). The research has been done at Underlined, the Architecture of Information Systems (AIS) group and in collaboration with CZ.

I would like to thank my supervisors Bart Hompes and Joos Buijs for this opportunity and their guidance during this project. I would also like to thank Theo van der Steen, Henrik Nijkamp and Gerdien Ridderbos from Underlined for guidance and valuable feedback. From CZ, I would like to thank Chantine Huigevoort and Wouter Wester for providing this opportunity.

Many thanks go to my friends for their continues support. Especially Tom, Koen and Erik for their support during my study. Last but not least, I would like to thank my family for their continuous support during my study.

Marco Cordewener
Eindhoven
2016
Contents

Contents vii
List of Figures xi
List of Tables xiii
List of Algorithms xv
List of Listings xvii
List of Terms xix

1 Introduction 1
  1.1 Project context .................................. 1
  1.2 Problem description .............................. 1
  1.3 Project goal ................................... 3
  1.4 Running example ................................ 3
  1.5 Project scope ................................ 5
  1.6 Outline ...................................... 6

2 Related work 7
  2.1 Customer journey analysis ......................... 7
     2.1.1 Service design ................................ 7
     2.1.2 Service analysis .............................. 8
     2.1.3 Online advertising ........................... 10
  2.2 Process mining ................................ 10
     2.2.1 Trace clustering ............................. 12
     2.2.2 Abstractions ................................. 15
     2.2.3 Mining process models when case identifiers are missing ................................... 16

3 Process mining on customer journeys 17
  3.1 CRM log files ................................ 17
     3.1.1 XES log ..................................... 17
     3.1.2 Log details .................................. 18
  3.2 Evaluation .................................... 19
     3.2.1 Trace clustering ............................. 19
     3.2.2 Abstraction techniques ....................... 21
  3.3 Identified problems ............................. 23
     3.3.1 Performance ................................. 23
     3.3.2 Diversity in customer behavior .................. 23

Customer journey identification through temporal patterns and Markov clustering vii
<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1.4 Rename excel columns</td>
</tr>
<tr>
<td>A.1.5 Create mijnCZ set</td>
</tr>
<tr>
<td>A.1.6 Create total set</td>
</tr>
<tr>
<td>A.1.7 Add missing data</td>
</tr>
<tr>
<td>A.1.8 Export</td>
</tr>
<tr>
<td>A.1.9 Subsets</td>
</tr>
<tr>
<td>A.2 ProM CZ Importer</td>
</tr>
</tbody>
</table>

**B Figures**

<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1 Figures in chapter 3</td>
</tr>
<tr>
<td>B.2 Figures in chapter 5</td>
</tr>
<tr>
<td>B.3 Figures in chapter 6</td>
</tr>
</tbody>
</table>

**C Event similarity matrices**

<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>C Event similarity matrices</td>
</tr>
</tbody>
</table>

**D Similarity matrix**

<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>D Similarity matrix</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Example of a Sanky Diagram</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Available data from MyCarService</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>Behavior of Bob</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>Process model of Bob</td>
<td>5</td>
</tr>
<tr>
<td>1.5</td>
<td>Scope of the project</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>Example of a Brand Touchpoint Wheel</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Example of a Customer Journey Map</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>Process mining in the real world</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>Example of a spaghetti model</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>Trace length Distribution</td>
<td>18</td>
</tr>
<tr>
<td>3.2</td>
<td>Clustering results from Trace Clustering plug-in</td>
<td>19</td>
</tr>
<tr>
<td>3.3</td>
<td>Episode miner results</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>Solution to the journey identification problem</td>
<td>25</td>
</tr>
<tr>
<td>4.2</td>
<td>Profiling a customer</td>
<td>26</td>
</tr>
<tr>
<td>4.3</td>
<td>Creating an event similarity matrix</td>
<td>27</td>
</tr>
<tr>
<td>4.4</td>
<td>Interpretation of the modified event similarity matrix</td>
<td>30</td>
</tr>
<tr>
<td>4.5</td>
<td>Profiling Bob</td>
<td>31</td>
</tr>
<tr>
<td>4.6</td>
<td>Creating the journeys of Bob</td>
<td>31</td>
</tr>
<tr>
<td>4.7</td>
<td>Temporal pattern recognition approach</td>
<td>33</td>
</tr>
<tr>
<td>4.8</td>
<td>Journeys of Bob</td>
<td>36</td>
</tr>
<tr>
<td>4.9</td>
<td>Schematic overview of the algorithm</td>
<td>37</td>
</tr>
<tr>
<td>4.10</td>
<td>Overview of the algorithm</td>
<td>38</td>
</tr>
<tr>
<td>5.1</td>
<td>ProM log viewer examples</td>
<td>39</td>
</tr>
<tr>
<td>5.2</td>
<td>Packages used in the implementation</td>
<td>40</td>
</tr>
<tr>
<td>5.3</td>
<td>Illustration of the algorithms package</td>
<td>42</td>
</tr>
<tr>
<td>5.4</td>
<td>Trace Finder Visualizer plug-in</td>
<td>44</td>
</tr>
<tr>
<td>5.5</td>
<td>Illustration of the visualization package</td>
<td>45</td>
</tr>
<tr>
<td>6.1</td>
<td>Pre-processing overview</td>
<td>48</td>
</tr>
<tr>
<td>6.2</td>
<td>Process model for the test customer</td>
<td>49</td>
</tr>
<tr>
<td>6.3</td>
<td>Process model for the journeys of the test customer</td>
<td>51</td>
</tr>
<tr>
<td>6.4</td>
<td>Trace length Distribution</td>
<td>52</td>
</tr>
<tr>
<td>6.5</td>
<td>Dotted chart on dental care event log</td>
<td>53</td>
</tr>
<tr>
<td>6.6</td>
<td>Process models obtained for the periodic behavior of 983 customers</td>
<td>53</td>
</tr>
<tr>
<td>6.7</td>
<td>Obtaining an event log containing only events related to dental care</td>
<td>54</td>
</tr>
<tr>
<td>6.8</td>
<td>Trace length distribution</td>
<td>55</td>
</tr>
<tr>
<td>6.9</td>
<td>Value distribution in the similarity matrix</td>
<td>56</td>
</tr>
<tr>
<td>6.10</td>
<td>Temporal patterns on dental care related events</td>
<td>57</td>
</tr>
<tr>
<td>6.11</td>
<td>Temporal patterns on events with a payment subject</td>
<td>58</td>
</tr>
</tbody>
</table>

*Customer journey identification through temporal patterns and Markov clustering*
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.12</td>
<td>Journeys containing events for compensation on dental care</td>
<td>59</td>
</tr>
<tr>
<td>6.13</td>
<td>Dotted Chart of journeys for compensation on dental care</td>
<td>59</td>
</tr>
<tr>
<td>A.1</td>
<td>The SAS Guide flow for CZ data pre-processing</td>
<td>73</td>
</tr>
<tr>
<td>B.1</td>
<td>Petri net obtained by the Inductive Miner</td>
<td>77</td>
</tr>
<tr>
<td>B.2</td>
<td>Settings dialog: Algorithm selection</td>
<td>78</td>
</tr>
<tr>
<td>B.3</td>
<td>Settings dialog: Settings for identifying temporal patterns</td>
<td>79</td>
</tr>
<tr>
<td>B.4</td>
<td>Settings dialog: Settings for clustering the remaining events</td>
<td>80</td>
</tr>
<tr>
<td>B.5</td>
<td>Additional settings for the TraceFinder plug-in</td>
<td>81</td>
</tr>
<tr>
<td>B.6</td>
<td>Full scale version of Figure 5.4</td>
<td>82</td>
</tr>
<tr>
<td>B.7</td>
<td>Full scale version of Figure 6.5</td>
<td>83</td>
</tr>
<tr>
<td>B.8</td>
<td>The full scale model of Figure 6.2</td>
<td>84</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Behavior of Bob</td>
<td>4</td>
</tr>
<tr>
<td>3.1</td>
<td>Example event log</td>
<td>18</td>
</tr>
<tr>
<td>3.2</td>
<td>Clustering details of Figure 3.2</td>
<td>20</td>
</tr>
<tr>
<td>4.1</td>
<td>Example similarity matrix for the subject perspective</td>
<td>33</td>
</tr>
<tr>
<td>4.2</td>
<td>EventClustering for Bob</td>
<td>33</td>
</tr>
<tr>
<td>4.3</td>
<td>Journeys of Bob</td>
<td>36</td>
</tr>
<tr>
<td>6.1</td>
<td>Journeys obtained for a single customer using both temporal patterns and clustering</td>
<td>50</td>
</tr>
<tr>
<td>6.2</td>
<td>Journeys obtained for a single customer using clustering</td>
<td>50</td>
</tr>
<tr>
<td>6.3</td>
<td>Top 5 subject &amp; (sub-) subject combinations</td>
<td>52</td>
</tr>
<tr>
<td>A.1</td>
<td>Subject matching</td>
<td>74</td>
</tr>
<tr>
<td>D.1</td>
<td>The similarity matrix for the subject perspective</td>
<td>89</td>
</tr>
</tbody>
</table>
## List of Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Journey identification for a single customer</td>
<td>26</td>
</tr>
<tr>
<td>2</td>
<td>Profiling a customer</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>Profiling a single event</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>Creating an event similarity matrix</td>
<td>28</td>
</tr>
<tr>
<td>5</td>
<td>Profile similarity</td>
<td>28</td>
</tr>
<tr>
<td>6</td>
<td>Markov cluster algorithm</td>
<td>29</td>
</tr>
<tr>
<td>7</td>
<td>Interpretation of the modified event similarity matrix</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>Obtaining traces</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>Creating a time distance matrix</td>
<td>34</td>
</tr>
<tr>
<td>10</td>
<td>Temporal pattern mining</td>
<td>35</td>
</tr>
<tr>
<td>11</td>
<td>Merging all results obtained by the algorithms in Section 5.4</td>
<td>43</td>
</tr>
<tr>
<td>12</td>
<td>Adding a cluster to a clustering</td>
<td>44</td>
</tr>
<tr>
<td>13</td>
<td>Converting “TraceFinderResult” to a model usable for visualization</td>
<td>46</td>
</tr>
<tr>
<td>14</td>
<td>Pseudo code for parsing a customer to a trace</td>
<td>76</td>
</tr>
</tbody>
</table>
List of Listings

A.1 Code in the program “Library” ...................................................... 67
A.2 Code in the program “Format data & rename columns” ......................... 67
A.3 Code in the program “Create set without mijnCZ” ............................. 68
A.4 Code in the program “Rename excel columns” .................................. 69
A.5 Code in the program “Create mijnCZ set” ...................................... 69
A.6 Code in the program “Create total set” ........................................ 70
A.7 Code in the program “Add missing data” ..................................... 70
A.8 Code in the program “Select 100 random customers” ......................... 71
A.9 Code in the program “Create subset 100 random customers” ................. 71
A.10 Code in the program “Select Mondzorg customers” .......................... 71
A.11 Code in the program “Create mondzorg subset” ............................ 72
List of Terms

Glossary

activity  A step that can be executed during the process.

case  An entity that is handled in the analyzed process. Each case has some data attributes and a trace.

customer journey  A sequence of events followed by customers to reach a certain goal, whether designed or not.

event  The execution of an activity at a certain moment in time. An event usually has at least three attributes: the activity that is executed, the time of the execution and a case identifier, stating to which case the event belongs.

event log  A multi-set of traces. Similar as with events and traces an event log contains also attributes, such as the name of the log.

pattern  An activity or sequence of activities that occur multiple times with a constant time interval between the occurrences.

perspective  attributes and measurements that can be used for clustering.

process instance  see case.

trace  A sequence of events in which each event occurs at most one time.

Acronyms

CRM  Customer Relationship Management.
Chapter 1

Introduction

Current techniques for analyzing customer journeys require a lot of manual labor or treat all behavior of a customer as a single journey. Hence, there is need for a tool that is able to identify the individual journeys of a customer from event data. Analysts should be able to use such a tool even without programming experience or knowledge. To verify the correctness of the obtained result, and the applicability of the tool, a case study on the customer data of Centraal Ziekenfonds (CZ) will be conducted.

The remainder of this chapter is as follows: Section 1.1 describes the context of the project. Section 1.2 describes the problem in more detail. In Section 1.3 the goal of the project is discussed and Section 1.4 introduces a running example for this thesis. Section 1.5 discusses the scope of the project. Finally, Section 1.6 discusses the outline of this thesis.

1.1 Project context

This thesis is the result of the graduation project for the master Business Information Systems at the Eindhoven University of Technology (TU/e). The research has been done within the Architecture of Information Systems (AIS) group at the TU/e, Underlined and CZ.

The AIS group is part of the Mathematics and Computer Science department of the TU/e. The group researches business processes and process-aware information systems that support those processes and is well known for the research in the field of process mining.

Underlined is located in ’s Hertogenbosch and is specialized in Customer Journey Management. Using a proven approach and tool set, customer experience during the customer journeys can be improved. Based on data from customer contact channels, online environments, customer feedback and research responds, the actual customer journeys and customer experience can be discovered, enabling measurement of the customers’ behavior and emotions.

CZ is a health insurance company in the Netherlands. For this thesis, CZ provided a dataset on which the case study (Chapter 6) is conducted. This dataset contains all incoming contacts on the three channels that can be used to contact CZ: e-mail, call center, and visiting one of the service desks throughout the country. Furthermore, the customer behavior in the online environment, called “Mijn CZ” is also added to this set. It covers the online portal in which the customers are signed in. Hence, it is possible to relate the customers in the set containing the online behavior to the customers in the set containing the other contact channels. In Chapter 6, this resulting dataset is discussed in more detail.

1.2 Problem description

Customers can use different communication channels to contact a company: from service desks and telephone calls to email and online environments. Because of this variability, companies often do not have many insights in the overall behavior of their customers. This makes optimization of
CHAPTER 1. INTRODUCTION

service levels increasingly difficult. Currently, customer journeys are analyzed through conducting in-depth interviews with customers and manually creating models that capture customer journeys. However, because persons are unique, every person has its own way of contacting a company. As a result, when interviewing only a few people, one will most likely capture only a small part of all the possible behavior. If one wants to capture the behavior of all customers, or at least the most common behavior, a data-based technique is needed. Analyzing such large datasets by hand is impossible. Hence, it is not possible to use traditional techniques for customer journey analysis to analyze such big data.

Typically, the different systems from a company keep logs of all contacts that a customer had with the company. There are already several tools that enable a statistical analysis of such datasets. Some of these tools even try to capture the behavior of customers. An example of such a tool is SAS Visual Analytics which uses a Sanky Diagram to visualize customer behavior. An example of a Sanky Diagram can be found in Figure 1.1. The major disadvantage of such a diagram is that it shows for which subject a customer will contact the company again, even though it might be the case that two following subjects have totally no relation with each other. Another disadvantage of Sanky Diagrams is that it is not possible to view other data-attributes which may influence the customer journeys. Since current customer journey analysis techniques require manual labor and Sanky diagrams fail to capture the actual customer journeys, a new technique is needed.

Process mining techniques are able to mine a process model from an event log containing events. Currently available process mining techniques require an event log containing nicely defined traces. A trace is a sequence of events. In this thesis, a trace contains all events of a single customer journey of a single customer. However, logs from communication systems do not contain traces. Using the customer relation number, all events of a customer can be combined into a single trace representing all the behavior of the customer. Such an event log can be used by existing process mining techniques. A customer can have multiple journeys in parallel. Existing process mining techniques do not use this fact. This might lead to incorrect results as non-related events are considered to be related. Hence, results obtained with existing process mining techniques are not useful for an analyst. As a result, the actual customer journeys are needed. Thus a technique is needed that is able to convert an event log containing all the moments a customer had contact with

Figure 1.1: Example of a Sanky Diagram. It shows for which subjects a customers contacted a company and for what the subject of the next contact is. The customers that did not contact the company anymore are shown as a path that exit a node, but are not connected to a next node. For example: of the customers that contacted the company for some compensation, 2,632 contacted the company the next time for another compensation, while 3,336 customers did not contact the company anymore.
a company, called events, to an event log that can be used by existing process mining techniques. Such a technique would need to be able to detect the different customer journeys of a single customer. At this moment, such a technique is not available.

**Problem statement:** Current process mining techniques are not suitable for modeling and analyzing customer journeys as system logs contain all events for any given customer and do not distinguish the different customer journeys.

1.3 Project goal

As stated in the previous section, there is need for a technique that can identify the different customer journeys. The process mining framework (ProM)\(^1\) can handle event logs, so implementing the technique as a ProM plug-in has the advantage that basic functionality is already implemented and can be reused. The resulting ProM plug-in will be used to analyze a dataset provided by CZ. This will show how the plug-in performs as well as the practical application of the technique.

The input of the technique is a dataset containing events for each moment a customer had contact with a company. The ProM plug-in will require an event log as input in which each trace contains all events of a single customer. There are already some tools that enable easy conversion from a simple CSV file or database to an event log. Most of the import functionality is already covered by the import functionality of ProM. The plug-in should also be built in such a way that the user only needs to choose which variant of the algorithm should be used and set some parameters for the selected algorithm. This ensures that users do not need to have any knowledge of programming or any algorithms that will be used by the plug-in.

**Project goal:** Develop and implement a technique that is able to discover customer journeys in an event log. Each trace in this log contains all events of a single customer. The resulting event log should contain a trace for each identified customer journey.

1.4 Running example

To give a better understanding of the problem and the developed solution, a running example is used. For this running example, a fictive company called *MyCarService* is used. *MyCarService* provides a service of repairing cars, for which customers need to pay a monthly subscription fee. An online portal is available to the customers. In this portal, customers can do almost everything that is needed. There are also other options for customers to contact *MyCarService*: calling, e-mail and visiting one of the service desks throughout the country.

*MyCarService* tracks the online behavior of the customers. The customer service department of *MyCarService* also tracks the behavior of the customers that call, e-mail or visit a service desk. For analysis purposes, the logs containing the online behavior and the logs from the customer service department are combined. Furthermore, *MyCarService* enriched the log with extra customer data, such as age and gender. Figure 1.2 shows a visual representation of all the available data.

*MyCarService* wants insights in the customer journeys in order to optimize the customer journey in terms of quality and time. As stated earlier, *MyCarService* does have all the data of their customers, but does not have an idea what the actual customer journeys are.

Bob is a customer of *MyCarService* and has contacted the company several times this year. The data that is available for customer Bob is shown in Table 1.1 and Figure 1.3.

*MyCarService* can create a process model using existing process mining technique on all the contact moments given in Table 1.1. Using the “Visual Inductive Miner” plug-in in ProM, a process model is obtained for the events in Table 1.1. Figure 1.4 shows this model. Clearly, this model relates activities that should not be related. For example, the process for changing the subscription is always followed by a car repair event.

\(^1\)See http://promtools.org

Customer journey identification through temporal patterns and Markov clustering
CHAPTER 1. INTRODUCTION

Table 1.1: The behavior of Bob. A detailed overview of the events by Bob. The behavior is also visualized in Figure 1.3.

<table>
<thead>
<tr>
<th>Event number</th>
<th>Channel</th>
<th>Date</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Online</td>
<td>30-03-2016</td>
<td>Payments</td>
</tr>
<tr>
<td>2</td>
<td>E-mail</td>
<td>26-04-2016</td>
<td>Available subscriptions</td>
</tr>
<tr>
<td>3</td>
<td>E-mail</td>
<td>27-04-2016</td>
<td>Change subscription</td>
</tr>
<tr>
<td>4</td>
<td>Telephone</td>
<td>28-04-2016</td>
<td>Car repair</td>
</tr>
<tr>
<td>5</td>
<td>Service desk</td>
<td>29-04-2016</td>
<td>Car repair</td>
</tr>
<tr>
<td>6</td>
<td>Online</td>
<td>30-04-2016</td>
<td>Payments</td>
</tr>
<tr>
<td>7</td>
<td>Telephone</td>
<td>28-05-2016</td>
<td>Car repair</td>
</tr>
<tr>
<td>8</td>
<td>Service desk</td>
<td>29-05-2016</td>
<td>Car repair</td>
</tr>
<tr>
<td>9</td>
<td>Online</td>
<td>30-05-2016</td>
<td>Payments</td>
</tr>
<tr>
<td>10</td>
<td>E-mail</td>
<td>27-06-2016</td>
<td>Damages abroad</td>
</tr>
<tr>
<td>11</td>
<td>Telephone</td>
<td>28-06-2016</td>
<td>Car repair</td>
</tr>
<tr>
<td>12</td>
<td>Service desk</td>
<td>29-06-2016</td>
<td>Car repair</td>
</tr>
<tr>
<td>13</td>
<td>Online</td>
<td>30-06-2016</td>
<td>Payments</td>
</tr>
<tr>
<td>14</td>
<td>Online</td>
<td>30-07-2016</td>
<td>Payments</td>
</tr>
<tr>
<td>15</td>
<td>Online</td>
<td>30-07-2016</td>
<td>Payments</td>
</tr>
</tbody>
</table>

When looking at customer journeys, four journeys can be identified from Table 1.1. The first journey captures the monthly subscription fees, the second journey is a short journey in which Bob has the goal to change his subscription. The third journey is for repairing the car. The last journey is retrieving some information on damages in a foreign country.

Figure 1.2: Available data from MyCarService. First, event data from all different channels are combined into a single dataset. This set is enriched using customer data, such as the gender and age of the customers.

Figure 1.3: Behavior of Bob. Visual representation of the events shown in Table 1.1. The world icon represents an event in the online environment, the envelope represents an incoming email, the phone represents an incoming phone call and the person icon represents a visit to the service desk.

Customer journey identification through temporal patterns and Markov clustering
CHAPTER 1. INTRODUCTION

Payments Available subscriptions Damages abroad
Change subscription Car repair

Figure 1.4: Process model of Bob. Using a process discovery algorithm directly on Table 1.1 results into a model in which some activities are related to each-other while they should not be related. For example, the sub-process for changing the subscription is always followed by a car repair event.

1.5 Project scope

As stated in the previous section, the result of this project will be a technique that is able to discover customer journeys from an event log. This technique is part of a larger framework for analyzing customer journeys. Figure 1.5 shows the scope of this project.

The process in the framework starts with collecting raw data from information systems. This data is converted to an event log in which each trace represents all events of a single customer. The next step is to apply our ProM plug-in to convert this log to an event log in which each trace represents a single customer journey. The result of this plug-in can further be used in ProM for analysis but also in other tools such as SAS\(^2\) or Tableau\(^3\).

---

\(^2\)See http://www.sas.com/
\(^3\)See http://www.tableau.com/

Figure 1.5: Scope of the project. First, some pre-processing is needed to obtain a set of customer contact data. Then, the customer journeys are discovered, enabling further analysis in tools such as ProM, SAS or Tableau. Marked in red, is the scope of this thesis.
1.6 Outline

The outline of the thesis reflects the steps that were taken during this graduation project. Chapter 2 discusses related work. Both work that is done in the area of customer journey analysis (Section 2.1) and in the area of process mining (Section 2.2) will be discussed. Chapter 3 shows how process mining techniques can be used for analyzing customer journeys and which problems can occur when applying existing process mining techniques for customer journeys analysis. Chapter 4 discusses the solution to solve the problems identified in Chapter 3. The implementation of our solution in ProM is discussed in Chapter 5. A case study is conducted to show how the technique performs. Also the practical application of the developed technique is shown during the case study. This is discussed in Chapter 6. Chapter 7 concludes the thesis, discusses limitations and outlines ideas for future work.
Chapter 2

Related work

This chapter discusses all the pre-existing work that is related to this thesis. The fields of customer journey analysis and process mining are combined. Hence, this section discusses related work from both fields. First a short introduction to each field is given, followed by a discussion of related work. Section 2.1 discusses the field of customer journey analysis. Section 2.2 discusses the field of process mining.

2.1 Customer journey analysis

This section gives a short introduction to the concept of customer journeys. A customer journey is defined by Norton et al. [37] as:

A customer journey is the sequence of events, whether designed or not, that customers go through to learn about, purchase and interact with company offerings, including commodities, goods, services or experiences.

The concept of customer journeys is currently used in multiple fields and for different kinds of analysis. In the field of web-science for example, customer journeys are used to analyze the effect and performance of advertisements [5,6,38,50,51]. Another field where customer journeys are used is the field of designing and analyzing services. When designing services, the customer journeys are determined such that the service will resolve around the customer and will give the best possible customer experience. By analyzing the customer journey for a service, the strong and weak points of the service, seen from the perspective of the customer, can be found. This enables improvement of services and therefore improvement in the customer experience.

Due to the differences of the fields where customer journey analysis are applied and the different needs, various techniques have been proposed for the analysis of customer journeys. Some techniques require in-depth interviews and manual labor, while others depend on big-data and automated algorithms to achieve good results. This section discusses the various techniques that have been identified for analyzing customer journeys.

The remainder of this section is organized as follows: Section 2.1.1 discusses the customer journey analysis methods that are used in the area of service design. Section 2.1.2 discusses the methods that are used in the area of service analysis and Section 2.1.3 discusses the methods used in the area of online advertising.

2.1.1 Service design

In service design, customer journeys are analyzed during the design of services. Steen et al. define service design as the process of planning and organizing people, infrastructure, communication and material components of a service [44]. By considering customer journeys during the design phase, services can be built to optimize customer experience. Steen et al. show the benefits
of involving customers when designing a service [44]. The identified benefits can be split into three categories: benefits for the service design project, benefits for the service’s customers or users and benefits for the organization(s) that are involved. Zomerdijk and Voss performed a case study on several experience-centric firms [53]. Six propositions were developed and tested to capture design principles for experience-centric services. Analyzing these propositions showed that the propositions regarding customer journeys are most supported. This shows that analyzing customer journeys is important when developing experience-centric services.

Bucolo and Matthews proposed a conceptual framework to assist companies to grow through a new design [12]. The goal of the framework is to gain a better understanding of customers through building deep insights across market segments. The framework consists of four stages: the observational stage, the re-framing stage, the narrative stage and the proposition stage. In the observational stage, the firm commits to engage with its customers early in the design process. Tools such as a Customer Journey Map [32] can be used in this stage. In the reframing stage, the firm selects an observation made in the previous step and translates this into meaning, i.e. ask “why did that observation occur?”. In the narrative stage, the firm communicates the meaning obtained in the reframing stage to the customers. The final narrative will be in the form of an opportunity, that is, it embeds the identified observations and meanings. The last phase is the propositions stage. In this stage, the firm links, with senior management, the proposition to the company strategy and checks whether the company strategy should be changed.

Service design methods have the advantage that they help companies in designing a new service in a way that optimizes the customer experience. However, they require interaction with customers and manual labor. Hence, these techniques can only be applied in a setting where the number of customers is small enough to analyze by hand. Another disadvantage of manual labor is that the result depends on the person executing the method: one person might give different results than another. The dataset used in this thesis is too large to analyze by hand. Customers do not follow a fixed process when contacting a company, as a result a small sample might not be representative. Hence, this technique is not suitable for identifying customer journeys in this dataset.

2.1.2 Service analysis

Techniques for analysing customer journeys are often used for analyzing services [13,27,32]. Several methods have been developed to analyze customer experience during a customer journey of a service. The most common methods are the Brand Touchpoint Wheel by Davis et al. [13] and the Customer Journey Map by Mangiaracina et al. [32]. The Brand Touchpoint Wheel divides the customer journey into a fixed amount of phases and identifies the touchpoints for each phase. The touchpoints of a journey are placed on the wheel in the same order as the customer journey. In this way, the Brand Touchpoint Wheel shows which touchpoints influence each other. An example of such a Brand Touchpoint Wheel is shown in Figure 2.1. The major disadvantage of this work is that it assumes that the customer journey always consists of the same number of (sequential)

Figure 2.1: Example of a Brand Touchpoint Wheel [13]. For each of the three phases, the touchpoints influencing that phase are shown. This technique is limited to a predefined set of non-optional, sequential steps that all have to occur.
steps.

The Customer Journey Map [32] is used to model a customer journey during the service. Figure 2.2 shows an example of a Customer Journey Map. First, the touchpoints are divided over a fixed number of phases. For each phase, the drivers are identified. Figure 2.2a shows an example of the five phases for a webshop and the touchpoints for each phase. Based on the quality of the drivers and user experience due to that driver, a quality score can be calculated for each phase. Using these quality scores, one can see how the user experience changes during the journey. Figure 2.2b shows how this can be visualized. Similar to the Brand Touchpoint Wheel, this method also requires manual work and assumes that the customer journey always consists of the same sequential steps.

In order to visualize the different contact channels used in the customer journey, Lee et al. proposed the Service Journey Modeling Language (SJML) [28]. This method has the advantage that it visualizes the different channels a customer goes through. Another advantage is that SJML is not fixed, i.e. two journeys with different lengths and steps can still be compared. As a result, it is more flexible than the Customer Journey Map and the Brand Touchpoint Wheel. The advantage of the Customer Journey Map over SJML is that it is able to display the quality and/or emotions of the customer during the journey, a concept that is missing in SJML. Another method similar to SJML is the customer journey method proposed by Nenonen et al. [35], where the journey of the customer is modeled in a customer path diagram. This method has the same advantage over the Customer Journey Map and the Brand Touchpoint Wheel as SJML: it does not depend on a fixed amount of categories and it visualizes how the customers use the various contact channels. All the methods mentioned before assume a customer journeys to always be a sequential process without repetition. Gudiksen et al. proposed the Service Ouroboros [22]. This method is quite similar to SJML and the customer journey method, with the exception that this
model allows for customer journeys to repeat themselves. Despite this advantage, the model still has the disadvantage that it assumes that the customer journey is always roughly the same. In cases where customers contact the company, there is no fixed process model that is followed by the customer. As a result, there are many deviations in the behavior. Hence, this method cannot be applied for identifying customer journeys in large datasets.

Another method for the analysis of services is service blue printing by Shostack [41]. In this method, the customer journey is drawn as a sequence of touchpoints. For all the drawn touchpoints, the connection to the backstage processes is drawn. This method illustrates the complexity of those backstage processes. However, this model can also only be used for a single sequential customer journey and is therefore not suitable for a situation in which several different journeys exists.

2.1.3 Online advertising

The analysis of customer journeys is well-known in the field of online advertising. By analyzing customer journeys, the effect and performance of different advertising channels can be measured [5, 6, 38, 50, 51]. In contrast to the techniques described in Section 2.1.1 and Section 2.1.2, which are based on interviews and manually creating a model, the techniques described in this section are based on data. Therefore, these techniques seem suitable for the identification of customer journeys in large datasets.

Techniques for analyzing customer journeys in the field of online advertising are often based on Markovian graphs [5, 38]. The different advertising channels are modeled as states, the transitions between two states are modeled based on the amount of customers that visit the pair of advertising channels in a single journey. This Markovian graph shows which customer journeys lead to a conversion, i.e. a sale, and therefore which customer channels have the most influence in leading to a conversion. Browser cookies are used to store information about ads that are shown to the user. Since it is known which ads have been shown, it is not the case that two parallel customer journeys are modeled as one single journey. However, these techniques have the disadvantage that if a customer uses multiple devices, a journey is created for each device, and not for the customer itself.

While some of the analyses focus on the conversion and the influence of each advertising channel, others focus on the value of a potential customer. The value of a potential customer depends on the state he or she is in. Based on this state of this and the investment needed to advertise to the potential customer, one can calculate the expected return of investment (ROI). Based on the ROI, one can decide whether further investment in advertisement to that potential customer is desired [5].

As stated before, Markovian graphs are used for analyzing customer journeys in the field of online advertising [5, 38]. This technique has the disadvantage that it only checks which ads are shown after each-other, so only a follow relation is used. For measuring the performance of online advertisements, this is not a problem. Cookies already can ensure the logging of actual customer journeys. However, this technique assumes that a customer journey is on a single device, i.e. the customer does not use multiple devices. For obtaining customer journeys from a large dataset it might be needed to take other attributes into account. As shown in the running example (Section 1.4), different customer journeys are separated by the subject and dates, and not through a follows relation. Hence, this technique is not suitable for identifying customer journeys in large datasets.

2.2 Process mining

Process mining is extracting non-trivial and useful information from event logs [1, 2]. Process mining techniques connect models, representing the real world, and event logs. Event logs contain events recorded by the real world. Analysts try to understand business processes by analyzing models discovered from the event logs. By comparing such models to normative models, i.e. by
comparing the reality with the “should be” scenario, deviations can be discovered. Figure 2.3 shows the role of process mining techniques in the real world. As Figure 2.3 indicates, there are three different areas within the area of process mining: discovery, conformance and extension.

The remainder of this section completes the introduction to the field of process mining by discussing the three areas in the area of process mining. Work that is already done in the field of process mining related to this thesis can be divided into three categories: trace clustering, abstractions, and mining process models when case identifiers are missing. After the discussion of the three different areas within process mining, the clustering of traces (Section 2.2.1) is discussed. Section 2.2.2 discusses the related work regarding the abstraction of events. Finally, Section 2.2.3 discusses the related work in which process mining techniques are applied when there are no case identifiers available.

Discovery
This type of process mining covers the control-flow discovery from an event log. In other words, automatically constructing a process model from an event log without using any a-priori information [3]. An example of such a process discovery algorithm is the \( \alpha \) algorithm [4], which takes only an event log as input. The output of the algorithm is a model describing the behavior seen in the event log. An example of such a resulting model is a Petri net. The \( \alpha \) algorithm can also be used to discover resource-related models such as a social network.

Conformance
This type of process mining requires an a-priori model, which is compared to the given event log.

\footnote{Source \url{http://processmining.org}}

Figure 2.3: Process mining in the real world\footnote{Source \url{http://processmining.org}}. Process mining techniques connect models representing the real world, and the event logs generated by the real world. This enables identification of differences between the models (the should-be situation) and the real world (the “as-is”) situation.
Conformance checking can be used for several reasons, including: measuring the quality of the process model and detecting, locating and explaining deviating behaviors in the event log [1]. In general, four quality dimensions for comparing model and log are considered when comparing the log and the model: fitness, simplicity, precision, and generalization.

**Extension**
The last type of process mining is extension and also requires an a-priori model. Here, the model is extended with a new aspect or perspective or the model is enhanced using information about the actual process recorded in the event log [1].

### 2.2.1 Trace clustering

Customers are usually unique from a control-flow perspective, especially in a healthcare setting. When modeling each customer as a trace in the event log and automatically discovering a model, so-called “spaghetti” models are obtained. As shown in Figure 2.4, such a model is completely unreadable and therefore not useful to the analyst. One solution to obtain better readable models is by somehow clustering traces and discovering a model for each individual cluster. The submodels are based on traces containing similar behavior. Since the submodels contain only traces with similar behavior, they are much more readable than the model describing the entire event log. Several trace clustering solutions have been proposed for this.

In [21], Greco et al. proposed a method that derives an initial process model and subsequently derives multiple sub process models in an iterative way. Similar work has been proposed by Medeiros et al. [34]. Their method consists of two steps, executed in an iterative way: first one or more models are mined from the event log using the Heuristics Miner [48]. In the second step, the quality of the mined model(s) is assessed. If the quality of the models is below a given threshold, the log will recursively be partitioned and new process models will be mined. The algorithm stops when all models have at least the desired quality. Hence, after applying the algorithm, one has a set of process models that each describe specific behavior in the original event log. De Weerdt et al. proposed a similar method [47]. Their method clusters an event log using a three-phase clustering algorithm. The goal of the selection phase is to add a new distinct process instance to the set of already selected instances, with the purpose of evaluating the process model discovered from this new sub-log. In the look ahead phase, the algorithm checks for all traces that were not in the selection phase if they fit perfectly in the model. If this is the case, the trace is added to the clusters. These first two steps are repeated until a predefined number of clusters is reached. In the last phase, the residual trace resolution phase, the traces that are not in a cluster are distributed over the clusters that are discovered in the previous steps. For the quality checks of the discovered clusters, the quality of the corresponding process model is used. By using the process model quality as a quality measure, the authors bridge the gap between the clustering bias and the evaluation bias. All methods described above use hierarchical clustering as a basis. However, as stated in [14], hierarchical clustering does not work well on very large datasets, as it is only effective at splitting small amounts of data.

The techniques described above partition the log based on the quality of the resulting model, 

![Figure 2.4: Example of a “spaghetti” model. The model is large and unstructured and therefore unusable to an analyst.](image)
CHAPTER 2. RELATED WORK

but do not take data attributes into account, while customer journeys are identified using data attributes. Song et al. proposed a trace clustering technique that does consider data-attributes [43]. By clustering on data-attributes, the clusters become more intuitively related to business mentality [15]. Song et al. model each trace as a vector model, called a profile. A vector space model is an algebraic model and is popular in the area of information retrieval [40]. By partitioning traces based on their profile, a process model is obtained for each group of similar traces. In the context of healthcare, this would mean that a profile represents a patient, and each resulting process model represents the behavior of a specific kind of patient. A more concrete example: each process model represents the behavior of a customer with a specific type of disease. A major advantage of this technique is that it is flexible. Different similarity measures and different clustering algorithms can be used. Hence, it is not always necessary to use a predefined number of clusters. As data attributes can be used for clustering, and this technique is able to work with a clustering algorithm that does not need a predefined number of clusters as input, this technique might be suitable for clustering events to identify customer journeys.

In [9], Bose and van der Aalst proposed a method for the clustering of traces based on a generic distance function. Since the generic edit distance framework is highly sensible to the cost of the edit operation, an automated, context aware approach for deriving the cost of the edit operations is proposed. Using the found costs for the edit operations, traces can be clustered using the generic edit distance framework. This technique relies on the costs derived for substitution, insertion or deletion of an activity. To use such a technique for identifying customer journeys, one would need to transform this to a framework that is able to derive the costs for substituting, inserting or deleting activity attributes. When identifying customer journeys, a comparison between events is needed, and not between traces. Hence, to be able to apply this technique for identifying customer journeys, one would need to derive an entire new formula for deriving the cost or substituting, inserting or deleting attributes of an event. Another disadvantage of this technique is that such a cost deriving function would look at all the data attributes, while in some cases some attributes might not be representative for identifying a customer journey.

As a trace is a sequence of events, each event describes a transition between the various states in a system. In [45], Veiga and Ferreira proposed a method that derives $k$ Markov chains from a given event log. The first step is to create $k$ clusters with all a random state transition matrix. By adding each trace to the cluster that fits best, clusters are improved. This is continued until the clusters do not change anymore, and a stable state is achieved. This method has the same disadvantage as the previous method that uses k-means clustering [43]: one needs to specify the number of clusters manually. However, the number of clusters are generally unknown. Due to the facts that transition matrices and Markovian chains are used and at only sequences of events/activities are looked at, this method is not able to discover some special workflow constructs such as parallelism. Rebuge and Ferreira also proposed a method for analyzing real-life healthcare processes [39]. During the development of their method, three challenges where identified: incompleteness and noise of clinical event logs, richness in process variants, and exceptional medical cases which must be captured and not disregarded. To solve these three challenges, the log is clustered using sequence clustering [45]. The authors extended this method by creating a methodology for analyzing an entire business process using multiple methods. Using a case-study on hospital data, it is shown that the proposed method is able to identify regular behavior, process variants, and exceptional medical cases. As stated before, the user needs to specify the number of clusters. As the number of journeys are unknown, this technique is unsuitable for identifying the different journeys of a single customer.

In [14], Delias et al. proposed a method that delivers compact and comprehensive synopses of flexible behaviors, keeping in mind the end goal to best support their analysis and improvement. For the clustering, the authors define a similarity metric based on a vector and a similarity matrix, but the proposed method should be able to work with all kinds of similarity metrics. The vector $V_{\cdot \cdot}$ is an ordered binary vector. The $k^{th}$ value is set to 1 if and only if trace $\tau$ contains activity $k$. Otherwise, the value is set to 0. The similarity matrix is a square matrix $M_{\tau}$ whose rows and columns are both equal to the number of activities. Each cell in $M_{\tau}$ in row $k$ and column $l$ is filled according to Equation 2.1, where $\text{dist}(k,l)$ is the distance between activities $k$ and $l$ in trace $\tau$. 

\begin{equation}
\text{dist}(k,l) = \begin{cases} 
1 & \text{if } k = l \\
0 & \text{otherwise}
\end{cases}
\end{equation}
CHAPTER 2. RELATED WORK

Using $M_{\tau}$, a second vector $V_{\tau,2}$ is created using Equation 2.2, where $n = (k - 1)K + l$ and $K$ is the number of activities. As a result, there are two vectors for each trace $\tau$: $V_{\tau,1}$ and $V_{\tau,2}$.

\[
M_{\tau}(k, l) = \frac{1}{\text{dist}(k, l)} \tag{2.1}
\]

\[
V_{\tau,2}(n) = M_{\tau}(k, l) \tag{2.2}
\]

\[
\text{Sim}(\tau_i, \tau_j) = w_a \cdot \text{cosineSim}(V_{\tau_i,1}, V_{\tau_j,1}) + w_t \cdot \text{cosineSim}(V_{\tau_i,2}, V_{\tau_j,2}) \tag{2.3}
\]

Using these two vectors, the similarity between two traces $\tau_i$ and $\tau_j$ can be calculated using Equation 2.3. When doing this for every pair $\tau_i$ and $\tau_j$, a similarity matrix is obtained. To cluster the traces, spectral clustering [36] is used. Using the ILP algorithm [49], a Petri net model is mined for each cluster. The technique proposed by Delias et al. is focused on clustering traces and so is the similarity metric. To be able to apply this technique for identifying customer journeys, one would need to define a similarity metric for events, indicating which events could belong to the same journey and which do not. Since this technique has the ability to work with different kind of similarity metrics, this technique might be suitable for identifying customer journeys. In this case, the quality of the obtained journeys would depend on the defined similarity metric.

In [15], Delias et al. proposed an effective exploratory tool for discovering the characteristics that are causing process variations. By clustering the log and mining a Petri net for each cluster, the two most dissimilar models can be found. Using these two models, a decision tree can be built in order to see which characteristics influence the flow variation the most. Similar work is presented in [30] by Leoni et al. In this work the authors proposed a framework to analyze an event log by providing: a broad and extendable set of characteristics related to control-flow, data-flow, time, resources, organizations and conformance, as well as a generic framework where any characteristic or dependent variable can be explained in terms of any set of other characteristics or independent variables. In [30], the log is partitioned using a decision tree. By choosing a dependent variable and independent variables, a decision tree is built where each leaf contains a multi-set of traces. The disadvantage of the work in [15] and [30] is that both methods are not suitable for global analysis, as a dependent variable needs to be chosen. Since the customer journeys are unknown, such dependent variable is also unknown. Hence, these techniques are not suitable for identifying customer journeys. However, these techniques do have the advantage of being more suitable for specific analysis as one needs to choose a dependent characteristic.

In [43] and [15], data attributes are used for the clustering of traces within the event log. Both these methods have the disadvantage that they rely on a clustering algorithm that takes the number of clusters as input. In most cases, the number of desired clusters is unknown, making these techniques unsuitable for most scenarios. Hompes et al. proposed a method that does not require the number of clusters as input [25]. Similar as the method in [14], a similarity matrix is used. The similarity matrix is created by modeling each trace as a profile, as proposed by Song et al. in [43], and computing the cosine similarity between the two profiles. The advantage of using the cosine similarity is that it always returns a number between 0 and 1. As a result, no normalization step is needed. After applying Markov clustering (MCL) algorithm [18] on the similarity matrix, clusters can be found in the matrix. MCL clustering is developed for graph clustering using flow simulation for simple graphs and weighted graphs [18,19]. It has previously been applied in the field of bio-informatics [20]. Another advantage of MCL is that only two parameters need to be specified. By changing the parameters, the granularity of the clustering is changed. A similarity matrix can also be constructed for events in a single trace. With such a modification, this technique can be applied for identifying the different customer journeys for a single customer.

The richness of process variants in an event log can also be the result of a process that changes over time. When the process of a company has changed, the journeys of their customers may also have changed. In [26], Hompes et al. proposed a method for detecting points in time where a process has changed. By dividing the log in time windows, a similarity matrix can be created for each time window as described in [25]. If the difference between the matrix of two time windows exceeds a certain threshold, it is an indication that the process is changed in the last time window.
This technique could be used for detecting change in the overall behavior of all customers, which might be an indication for a change in the process. For this technique to work in customer journey analysis, the actual customer journeys need to be discovered. Hence, an event log containing in which each trace represent one journey is needed. As a result, this technique is not suitable for identifying customer journeys, but it is useful for investigating the diversity in customer journeys, and their evolution over time.

### 2.2.2 Abstractions

“Spaghetti”-like models often contain multiple patterns, i.e., some activities that are always executed after each other. Such patterns can be abstracted to a single abstract event. The result of applying such abstractions is that the number of activities in the model will be reduced, therefore decreasing the amount of edges. Having less activities and edges results in a less “spaghetti”-like model. Günther et al. proposed such a technique in [23]. Using clustering, the low-level events are combined to a single higher-level activity. This method has one major disadvantage that other abstraction methods do not have: a timewindow parameters needs to be chosen by the user. If this parameter is not chosen in a correct way, this method might return unwanted results. Another disadvantage is that this technique only looks at the order in which events occur. For the identification of customer journeys, other data attributes, such as the subject and the time, should be used. Hence, this technique is not suitable for identifying customer journeys.

Bose et al. proposed a technique for the abstraction of events [8]. The technique proposed in their paper pre-processes the event log such that the resulting process model is less “spaghetti”-like. The method identifies so called tandem traces and repeats as defined in [10] and replaces it with a single abstract activity. In [11], Bose et al. show how it is possible to obtain a hierarchical process model using a set of interrelated plugins in ProM by abstracting low-level activities to a single high-level activity. Similar as the previous technique, only the time of events is considered, so also this technique is not suitable for identifying customer journeys.

The principle of abstraction has also been applied in [33], where Mans et al. performed a case study on the billing system of a hospital. Since directly applying process discovery algorithms on the event log would result in “spaghetti”-like models, groups of low level activities are abstracted to a single abstract activity. To obtain process models from the processed log, the Heuristics Miner [48] was used because of its robustness to noise, in combination with clustering. This results in a collection of process models. Each of the process models and patterns that were obtained, were recognized by domain experts of the hospital. A similar approach has been proposed by Smirnov et al. in [42]. By defining a distance measure between two vectors representing two activities, a clustering algorithm can cluster activities. Such a cluster of activities can be grouped into a single abstract activity. Since this technique uses the distance between two vectors for determining whether activity abstraction can be done, it allows for all kind of data attributes, such as the timestamp and the activity, to be used as well.

In some cases, the activities that are logged, can directly be mapped to higher level activities. For example, when using the log from a service consisting of various modules, each activity can be mapped to a specific module. Yang and Shan have proposed a method to identify such modules from an event log [52]. Using Ward’s method and k-means clustering, the activities are used together to determine clusters for identifying service modules. However, the modules are identified by clustering the different activities in the event log, while for the identification of customer journeys, events need to be clustered into journeys. Hence, this technique is not suitable for identifying customer journeys.

Rather than clustering low-level activities into a single high-level activity, Leemans et al. proposed a method for extracting patterns called episodes [29]. An episode is a partially ordered set of activities in an event log. An episode is called frequent if it is “embedded” in many sliding time windows. Mining episodes is useful as it identifies local patterns in the event log. If there are local patterns in the event log, it might be the case that there is a sequence of events that multiple customers encounter. Hence, it might be the case that a journey is found. The disadvantage of this technique is similar to the previously described techniques: only the time of the events are
CHAPTER 2. RELATED WORK

used. Because of this, it might be the case that the events in such a pattern are actually not related at all in such a local pattern. Hence, this technique is not directly suitable for identifying customer journeys.

In [16], Diamantini et al. proposed an approach called Behavioral Process Mining as new approach to enlighten relevant sub-processes, representing meaningful collaboration work practices. The proposed approach is based on the application of hierarchical graph clustering to the set of instance graphs generated by a process. The authors also proposed a technique for building instance graphs from traces. The technique for building instance graphs from traces, which is based on Finite State Machines, is also the major disadvantage of this technique. This technique shares the disadvantage that only the order of events is used, while for the correct identification of customer journeys, other data attributes should be used as well.

2.2.3 Mining process models when case identifiers are missing

A case identifier is essential for traditional process mining techniques. However, in some settings, events in a dataset are not labeled. Hence, traditional process mining techniques cannot be applied. Similarly, for the identification of customer journeys, only the identifiers for customers are known, while the identifiers for the journeys are not known. Several techniques have been proposed for extending the event log with trace identifiers.

Bayomie et al. proposed a method in [7] to deduce case identifiers for an event log where these identifiers are missing. They proposed an automatic approach for adding case identifiers to an unlabeled event log. Based on the given process model representing the executed process, and heuristic information about the execution time of the different activities within the process, the approach is able to identify the missing case identifiers. In [24], Helal et al. proposed an extension to the work done in [7] by creating a method that can handle streaming data. Both these methods could be changed such that they can be used to discover customer journey identifiers, instead of trace identifiers. The main disadvantage of these methods is that they require a process model as input, making them unsuitable for discovering customer journeys, as in this case, the journeys are unknown. As a result, this technique is not suitable for identifying customer journeys.

Liu et al. presented a two-phase process model learning process [31]. Using a probabilistic approach, links between the executed activities are learned together with the unknown process models. Contrary to the two approaches described above, this method does not need the executed process models as input. However, it does require the total number of process models as prior knowledge, which is a major disadvantage for this technique. As the number of unique journeys of a customer is unknown, the total number of process models is also unknown, which makes this proposed technique unsuitable for our analysis.
Chapter 3

Process mining on customer journeys

Chapter 2 discussed relevant techniques in the field of customer journey analytics and process mining. The techniques for analyzing customer journeys require manual labor. As Section 3.1 shows, the event logs used in this thesis contains \( \sim 77 \) thousand customers with \( \sim 750 \) thousand contact moments. As a result, the techniques described in Section 2.1 are not suitable for analyzing this type of event logs. Hence, process mining techniques will be used to identify customer journeys.

In the remainder of this chapter the application of process mining techniques on customer journey data is described and is organized as follows. First, Section 3.1 discusses the log that can be obtained from Customer Relationship Management (CRM) systems. Section 3.2 discusses the application of various existing process mining techniques on the event logs discussed in Section 3.1. During this application, problems will arise. These are discussed in Section 3.3.

3.1 CRM log files

The type of log files used for this thesis are obtained from CRM systems. Each time a customer contacts a company, the date and time, the subject and the contact channel are stored in a database. The last attribute can be, for example, calling service center, e-mailing or visiting a service desk. Many companies also have an online portal on which their customers can sign in. In this online portal, customers can perform various actions, such as changing their subscription. In case the behavior of the customers on such an online portal is tracked, this data can also be added to the log. The subject of such a contact moment can be derived from the URL of the visited page.

In this thesis, a dataset is used that contains only the contact moment in which the customer contacts the company. Table 3.1 shows a part of an example event log. In theory, also moments when the company contacts the customer can be used in the log file.

Process mining tools use files in the MXML [17] or XES [46] format. Unfortunately, this format is not shared by CRM systems. Hence, a conversion is needed. For the case study on CZ data (Chapter 6) such a conversion is performed. This conversion is described in Section 6.1 and can be found in Appendix A.

3.1.1 XES log

The event log obtained after the conversion contains a trace for each customer that was in the original dataset. The events in each trace represent individual moments on which the customer contacted the company. In the case of CZ, a customer contacts CZ if he sends an e-mail, calls or visits the service desk for a single subject. It may be the case that the customer asks two questions in a single e-mail. In such a case, this is seen as two contacts using the same channel.

Customer journey identification through temporal patterns and Markov clustering
CHAPTER 3. PROCESS MINING ON CUSTOMER JOURNEYS

Table 3.1: Example event log.

<table>
<thead>
<tr>
<th>Customer</th>
<th>Date</th>
<th>Channel</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>30-02-2015</td>
<td>Service desk</td>
<td>Damages abroad</td>
</tr>
<tr>
<td>Alice</td>
<td>31-02-2016</td>
<td>Online</td>
<td>Damages abroad</td>
</tr>
<tr>
<td>Bob</td>
<td>30-03-2016</td>
<td>Online</td>
<td>Payments</td>
</tr>
<tr>
<td>Bob</td>
<td>26-04-2016</td>
<td>E-mail</td>
<td>Available subscriptions</td>
</tr>
<tr>
<td>Bob</td>
<td>27-04-2016</td>
<td>E-mail</td>
<td>Change subscriptions</td>
</tr>
<tr>
<td>Bob</td>
<td>28-04-2016</td>
<td>Telephone</td>
<td>Car repair</td>
</tr>
<tr>
<td>Bob</td>
<td>28-04-2016</td>
<td>Service desk</td>
<td>Car repair</td>
</tr>
<tr>
<td>Charlie</td>
<td>20-11-2015</td>
<td>Online</td>
<td>Payments</td>
</tr>
<tr>
<td>Charlie</td>
<td>20-12-2015</td>
<td>Online</td>
<td>Payments</td>
</tr>
<tr>
<td>Charlie</td>
<td>22-05-2016</td>
<td>Online</td>
<td>Payments</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

and with the same timestamp. The customer can also contact CZ via the online environment. In this case, each page visit is counted as a contact. The data attributes concerning the contact moment, such as subject, contact channel and date & time, are stored as attributes of an event. Customer attributes, such as age, gender are stored as attributes of a case.

3.1.2 Log details

Using the “Log Visualizer” plug-in in ProM, the log file from CZ is analyzed. First to notice is that most of the traces are relatively short. Figure 3.1 shows how the length of the traces is distributed. In the period of 1 year and 8 months, most of the customers contacted the company ten times, or even less. There are \( \sim 77 \) thousand traces in the event log and \( \sim 750 \) thousand events, meaning that there are \( \sim 77 \) thousand different customers that contacted CZ at least ones during a period of 1 year and 8 months.

Despite that the traces are short in most of the cases, the obtained process model gives limited insights. Figure B.1 shows the Petri net obtained with the Inductive Miner. Only 10 activities do not support the behavior of a “flower” model. Even though these events do not cover the behavior of a “flower” model, they give limited insights. Chapter 2 discussed some existing process mining techniques for obtaining process models with a higher quality, which could lead to more insights. A subset of these plug-in have been used on the event log described above. In Section 3.2 the application of these techniques is discussed.

Figure 3.1: Trace length Distribution. The x-axis shows the length and the y-axis shows the amount of traces. Most of the traces have a short length, meaning that the amount of contacts per customer with CZ is small. On average, 10 events are recorded per customer.
CHAPTER 3. PROCESS MINING ON CUSTOMER JOURNEYS

3.2 Evaluation

This section describes the application of existing process mining techniques on event logs as described in Section 3.1. To show the effect of applying existing process mining techniques, the CZ data is used. Some of the trace clustering techniques discussed in Section 2.2 are applied on this event log. This is discussed in Section 3.2.1. Similar to the application and discussion of trace clustering techniques, techniques for abstractions are also applied on such an event log. This is discussed in Section 3.2.2.

3.2.1 Trace clustering

Applying process discovery algorithms directly on a dataset as described in Section 3.1, leads to “spaghetti” models. As Figure B.1 shows, these models are difficult to interpret. In Section 2.2.1 some techniques for obtaining more readable process models are discussed. The techniques in this section use the clustering of cases as a basis and use process discovery algorithms on each cluster to obtain a set of process models.

Hompes et al. proposed a technique that clusters cases using the Markov cluster (MCL) algorithm [25]. This technique is implemented as a ProM plug-in in the TraceClustering package. The plug-in requires three parameters: the expansion parameters, the inflation parameter and the clustering perspectives. The expansion and inflation parameter influence the granularity of the resulting clustering. MCL and its parameters are discussed in Section 4.2.3. Similar as in [25], the expansion parameter is set to 2 and the inflation parameter is set to 15. The perspectives parameter of the plug-in indicates which perspectives are used when building the similarity matrix. There are three different types of perspectives that can be chosen: the frequency of an event attribute value in a trace, the occurrence of an event attribute in a trace and attributes of a trace. The remainder of this section discusses several chosen combinations of perspectives aimed at obtaining more readable process models.

Even though MCL is a fast and scalable cluster algorithm for graphs [18, 25], the dataset used in this thesis is too large for the “TraceClustering” plug-in. Hence, for this analysis a subset of 1,000 randomly selected customers is used.

![Figure 3.2: Clustering results from Trace Clustering plug-in. Two different sets of perspectives have been used. On the left, the attributes “property1” and “group” are used for clustering and resulted in seven clusters. On the right, the “property1” and “gender” and resulted in two clusters. The size of the clusters depend on the amount of traces in the cluster.](https://svn.win.tue.nl/trac/prom/browser/Packages/TraceClustering)
Table 3.2: Clustering details of Figure 3.2. On the left the clustering of Figure 3.2a is shown. On the right, the clustering of Figure 3.2b is shown.

(a) Explanation of the 7 clusters in Figure 3.2a.

<table>
<thead>
<tr>
<th>C</th>
<th>Perspective</th>
<th>Value(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>Property 1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Group</td>
<td>“3.Volwassenen”</td>
</tr>
<tr>
<td>c2</td>
<td>Property 1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Group</td>
<td>“2.Jongvolwassen”</td>
</tr>
<tr>
<td></td>
<td>Property 1</td>
<td>1,3,4,5,6,7</td>
</tr>
<tr>
<td>c3</td>
<td>Property 1</td>
<td>2</td>
</tr>
<tr>
<td>c4</td>
<td>Property 1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Group</td>
<td>“4.Gezinnen”</td>
</tr>
<tr>
<td>c5</td>
<td>Property 1</td>
<td>1,3,4,5</td>
</tr>
<tr>
<td></td>
<td>Group</td>
<td>“4.Gezinnen”</td>
</tr>
<tr>
<td>c6</td>
<td>Property 1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Group</td>
<td>“6.Senioren”</td>
</tr>
<tr>
<td>c7</td>
<td>Property 1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Group</td>
<td>“6.Senioren”</td>
</tr>
</tbody>
</table>

(b) Explanation of the 2 clusters in Figure 3.2b.

<table>
<thead>
<tr>
<th>C</th>
<th>Perspective</th>
<th>Value(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>Property 1</td>
<td>1,2,3,4,5,6</td>
</tr>
<tr>
<td></td>
<td>Gender</td>
<td>“V”</td>
</tr>
<tr>
<td>c2</td>
<td>Property 1</td>
<td>1,2,3,4,5,6</td>
</tr>
<tr>
<td></td>
<td>Group</td>
<td>“M”</td>
</tr>
</tbody>
</table>

Clustering using customer attributes

First, clustering on all customer specific attributes is used in combination with the expansion and inflation parameters as given above. Using all perspectives regarding trace attributes results into a single cluster. Hence, a combination of perspectives regarding to trace attributes is used. The result of this is shown in Figure 3.2. Table 3.2 shows, for both clusterings, the values of the used perspectives per cluster.

Using the Inductive Miner plug-in in ProM, a process model is obtained for the two largest clusters \((c_3 \text{ and } c_6)\) in Figure 3.2a and the two clusters in Figure 3.2b. Based on the noise threshold setting, different process models are derived. However, even with the highest possible noise threshold, the miner returns large unstructured models. In these models, groups of activities (10+) form “flower” models within the entire model. As a result, it is hard to obtain useful insights from such models. An explanation for this behavior is that the actual customer journeys are not identified. In the used dataset all events of a single customer are stored in a single trace. As a result, non-related events are still considered to be related. In large datasets, this might cause a relation between all pairs of events.

Using a process discovery algorithm on a cluster of traces results still into models with no or little structure. Event when looking at smaller parts of the resulting process models does not reveal any sub-processes that might be an indication for a single customer journeys. Hence, this technique cannot be used for identifying journeys.

Event attribute clustering

As shown previously, using customer attributes as clustering perspectives does not result in good clusters since the process models obtained for those clusters still have a low quality. Another possible option for clustering is using perspectives regarding event attributes. Due to the diversity in the behavior of each customer, there are always two events in different traces that have a perspective with a similar value. When this is the case, MCL adds both these traces to the same
cluster. As a result, all traces are added to the same cluster. In some cases, one or two traces are not added to this large cluster. So, clustering customers/traces based on attributes of the events of the customer does not lead to a good clustering. As a result, it is not possible to obtain high quality process models for these clusters. Even if the technique was able to cluster some cases together when using event attributes as perspectives, similar results as described in the previous section are expected, due to the fact that a case contains all events of a single customer and not a single customer journey. Hence, trace clustering techniques, in combination with event attributes as perspectives, are not suitable for identifying customer journeys.

Conclusions
Clusters are based on data-attributes to become more intuitively related to business mentality [15]. To test the effect of clustering traces, an event log from CZ, as described in Section 3.1, is used in combination with the MCL algorithm as proposed in [25]. Due to the diversity in the behavior of customers, it is difficult to find a set of perspectives that leads to a good clustering. As shown in the previous paragraphs, even if the traces are divided nicely over a set of clusters, the resulting process models are closely related to “flower” models and therefore, give limited insights. Only a few activities describe behavior that is not in a “flower” model. As a result, it is not possible to obtain customer journeys through the trace clustering.

Due to the fact that each trace contains all the behavior of a single customer, it is likely that a single trace contains multiple customer journeys. Hence, a technique is needed that is able to detect those journeys and creates a trace for each journey. In this case, a trace represents a single customer journey. Hence, each trace contains events that are related to each other. When applying directly a process discovery algorithm on all the traces, still large and hard to interpret models can be found, due to the high variability between the traces. Hence, an algorithm is needed that identifies the customer journeys of each customer.

3.2.2 Abstraction techniques
Section 2.2.2 discusses techniques for creating better readable process models based on abstracting groups of low-level events into a single higher-level event. In this section, some of these techniques are applied. Similar to the previous section, the CZ event log is used. First, the technique by Bose et al. [8] is used on such a log file. After an analysis with this technique, the frequent episode mining technique by Leemans et al. [29] is used on the same event log.

Event abstraction
First, a technique by Bose et al. is used. This technique pre-processes the event logs such that, when using a process discovery algorithm, the resulting process model is less “spaghetti”-like. Clusters of low-level events are replaced by a single higher-level event. The identification these clusters is done by identifying tandem traces and repeats [10]. This technique is available as a ProM plug-in. Similar as in Section 3.2.1, this technique is not able to process all 77 thousand traces. Hence, the same subset of 1,000 randomly selected customers is used. For the settings of the plug-in the following pattern types are selected: “Tandem Arrays (loops)” and “Maximal Repeats”. The default settings are used for all other settings. Using these settings, 150 alphabets are discovered. Based on these alphabets, pattern abstractions can be created. Similar as for identifying the alphabets, the default settings are used. The plug-in discovered 90 pattern abstracts in the log using the 150 discovered alphabets. The new event log is exported and using the Inductive Miner plug-in in ProM a process model is obtained. Similar as to the models obtained with trace clustering, the resulting model is unstructured and is closely related to a “flower” model.

In the event log used to mine the model, each trace represents all the behavior of a single customer. As a result, a single trace can contain multiple customer journeys. If there was a clear

---

2See https://svn.win.tue.nl/trac/prom/browser/Packages/PatternAbstractions
pattern that occurred in the behavior of multiple customers, this might have been a customer journey. However, the algorithm fails to find a clear pattern as it returns a “flower” model. As a result, this technique fails to capture different customer journeys from an event log.

**Frequent episode mining**

As stated in Section 2.2.2, Leemans et al. proposed a technique for discovering local patterns in an event log [29]. This technique is available as in the “Leemans Episode miner” plug-in for ProM\(^3\). This plug-in is used on the event log described in Section 3.1.

Due to the diversity between all the customers and the diversity of the behavior of a single customer, this plug-in is not able to detect any local patterns in the event log. To obtain any patterns, the frequency threshold needs to be decreased to a low value (~ 0.1). Using this setting, and default values for the other settings episodes are discovered in the event log.

Using these settings result into a large number of episodes. However, most of the episodes are quite similar. Figure 3.3 shows two different episodes from that result. The first episode (Figure 3.3a) is an episode in which three times the activity “Inkomend telefonisch/Algemeen” is executed. This episode is one of the episodes with the highest frequency (0.262). The other episodes with the same frequency, are also based on executing three times this activity.

The Leemans Episode miner is only able to obtain some episodes that have a low frequency (\(< 0.27\)). This means that there is not a single episode in the event log that occurs for a relative high number of customers, i.e. it is not possible to find similar behavior between the customers. Hence, it can be concluded that there is no behavior shared between customers, which might have been a customer journey. Furthermore, it might be the case that two events in an episode are not related to each other. As a result, they should not be in the same episode. For example, the two events “Inkomend telefonisch/Betaling” and “Inkomend telefonisch/Algemeen” in Figure 3.3b, may not be related at all. The first event could be related to the monthly fee, while the second event is a question regarding whether the insurance holds in a foreign country and occurred two years later. In this case, these two events should not be considered to be in the same episode, as they should not be considered to be in the same customer journey.

**Conclusions**

In Section 2.2.2 techniques for abstraction of events are discussed. Two of these techniques have been applied on the dataset supplied by CZ. Similar to the techniques described in Section 3.2.1,

\(^3\)See [https://svn.win.tue.nl/trac/prom/browser/Packages/EpisodeMiner](https://svn.win.tue.nl/trac/prom/browser/Packages/EpisodeMiner)

![Inkomend telefonisch/Algemeen](a) Frequent episode with a frequency of 0.262

![Inkomend telefonisch/Algemeen](b) Frequent episode with a frequency of 0.139

Figure 3.3: Episode miner results. Using a low minimal threshold for the frequency, some frequent episodes can be obtained. Two of such episodes are shown. Both these episodes have a low frequency and deliver limited insights.
the techniques described in this section are not able to handle the full size dataset. Hence, a subset of 1,000 randomly selected customers is used. Due to the diversity in the behavior of all customers, both techniques are not able to obtain a high quality models.

The different customer journeys of may overlap each-other, i.e. it is possible that two or more journeys are executed in parallel. Currently it is unknown which events belong to which journey in such a case. Due to this fact, it may not be possible anymore for existing techniques to find groups of activities that can be abstracted to a single abstract activity. Hence, an algorithm is needed that identifies the customer journeys of each customer.

3.3 Identified problems

As shown in the previous section, some problems have been identified. This section will discuss some of these problems in more detail. The first encountered problem is the performance of existing techniques. Another identified problem is the diversity in the behavior of the customers.

3.3.1 Performance

The first identified problem is related to the performance of existing techniques. Both the techniques in Section 3.2.1 and in Section 3.2.2 cannot deal with the full-size event log. In all cases a subset of only 1,000 of the ~ 77 thousand customers was taken.

3.3.2 Diversity in customer behavior

First, the technique by Hompes et al. [25] was used on subset of 1,000 customers from the log file described in Section 3.1. In this technique, a set of selected perspectives is needed as input for the algorithm. During the search for a good set of perspectives, the second problem occurred: the behavior of a single customer is too diverse and no two customers have similar behavior. Therefore, there is almost always at least one attribute equal between two customers. As a result, clustering algorithms have troubles creating good quality clusters. As shown in Section 3.2.1, it is still possible to obtain a good clustering. However, the process models obtained for each cluster do have a low quality.

The low quality of the process models is caused by the diversity in the behavior of a single customer. In the running example (Section 1.4), Bob is shown. In this example, Bob has also some diverse behavior: Bob checks payments monthly. In the same time, Bob tries to change his subscription, needs to have his car repaired a few times and needs information on how damages in another country are handled. Hence, the behavior of Bob can be divided into four categories: the monthly payments, the subscription change, the car repairs and the information question. Suppose that Alice is also a customer of MyCarService, and has a behavior similar to Bob. Existing trace clustering techniques will put Alice and Bob in the same cluster. Using an existing process discovery algorithm, will result in a model as is shown in Figure 1.4. The technique does not identify the three customer journeys. Hence, a technique is needed that is able to split the behavior of Bob and Alice into the three categories. In this case, a process model can be obtained in which the three journeys are separated and hence, it will have a higher quality and is more useful to an analyst.
Chapter 4

Solution

As discussed in the previous chapters, a technique is needed that is able to identify customer journeys from event logs. This chapter discusses the theoretical solution for identifying customer journeys. First, Section 4.1 discusses the approach on how to solve this problem and the main idea of the algorithm. Section 4.2 describes the algorithm for identifying the different customer journeys for a single customer. Section 4.3 demonstrates the inner workings of the algorithm using the running example. In Section 4.4 and Section 4.5 some drawbacks of the algorithm are discussed and some extensions to the solution are proposed. At last, Section 4.6 gives an overview of the full solution.

4.1 Solution approach

As stated before, a technique is needed that is able to split the behavior of customers into customer journeys, i.e. a technique that is able to split cases into multiple traces, each representing a single customer journey. The events that should be in the same journey have something in common that states that they belong together. Since events have attributes, the value of at least one attribute or measure should be equal between the events in a single customer journey. In the remainder of this thesis, perspective is used as reference to attributes and measurements that can be used for clustering. As a result, a clustering algorithm that clusters the events of a single customer into clusters can be used for identifying customer journey. In this case, each cluster is a single customer journey.

It might be the case that for two different event logs different attributes need to be considered. Hence, the user needs to specify a set of perspectives that can be used by the clustering algorithm. Figure 4.1 shows the global overview of the approach for identifying customer journeys. The remainder of this chapter discusses the solution of identifying the journeys for a single customer (“Identify journeys for c using p” in Figure 4.1).

Figure 4.1: Global overview of the customer journey identification approach. Using a set of selected perspectives, and a clustering algorithm, the journeys of a single customer are obtained.
4.2 Identifying journeys per customer

In Section 4.1 it is argued that the customer journeys are obtained per customer and that this can be done by clustering the events of a single customer into customer journeys. This section discusses how the customer journeys are obtained for a single customer.

The events of a single customer are divided into customer journeys using a clustering technique. Clustering of the behavior is done in a similar way traces are clustered in [25]. In [25], the user needs to select perspectives that are used for the clustering. Based on these perspectives, a profile is created for each trace, as proposed by Song et al. [43]. Both Delias et al. and Hompes et al. use a similarity matrix to store similarity between traces [14, 25]. Using a clustering algorithm on this similarity matrix, clusters are formed. The resulting matrix, can be used to divide traces over clusters. The result of this is a set of clusters, each containing traces with similar values for the selected perspectives.

The same concept is used for dividing the events of a customer into customer journeys. Instead of creating a profile for a trace, a profile is created for a single event (line 6 in Algorithm 1). Using all profiles, a similarity matrix is constructed. In this case, the similarity matrix does not represent the similarity between all pairs of traces, but the similarity between all pairs of events (line 7 in Algorithm 1). Similar as in [25], Markov clustering is used to create clusters from this matrix (line 8 in Algorithm 1). At last, the clustering is interpreted and for each cluster of events a trace is created (lines 9 and 10 in Algorithm 1). Each of these traces represents a single customer journey. The remainder of this section discusses the procedures mentioned above in more detail.

Algorithm 1 Journey identification for a single customer

\begin{algorithm}
\begin{algorithmic}[1]
\Function{IdentifyJourneys}{customer, userSettings}
\State perspectiveList $\leftarrow$ userSettings.selectedPerspectives
\If{perspectiveList is empty}
\State return original customer trace \Comment{Clustering is not possible, so return original trace}
\EndIf
\State profiles $\leftarrow$ \Call{BuildProfiles}{customer, perspectiveList}
\State similarityMatrix $\leftarrow$ \Call{BuildMatrix}{profiles}
\State clusteredMatrix $\leftarrow$ \Call{ComputeClustering}{similarityMatrix, userSettings}
\State eventClustering $\leftarrow$ \Call{InterpretClustering}{customer, clusteredMatrix}
\State journeyTraces $\leftarrow$ \Call{CreateTraces}{eventClustering}
\State return journeyTraces
\EndFunction
\end{algorithmic}
\end{algorithm}

4.2.1 Profiling events

In line 6 of Algorithm 1 the procedure “BuildProfiles” is used. In this procedure, a profile is built for each event in the trace. Figure 4.2 shows the idea of building a profile for event $c$ in the trace using the perspectives $P_1$ until $P_u$.

The procedure for creating profiles for each event in the trace is relative simple. Algorithm 2 shows how this is done by iterating over all events in the trace. Note that \texttt{EVENTS(customer)} is a

\[
\begin{array}{c}
\text{Trace} \\
\hline
\text{a} \\
\text{b} \\
\vdots \\
\text{c} \\
\text{w} \\
\hline
\end{array}
\rightarrow
\begin{array}{c}
\text{profile} \\
\hline
\text{[P_1 | c, P_2 | c, \ldots, P_u | c]} \\
\hline
\end{array}
\]

Figure 4.2: Profiling a customer. A profile is created for event $c$ using perspectives 1 until $u$. $P_1 | c$ means projecting event $c$ on perspective $P_1$. 

Customer journey identification through temporal patterns and Markov clustering
Algorithm 2 Profiling a customer

1: function BuildProfiles(perspectives,customer)
2:     profiles ← new list
3:     for all event in Events(customer) do
4:         profile ← BuildProfile(perspectives,event)
5:         add profile to profiles
6:     end for
7:     return profiles
8: end function

Algorithm 3 Profiling a single event

1: function BuildProfile(perspectives,event)
2:     profileVector ← new Vector with length of the perspectives list
3:     for all perspective in perspectives do
4:         value ← perspective ↾ event
5:         add value to profileVector
6:     end for
7:     return profileVector
8: end function

function that returns all events of the given customer. For each event in the trace the procedure in Algorithm 3 is used. Creating a profile \([P_1 \ | \ e, P_2 \ | \ e, \ldots, P_u \ | \ e]\) for an event \(e\) is also a quite straightforward procedure: iterating over all selected perspectives, project \(e\) on each perspective and add the resulting value to the profile. Projecting an event \(e\) on a perspective \(P\) is denoted as \(P \ | \ e\).

4.2.2 Build event similarity matrix

After the event profiles are built for each event of the customer, an event similarity matrix is constructed (line 7 in Algorithm 1). An event similarity matrix is a square matrix in which each cell represents the similarity between two events. Hence, the number of columns and rows of the matrix are equal to the number of events of customer. Figure 4.3 shows a visual representation of this step. The procedure for creating such a matrix from the set of profiles is shown in Algorithm 4. The idea of building an event similarity matrix is straightforward. By taking each combination of profiles, the similarity between those two profiles can be calculated. The similarity between two profiles is calculated using the procedure “ComputeSimilarity”, which uses cosine similarity. The

\[
\begin{pmatrix}
1 & \text{sim}(\alpha,\beta) & \text{sim}(\alpha,\gamma) & \ldots & \text{sim}(\alpha,\delta) \\
\text{sim}(\beta,\alpha) & 1 & \text{sim}(\beta,\gamma) & \ldots & \text{sim}(\beta,\delta) \\
\text{sim}(\gamma,\alpha) & \text{sim}(\gamma,\beta) & 1 & \ldots & \text{sim}(\gamma,\delta) \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\text{sim}(\delta,\alpha) & \text{sim}(\delta,\beta) & 1 & \ldots & 1 \\
\end{pmatrix}
\]

Figure 4.3: Creating an event similarity matrix. After the profiles are built for each event in the trace, a similarity matrix is constructed. On the left is the set of profiles from Figure 4.2. Using these profiles, the similarity matrix on the right is obtained. \(\text{sim}(\alpha,\beta)\) is the similarity between profiles \(\alpha\) and \(\beta\). The numerical value can be obtained using Algorithm 5.
procedure for calculating the difference between two profiles is shown in Algorithm 5.

The algorithm first checks the length of the two profile vectors. If these two lengths are not the same, the vectors are incomparable. Moreover, the algorithm checks whether the vectors are zero vectors, i.e., if both vectors contain only null values. If both profile vectors are zero vectors, they are equal and if only one of them is a zero vector, they are distinct. For both profiles a vector is created. Next, the algorithm iterates over both the profile vectors and calculates per index the similarity between the two values using the corresponding perspective. That is, the perspective, that is used to obtain the two values currently used in the profile vectors, is used to calculate the similarity between the two values. This calculation depends on the type of the perspective. If the calculated similarity is higher than a given threshold, the similarity value is added to the second vector, otherwise 0 is added to the second vector. In both cases 1 is added to the first vector.

After the iteration on both profile vectors, two vectors containing only numbers are obtained. The next step is to compute the similarity of these two vectors and hence the similarity between
CHAPTER 4. SOLUTION

the profile vectors. The calculation of the similarity is done using cosine similarity (Equation 4.1), however any method for computing the similarity between two vectors can be used. Cosine similarity is used because it is efficient and bounded to $[0, 1]$. As a result, no normalization step is required afterwards.

$$sim(v_1, v_2) = \frac{v_1 \cdot v_2}{||v_1|| ||v_2||} = \frac{\sum_{i=1}^{n} v_{1i} v_{2i}}{\sqrt{\sum_{i=1}^{n} v_{1i}^2} \sqrt{\sum_{i=1}^{n} v_{2i}^2}}$$  (4.1)

The result of Algorithm 5 is an event similarity matrix that can be used to split the trace into customer journeys.

4.2.3 Grouping related events

The events in a single trace are clustered into groups of events, each representing a single customer journey (line 8 in Algorithm 1). In theory any clustering algorithm can be used for the implementation of our technique. However, since the amount of customer journeys are unknown for a single customer, a clustering algorithm that does not require the amount of clusters as a parameter is desired. The Markov cluster algorithm (MCL) is such an algorithm. MCL is an algorithm for clustering graphs using flow simulation for simple and weighted graphs [18, 19]. MCL also has the added advantage that it is fast and scalable.

Algorithm 6 shows the MCL algorithm in pseudo code. The input for this clustering algorithm is an event similarity matrix and two parameters: expansion and the inflation parameter. The first step is to ensure that the matrix is column normalized, i.e. ensure that the matrix is stochastic. Next, MCL executes the expansion and inflation step in an iterative way until the matrix converges. That is, the change in the matrix from iteration $i$ to the matrix from iteration $j = i + 1$ is smaller than some threshold $\Delta_s$. In the expansion step, the matrix is raised to the expansion parameter, which is one of the parameters of MCL. After the expansion step, the inflation step is done. In this step, each entry in the matrix is raised to the inflation parameter, the other parameter of MCL. After this, the matrix is normalized such that it is stochastic again. After the last inflation step, the modified matrix is returned. The resulting matrix contains the different clusters that are obtained. Section 4.2.4 shows how this matrix is interpreted.

Algorithm 6 Markov cluster algorithm

1: function ComputeClustering(eventSimilarityMatrix, userSettings)
2:    matrix ← ColumnNormalizeMatrix(eventSimilarityMatrix)
3:    oldMatrix ← NULL
4:    e ← userSettings.expansionParameter
5:    i ← userSettings.inflationParameter
6:    while DIFFERENCE(matrix, oldMatrix) ≤ $\Delta_s$ do  ▶ Loop until matrix does not change
7:        oldMatrix ← matrix
8:        matrix ← matrix$^e$  ▶ Expansion step
9:        for row ← 0 to SIZE(profiles) do  ▶ Inflation step
10:            for column ← row to SIZE(profiles) do
11:                value ← matrix.Get(row, column)
12:                matrix.Set(row, column, value$^i$)
13:          end for
14:    end while
15:    matrix ← ColumnNormalizeMatrix(matrix)  ▶ Make matrix stochastic again
16:    return matrix
17: end function

Customer journey identification through temporal patterns and Markov clustering 29
4.2.4 Splitting a trace into customer journeys

The result of the clustering in line 8 in Algorithm 1 is a modified event similarity matrix. The last step is to create traces, each containing a single customer journey, from the modified event similarity matrix. The idea of this is illustrated in Figure 4.4. Algorithm 7 shows how such a matrix is converted to a set of clusters. For each row, i.e., for each event, a new cluster is created and the \( row^{th} \) event of the customer is added to this cluster, i.e. each event is used as an attractor. Next, the entire row is read and attractees are found. If the value in the corresponding cell is larger than a certain threshold value, the \( column^{th} \) event is seen as an attractee and is added to the cluster. Algorithm 7 uses the function \( \text{customer}.\text{Get}(column) \), this function gets the \( column^{th} \) event of the customer.

As soon as the entire row is read, and the cluster is filled, the new cluster is added to the set of clusters. This add function ensures that no two similar clusters are in the resulting set. Also, if

\[
\begin{pmatrix}
\alpha & \beta & \ldots & \gamma \\
\delta & \epsilon & \ldots & \zeta \\
\vdots & \vdots & \ddots & \vdots \\
\eta & \theta & \ldots & \iota \\
\end{pmatrix}
\Rightarrow
\begin{cases}
<\alpha, \ldots> \\
<\beta, \ldots> \\
<\ldots> \\
<\ldots, \omega>
\end{cases}
\]

Figure 4.4: Interpretation of the modified event similarity matrix. The modified event similarity matrix is the result of applying Algorithm 6. The interpretation of this matrix and the creation of traces, i.e. obtaining customer journeys is shown in Algorithm 7 and 8.

### Algorithm 7 Interpretation of the modified event similarity matrix

1: function \( \text{INTERPRETCLUSTERING}(\text{customer}, \text{clusteredMatrix}, \text{userSettings}) \)
2: \( \text{eventClustering} \leftarrow \text{new} \text{EventClustering} \)
3: for \( \text{row} \leftarrow 0 \) to \( \text{SIZE}(\text{profiles}) \) do
4: \( \text{cluster} \leftarrow \text{new} \text{EventCluster} \)
5: add \( \text{customer}.\text{Get}(\text{row}) \) to \( \text{cluster} \)
6: for \( \text{column} \leftarrow 0 \) to \( \text{SIZE}(\text{profiles}) \) do
7: \( \text{if} \ \text{clusteredMatrix}.\text{Get}(\text{row}, \text{column}) > \text{userSettings.stopCriterion} \) then
8: \( \text{event} \leftarrow \text{customer}.\text{Get}(\text{column}) \)
9: add \( \text{event} \) to \( \text{cluster} \)
10: end if
11: end for
12: add \( \text{cluster} \) to \( \text{eventClustering} \)
13: end for
14: return \( \text{eventClustering} \)
15: end function

### Algorithm 8 Obtaining traces

1: function \( \text{CREATETRACES}(\text{eventClustering}) \)
2: \( \text{traces} \leftarrow \text{new} \text{set of traces} \)
3: for all \( \text{cluster} \) in \( \text{eventClustering} \) do
4: \( \text{trace} \leftarrow \text{new} \text{Trace} \)
5: for all \( \text{event} \) in \( \text{cluster} \) do
6: add \( \text{event} \) to \( \text{trace} \)
7: end for
8: add \( \text{trace} \) to \( \text{traces} \)
9: end for
10: return \( \text{traces} \)
11: end function

Customer journey identification through temporal patterns and Markov clustering
the given cluster is a sub-cluster of an existing cluster or the resulting set contains a sub-cluster of the given cluster, the sub-cluster is removed and the only larger cluster is kept. When all rows are read and all clusters are created, the set of clusters is returned.

The last step in Algorithm 1 is converting the set of clusters to a multi-set of traces in which each trace represents a customer journey. For each identified cluster, a new trace is created and all events in the current cluster are added to the new trace. As soon as all events are added to the trace, the trace is added to the resulting set of traces. The algorithm thus finishes when all clusters are handled.

4.3 Application on running example

In our running example (Section 1.4) a company called MyCarService and its customer Bob were introduced. This section uses Bob to illustrate how the algorithm described in Section 4.2 works.

Algorithm 2 (Section 4.2.1) is applied on the trace containing all events related to Bob. In this example, one similarity perspective is used: the subject of the contact. In this example, a profile vector is built for the first and the last event of Bob using the selected perspective. The values of these two profile vectors are shown in Figure 4.5.

The next step of the algorithm is to build an event similarity matrix that contains the similarity between all events of a single customer. A part of the resulting matrix is shown in Equation 4.2. Note that only the similarities for the first two and last two events are shown. The full event similarity matrix is shown in (C.1) in Appendix C. Every event is equal to itself. As a result, all values on the diagonal in (4.2) are equal to 1.

\[
\begin{pmatrix}
1.00 & 0.00 & \ldots & 1.00 & 1.00 \\
0.00 & 1.00 & \ldots & 0.00 & 0.00 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
1.00 & 0.00 & \ldots & 1.00 & 1.00 \\
1.00 & 0.00 & \ldots & 1.00 & 1.00 \\
\end{pmatrix}
\] (4.2)

Each cell in the matrix is filled by a creating two vectors based on the selected profiles and

<table>
<thead>
<tr>
<th>Trace</th>
<th>profile</th>
<th>[Payments]</th>
<th>[Available subscriptions]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>profile</td>
<td>[Payments]</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>profile</td>
<td>[Available subscriptions]</td>
<td></td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>profile</td>
<td>[Payments]</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.5: Profiling Bob. The numbers in the trace correspond to the event numbers in Table 1.1. For the first two and the last event, the resulting profiles are shown. The profiles are obtained by projecting all events on the subject perspective.

\[
\begin{pmatrix}
0.17 & 0.00 & \ldots & 0.17 & 0.17 \\
0.00 & 1.00 & \ldots & 0.00 & 0.00 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0.17 & 0.00 & \ldots & 0.17 & 0.17 \\
0.17 & 0.00 & \ldots & 0.17 & 0.17 \\
\end{pmatrix} \Rightarrow \begin{pmatrix}
< 3 >  \\
< 4, 5, 7, 8, 11, 12 >  \\
< 2 >  \\
< 10 >  \\
< 1, 6, 9, 13, 14 >  \\
\end{pmatrix}
\]

Figure 4.6: Creating the journeys of Bob. The modified similarity matrix is interpreted and for each cluster a trace, each representing a single customer journey, is created. Note that the numbers on the right hand side refer to the event numbers in Table 1.1.
computing the cosine similarity on these two vectors. To illustrate how Algorithm 5 computes
the similarity between two events, the first and last event of Bob are used. In the first and last
event, Bob checks his monthly payments. As a result, the two events are equal for the subject
perspective. (4.3) shows the two vectors obtained by Algorithm 5.

\[ v_1 = \begin{pmatrix} 1 \\ \hline \end{pmatrix}, \quad v_{15} = \begin{pmatrix} 1 \\ \hline \end{pmatrix} \]  

Equation (4.4) shows the calculation for the similarity between the first and last event.

\[
\text{cosinesimilarity}(\text{event}_1, \text{event}_{15}) = \frac{v_1 \cdot v_{15}}{|v_1|||v_{15}||} = \frac{\sum_{i=1}^{1} v_1_i v_{15_i}}{\sqrt{\sum_{i=1}^{1} v_1_i^2 \sqrt{\sum_{i=1}^{1} v_{15_i}^2}}} = \frac{1 \cdot 1}{1 \cdot 1} = 1
\]

The next part of Algorithm 5 is to compute the actual similarity score using cosine similarity.

The last step is to obtain traces, each containing a single customer journey, from the modified
event similarity matrix shown in (4.5). This is done by executing Algorithm 7 and 8. Figure 4.6
illustrates this last step and shows the resulting customer journeys of Bob.

4.4 Non-binary matching

In the previous section, the algorithm compares events on multiple perspectives. It may be the case
that for a perspective two events have a different value, but should still be considered somewhat
similar, i.e. similarity might not always be binary.

In the running example (Section 1.4), despite that events 2 and 3 have different subjects,
they are related. When using the same perspective as in Section 4.3 (subject), events 2 and 3
should be in the same customer journey. However, the algorithm splits these two events into
separate journeys. Hence, an extension to the algorithm is needed in which the user can learn the
algorithm for certain perspectives which values are not exactly equal to each-other, but should
still be considered to be related, i.e. domain knowledge can be used as optional input.

This problem is solved by introducing the option for using a similarity matrix for a single
perspective. In line 8 of Algorithm 5, the selected perspective computes the similarity between
two values. Instead of using an exact match, a matrix is used for checking the similarity. Table 4.1
shows an example similarity matrix for the subject perspective. Each cell in the matrix is a
number between 0 and 1, where 0 indicates that the two values are not similar and 1 indicates
that the two are similar. Note that the entry “Available subscriptions”, “Change subscription”
is 0.9 while “Change subscription”, “Available subscriptions” is 0.1. This means that it is more
likely that “Available subscriptions” is followed by “Change subscription”, i.e. a customer contacts
MyCarService for the subject “Available subscriptions” and the next contact is for the subject
“Change subscription”, than the other way around.
CHAPTER 4. SOLUTION

Table 4.1: Example similarity matrix for the subject perspective. \textbf{pm}: Payments; \textbf{as}: Available subscriptions; \textbf{cs}: Change subscription; \textbf{cr}: Car repair; \textbf{da}: Damages abroad

\begin{tabular}{|c|c|c|c|c|}
\hline
 & \textbf{pm} & \textbf{as} & \textbf{cs} & \textbf{cr} & \textbf{da} \\
\hline
\textbf{pm} & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
\textbf{as} & 0.0 & 1.0 & 0.9 & 0.0 & 0.0 \\
\textbf{cs} & 0.0 & 0.1 & 1.0 & 0.0 & 0.0 \\
\textbf{cr} & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\
\textbf{da} & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\
\hline
\end{tabular}

Table 4.2: Event Clustering obtained for Bob using subject perspective. For the subject perspective, the matrix in Table 4.1 is used.

<table>
<thead>
<tr>
<th>Cluster number</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,6,9,13,14,15</td>
</tr>
<tr>
<td>2</td>
<td>4,5,7,8,11,12</td>
</tr>
<tr>
<td>3</td>
<td>2,3</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

Using the matrix shown in Table 4.1, an event similarity matrix is obtained. Next, using Algorithm 6 (Section 4.2.3) clusters are formed in this matrix. From the modified event similarity matrix, clusters are identified using Algorithm 7 (Section 4.2.4).

The event similarity matrix is shown in (C.3) in Appendix C. When comparing (C.3) and the original event similarity matrix in (C.1), the similarity matrix for Bob has changed due to the matrix given in (4.1). As a result, the resulting clustering, obtained by using MCL, is also different. The resulting matrix containing the clusters is shown in (C.4) in Appendix C.

The difference between the clustering in Table 4.2 and Figure 4.6 is that the third and fourth cluster in Figure 4.6 are merged into a single cluster and hence, the journey of Bob changing his subscription is found.

4.5 Periodic behavior recognition

Temporally recursive behavior can be considered as a single customer journey. The approach described in Section 4.2.3 separates such behavior. Hence an extension to the current approach is needed that captures such temporal behavior.

In the running example (Section 1.4) the behavior of Bob is shown. A close inspection on the behavior of Bob (Figure 1.3 and Table 1.1), shows that Bob inspects his payments at the end of each month. Hence, there is a single temporal pattern. As this is considered to be a customer journey, an extension to the algorithm in Section 4.2 is needed. The remainder of this section discusses this extension for discovering periodic behavior, also called patterns.

4.5.1 Overview

To find temporal patterns, groups of events needs to be found that all have a distance of $\Delta x$ days between them. In the example of Bob (Section 1.4), at the end of each month the payments are

![Figure 4.7: Temporal pattern recognition approach. The flow starts with getting the events of the customer, and building time matrix $m$. Using $m$, temporal patterns are obtained and returned.](image)

Customer journey identification through temporal patterns and Markov clustering 33
checked. Hence, $\Delta x$ would be $\sim 30$ days in this case. However, it can also be the case that Bob visited the online portal of MyCarService twice on 30-07-2016 for checking the payments. In this case, both events are considered to be in the pattern. An overview of the solution for discovering patterns is shown in Figure 4.7.

The algorithm for identifying such patterns in time consists of two steps: building a time distance matrix for the customer and using this matrix to discover the patterns. Section 4.5.2 shows how such a time distance matrix is build and Section 4.5.3 shows how the temporal patterns are obtained from this time distance matrix.

4.5.2 Build time distance matrix

The first step for identifying patterns is to build the time distance matrix $m$. This square matrix has the size of the number of events. Each entry $m(i,j)$ represents the absolute distance in time between the $i$th and $j$th event of the customer. Because the cells contain the absolute values in time, only the upper triangle of the matrix is needed. Algorithm 9 shows the pseudo code for obtaining this matrix.

Algorithm 9: Creating a time distance matrix

This technique is also applied on all events of Bob. Equation 4.6 shows the calculation for the distance in time between events 1 and 2 of Bob. The resulting time distance matrix is shown in (4.7). The full matrix is shown in (C.5) in Appendix C.

$$
timeDistance(event_1, event_2) = |time(event_1) - time(event_2)|
= |(30 - 03 - 2016) - (27 - 04 - 2016)|
= |-27| = 27 \text{days}
$$

\begin{pmatrix}
0.00 & 27.00 & \ldots & 122.00 & 122.00 \\
0.00 & 0.00 & \ldots & 95.00 & 95.00 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0.00 & 0.00 & \ldots & 0.00 & 0.00 \\
0.00 & 0.00 & \ldots & 0.00 & 0.00 \\
\end{pmatrix}

(4.7)

4.5.3 Periodic behavior mining

The result of the first step is a time distance matrix where each cell holds the absolute distance in time, between the two corresponding events. By iterating over the cells of this matrix, periodic behavior can be identified. Algorithm 10 shows how periodic behavior is discovered.

The algorithm iterates over all the rows in the time distance matrix and takes event $e_1$ with index row as the starting event, i.e. all events are used once as starting point. In the next step,
Algorithm 10 Temporal pattern mining

1: procedure DiscoverPatterns(distanceMatrix) 
2:     patterns ← new list of patterns 
3:     for row ← 0 to Size(distanceMatrix) do 
4:         e₁ ← the event at position row of the customer 
5:     for column ← row + 1 to Size(distanceMatrix) do 
6:         e₂ ← the event at position column of the customer 
7:         if e₁ and e₂ do not match on all selected perspectives then 
8:             continue; 
9:         end if 
10:            pattern ← new Pattern with events e₁ and e₂ 
11:            ∆₁ ← distanceMatrix(row, column) 
12:     for column₂ ← column + 1 to Size(distanceMatrix) do 
13:         e₃ ← the event at position column₂ of the customer 
14:         if e₁ and e₃ do not match on all selected perspectives then 
15:             continue; 
16:         end if 
17:            ∆₂ ← distanceMatrix(row, column₂) 
18:         if ∆₂ mod ∆₁ is not within margin then 
19:             continue 
20:         end if 
21:             add e₃ to pattern 
22:     end for 
23:     if length of pattern ≥ 2 · ∆₁ then 
24:         add pattern to patterns 
25: end if 
26: end for 
27: end procedure 

In Equation 4.7 the time distance matrix containing the time-interval, in days, between all pairs of events is shown. When using Algorithm 10 with Equation 4.7 as input, two temporal patterns are discovered in the behavior of Bob. The first obtained pattern is the periodic behavior of checking payments at the end of the month. The second pattern contains the repair of the car at the end of April, May and June.

When using the clustering algorithm with non-binary matching on the remaining events, all journeys, both periodic and non-periodic, of Bob can be discovered.

To visualize the behavior of Bob, a process model is obtained using a log in which each journey in Table 4.3 is a trace. The resulting model is shown in Figure 4.8. This model describes the four different customer journeys of Bob. The journey on the top and bottom are the two journeys representing periodic behavior. As this behavior is repeated over time, a loop is visible in the process model. The two possible paths in the middle part of the model represent the two journeys

the algorithm iterates over the events with an index higher than row in the row and takes the event e₂. If there is a selected perspective for which e₁ and e₂ are not related, the algorithm continues with the next event. The absolute distance in time between e₁ and e₂ is taken as ∆₁ and new pattern p is created. For the remainder of the events in the row, it is checked if the distance between these events in time, ∆₂ is equal to k · ∆₁ including some margin ϵ, where k can be any non-negative integer. A margin is allowed because in some cases the time interval varies over time due, for example the number of days in a month. If ∆₂ is equal to k · ∆₁ ± ϵ, the third event is added to p. As soon as the algorithm finishes iterating over the remaining events, and the length of p is greater than 2 · ∆₁, i.e. the distance in time between the earliest event and the latest event in p is more than two times the time-interval ∆₁, p is added to the set of found patterns.
Table 4.3: The journeys of Bob obtained by identifying temporal patterns and clustering the remaining events. The numbers in the “Events” column are the event numbers as shown in Table 1.1.

<table>
<thead>
<tr>
<th>Journey number</th>
<th>Type</th>
<th>Events</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Periodic</td>
<td>1,6,9,13,14,15</td>
<td>Checking payments at the end of each month</td>
</tr>
<tr>
<td>2</td>
<td>Periodic</td>
<td>4,5,7,8,11,12</td>
<td>Car repair service at the end of each month</td>
</tr>
<tr>
<td>3</td>
<td>Cluster</td>
<td>2,3</td>
<td>Change subscription</td>
</tr>
<tr>
<td>4</td>
<td>Cluster</td>
<td>10</td>
<td>Damages abroad information</td>
</tr>
</tbody>
</table>

obtained by clustering the events that are not contained by any periodic behavior. As a result, this process model does indeed describe the four customer journeys as described in Section 1.4.

The process model in Figure 1.4 had some relations between activities, while the activities were not related. For example, the sub-process for changing the subscription is always followed by a car repair event. In Figure 4.8, these incorrect relations are not shown. In terms of customer journeys, the technique is indeed able to obtain a better process model.

Figure 4.8: Four journeys of Bob. The four different paths (excluding loops) represent the four different journeys of Bob. Two out of the four journeys represent the two obtained temporal patterns. Hence, the loop in those two journeys.

4.6 Solution overview

In Section 4.1 the approach to solving the problem of customer journey identification is discussed. Section 4.2 discussed the solution of identifying the customer journeys for a single customer. At last, Section 4.4 and 4.5 discussed some drawbacks of this algorithm and some improvements. This section shows a quick overview of steps of the full algorithm.

Figure 4.9 shows a high-level overview of the algorithm. In this overview the steps used for mining temporal patterns and clustering the events are abstracted to a single high level step. Figure 4.10 gives the same overview as Figure 4.9 without the abstraction. The algorithm starts with retrieving data from all the customers. As discussed in Section 4.1, the algorithm identifies the customer journeys for each customer separately and joins all the results in the end. Based on the settings set by the user, periodic behavior is discovered and/or clustering is applied.
As explained before, the user has the ability to choose whether the algorithm should search for temporal patterns, cluster all events or do both steps. If the user opted to use patterns, a time distance matrix (Section 4.5.2) is built. Using this matrix, periodic behavior is discovered (Section 4.5.3). Some events of the customer are covered by such behavior and some events are not. The next step of the algorithm is to separate the events that are not covered by any temporal pattern. If the user opted to use clustering and patterns, these remaining events are clustered. If the user did not opt to search for periodic behavior, all events of the customer are clustered. After the clustering, the results of the periodic behavior and the clustering result are merged, if applicable, and the algorithm continues to the next customer.

Figure 4.9: Schematic overview of the algorithm
CHAPTER 4. SOLUTION

Figure 4.10: A detailed overview of the algorithm described in this chapter.
Chapter 5

Implementation

Chapter 4 discussed the solution for the problem described in Chapter 3. This chapter describes the implementation of this solution. Section 5.1 discusses the main application decision taken before the start of the implementation. In Section 5.2 the structure of the implementation is discussed. There are three important parts in this structure: the plug-ins (Section 5.3), the algorithms (Section 5.4) and the visualization (Section 5.5).

5.1 Implementation decision

This section discusses the main decision that has been made for the implementation of the solution that is described in Chapter 4. The first decision that has to be made was which programming language is used. The only, to the author known, available library for event logs is the OpenXES library, which is written in Java. Furthermore, ProM is written in Java as well. Another advantage of Java is that it is able run on most operating systems. The remainder of this section discusses ProM in more detail.

ProM

Process Mining framework, or in short: ProM, is an open-source framework for process mining. ProM is a plug-in based framework. Meaning that every researcher in the world, that has a new algorithm or technique, can implement this as a ProM plug-in and add it to ProM. As a result, ProM contains always the newest process mining algorithms and techniques. Another advantage of being plug-in based is that it is easy to use. A user installs the plug-ins he or she needs, and uses these plug-ins. For most of the plug-ins, an event log is needed as input. ProM can handle

![Log visualizer example](image1)

![Dotted Chart example](image2)

Figure 5.1: Two ProM plug-ins that can be used to visualize an event log. Figure 5.1a shows a screen of the log inspector plug-in. With this plugin, one can obtain a quick summary of the event log. Figure 5.1b shows an example of a Dotted Chart, which can be used for identifying some global patterns in time for example.
both the MXML [17] and the eXtensible Event Stream (XES) [46] formats. ProM has also built-in functionality to investigate and explore event logs. An example of such a functionality is the log inspector, in which one can inspect the log in full detail, but also a quick summary of the log can be obtained.

Figure 5.1 shows two example plug-ins that are available in ProM. Figure 5.1a shows the Log Visualizer of ProM. With this plug-in one can inspect an event log on a very high level, but also on a low level per trace and view each event separately. A second example of a plug-in is shown in Figure 5.1b, which shows a Dotted Chart. The horizontal axis of this plot is the time and the customers are shown on the vertical axis. Each dot in the chart represents an event that has happened for a certain customer on a certain time. In this case an event is a moment on which the customer contacted a company and the color of the dot indicates the contact channel that was used when the customer contacted the company. The Dotted Chart can be used to inspect whether there are some global patterns that happen in time. For example, Figure 5.1b shows that in the middle, there are more events at the same time than on other times.

5.2 Application structure

The solution of Chapter 4 is implemented as a plug-in for ProM. This section gives a quick overview on the structure of the plug-in. Figure 5.2 shows an overview of the top-level packages. Highlighted are the three packages that are important: algorithms, plugins and visualization.

In the helpers package, a number of classes are defined that contain some functionality that is needed in the other packages. The main difference with the classes in the utils package is that the classes in the utils package need to be instantiated and have a state. While the classes in the helpers package do not have a state. The three packages marked in red are more important and the remainder of this chapter discusses these three packages. The plugins package is described in Section 5.3. The algorithms package is discussed in Section 5.4. At last, the visualization package is described in Section 5.5.

![Figure 5.2: Packages used in the implementation. Highlighted in red are the three most important packages.](image)

5.3 Plug-ins package

The first important part of the implementation is covered by the “plugins” package. This package contains the definitions for the two plug-ins used: the “TraceFinder” plug-in and the “Visualize Trace Finder Result” plug-in. In these definitions, the input and outputs of the plug-ins are defined. For the “TraceFinder” plug-in, there are two variants: both require an event log as input and the output is a “TraceFinderResult” object, which is used as input for the “Visualize Trace Finder Result” plug-in. One variant of the “TraceFinder” plug-in, requires a parameters object as input. The other variant starts with the default parameters and shows a settings wizard to the user.
5.3.1 Settings

If the parameters for the algorithm are not given, the plug-in shows a settings dialog to the user. The first setting is the selection of the algorithm to be used. There are two algorithms available to the user: “TraceFinder” and “TraceFinder++”. The first algorithm uses clustering for the identification of customer journeys, as described in Section 4.2. The second algorithm extends the first algorithm by identifying the temporal patterns of the customers first, as is described in Section 4.5. If desired, the algorithm has the ability to cluster all the remaining events into journeys. The settings dialog is shown in Figure B.2.

Temporal pattern settings

Depending on the chosen algorithm, the next setting screen is shown. In this setting screen the user can select perspectives that should be used by the algorithm for identifying temporal patterns (Figure B.3a). If a perspective is selected, the algorithm for mining periodic behavior (Section 4.5.3) uses this perspective when determining if two events can be in the same pattern. Section 4.4 discussed an extension to the algorithm using similarity matrices for non-binary matching. For each of the perspectives shown in Figure B.4a, the user has the ability to enter the path to such a similarity matrix stored in a CSV file. If the user opted to use the “TraceFinder” algorithm and hence, not discovering periodic behavior, the option for selecting perspectives is not shown.

Some customers have a high activity in the online environment in a small timeframe. As a result, the algorithm for identifying temporal patterns can find some patterns with a small time-interval (\(\leq 1\) hour). In Algorithm 10 this time-interval is denoted as \(\Delta_1\). To discard these results, a setting for the minimal value for the time-interval is introduced (Figure B.3b).

Cluster settings

After the settings for identifying temporal patterns, settings for the clustering of events are shown. Similar as to the perspective selection for identifying temporal patterns, the user can select the perspectives that should be used for clustering and optionally enter paths to similarity matrices stored on the hard drive. Figure B.4a shows the screen for selecting the perspectives.

Figure B.4b shows the second settings screen for the clustering of events. This screen contains some settings that are specifically for the MCL algorithm. For more information on the expansion and inflation parameters, the reader is referred to Section 4.2.3 and [25]. It is possible to not select any perspective. As a result, there is no perspective to cluster on and hence, the clustering algorithm will not be executed.

Other settings

After the settings for the clustering, one last setting screen is shown to the user. On this screen, there are three different settings. The first setting is for determining whether two events are equal in time or not. The user enters the maximal value, in days, that two events can be apart while still being considered to be close in time.

The second setting shown is for specifying the threadpool size. The algorithm is executed for each customer individually. By increasing the threadpool size, more customers can be handled in parallel. As a result, increasing the threadpool size may decrease the time needed until the algorithm completes.

The last setting in Figure B.5 is the similarity threshold. This similarity threshold indicates which the similarity between two items in the similarity matrix at least should be to be considered as similar. As stated before, the user has the ability to use similarity matrices and such a matrix contains doubles representing the similarity (Section 4.4).
5.4 Algorithms package

The second important package in the implementation is the “algorithms” package. The solution discussed in Chapter 4 is implemented in this package. The most important class in the algorithms package is the “TraceFinderAlgorithm” class. This is used by the “TraceFinder” plug-in and contains only one method: apply. The apply method has three parameters: the plugin context, the input log and parameters.

The apply method retrieves the algorithm, selected by the user and executes this algorithm for every trace in the event log. As soon as all customers have been processed, all the different sub-results are merged together. Figure 5.3 shows an overview of the different algorithm related packages and classes, which the remainder of this section will discuss.

5.4.1 TraceSplitter package

Section 5.3.1 discussed the ability to choose between two algorithms. The package “tracesplitter” contains these two algorithms. The “TraceFinder” algorithm is implemented in “TraceSplitterBasic.java” and “TraceFinder++” is implemented in “TraceSplitterBasicPlus.java”. Both these java classes implement the “TraceSplitterAlgorithm” interface. As the main algorithm uses this interface, any algorithm can be used for identifying customer journeys. As a result, it is possible to easily extend this plug-in with new algorithms. “TraceSplitterAlgorithms.java” contains an enumeration, containing all available algorithms (“TraceFinder” and “TraceFinder++”). This enumeration is used by the settings of the plug-in (Section 5.3.1). If a new algorithm is added that implements the “TraceSplitterAlgorithm” interface, it can also be added to this enumeration. As a result, the new algorithm is directly available to the user in the settings screen.
5.4.2 Patterns package

Chapter 4 discusses also two extensions to the basic algorithm: non-binary matching (Section 4.4) and periodic behavior (Section 4.5). The periodic behavior extension is implemented in the “patterns” package. Similar as the package “tracesplitter”, an interface is used. This enabled an easy addition of additional algorithms for identifying patterns. The solution for identifying periodic behavior (Section 4.5) is implemented in “TimePatternFinder.java”.

5.4.3 Merging customer results

The algorithms in this thesis result into a solution per customer. In order to obtain useful results, a merging step is needed. Algorithm 11 shows how the results for all customers are merged.

Algorithm 11 has one parameter: a list of the results obtained by the algorithms described in Section 5.4. Initially an empty list of clusters and an empty list of remaining traces is created. Next, the algorithm iterates over all the customers that are processed by the algorithms in Section 5.4. For each customer it is checked if clusters were obtained. Note that results found by the temporal pattern extension (Section 5.4.2) are also stored in a cluster object. For each cluster of the current customer it is checked if clusteringList already contains a clustering of events that is similar to the current cluster. If this is the case, the cluster is added to the clustering. If there is not a single clustering which contains a similar cluster, a new clustering is created with the current cluster and is added to the clusteringList. The second step for each customer is to check if there are some events that are not contained in any cluster. If this is the case, these events are stored in a trace and the trace is added to the list of remaining traces. As soon as all customers have been processed, the clusteringList and the list of remaining traces are added to a new “TraceFinderResult” object, which is returned.

Algorithm 11 Merging all results obtained by the algorithms in Section 5.4

1: function MergeCustomerResults(customerResults)
2:   clusteringList ← new list of clusterings
3:   remainingList ← new list of traces
4:   for all customer in customerResults do
5:     if clusters found for customer then
6:       for all cluster of the customer do
7:         AddClusterToResult(clusteringList, cluster)
8:       end for
9:     end if
10:    if customer has remaining events then
11:      trace ← new trace with the customer’s attributes
12:      for all event in the remaining events of the customer do
13:       add event to trace
14:      end for
15:      add trace to remainingList
16:    end if
17:  end for
18: return new TraceFinderResult containing clusteringList and remainingList
19: end function
Algorithm 12 Adding a cluster to a clustering

1: procedure AddClusterToResult(clusteringList, cluster)
2:     for all clustering in clusteringList do
3:         if clustering.ContainsSimilarCluster(cluster) then
4:             add cluster to clustering
5:         end if
6:     end for
7:     newClustering ← new EventClustering
8:     add cluster to newClustering
9:     add newClustering to clusteringList
10: end procedure

5.5 Visualization package

The result of the plug-in containing the algorithms is an object of the type “TraceFinderResult”. To be able to use this result, a second plug-in is created: “Visualize Trace Finder Result”. This plug-in is implemented in the “visualization” package. This section discusses this package and the resulting visualization. Figure 5.4 shows the screen that is shown after the user has applied the “TraceFinder” plug-in.

The view shown in Figure 5.4 contains three parts: the left bar, the bottom bar and the visualization view. In the left bar, the user can filter the results shown in the visualizer. The user has the ability to filter the results based on the minimal and maximal trace length. Another filter option is the time-interval. The user can enter multiple values that should be kept. A row in the visual component can contain multiple traces, each with their own time-interval. If a row contains a trace with one of the intervals entered by the user, the row is kept in the visualization. For each perspective chosen by the user, except for the “time:timestamp” perspective, a filter option is also added. Similar to the time-interval filter, the user can enter multiple values that should

Figure 5.4: “Trace Finder Visualizer plug-in”. On the left the filter options and some statistics are shown. On the bottom, different export options are opted to the user. The remaining of the screen contains the visualization of the results and consists of two parts. The first part is a table showing all the results. The second part visualizes the results selected in the table. Depending on the selected tab, a Dotted Chart is used for the visualization or the Inductive Visual Miner. Appendix B Figure B.6 shows this figure in a larger format.
be kept. Furthermore, some statistics about the number of results, the minimal, maximal and average trace length are shown.

The bottom bar in the visualization contains four buttons for exporting the results. With this the user has the ability to export an event log containing all the results that are found. The user has also the ability to export an event log containing only the results that are selected by the user. On the right of this bar, there are two buttons for exporting event logs containing all the events that are not contained in any result, but were in the original dataset. Note, that if clustering is used, all events are contained in at least one result. For the exporting the user has two options: exporting all remaining events and exporting all events that are not contained in any of the selected results.

The last part of the view is the visualization of the results. In the top of the visualization, the obtained results are shown in a table. The first columns of the table depend on the perspectives selected by the user. After all the columns to show the different values for the different perspectives, a column for the time interval is added. If a row is the result of periodic behavior, this cell is filled with the lowest and highest time-interval. If a row is the result of clustering, the value for the time-interval is equal to -1. In the lower part of the visualization, the selected rows are visualized using the Dotted Chart\(^1\) and the Inductive Visual Miner\(^2\). Using the different tabs, the user can switch between the two visualizations.

5.5.1 Code

The structure of the visualization package is shown in Figure 5.5. The most interesting part of this package are the package “panels.externals” and the “ModelConverter” class.

In the class “ModelConverter” the result of applying the algorithm (Section 5.4.3) is converted to a model that can be used by the visualization plug-in. Algorithm 13 shows this conversion. The result of this conversions is an object of the class “TableModel” which extends “AbstractTableModel”. As a result, any Java table can use this model and visualize the result.

After the conversion to the data-model that can be used by the visualization, the visualization

\(^1\)See https://svn.win.tue.nl/trac/prom/browser/Packages/LogProjection

\(^2\)See https://svn.win.tue.nl/trac/prom/browser/Packages/InductiveVisualMiner

---

Figure 5.5: Illustration of the visualization package

Customer journey identification through temporal patterns and Markov clustering 45
is built. As stated before, the visualization contains three parts: the left bar, the bottom bar and the visualization view. Each of these three views are built in a separate panel and the panels are added to the main panel of the visualization. The left panel is built on top of three other panels: the filter panel, the panel for the visual settings and the statistics panel. These three panels and the bottom bar panel and the visualization panel can be found in the “panels” package.

The visualization panel contains two parts: the table and the visualizations using the Dotted Chart and the Inductive Visual Miner. The table in the upper part is a standard JTable from the Java Software Development Kit (SDK), which can interpret and visualize the table model obtained by Algorithm 13. The lower part of the visualization is another panel called “VisualPanel”. This panel contains both the visualization for the Dotted Chart as the visualization for the Inductive Visual Miner in a tabbed pane. Both these visualizations are defined in the package “external” inside the “panels” package. These panels are created using the corresponding create function in the “VisualizeLogPaneFactory” class. By adding a panel to this tabbed pane and a method in this factory, a new plug-in that visualizes an event log can be added as a new tab. As a result, this visualization is easily extendable.

Algorithm 13 Converting “TraceFinderResult” to a model usable for visualization

```
1: function ConvertData(data)
2:    rowModelList ← new list of row models
3:    if data.ContainsClusters then
4:        for all clustering in data do
5:            minLength ← minimal trace length in clustering
6:            maxLength ← maximal trace length in clustering
7:            traces ← new list of traces
8:        for all cluster in clustering do
9:            trace ← new Trace
10:           for all event in cluster do
11:              add event to trace
12:          end for
13:          add trace to traces
14:        end for
15:        rowModel ← new RowModel containing traces, minLength, maxLength
16:        add rowModel to rowModelList
17:    end if
18: end function
```
Chapter 6

Case study

This chapter discusses a case study, which is performed on data from CZ. Customers can contact CZ via several contact channels: e-mail, call center, visiting one of the service desks throughout the country, and using the online environment, called “Mijn CZ”. CZ supplied a dataset that contains incoming contacts from customers using either one of these four channels. This dataset contains \( \sim 5 \) million contacts from \( \sim 78 \) thousand customers. Each contact has a few attributes: a randomized customer id, a subject, and a (sub-) subject. The subjects and (sub-) subjects are derived in a different way depending on the contact channel. For the contacts in the online environment, the subjects and (sub-) subjects are derived from the URL of the visited pages. For the other three contact channels, the subjects and (sub-) subjects were derived using text mining techniques. Due to this randomization and this classification, the dataset is completely anonymous and does not contain any personal details.

The reason for the large amount of contacts is the online environment. Each time a customer visits a page in the online environment, this visit is recorded in the log with a subject and (sub-) subject, which are derived from the URL of the visited page. A problem with the records from the online environment is that the subjects do not match with the subjects used in the other three contact channels. Due to the difference in the granularity of the logging and the non-overlapping set of subjects, a pre-processing step is needed.

The remainder of this chapter is organized as follows: Section 6.1 discusses the pre-processing of the data. To show that our technique is indeed able to distinguish the different customer journeys, a single customer is taken from the dataset and the technique is applied on this single customer. Section 6.2 discusses the application and results of this. The technique discussed in Chapter 4 is used to analyze a subset, related to the dental care, of the given dataset. This is discussed in Section 6.3. At last, Section 6.4 concludes this chapter.

6.1 Data pre-processing

As is stated in the introduction of this chapter, the log from CZ needs some pre-processing before our technique can be applied. This section gives a short overview of the steps done during the pre-processing. Appendix A describes the full steps taken during the pre-processing, including the PROC SQL queries that are executed in each step.

The dataset from CZ was stored in a SAS dataset. As the processing could easily be done with SQL-like queries, the choice was made to do the pre-processing in SAS Enterprise Guide. The main idea of this pre-processing is that the data is loaded in SAS Enterprise Guide, a series of queries are executed on the dataset and a CSV file is exported. The pre-processing consists of four distinct steps: formatting the data, ensuring subject equality between the various contact channels, adding missing data and converting the dataset to an event log. Figure 6.1 illustrates the steps done during the pre-processing.
CHAPTER 6. CASE STUDY

6.1.1 Data formatting

The first step is to format the data given in the CZ dataset. Columns are renamed to a consistent naming and string values in the data are trimmed. When exporting the dataset directly from SAS Enterprise Guide to a CSV file, the ids of the customers are exported as a number and therefore, rounded. To avoid this problem, the ids are stored as a string instead of a number. SAS Enterprise Guide uses an entirely different formats for dates and times than process mining tools such as ProM and Disco. Hence, this step also converts these date-time values to a format supported by process mining tools.

6.1.2 Subject classification

The subject and (sub-) subjects of the events in “Mijn CZ” are classified in a different way than the subject and (sub-) subjects of all other events. As a result, an event from the online environment can never be in the same journey as an event from, for example, the service desk. The second step of the pre-processing changes the subjects and (sub-) subjects of the events regarding the online environment. This is done by creating a mapping between the subjects and (sub-) subjects from the online environment to the subjects and (sub-) subjects from the other three contact channels. Such a mapping can be found in Table A.1. If the subject and (sub-) subject of a contact in the online environment are contained in this mapping, the contact is kept and its subject and (sub-) subject are changed. If the combination of the subject and (sub-) subject is not in the mapping, the contact is discarded. After all contacts in the online environment have been handled, the remaining contacts are merged again with the events from the other three contact channels.

6.1.3 Missing data

For some events (~ 30%), not all attributes are filled, i.e. for some attributes the values are unknown. The third pre-processing step adds values for the missing attributes. For example, a contact in the online environment does not have a value for the department attribute. Hence, this value is set to “Web” in this case.

6.1.4 Conversion to an event log

In the last step, the data is exported from SAS Enterprise Guide to a CSV file. Process mining tools have the ability to import CSV files. However, users need to specify which attributes need to be considered as trace attributes and which as event attributes. Section A.2 discusses a simple ProM plug-in called “CZ Importer” that has this knowledge for the CZ data and is able to convert a CSV file to a XES log without user input.

Figure 6.1: Schematic overview of the data pre-processing. First the data is loaded and formatted in such a way that process mining tools can use it. Next, equal subject classification is ensured between the various contact channels and missing data is added to the set. Finally, the dataset is converted to an event log such that it can be used by process mining tools such as ProM. The first three processing steps are done in SAS Enterprise Guide and are described in Section A.1. The conversion to an event log is done using a custom made plug-in for ProM (Section A.2).
6.2 Single customer

Chapter 4 discusses the theoretical solution and shows that the technique works using Bob from the running example. As stated in Chapter 5, the technique described in Chapter 4 is implemented as a plug-in for ProM. The result of the pre-processing described in Section 6.1 is an event log. To demonstrate that our technique is applicable on real-life datasets and returns valuable results, a single customer from this event log is used.

6.2.1 Input model

Using the Inductive Miner plug-in in ProM, a process model is obtained for this single customer. Figure 6.2 shows this process model. As all events of this single customer are in a single trace, a single flow is discovered. Due to the fact that a trace captures all events instead of a single journey, some activities are related in the process model, but should not be related. For example, “Wijzigingen doorgeven/Polis” is followed by “Algemeen/Eigen risico”, despite that the time between these two events was more than two months. When considering customer journeys, these two activities do not have to be related.

Figure 6.2: Part of the process model for the test customer. The model is too large to show here. Hence, the middle part is replaced with dots. The full model is shown in Appendix B, Figure B.8. All the events of the customer are captured in a single trace. As a result, the model shows relations between activities that should not be there. For example, “Wijzigingen doorgeven/Polis” is followed by “Algemeen/Eigen risico”, despite that the time between these two events was more than 2 months. Hence, these two activities should not be related.

6.2.2 Output model

Figure 6.2 shows the process model obtained after applying a process discovery algorithm directly on the event log of a single customer. To obtain the different customer journeys for this customer, our plug-in is used. The technique is applied with the “TraceFinder++” algorithm, with the “concept:name” attribute as perspective for obtaining temporal patterns with a similarity matrix. This attribute contains the subject and (sub-) subject. As a result, this matrix is too large to create manually. In Section 6.3.2, such a similarity matrix is created based on data. This similarity matrix is also used here. For the clustering of the remaining events, the time and “concept:name” attributes are used. For the “concept:name” attribute, the same similarity matrix is used as with the perspective for temporal patterns.

Table 6.1 shows the results obtained by the “TraceFinder” algorithm. Interesting are the results with id 5 and 6 with the subjects “Nota/Farmacie” and “Vergoedingen/Farmacie”. One might expect that these two events are related and should be in the same journey. Further inspection of the customer revealed that these two events are not related in time. The first event was at 02-09-2014 and the second at 06-05-2015. Hence, the algorithm did create two separate journeys for these two events. The events regarding the activity “Algemeen/Eigen risico” are split into two journeys, as there are three groups of events with a similar distance in time between them. The
Table 6.1: Journeys obtained for a single customer using both temporal patterns and clustering. The results regarding the activity “Algemeen/Eigen risico” are split into two journeys: one temporal pattern and a cluster containing the remaining events.

<table>
<thead>
<tr>
<th>Id</th>
<th>Subjects</th>
<th>Temporal pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aanvraag/Declaratieformulier,</td>
<td>✗</td>
</tr>
<tr>
<td></td>
<td>Nota/Alternatieve zorg,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Vergoedingen/Gezichtshulpmiddelen</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Aanvraag/Mondzorg</td>
<td>✗</td>
</tr>
<tr>
<td>3</td>
<td>Algemeen/Eigen risico</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>Algemeen/Eigen risico</td>
<td>✗</td>
</tr>
<tr>
<td>5</td>
<td>Algemeen/Ziekenhuis</td>
<td>✗</td>
</tr>
<tr>
<td>6</td>
<td>Nota/Alternatieve zorg</td>
<td>✗</td>
</tr>
<tr>
<td>7</td>
<td>Nota/Farmacie</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Vergoedingen/Farmacie</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Wijzigingen doorgeven/Polis,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Aanvraag/Pakketvergelijker</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2: Journeys obtained for a single customer using clustering techniques on events. The only difference with Table 6.1 is that there is no temporal pattern result, i.e. the row with id 3 from Table 6.1 is missing.

<table>
<thead>
<tr>
<th>Id</th>
<th>Subjects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aanvraag/Declaratieformulier,</td>
</tr>
<tr>
<td></td>
<td>Nota/Alternatieve zorg,</td>
</tr>
<tr>
<td></td>
<td>Vergoedingen/Gezichtshulpmiddelen</td>
</tr>
<tr>
<td>2</td>
<td>Aanvraag/Mondzorg</td>
</tr>
<tr>
<td>3</td>
<td>Algemeen/Eigen risico</td>
</tr>
<tr>
<td>4</td>
<td>Algemeen/Eigen risico</td>
</tr>
<tr>
<td>5</td>
<td>Algemeen/Ziekenhuis</td>
</tr>
<tr>
<td>6</td>
<td>Nota/Alternatieve zorg</td>
</tr>
<tr>
<td>7</td>
<td>Nota/Farmacie</td>
</tr>
<tr>
<td>8</td>
<td>Vergoedingen/Farmacie</td>
</tr>
<tr>
<td>9</td>
<td>Wijzigingen doorgeven/Polis,</td>
</tr>
<tr>
<td></td>
<td>Aanvraag/Pakketvergelijker</td>
</tr>
</tbody>
</table>

second result contains the three events with the same activity, but that could not be added to one of the three groups in the temporal pattern. However, one would expect that all these events are captured in a single journey.

Figure 6.3 shows the process model obtained for the journeys in Table 6.1. Due to the fact that both journey 1 and 9 only contains the activity “Algemeen/Eigen risico”, and in both journeys the activity is executed one or more times, the two journeys are drawn in a similar way in the process model. Hence, the model in Figure 6.3 covers all the distinct journeys.

As shown before, identifying temporal patterns does not give desired results in this case. Hence, only clustering is applied on this customer, i.e. the “TraceFinder” algorithm is used. The same settings are used as when using the “TraceFinder++” algorithm. Applying this algorithm results in 8 different customer journeys as shown in Table 6.2.

As expected, the results are similar as when finding temporal patterns. The “Algemeen/Eigen risico” events are split into a temporal pattern with some remaining events in the pattern. Hence, this result might be seen as a better result than when identifying patterns first.
Customer journey identification through temporal patterns and Markov clustering
Table 6.3: Top 5 subject & (sub-) subject combinations

<table>
<thead>
<tr>
<th>Subject</th>
<th>(sub-) subject</th>
<th>Occurrences</th>
<th>Occurrences (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vergoedingen</td>
<td>Overig</td>
<td>29,933</td>
<td>12.03</td>
</tr>
<tr>
<td>Nota</td>
<td>Overig</td>
<td>28,062</td>
<td>11.28</td>
</tr>
<tr>
<td>Algemeen</td>
<td>Overig</td>
<td>26,039</td>
<td>10.47</td>
</tr>
<tr>
<td>Machtinging</td>
<td>Overig</td>
<td>11,931</td>
<td>4.80</td>
</tr>
<tr>
<td>Vergoedingen</td>
<td>Mondzorg</td>
<td>11,212</td>
<td>4.51</td>
</tr>
</tbody>
</table>

6.3 Dental care

The result of the pre-processing steps described in Section 6.1 is an event log. To show the applicability of the developed technique, a subset of this event log has been created. This subset contains all the customers that contacted CZ at least once for the subject “Mondzorg” (dental care). The resulting event log contains ~17 thousand customers. These customers contacted CZ ~250 thousand times using either one of the four available contact channels. Figure 6.4 shows the distribution of the different lengths of the traces in the event log, i.e. the number of times a customer contacted CZ during a time period of ~1.5 years.

For each of the 150 thousand contacts, the subject and the (sub-) subject is recorded. Table 6.3 shows the top 5 combinations of subject and (sub-) subject that occur the most. The subjects in Table 6.3 are nicely categorized. However, the (sub-) subjects are not. Four out of five (sub-) subjects are “Overig” (other). As a result, it is unknown what the exact subject of the contact was. The subjects and (sub-) subjects are derived in two different ways depending on the contact channel. The events using call center, e-mail or the service desk are based on the logging of the CZ employee. Using text mining techniques these logs are classified to subjects and (sub-) subjects. The subjects and (sub-) subjects of events in “Mijn CZ” are derived from the URL of the visited page. As a result, events from the online environment did have entirely different subject and (sub-) subjects. During the pre-processing (Section 6.1) the subjects and (sub-) subjects are changed. However, the logging of the (sub-) subjects is not done, hence all (sub-) subjects are set to “Overig”. Figure 6.5 shows a Dotted Chart for this event log. There is an event in the dataset for each single page visit in the online environment. As a result, there are ~66 thousand events using the online environment, which explains the high number of events with the (sub-) subject “Overig”.

6.3.1 Subjects

Our plug-in is executed on this dataset. Using the “TraceFinder++” algorithm, temporal patterns are discovered in the event log. The subject perspective is used in combination with a similarity
matrix. The matrix is shown in Appendix D, Table D.1. Furthermore, the minimal delta in
days is set to 1 day. No perspectives are selected for clustering. Hence, only periodic behavior is
discovered. 180 results are discovered after applying the algorithm. The shortest trace length is 3
events and the maximum trace length is 111, with an average of 7 events per trace. Some of the
results only support a single customer. However, there is also some temporal pattern identified
that covers 983 customers.

The result covering 983 customers covers a time-interval range from 1 until 310 days and covers
the subjects “Algemeen”, “Nota” and “Vergoedingen”. This means that there are customers that
contacted CZ for either one, or a combination, of these subjects on a daily basis. However, there
are also customers contacting CZ with a time-interval of 310 days. To illustrate the behavior of
these 983 customers, an event log is created for these journeys and the Inductive Miner has been

![Dotted chart on dental care event log](Figure B.7)

The color is based on the contact channel used for the contact. The online environment is used a lot, which explains the high number of events with the (sub-) subject “Overig”.

![Process models obtained for periodic behavior](a)

(a) Process model obtained for periodic behavior

![Process models obtained for periodic behavior](b)

(b) Process model obtained for periodic behavior

Figure 6.5: Dotted chart on dental care event log (Figure B.7). The color is based on the contact channel used for the contact. The online environment is used a lot, which explains the high number of events with the (sub-) subject “Overig”.

Customer journey identification through temporal patterns and Markov clustering
used to obtain a process model. Figure 6.6a shows the Petri net obtained when using the subject attribute as event classifier. This model shows clearly the behavior of the customer and has a high precision when comparing it to the event log (0.99376). However, this model does not distinguish between the different (sub-)subjects of a single subject. To obtain more detail in the behavior of these customers, the (sub-)subjects are also included in the event name classifier. Similar as to the previous process model, a Petri net is obtained using the Inductive Miner. The resulting model is shown in Figure 6.6b.

The model shown in Figure 6.6b is more complex than when only using the subjects as event name classifiers. A close inspection of the model shows that this model is closely related to a “flower” model. The only two activities that violate the behavior of a “flower” model are the first and last activity. As a result, the precision score is low (0.19603) and the generalization score is high (0.99979).

The technique is able to obtain some temporal patterns for customers in this event log. Since only the subject attribute is used as a perspective, it might be the case the case that two events with the subject “Vergoedingen” are not related at all. When using the (sub-)subject attribute as well for the discovery of a process model, a “flower” like model is obtained (Figure 6.6b).

6.3.2 Dental care related events

As shown in the previous section, using only the subject attribute of events does not result into good process models and insights. In order to obtain useful insights in the CZ data, a smaller subset is created with events related to dental care. The remainder of this section discusses the filtering of these events, the creation of a similarity matrix. At last, an analysis is done using the obtained subset and similarity matrix.

Filtering

To obtain better results, a new subset is created. This subset contains all events that are related to dental care and events that are near a dental care related event. An event is related to dental care if its subject is “Mondzorg”. The range for this allowance is set to maximum 14 days before a dental care related event and a maximum of 14 after a dental care related event. Using this restriction allows to do a more detailed analysis on the behavior of customers in the dental care health area. Figure 6.7 shows an example of 5 events for this filtering.

Table 6.3 shows that most (sub-)subjects are “Overig”, i.e. they could not be classified to a (sub-) subject. As a result, these events can be related to all events having the same subject. Hence, it is hard to create separate journeys for all the different health care areas as they are all connected via events with the (sub-) subject “Overig”. To be able to obtain separate journeys for the different health care areas, all events with the (sub-) subject “Overig” are removed.

Figure 6.4 shows that a large percentage of the customer contacted CZ only once. In some cases, a customer has multiple events, but all events are on the exact same date and time. In

![Figure 6.7](image-url)

**Figure 6.7**: A visual representation of the filtering step to obtain a set contain dental care related events. The events with a ✓ in the lowest row remain after the filtering and the events with a ✗ will not. Events $e_2$ and $e_6$ are related to dental care. All other events are not. Hence these two events are kept after the filtering. Event $e_1$ is only 2 days before $e_2$, as a result it is kept in the new subset, despite that it is not related to dental care.
In this case a customer contacted CZ only once, but asked multiple questions on different subjects for example. These cases do not contain any journeys. The goal is to obtain customer journeys. Hence, all events of customers that only contacted CZ once are removed.

After all the filtering steps, the custom made ProM plug-in (Section A.2) is used to convert the set to an event log. This event log contains \(~ 14\) thousand customers and \(~ 90\) thousand events. Figure 6.8 shows how the events are distributed over all the traces.

**Similarity matrix creation**

In Section 6.3.1, the similarity matrix was created manually. In this case, the “concept:name” attribute of events will be used as a perspective. In the event log, this is a combination of the subject and the (sub-) subject. As a result, there are over 100 unique activities. When creating a similarity matrix, this would mean that over 10 thousand cells need to be filled. Hence, a technique based on data is needed to generate such a similarity matrix.

One way to build such a similarity matrix is to check whether the subject, (sub-) subject or both of two activities are equal and give a score based on this. This method has the advantage that it promotes the similarity between two different activities in the same health care area. However, this method fails to capture some important relations. For example, when a customer asks for a comparison between insurance packages, the next contact might be for changing his insurance. Both the subject and (sub-) subject would be different for these two activities. Hence, this technique would not allow this behavior in the similarity matrix.

Another option to build a similarity matrix is to use a variant of the market basket analysis. Instead of counting how often \(A\) and \(B\) occur together, counting how often \(A\) is followed by \(B\). The normalization step can be done in a similar way as with the normal market basket analysis: dividing the number of occurrences of \(A\) followed by \(B\) by the number of occurrences of \(A\). Equation 6.1 shows the calculation for the value in the matrix \(M\) for the activities \(act_1\) and \(act_2\).

\[
M(\text{act}_1, \text{act}_2) = \frac{\#(\text{act}_1 \rightarrow \text{act}_2)}{\#\text{act}_1} \tag{6.1}
\]

The advantage of this method is that it is not limited to the name of the activities. For example, if customers ask for a comparison between insurance packages, and their next contact is for changing their insurance, this is also captured in the similarity matrix. The disadvantage of this method is that it can promote some unwanted behavior as somehow two activities may follow each other while they should not be related. For example, after applying this calculation, a dental care event followed by a physiotherapy event has a higher score than followed by another dental care event. The second disadvantage of this method is that an activity \(act\) is never 100% similar to itself, i.e. \(M(\text{act}, \text{act}) \neq 1\).

Figure 6.8: Trace length distribution for the set containing events related to dental care and events close in time to a dental care event. Furthermore, each trace contains at least two events at two different dates or times. The x-axis shows the length and the y-axis shows the amount of traces.
Both the methods described above have their advantages and disadvantages. Hence, a technique that combines the previous two techniques is proposed. First, an initial matrix $M$ is created using Equation 6.1. In the next step, a second matrix $M'$ is obtained using $M$ as input. Based on whether the subject or (sub-) subject are equal for a pair of activities, the values in $M$ are changed according to Equation 6.2. Note that the subject of activity $act_1$ is denoted as $act_1[subject]$.

$$M'(act_1, act_2) = \begin{cases} M(act_1, act_2) & \text{if } act_1[subject] = act_2[subject] \text{ and } act_1[subsubject] = act_2[subsubject] \\ \frac{M(act_1, act_2)}{4} & \text{if } act_1[subject] \neq act_2[subject] \text{ and } act_1[subsubject] \neq act_2[subsubject] \\ M(act_1, act_2) & \text{otherwise} \end{cases} \tag{6.2}$$

Most of the values in $M'$ are low ($< 0.1$). Hence, all rows are scaled such that the maximum value in each row is 1 using Equation 6.3.

$$M''(act_1, act_2) = \frac{M'(act_1, act_2)}{\max_{act_i \in acts}(M'(act_i, act_2))} \forall act_i \in acts$$

Where $acts$ is the set of activities.

The last step for building the matrix is to set all values on the diagonal to 1, such that an activity is always equal to itself. The similarity between two activities is shown in Equation 6.4.

$$M'''(act_1, act_2) = \begin{cases} 1 & \text{if } act_1 \equiv act_2 \\ M''(act_1, act_2) & \text{otherwise} \end{cases} \tag{6.4}$$

When calculating $M'''$ using all possible pairs of activities, the similarity matrix is obtained. Despite the scaling done in Equation 6.3, most of the values are still low in the similarity matrix. Figure 6.9 shows the distribution of the values in the matrix $M'''$. Interesting is the small horizontal line indicating the values that are exactly one. This is caused by the scaling step Equation 6.3 and the step in which the diagonal is set to 1, Equation 6.4.

**Journey detection**

Using the filtering steps described above, an event log is obtained. This log contains $\sim 90$ thousand events generated by a total of $\sim 14$ thousand customers. Figure 6.8 shows how the length of the traces are distributed. When comparing this distribution to the one shown in Figure 6.4, it is noticeable that the shortest trace is longer and the longest trace is shorter. Using this subset, a similarity matrix for the “concept:name” attribute is obtained as described above. Using the

![Figure 6.9: Value distribution in the similarity matrix. Most values are 0 or close to 0. Only a small fraction of the values has a value larger than 0.1.](image-url)
“concept:name” attribute has the advantage that both the subject and (sub-) subject are covered. Using the “TraceFinder++” algorithm, the customer journeys are obtained. For the temporal patterns part, the “concept:name” attribute is used as perspective and \( \Delta_{\text{min}} = 1 \) day. All events that are not contained in a temporal pattern, are clustered using the “time:timestamp” and “concept:name” attributes as perspectives. For the clustering, the expansion parameter is set to 2 and the inflation parameter is set to 15. Similar as done with the filtering step, two events are considered to be close in time if there is at most 14 days between them. At last, a similarity threshold needs to be chosen. Figure 6.9 shows the distribution of the values in the obtained similarity matrix. Clearly, a relative low value (\(< 0.1\)) needs to be chosen as otherwise too many values would be excluded. However, if a too low value is chosen, too many values are included, which might have the result that the algorithm will put events in the same journey while they should be in different ones. Hence, a similarity threshold of 0.049 is chosen.

**Temporal patterns**

Applying the algorithm results into 4,815 unique identified journeys. Of these 4,815 unique journeys, 525 are obtained using the algorithm for finding temporal patterns. Unfortunately, these temporal patterns often cover 10 customers or less. In fact, most of them cover only a single customer. Only two temporal patterns cover more than 80 customers: one covers 86 customers on the activities “Nota/Mondzorg” and “Vergoedingen/Mondzorg” and one covering 116 customers with the activities “Betaling/BetalingsRegeling” and “Betaling/Premiebetaling”. In the filtering step, only dental care related events and events near a dental care related event are kept. As a result, events for a regular payment for example are removed if they are not near a dental care related event. Hence, it is not possible to discover such temporal patterns anymore.

Figure 6.10 shows the process model, obtained with the Inductive Miner, for the result on the activities “Nota/Mondzorg” and “Vergoedingen/Mondzorg”. The lowest time interval for such behavior is 6 days and the longest time interval is 293 days. Interestingly, there is no clear process or order in which the activities are executed. As a result, the process model shown in Figure 6.10 has the two activities modeled in parallel. A closer inspection of the Dotted Chart revealed that \( \sim 50\% \) of the customers did first the activity “Nota/Mondzorg” and the others did first the activity “Vergoedingen/Mondzorg”. Hence, it can be concluded that there is indeed not a sequence of activities that occurs the most and might have been an indication for a “fixed” process. As the behavior is periodic, both activities are modeled in a loop. Due to the loops and parallelism in the process model, any activity can be executed in any order. Hence, the process model is almost an unfolded “flower” model. The only difference with a “flower” model is that in this model all activities need to be executed at least once.

Another temporal pattern obtained with a high support on the number of customers contains the activities “Betaling/Betalingsregeling” and “Betaling/Premiebetaling”. Figure 6.11 shows a process model, also obtained using the Inductive Miner, and a Dotted Chart for all journeys covering this pattern. The time interval between groups of events in this pattern varies from 6 until 289 days. Similar as to the pattern containing dental care events, there is no clear sequence

![Figure 6.10: Temporal patterns on dental care related events. The Inductive Miner is used to create a process model for these journeys.](image-url)
of activities that is followed by these customers. \( \sim 40\% \) of the customers start with the activity “Betaling/Betalingsregeling”, while the other \( \sim 60\% \) start with “Betaling/Premiebetaling”. As a result, the two activities are modeled in parallel in the process model. Similar to the previous temporal pattern, both activities can be executed multiple times, which explains both loops around the activities.

**Clusters**

After all temporal patterns have been identified, all events that could not be added to a temporal pattern are clustered into customer journeys. Similar as to the temporal pattern results, most results have a low support (< 10) customers. However, there are still 83 unique journeys found with a support higher than 100 customers. However, most of results with the highest support contain only a single activity.

For 53 unique customers, the algorithm obtained a journey using the activities “Nota/Mondzorg”, “Aanvraag/Declaratieformulier” and “Vergoedingen/Mondzorg”. When looking at these three activities, it suggests that this covers the process that a customer follows when trying to get a compensation for dental care related costs. Hence, one might expect a sequential process flow for this. Figure 6.12 shows a process model obtained for these journeys. This model shows that there is not a single sequential process. All three activities are modeled in parallel and are contained in a loop. Hence, all three activities can be executed in any order and infinitely often. The only difference between this model and a “flower” model is that this model requires all activities to be executed at least once. Hence, there is no structure in the process for trying to get a compensation for dental care related costs. The Dotted Chart in Figure 6.13 confirms this insight as there is indeed no sequence of activities that occurs for multiple customers. Similar to the temporal patterns, for each of the activities a similar amount of customers starts with that activity, i.e. \( \sim 33\% \) of the customers start with “Nota/Mondzorg”, \( \sim 33\% \) start with “Vergoedingen/Mondzorg” and \( \sim 33\% \) start with “Aanvraag/Declaratieformulier”.

Multiple results containing more than one activity have been analyzed. Even when the result covers a low number of customers (< 10), still the same observations can be made: all activities are modeled in parallel and are contained in a loop such that they need to be executed at least once. Hence, it can be concluded that even if the different customer journeys are obtained for all the customers, it is still not possible to obtain high quality process models that deliver useful insights to an analyst.

As stated before, most of the results with a high support cover only one activity. For these results, it is not interesting to show a process model as this model would contain only a single activity with a self-loop. This self-loop would be included as there are always some customers that contacted CZ a second time for the same subject and (sub)subject. A reason for this might be that a customer has another question on the same subject after the first contact.

---

**Figure 6.11: Temporal patterns on events with a payment subject. The Inductive Miner is used to create a process model for these journeys.**
Figure 6.12: Journeys containing events for compensation on dental care. The process model is obtained using the Inductive Miner.

Figure 6.13: Dotted Chart of journeys for compensation on dental care. The colors of the events are based on the activity and the shape are based on the contact channel that is used. When multiple events occur at the same time, only one color is used.
6.4 Conclusions

CZ supplied a dataset containing all incoming contacts on four contact channels: e-mail, call center, visiting one of the service desks throughout the country, and using the online environment. After the pre-processing (Section 6.1) an event log is obtained. A single customer in this event log is used to show that our technique is indeed able to identify the different customer journeys. In this chapter, it is also shown that identifying temporal patterns first, might not always give the desired result. Some events of a customer are capture in a temporal pattern, while there are similar events, that are not captured in the pattern. Even if these events occurred between two groups of events in the temporal pattern.

The developed technique is used on a subset of the supplied dataset. This subset contains all events that are related to dental care. The technique is able to distinguish the different journeys for each customer separately. However, when merging similar journeys, the obtained process models do not give many insights. This is due to the fact that each customer behaves slightly different and executes the same activities in a slightly different order.
Chapter 7

Conclusions & Future work

This chapter concludes this thesis. First, Section 7.1 discusses the conclusions and limitations of the work done in this thesis. Section 7.2 discusses the future work.

7.1 Conclusions

Current process mining techniques are not suitable for modeling and analyzing customer journeys as the different journeys are not distinguished. Hence, a technique for identifying customer journeys is needed. As a result, the goal of this project was:

**Project goal:** *Develop and implement a technique that is able to discover customer journeys in an event log. Each trace in this log contains all events of a single customer. The resulting event log should contain a trace for each identified customer journey.*

To identify customer journeys in a dataset, a new technique is proposed. The technique discovers the journeys for each customer separately. This is done by detecting temporal patterns first. Events that are not covered by a temporal pattern, are clustered into journeys. To show the applicability of this technique, it is implemented as a plug-in for ProM. After the different journeys are identified for all customers, the results are directly loaded into a visualized. This enables further analyses for analysts.

In Chapter 4 it is shown that the technique does create different journeys for different, but related, activities. This is solved by using a similarity matrix. Creating a matrix for all pairs of activities may become a time consuming task. Hence, an approach for deriving such a similarity matrix is proposed.

A case study is conducted to show the practical applicability of our solution. This case study is conducted using customer contact data from CZ, which is an insurance company in the Netherlands. During this case study, the technique is used on a single customer to show that it is indeed able to discover the different journeys. When identifying temporal patterns first, it might be the case that some events are captured in a temporal pattern, while some other related events are not. Hence, detecting temporal patterns first, might not always give the desired result.

The technique is used on a subset of the supplied dataset, containing all events related to dental care. This showed that the technique is able to identify the different journeys for each customer separately. However, when merging similar journeys, the obtained process models do not give many insights. This is due to the fact that each customer behaves slightly different and executes the same activities in a different order.

7.2 Future work

During the case study some limitations of the developed technique were discovered. A limitation of the technique is that it heavily depends on the similarity matrix and corresponding similarity
threshold. An approach for creating such a similarity matrix is proposed. This approach could be incorporated in the current technique for the identification of customer journeys. The output of the technique heavily depends on the chosen value for the similarity threshold. Hence, a study could be conducted on how to derive this threshold automatically.

The case study on the CZ data revealed that the customer journeys are discovered correctly for each customer separately. However, when merging similar journeys, the obtained process models do not give many insights. It is shown that this is due to the fact that each customer behaves slightly different and executes the same activities in a slightly different order. In a next study, one could derive a better approach for comparing the results of different customers. This could result into more readable process models and hence, more insights in the behavior of customers.

During the case study it was also shown that in some cases the temporal pattern extension returns patterns that should not be considered as a temporal pattern. In such a case, the events are captured in the pattern, while there are similar events, that are not captured in the pattern. Even if these events occurred between two groups of events in the temporal pattern. In a future study, one could develop a technique that checks whether an obtained temporal pattern is indeed a valid temporal pattern, or just a coincidence that the events have a similar time interval between them.
Bibliography


BIBLIOGRAPHY


[12] S. Bucolo and J.H. Matthews. A conceptual model to link deep customer insights to both growth opportunities and organisational strategy in smes as part of a design led transformation journey. Design management toward a new Era of innovation, 2011. 8


[50] D.A. Wooff and J.M. Anderson. Inferring marketing channel relevance in the customer journey to online purchase. 2013. 7, 10


Appendix A

Pre-processing CZ data

This appendix describes how the CZ data is pre-processed. The first part of the pre-processing is done using SAS Enterprise Guide. Section A.1 discusses this part of the pre-processing. The second step of the pre-processing is done using a customer made ProM plug-in. Section A.2 discusses this plug-in.

A.1 SAS Guide

The pre-processing in SAS Enterprise Guide is done by creating a process flow. The entire flow is shown in Figure A.1 on page 73.

A.1.1 Library

The flow starts with a library. In this library, the location of the input files is defined, ensuring that SAS Enterprise Guide can access the source files. The code for this library can be found in Listing A.1.

```sas
LIBNAME input '<Path to dataset>'; Quit;

Listing A.1: Code in the program “Library”
```

A.1.2 Format data & rename columns

After the definition of the input location, the data from CZ is imported in the block “SET MARCO_ZORG_V4”, and the program “Format data & rename columns” is called. This program executes PROC SQL code that ensures a nice column naming in the dataset. Furthermore, the id of the customers is set in a correct way, i.e. the ids are stored as a string and not as a number. Storing ids as a number might cause problems due to rounding the numbers when exporting the dataset. The last important part of this program is that it sets the date of the contact moment in a format that can be read by process mining tools such as ProM and Disco. The code for this program can be found in Listing A.2.

```sas
PROC SQL;
    CREATE TABLE WORK.SET_CORRECT_FORMAT AS
    SELECT
            TRIM(PUT(Fictief_ID , 16.0)) AS customerId,
            TRIM(PUT(DATEPART( datumtijd ) , DDMMYYDY. ) || " : " || PUT(TIMEPART( datumtijd ) , TIME.) ) AS timestamp,
            TRIM( Hoofdonderwerp ) AS subject,
            TRIM( Subonderwerp ) AS subsSubject,
            fictief_mijnCzsid AS mijnCzSessionId,
            TRIM( Kanaal ) AS contactChannel,
```

Listing A.2: Code in the program “Format data & rename columns”
### APPENDIX A. PRE-PROCESSING CZ DATA

| TRIM(soort_klant) AS customerType, TRIM(doelgrp_omschr) AS group, TRIM(geslacht) AS gender, TRIM(provincie) AS state, TA_pakketwaarde, AVTA_pakketwaarde, TRIM(con_locatie) AS location, Advies AS advice, FTR, con_type AS contactType, Tevredenheid AS satisfaction, property1, soort_case AS caseType, TRIM(Afdeling) AS department FROM INPUT.SETMARCO_ZORG_V4; |

Listing A.2: Code in the program “Format data & rename columns”

### A.1.3 Create set without mijnCZ

In the original dataset, the subjects from the set regarding “Mijn CZ” are completely different than the subjects in the other set. As a result, there is never a match when comparing a subject from MijnCZ and the other set. Hence, the subjects of “Mijn CZ” need to be changed. To achieve this, all contacts that are not via “Mijn CZ” are extracted from the dataset, and stored in a temporary table as is done with the program “Create set without mijnCZ”. See Listing A.3 for the code of this program.

### PROC SQL feedback:

```
CREATE TABLE WORK.SET WITHOUT_MIJN_CZ AS
SELECT customerId, timestamp, subject, subsubject, mijnCzSessionId, contactChannel, customerType, group, gender, state, TA_pakketwaarde, AVTA_pakketwaarde, location, advice, FTR, contactType, satisfaction, property1, caseType, department FROM WORK.SET_CORRECT_FORMAT WHERE contactChannel <> "Mijn CZ";
```

Listing A.3: Code in the program “Create set without mijnCZ”

### A.1.4 Rename excel columns

As stated in Section A.1.3, the subjects of the contacts via “Mijn CZ” need to be changed. The flow imports an Excel file containing a mapping from the subjects and (sub)subjects to the new subjects and (sub)subjects. The table in the Excel file can be found in Table A.1 on page 74. The importer (“Import Data (Mapping_MijnCZ_Subject.xlsx[Blad1])” gives standard names to
APPENDIX A. PRE-PROCESSING CZ DATA

the columns found in the Excel file. For easy use later on, the program “Rename excel columns” is introduced which renames these column names. The code for this PROC SQL can be found in Listing A.4.

```
PROC SQL feedback;
CREATE TABLE WORK.SUBJECT MATCHING AS
SELECT
    F1 AS subjectOld,
    F2 AS subsubjectOld,
    F3 AS subjectNew,
    F4 AS subsubjectNew
FROM WORK.MAPPING MIJNCZ SUBJECT;
QUIT;
```

Listing A.4: Code in the program “Rename excel columns”

A.1.5 Create mijnCZ set

As stated before in Section A.1.3, the subjects of the “Mijn CZ” set need to be changed. The program “Create mijnCZ set” separates the contacts via “Mijn CZ” from the other contacts. In the same PROC SQL query, the subjects are changed such that they match the subjects used for the other contact channels. The changes of the subject and (sub)subject values are based on the result of the program described in Section A.1.4. If a subject of “Mijn CZ”, is not in the result of the program Section A.1.4, the contact moment is discarded as it is not considered as relevance in the customer journey. The code for the program “Create mijnCZ set” can be found in Listing A.5.

```
PROC SQL feedback;
CREATE TABLE WORK.SET_MIJNCZ AS
SELECT
    data.customerId,
    data.timestamp,
    matching.subjectNew AS subject,
    matching.subsubjectNew AS subsubject,
    data.mijnCzSessionId,
    data.contactChannel,
    data.customerType,
    data.group,
    data.gender,
    data.state,
    data.TA_pakketwaarde,
    data.AVTA_pakketwaarde,
    data.location,
    data.advice,
    data.FTR,
    data.contactType,
    data.satisfaction,
    data.property1,
    data.caseType,
    data.department
FROM WORK.SET_CORRECT_FORMAT data, WORK.SUBJECT MATCHING matching
WHERE data.contactChannel = "Mijn CZ"
AND matching.subjectOld = data.subject
AND matching.subsubjectOld = data.subsubject;
QUIT;
```

Listing A.5: Code in the program “Create mijnCZ set”

A.1.6 Create total set

The programs in Section A.1.3 and A.1.5, split the data in two groups: contacts via “Mijn CZ” and other. Since the subjects in both groups are now similar, these two groups can be merged together. The PROC SQL code for this is shown in Listing A.6.
APPENDIX A. PRE-PROCESSING CZ DATA

Listing A.6: Code in the program “Create total set”

```
PROC SQL feedback;
   CREATE TABLE WORK.SET_ZORG AS
   SELECT * FROM WORK.SET_WITHOUT_MIJNCZ
   UNION
   SELECT * FROM WORK.SET_MIJNCZ;
QUIT;
```

A.1.7 Add missing data

The last program in the main flow of SAS Enterprise Guide is the “Add missing data” program. This program adds an activity column to the data. For this thesis, the activity name consists of a combination of the subject and the (sub)subject of the contact. If the customer was on “Mijn CZ”, the department is unknown. In some cases, it might be desired to see which departments are contacted. Since this is unknown in some cases, the department is set to “Web” if it is unknown. The code for this last program is shown in Listing A.7.

```
PROC SQL;
   CREATE TABLE WORK.ZORG_FINAL AS
   SELECT customerId ,
          timestamp ,
          TRIM(subject) || "/" || TRIM(subsubject) AS activity ,
          subject ,
          subsubject ,
          mijnCzSessionId ,
          contactChannel ,
          customerType ,
          group ,
          gender ,
          state ,
          TA_pakketwaarde ,
          AVTA_pakketwaarde ,
          location ,
          advice ,
          FTR ,
          contactType ,
          satisfaction ,
          property1 ,
          caseType ,
          department
   FROM WORK.SET_ZORG
   ORDER BY customerId;
   UPDATE WORK.ZORG_FINAL SET department = "Web" WHERE department = "";
QUIT;
```

Listing A.7: Code in the program “Add missing data”

A.1.8 Export

The last important module in the flow is the “Export file” module. This module exports the result of Section A.1.7 to a CSV file. There are four steps for defining the export module:

1. Select file to export. The dataset that is the result of Section A.1.7 or “ZORG_FINAL” in Figure A.1.
2. Select a file type for the output file. For this thesis, csv files are used. Hence “Text Files (Comma delimited)(*.csv)” needs to be selected.
3. Modify additional options for the output file. Only one option is available here: “Use labels for column names”. This checkbox is not checked.
4. Specify the location and name for the output file. Any file location on the local computer or SAS server can be chosen here, as long as the folder exists.

After running this module, a csv file containing all the data is stored on the given location.

A.1.9 Subsets

To test the algorithm some subsets are generated. The generation of these subsets are in the right side in Figure A.1.

The first subset is a random sample of 100 customers. The program “Select 100 random customers” (Listing A.8) takes randomly 100 different customer ids. After this, the program “Create subset 100 customers” (Listing A.9) is executed to create a table containing all events of these 100 customers. The export of this subset is done in a similar way as described in Section A.1.8.

```
PROC SQL outobs=100;
  CREATE TABLE WORK.FINAL\_CUSTOMER\_SUBSET AS
    SELECT
      UNIQUE(customerId)
    FROM WORK.FINAL;
QUIT;
```

Listing A.8: Code in the program “Select 100 random customers”

```
PROC SQL;
  CREATE TABLE WORK.FINAL\_SUBSET AS
    SELECT
      customerId, timestamp, activity, subject, subsubject, mijnCzSessionId, contactChannel, customerType, group, gender, state, TA_pakketwaarde, AVTA_pakketwaarde, location, advice, FTR, contactType, satisfaction, property1, caseType, department
    FROM WORK.FINAL AS full, WORK.FINAL\_CUSTOMER\_SUBSET AS ids
    WHERE full.customerId = ids.customerId
    ORDER BY full.customerId;
QUIT;
```

Listing A.9: Code in the program “Create subset 100 random customers”

The second subset that is created is a dataset containing all customers that contacted CZ with the subject “Mondzorg” at least once. Similar as the previous subset, first a set containing all the customer ids is created and using this subset a dataset containing all events of these customers. Similar as the previous subset and the full dataset, the export of this subset is done as described in Section A.1.8.

```
PROC SQL;
  CREATE TABLE WORK.MONDZORG\_CUSTOMERS AS
    SELECT
      UNIQUE(customerId)
```

Customer journey identification through temporal patterns and Markov clustering
APPENDIX A. PRE-PROCESSING CZ DATA

Listing A.10: Code in the program “Select Mondzorg customers”

```sql
PROC SQL;
CREATE TABLE WORK.MONDZORG AS
SELECT
  full.customerId,
  timestamp,
  activity,
  subject,
  subsubject,
  mijnCzSessionId,
  contactChannel,
  customerType,
  group,
  gender,
  state,
  TA_pakketwaarde,
  AVTA_pakketwaarde,
  location,
  advice,
  FTR,
  contactType,
  satisfaction,
  property1,
  caseType,
  department
FROM WORK.FINAL AS full, WORK.MONDZORG_CUSTOMERS AS ids
WHERE full.customerId = ids.customerId
ORDER BY full.customerId;
QUIT;
```

Listing A.11: Code in the program “Create mondzorg subset”

```sql
Listing A.11: Code in the program “Create mondzorg subset”

```
Customer journey identification through temporal patterns and Markov clustering
Table A.1: Subject matching. The two columns prefixed with "Old" are the subjects and (sub-) subjects obtained from the pages in the online environment "Mijn CZ". These subjects and (sub-) subjects are replaced with the corresponding values in the two columns prefixed with "New".

<table>
<thead>
<tr>
<th>Old subject</th>
<th>New subject</th>
<th>Old (sub-) subject</th>
<th>New (sub-) subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>aanvragenzorgpas</td>
<td>Aanvraag</td>
<td>aanvragen</td>
<td>Aanvraag Zorgpas</td>
</tr>
<tr>
<td>aanvragenzorgpas</td>
<td>index</td>
<td>Wijzigingen doorgeven</td>
<td>Klacht Overig</td>
</tr>
<tr>
<td>beeindigverzekering</td>
<td>Polis</td>
<td>Verziening</td>
<td>Overig</td>
</tr>
<tr>
<td>bijschrijven</td>
<td>Polis</td>
<td>Wijzigingen doorgeven</td>
<td>Overig</td>
</tr>
<tr>
<td>contact</td>
<td>bevestiging</td>
<td>Wijzigingen doorgeven</td>
<td>Overig</td>
</tr>
<tr>
<td>doorgevenhuwelijksamenwonen</td>
<td>Geboortezorg</td>
<td>Overig</td>
<td>Overig</td>
</tr>
<tr>
<td>eigenrisico</td>
<td>Vergoeding</td>
<td>Overig</td>
<td>Overig</td>
</tr>
<tr>
<td>kraamzorg</td>
<td>Geboortezorg</td>
<td>Overig</td>
<td>Overig</td>
</tr>
<tr>
<td>machtigingen</td>
<td>Machtiging</td>
<td>Overig</td>
<td>Overig</td>
</tr>
<tr>
<td>notamelden</td>
<td>bevestiging</td>
<td>Wijzigingen doorgeven</td>
<td>Overig</td>
</tr>
<tr>
<td>overlijden</td>
<td>Overlijden</td>
<td>Wijzigingen doorgeven</td>
<td>Overig</td>
</tr>
<tr>
<td>regelingeigenrisico</td>
<td>Eigen risico</td>
<td>Overig</td>
<td>Overig</td>
</tr>
<tr>
<td>rekeningnummer</td>
<td>Rekeningnummer</td>
<td>Overig</td>
<td>Overig</td>
</tr>
<tr>
<td>wijzigbetaalgegevens</td>
<td>Rekeningnummer</td>
<td>Overig</td>
<td>Overig</td>
</tr>
<tr>
<td>wijzigcollectiviteit</td>
<td>Collectiviteit</td>
<td>Overig</td>
<td>Overig</td>
</tr>
<tr>
<td>wijzigemail</td>
<td>Adresgegevens</td>
<td>Overig</td>
<td>Overig</td>
</tr>
<tr>
<td>wijzigingoverig</td>
<td>Overig</td>
<td>Overig</td>
<td>Overig</td>
</tr>
<tr>
<td>wijzigingverzekering</td>
<td>Verzekering</td>
<td>Overig</td>
<td>Overig</td>
</tr>
<tr>
<td>wijzigkanaalkeuze</td>
<td>Wijzigingen doorgeven</td>
<td>Overig</td>
<td>Overig</td>
</tr>
<tr>
<td>wijzigingcollectiviteit</td>
<td>Collectiviteit</td>
<td>Overig</td>
<td>Overig</td>
</tr>
<tr>
<td>wijzigingpersoonsgegevens</td>
<td>Adresgegevens</td>
<td>Overig</td>
<td>Overig</td>
</tr>
</tbody>
</table>

74 Customer journey identification through temporal patterns and Markov clustering
A.2 ProM CZ Importer

The data obtained from CZ needed some pre-processing. First, some pre-processing steps have been done in SAS. Section A.1 describes the first pre-processing steps in SAS Enterprise Guide. The result of these steps is a CSV file, containing an entry for each moment a customer contacted CZ. Some process mining have the ability to import CSV files like the ProM import framework and Disco. In the CSV file, each row corresponds to a moment in which a customer contacted CZ. All the columns represent an attribute for this contact. Some of these attributes are the same for all contacts by the same customer, while some are dependent on the contact itself. Hence, an importer is needed that is able to distinguish event attributes from trace attributes.

To enable an easy way for importing the csv files in ProM, a small plug-in is created. The input of this plug-in is a csv file object and the output of this plug-in is an event log based on the naive implementation of the XLog interface from the OpenXES library\(^1\). The following list of keys, represent attributes that are considered as trace attributes:

- customerType
- group
- gender
- state
- TA_Paketwaarde
- AVTA_pakketwaarde
- property1

All the attributes in a row that are not in the list above, are considered to be event attributes. The idea of the import plug-in is that it reads all rows for a single customer and builds a trace for that customer. This process is repeated until all customers are modeled as a trace. Note that this method assumes that all rows of a customer are grouped together, i.e. the rows are sorted on the customerId attribute. Algorithm 14 shows the idea of parsing an array of lines representing a single customer to a trace representing the customer.

\(^1\)See http://www.xes-standard.org/openxes/start
Algorithm 14 Pseudo code for parsing a customer to a trace

1: function AddClusterToResult(customerLines)
2:     traceAttributes ← new AttributeSet
3:     trace ← new Trace
4:     for all line in customerLines do
5:         eventAttributes ← new AttributeSet
6:             for all attribute in line do
7:                 if attribute is a trace attribute then
8:                     add attribute to traceAttributes
9:                 else
10:                     add attribute to eventAttributes
11:             end if
12:         end for
13:         event ← new Event(eventAttributes)
14:         add event to trace
15:     end for
16:     add traceAttributes to trace
17:     return trace
18: end function
Appendix B

Figures

In the thesis, various process models have been shown. In some cases, these models are not readable in the small images throughout the thesis. For the interested reader, those process models are included in this section at a page size scale.

B.1 Figures in Chapter 3

Figure B.1: Petri net obtained by the Inductive Miner. Almost all activities are modeled as a single choice and are contained in a loop, i.e. they cover the behavior of a “flower” model. There are 10 activities that do not support this behavior. Hence, this model gives limited insights in the process.
B.2 Figures in Chapter 5

Figure B.2: Settings dialog: Algorithm selection. The user can select the desired algorithm. Depending on the selected algorithm, some information about the algorithm is shown to the user.
Figure B.3: Settings dialog: Settings for identifying temporal patterns. Figure B.3a shows the screen in which the user can select perspectives and optionally similarity matrices is shown. Figure B.3b shows the screen in which users can enter a value for the $\Delta_1$. 

(a) Ability to chose perspectives and a similarity matrix for each perspective.

(b) Minimal time-interval setting.
APPENDIX B. FIGURES

(a) Ability to chose perspectives and a similarity matrix for each perspective.

(b) Settings for MCL clustering. The inflation and expansion parameter can be specified.

Figure B.4: Settings dialog: Settings for clustering the remaining events. Figure B.4a shows the screen in which the user can select perspectives and optionally similarity matrices is shown. Figure B.3b shows the screen in which two settings specific for the Markov Cluster algorithm are shown.
Figure B.5: Additional settings for the TraceFinder plug-in
Figure B.6: Full scale version of Figure 5.4

Customer journey identification through temporal patterns and Markov clustering
B.3 Figures in Chapter 6

Figure B.7: Full scale version of Figure 6.5
Figure B.8: The full scale model of Figure 6.2

Customer journey identification through temporal patterns and Markov clustering
Appendix C

Event similarity matrices

Some matrices are too large to show in the thesis. This chapter contains these matrices in their full size and with all the data. Note that all 0.00 values are replaced with a single – to increase the readability.
## APPENDIX C. EVENT SIMILARITY MATRICES

(C.1) 

\[
\begin{bmatrix}
1 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
1 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
1 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
1 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
1 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
1 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
1 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 0.17 \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 0.17 \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 0.17 \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 0.17 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 0.17 \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 0.17 \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 0.17 \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 0.17 \\
\end{bmatrix}
\]

(C.2) 

\[
\begin{bmatrix}
1 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
1 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
1 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
1 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
1 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
1 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
1 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
0.17 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\
\end{bmatrix}
\]

Customer journey identification through temporal patterns and Markov clustering
### APPENDIX C. EVENT SIMILARITY MATRICES

\[ \begin{pmatrix}
\text{(C.3)} \\
0 & 1.00 & 1.00 \\
1.00 & 0 & 1.00 \\
1.00 & 1.00 & 0 \\
1.00 & 1.00 & 1.00 \\
0.17 & 1.00 & 1.00 \\
\end{pmatrix} \]

\[ \begin{pmatrix}
\text{(C.4)} \\
0 & 1.00 & 1.00 & 1.00 \\
1.00 & 0 & 1.00 & 1.00 \\
1.00 & 1.00 & 0 & 1.00 \\
1.00 & 1.00 & 1.00 & 0 \\
0.17 & 0.17 & 0.17 & 0.17 \\
\end{pmatrix} \]
APPENDIX C. EVENT SIMILARITY MATRICES

\[
\begin{bmatrix}
\vdots \\
-27.00 & 28.00 & 29.00 & 30.00 & 31.00 & 32.00 & 33.00 & 34.00 & 35.00 & 36.00 & 37.00 & 38.00 & 39.00 & 40.00 & 41.00 & 42.00 & 43.00 & 44.00 & 45.00 & 46.00 & 47.00 & 48.00 & 49.00 & 50.00 & 51.00 & 52.00 & 53.00 & 54.00 & 55.00 & 56.00 & 57.00 & 58.00 & 59.00 & 60.00 & 61.00 & 62.00 & 63.00 & 64.00 & 65.00 & 66.00 & 67.00 & 68.00 & 69.00 & 70.00 & 71.00 & 72.00 & 73.00 & 74.00 & 75.00 & 76.00 & 77.00 & 78.00 & 79.00 & 80.00 & 81.00 & 82.00 & 83.00 & 84.00 & 85.00 & 86.00 & 87.00 & 88.00 & 89.00 & 90.00 & 91.00 & 92.00 & 93.00 & 94.00 & 95.00 & 96.00 & 97.00 & 98.00 & 99.00 & 100.00 & 101.00 & 102.00 & 103.00 & 104.00 & 105.00 & 106.00 & 107.00 & 108.00 & 109.00 & 110.00 & 111.00 & 112.00 & 113.00 & 114.00 & 115.00 & 116.00 & 117.00 & 118.00 & 119.00 & 120.00 & 121.00 & 122.00 \\
\vdots \\
\end{bmatrix}
\]

(C.5)
Appendix D

Similarity matrix

Table D.1: The similarity matrix for the subject perspective. AV (Aanvraag), AG (Algemeen), BT (Betaling), MA (Machtiging), NO (Nota), VG (Vergoedingen), WD (Wijzigingen doorgeven)

<table>
<thead>
<tr>
<th></th>
<th>AV</th>
<th>AG</th>
<th>BT</th>
<th>MA</th>
<th>NO</th>
<th>VG</th>
<th>WD</th>
</tr>
</thead>
<tbody>
<tr>
<td>AV</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>AG</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>BT</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>MA</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NO</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>VG</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WD</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Customer journey identification through temporal patterns and Markov clustering