Chair of Process and Data Science

Applying Process Mining to User Behaviour Analysis: The Event-Case Attribution Problem

Master's Thesis

Author : **Tom-Hendrik Hülsmann**

Supervisors : Marco Pegoraro, M. Sc.
Dr.-Ing. M. Seran Uysal

Examiners : Prof. Dr. Wil van der Aalst
Dr. Simon Völker

Registration date : 18.05.2021

Submission date : 18.11.2021

This work is submitted to the institute

i9 Process and Data Science (PADS) Chair, RWTH Aachen University

Abstract

Recorded user interaction data today is a valuable source of information for software system providers. However, in most cases only basic analyzes are carried out based on these recordings, not using the available data to its full potential. The rich process mining ecosystem offers a wide array of different techniques and analysis methods that could be used for an advanced analysis of the user behaviour based on these interaction recordings. Until now, it was however not easily possible to use this data in process mining applications, because it does not contain the clear task identifiers required for process mining. In order to enable process mining for user interaction data, this thesis presents a novel case attribution approach based on neural action embeddings. The introduced method is presented and evaluated against a set of different test logs with varying different log characteristics and compared to two baseline approaches. In a case study based on a real dataset from a mobility sharing smartphone application, the utility of the produced segmentation in the context of a process mining analysis is evaluated. It is shown that process experts familiar with the application are able to derive concrete and actionable findings from the results of the conducted analysis, thereby emphasizing the large potential of applying process mining techniques in the field of user behaviour analysis.

Contents

Chapter 1

Introduction

1.1 Motivation

The performance of computing devices has continued to exponentially increase over the course of the last decades. This significant increase of raw computing power has enabled a substantial gain of capabilities for all kinds of different devices in all kinds of areas. Following this development, a large increase in the complexity of modern software systems is observed. The effects of this continuing trend can be found on all of the different levels, from software development to user interaction design. The HTML specification that powers the web has, for example, become twenty-five times longer since 1997 [1, 2] and the average size of smartphone applications has increased tenfold over the course of only four years [3].

As different software systems make use of the new capabilities that are offered by these modern devices, more and more features and functionalities are continuously offered to the users. This has made modern software systems increasingly more complex from the users' perspective. At the same time, the main mode of computing has shifted away from large screen devices such as desktop computers and laptops, to smartphones and tablets for a large part of the population [4]. These simultaneous developments have led to the current situation, in which a growing number of ever more complex features have to be accommodated on an increasingly smaller screen area. This poses significant challenges for the areas of modern *user interface design* (UI-design) and *user experience design* (UX-design).

Understanding the way that real users are interacting with a software system is one potential way to counter this problem. The computing power of modern devices and the essentially unlimited availability of network access today, allow software providers to track the behaviour of their users to an extent that was never possible before. In the past, the user feedback that a software provider could collect was limited to direct problem reports and surveys, both of which can only ever cover a small fraction of the overall user base. Today it is possible to observe the detailed behaviour of all users (e.g. clicks, views, searches, etc.) at any time by recording the users interactions with the software and storing these recordings in a central database. This interaction data contains valuable insights that can be used as the basis for improvements to the software system, founded on the real usage of the application. However, the tools and techniques that are available and in use today in order to provide such insights are rather simple and cannot unlock the full potential of the recorded data. Basic data mining techniques such as pattern mining are employed in conjunction with various time based measures, in order to obtain basic knowledge about the users and their usage of the software system [5]. Examples for such common metrics are the time a user spends on a certain screen or menu, which screens are visited from a certain screen and which percentage of users actually use a certain feature. While these metrics are able to provide a first set of basic insights into the real usage of the applications to the developers and designers of the software systems, they are not able to fully unlock the potential that lies in the recorded interaction data.

The field of *process mining* promises a whole new set of capabilities in this context. Process mining is concerned with automatically extracting knowledge out of data that is generated during the execution of a process. Traditionally, it is applied to business processes such as the classical purchase to pay process. In the context of observing the behaviour of users interacting with a software system, the totality of recorded interaction sequences that correspond to specific tasks of the users, can be considered as a *process*. Using this definition, the recorded interaction data can be used in process mining and therefore benefit from the rich existing process mining ecosystem. This enables new kinds of advanced applications such as sequence analysis, process model generation, conformance checking, running case prediction, or root cause analysis for interaction data. With all of these possibilities, process mining goes far beyond what is possible using the basic tools and techniques that are available and commonly used for user behaviour analysis today. However, there is one unfortunate characteristic of user interaction recordings that until now prevents the application of process mining to this type of data. Interaction recordings do not typically include any case information that could be used in order to separate the different process instances (tasks) that are contained in the log from each other. This is, however, a basic requirement for the majority of process mining techniques. It is therefore not easily possible today to leverage the potential of process mining for the analysis of user behaviour based on interaction recordings. The problem statement that results from this in the context of this thesis is presented in the following.

1.2 Problem Statement

Modern software design and development can be described as a continuous cycle between the release of an application, the collection of suggestions and user feedback and the design and development of updates based on this feedback. This cycle has changed over the years and today not only includes direct feedback from the users, but also the collection of automated usage metrics in the form of logs. Such interaction logs are an important part of the feedback process and provide the software providers with valuable information about how their software is used in a real setting [6]. As introduced before, the common analysis techniques that are applied to this data today are however somewhat limited and are not able to make use of the data to its full potential. This is why, in the scope of this thesis, the possibility of applying process mining to user interaction recordings in the analysis step of the iterative software development life cycle is explored. As already introduced, this data cannot be used for process mining directly. An additional step in the development cycle is therefore introduced, in which all of the recorded interactions are assigned to the corresponding task of the users.
A representation of this extended iterative software development life cycle can be found in Figure 1.1. Before the analysis step, a case attribution step is introduced in which the

recorded data is preprocessed, in order to make it suitable for process mining. Through this the available techniques in the analysis phase are extended by the rich ecosystem of existing process mining approaches. Following these extensions of the iterative software development life cycle, the two main areas of focus of this thesis are:

- The introduction of a novel approach to the case attribution problem, designed with the special properties of user interaction data in mind, therefore enabling process mining workflows for this kind of data.
- The evaluation of the real potential of process mining approaches in the context of user behaviour analysis.

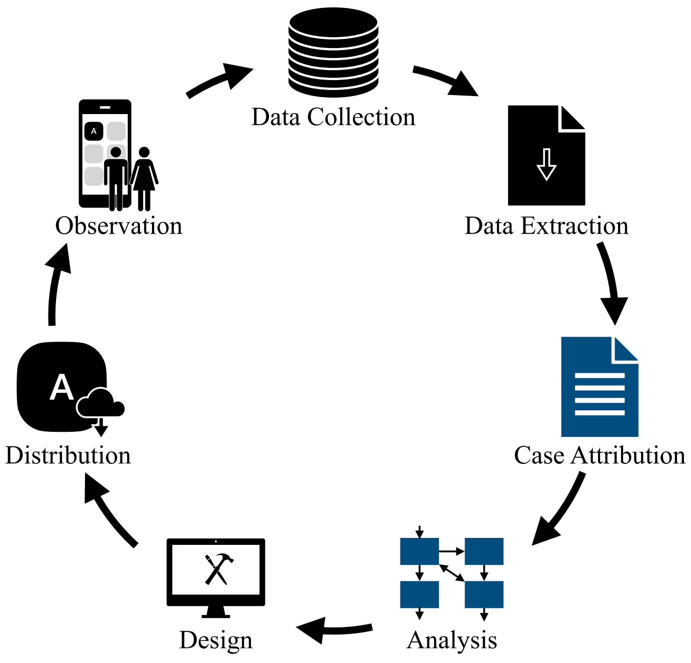


Figure 1.1: A representation of the modern iterative software development life cycle [6], including the additional case attribution phase that is proposed in this thesis.

The absence of a task identifier in a dataset one desires to analyze with process mining is not only a problem in the context of user behaviour analysis, but is also relevant for various other process mining applications such as *Robotic Process Automation* (RPA). In RPA, repetitive tasks of users are sought to be automated through the use of software bots. The behaviour of these bots is based on recordings of real users performing the task. In other areas of application, the recorded events may originate from multiple different systems, that do not have a single identifier for a shared process instance. The common underlying problem of all of these areas is the assignment of the single events to their corresponding process instance, this problem is called the *event-case attribution problem* or *event-case correlation problem* [7]. An example for this can be found in Figure 1.2. Because of this general importance, a number of different approaches to the event-case attribution problem have been proposed in the past. These approaches, are however in many cases relying on additional attributes in the event data that are not generally available for interaction logs, such as in [8]. Other approaches are fundamentally based on the relation between the timestamps of the recorded events, such as in [9, 10]. These approaches provide good results for event logs that have especially homogeneous activity durations, but struggle with identifying outliers and processes in which the duration of certain activities may vary significantly. Further approaches make use of certain patterns in the interaction sequences, such as predefined start and end events [11]. Those approaches are however only applicable to a small subset of software systems, in which clear start and endpoints exist. Many of the best performing case attribution approaches also suffer from a high time complexity, which renders them unusable for larger datasets.

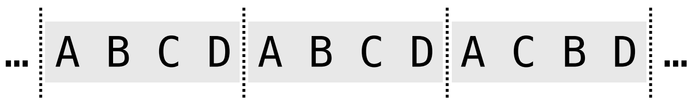


Figure 1.2: An example sequence of user interactions. Each character corresponds to a user interaction and the interactions are ordered based on their timestamp. *Case attribution* is the process of identifying those interactions that belong to the same process instance. Related interactions belonging to the same process instance are indicated separated by the dashed lines in the graphic.

Compared to the existing approaches, the case attribution method that is proposed in this paper should therefore make use of the special properties of user interaction data, such as the missing concurrency. A user is only able to perform one action at a time, and all of the interactions have a clear time based ordering to them. Additionally, the average duration of certain actions may vary drastically between different users and may be influenced by external factors, such as distractions of the user (e.g., incoming emails or phone calls). The goal of the thesis therefore is to propose a method that mainly considers the sequence of interactions, rather than the corresponding timestamps. Based on the various relations between interactions and patterns that may be observed in the log data, the method should be able to assign interactions that are related to the same task of the user to the same process instance. Additionally, the method should also be able to be applied to large interaction logs containing hundreds of thousand of interactions in a reasonable amount of time. In this way, the method should be able to make existing user interaction data accessible to different process mining techniques that could not have been applied to this kind of data otherwise.

In order to evaluate if process mining is indeed able to provide the expected benefits to the area of user behaviour analysis, a case study based on a real interaction log has to be conducted. The case study should consist of the following:

- A real interaction log should be attributed with case information using the proposed method.
- Basic process mining techniques should be applied to the attributed log in order to obtain insights about the real usage of the application.
- The resulting findings should be presented to, and be discussed with a number of process experts.
- The observations of the process experts should provide deep insights into the actual usage of the application, that go beyond the capabilities of existing approaches.
- Additionally, the experts should be able to identify concrete areas of the application that are not ideal in regard to the user experience. Based on this, they should be able to propose actionable improvements to the usability of the application.

1.3 Contributions

This thesis makes contributions to the field in two main areas: The conceptualization of a novel case attribution approach, including a reference implementation, and the subsequent application of this method to real interaction data, in order to evaluate the potential of process mining in the area of user behaviour analysis. These two main areas include the following individual contributions:

- A procedure to generate an arbitrarily-sized artificial event log using an existing log, combined with a directly-follows-graph of the process under consideration.
- A word2vec based method that is able to detect case boundaries in an unsegmented sequence of events without the use of timestamps.
- Multiple evaluation metrics that can be used in order to compare the quality of different case attribution approaches against each other.
- A Python based reference implementation of the proposed method, including the artificial log generation, word2vec model training, model based case boundary detection and the final segmentation of the input log.
- An evaluation of several different case attribution approaches using interaction logs with a wide range of different log characteristics.
- The assessment of process experts, regarding the actual utility of real recorded user interaction data when used in conjunction with process mining techniques.

In the following section the overall structure of the thesis and its contents is presented in more detail.

1.4 Structure

Initially, key concepts, definitions and notations that are used throughout the thesis are introduced in Chapter 2. Following this, in Chapter 3 existing applications of process mining regarding the analysis of interaction data are presented and various different approaches to the event-case attribution problem are introduced and compared to the method that is proposed in this thesis. The proposed method is then explained in detail in Chapter 4 and its various different phases are introduced. Additionally, details about the reference implementation and the used technology are discussed.
In Chapter 5, the proposed method is applied to multiple different artificial test logs, having a diverse set of log characteristics. The results are compared to those of two baseline case attribution approaches and the advantages and limitations of the proposed method compared to these baselines are considered. In a case study that is based on a real dataset from the mobility sharing smartphone application *MOQO*, the utility of the segmented interaction log in the context of user behaviour analysis is examined and the resulting observations are shared with process experts from the software provider.
Finally, the results and findings of this thesis are recapitulated and questioned in Chapter 6. Possible directions for improvement of the proposed method are discussed and opportunities for future work in the areas of case attribution and user behaviour analysis using process mining are considered.

Chapter 2

Preliminaries

2.1 Process Mining

Process mining is a relatively new field that lies at the intersection of established process sciences such as *business process management* and the area of *data science*. Its goal is to extract knowledge from so-called *event data* which is continuously collected during the execution of a process. A process can be any sequence of events that are carried out in order to reach a goal. Common examples include business processes such as the purchase to pay process. However, a wide range of different procedures in many other areas may also be considered as processes. In recent times information systems have become ubiquitous and are involved in almost every aspect of modern life. Because of this omnipresence of software systems in processes, they are a prime source for event data. During their execution, such information systems produce large amounts of data in the form of logs that contain information about what actions or tasks were performed at which point in time. Process mining techniques utilize this event data in order to automatically discover new information about the underlying process. This information may then be used in order to improve the observed process in different ways. Despite its young age, the field of process mining already offers a rich ecosystem of algorithms and techniques in areas such as *process discovery*, *conformance checking*, *process enhancement* and others [12].

The logs containing the event data that is collected during the execution of the process are called *event logs*. Event logs are a collection of individual events that at least consist of a *timestamp*, the carried out *activity* and a *case identifier*. These attributes represent the absolute minimum amount of information that is required for most process mining applications. Additionally, there may be other properties associated with the events, for example who carried out the activity or how long its execution did take. An exemplary event log can be found in Table 2.1.

Definition 2.1 (Event Log)**.** [13, 14] An event log L is a set $L \subseteq C \times A \times T \times V_1 \times ... \times V_n$ with C being a set of case identifiers, A being a set of activity names, T being a set of timestamps and V_i being sets of additional attribute values. The elements $e = (c_{id}, a, t, v_1, ..., v_n) \subseteq L$ are called *events* and have values from the corresponding domains.

In order to be able to follow a single process instance throughout the process, each event is labeled with a case identifier that is shared among all events belonging to the same process instance. Based on this, the complete event log can be grouped into multiple distinct so called *cases* that consist of sequences of events with varying lengths.

Definition 2.2 (Case)**.** A *case* c of length n in process mining is one instance of a process execution. It consists of a set of n events $c = \{e_1, ..., e_n\}$ where for all $i, j \in \{1, ..., n\} \mid i < j$ it holds that $c^i_{id} = c^j_{id}$ and $t_i \leq t_j$.

All of the events of a case share the same case identifier. In addition to the events themselves, a case may also be associated metadata that concerns all events of the case and can be used to further describe the underlying process instance. (e.g., an order number or a customer identifier). The first event in a case is called the *start event*, while the last event is called the *end event*.

Table 2.1: An excerpt of a fictional event log. The case identifier is represented by the `Order No.` column, the activity name by the `Status` column and the timestamp by the `Time` column. The `Product No.` column represents an additional attribute that provides further information about the process instance. Based on [14].

Order No.	Status	Product No.	Time
84578	Order Received	9091	2019-15-02 08:12:52+02:00
84590	Order Cancelled	4056	2019-15-02 08:23:43+02:00
84578	In Processing	9091	2019-15-02 09:05:11+02:00
⋮	⋮	⋮	⋮

As introduced before, the existence of a timestamp, an activity, and a case identifier is generally a requirement for the majority of process mining operations. Most process mining techniques rely on the fact that a grouping of events based on the case identifier is possible. For example, consider conformance checking techniques: In order to assess if a process instance is fitting the constraints of the assumed process model, it is a requirement to be able to distinguish between the different process instances. Since this distinction is based on the case identifier, conformance checking is not possible if no such identifier is available. The same is also true for process discovery techniques, in which it is of importance to be able to identify the start and end events. In many areas of application a suitable case identifier is easily available. For example, there might be an order number, a part identifier or a distinct process id. Since these identifiers are in many cases needed during the execution of the process in order to handle the different process instances accordingly, they are generally known to the involved information systems.

However, this is not the case in all circumstances and there exists a significant number of information systems that are involved in processes, but are not process-aware. Examples of such systems include e-mail clients, that may be aware of the recipient but not the concrete case, or machines in production environments that do not have an understanding of the whole production line. In addition to that, there also exist use cases in which the definition of a case is not straightforward and it is therefore not possible to directly assign case identifiers. As introduced before, the analysis of user behaviour based on recorded interaction data is an example for such a situation. A case here represents a task that the user performs. At the time of recording, it is not known when a task starts or ends. In such situations, process mining techniques cannot be applied directly to the recorded data. A preprocessing step that correlates events with cases is therefore required.

Definition 2.3 (Event-Case Attribution Problem)**.** The Event-case attribution problem, or event-case correlation problem, is concerned with associating events in an event log that do not have an unambiguous case identifier with their respective process instances (cases). The goal is to find an optimal segmentation of the event log in such a way, that each event in the log can be clearly correlated with a unique case based on the attributes of the event.

The information that is contained in an event log is often transformed and modeled in different ways in process mining. One common abstraction model for the control-flow perspective of a process is the so-called *Directly-Follows Graph* (DFG) which is based on the directly-follows relations in an event log.

Definition 2.4 (Directly-Follows Relation)**.** [14] Given an event log $L \subseteq C \times A \times T$, the directly-follows relation $>_L$ is defined in the following way: $>_L = \{(a_1, a_2) \in A \times A\}$ such that $\exists(c_{id}, a_1, t_1), (c_{id}, a_2, t_2) \in L$ where $t_1 < t_2$ and $\neg\exists(c_{id}, a_3, t_3) \in L$ where $t_1 < t_3 < t_2$.

Two activities are said to be in a directly-follows relation if there exist two events with the corresponding activities that belong to the same case and occur one after another, with no other event in between them. The DFG of an event log is build based on the directly-follows relations of the log.

Definition 2.5 (Directly-Follows Graph)**.** [14] The *directly-follows graph (DFG)* of an event log L is a directed graph $G = (V, E)$, where V is a set of vertices and E a set of edges with:

- $V = \{a \,|\, \text{a is an activity in } L\}$
- $E = \{(a_1, a_2) \,|\, a_1, a_2 \in V : a_1 >_L a_2\}$

A DFG additionally often includes a weighting function that assigns a value to each edge of the graph. Different variants of DFGs using different weighting functions do exist. There are for example frequency-based and time-based variants. A closely related type of event log abstraction model is the *transition system.* In contrast to the events in the event log, which model single events in the process, transition systems aim to encode the current state of the process and the transitions between these different states.

Definition 2.6 (Transition System)**.** A *transition system* is a tuple (S, E, T, i) where S is a set of states that represent a configuration of the process, E is a set consisting of the activities that can be performed in order to transition between different configurations of the process, $T \subseteq S \times E \times S$ is a set containing the transitions between configurations and $i \in S$ is the initial configuration of the process. Starting from the initial state i, the transition system can move between states according to the transition rules that are defined in T.

A transition system can be obtained from an event log through different types of abstractions. The assumption for these abstractions is, that every specific state of the process corresponds to a sequence of events in the log. In general the abstraction is either based on a window of past events, future events or both. The size of the window is flexible and can be chosen based on the task. The most basic abstraction considers the activity of the current event as a state. This corresponds to a past abstraction with a window size of one. The corresponding transition system is therefore equal to the directly-follows graph of the same log. When there is more than a single event in the window, one has to additionally

choose a representation for the events in the window. Common representations include sets, multisets and sequences of events [15].
In the following section, a specific type of event log that is used in the context of automated user interaction analysis is introduced.

2.2 User Interaction Logs

While users are operating a software system they continuously interact with it in different ways and perform diverse actions. These actions change the state of the software system and allow the users to progress with their tasks. Such sequences of actions may be viewed as a process and are therefore suitable for the application in process mining. Actions may range from simple clicks or taps on buttons, over text entry and manipulation, to more complex actions such as gestures or voice commands. Each action a user performs at a certain point in time, represents an interaction between the software system and the user. These actions are therefore called *user interactions*.

Definition 2.7 (User Interaction)**.** [16] A *user interaction* i describes an action that a user performs in a software system. It is denoted as a tuple $i = (u, a, t, p_1, ..., p_n)$ where u uniquely identifies the user that performed the action, $a \in A$ is an identifier for the type of action out of all the possible actions A (e.g. which button was pressed or which screen was displayed), $t \in T$ is a timestamp that captures the point in time at which the action occurred and p_i for $i \in \{1, ..., n\}$ are additional optional parameters that further describe the action e.g. how long it did last or in which context it was performed. The action identifier a corresponds to the activity attribute that is associated with an event in the context of process mining.

A user interaction is uniquely identified by the combination of the user, the timestamp and the action identifier. Because user interactions are frequently performed in quick succession (e.g. a double click lasts less than a second) and the correct ordering of the interactions is highly important, a high timestamp resolution is required in order to correctly differentiate between distinct interactions. In order to learn more about the behaviour of the users in relation to the software system, the observation of single user interactions is not sufficient. A sequence of user interactions, obtained from multiple different users is therefore recorded. The recorded user interactions are combined and collected in a so called *user interaction log*. An example of such an user interaction log can be found in Table 2.2.

Definition 2.8 (User Interaction Log)**.** [16] A *user interaction log (UI Log)* L is a collection of ordered user interactions. It is denoted by a tuple $L = (I, \sigma)$ where $I = \langle i_1, ..., i_n \rangle$ is an ordered sequence of user interactions $i_j = (u_j, a_j, t_j)$ with $(u_j, a_j, t_j) < (u_k, a_k, t_k) \Rightarrow t_j \leq t_k$ and $\sigma : I \rightarrow \mathbb{N}$ is a segmentation function that assigns a case identifier to every user interaction. Interactions with the same case identifier according to σ are belonging to the same case.

In addition to the individual interactions, user interaction logs also contain information about the relations between the different interactions. Similar to the basic concept of event data that was presented in the prior sections, each interaction is usually also associated with a case. A case describes a sequence of related user interactions that the user performed in order to complete a task. For example, the sequence of clicks that a user needs to perform in order to reach a certain functionality, starting from the main menu of the

Table 2.2: An example of a segmented user interaction log. The `case_id` column contains information about the case identifier that is associated with the interaction. The `user_interaction` column contains the user interactions with the corresponding attributes: `user identifier`, `action identifier` and `timestamp`.

`case_id`	`user_interaction`
1	(#012, menu, 2021-01-30 12:23:10.101)
1	(#012, search, 2021-01-30 12:23:11.512)
2	(#124, map, 2021-01-30 12:23:13.431)
1	(#012, search_result, 2021-01-30 12:23:13.602)
2	(#124, details, 2021-01-30 12:23:15.020)

software interface. However, suitable information about the case is generally not always known at the time of recording in such a setting. This is especially true when the goal is to fully automate the recording process without relying on any human annotation of the log. UI logs that do not contain case information are called *unsegmented*.

Definition 2.9 (Unsegmented UI Log)**.** A user interaction log $L = (I, \sigma)$ is called *unsegmented* or *uncorrelated* if the segmentation function σ is constant, meaning that $\sigma(i_1) = \sigma(i_2) \mid \forall i_1, i_2 \in I$.

The problem of transforming an unsegmented UI log into a segmented UI log corresponds to the event-case attribution problem of general process mining that was introduced earlier. In order to apply existing process mining techniques to an unsegmented UI log, the event-case attribution problem has to be solved by performing some kind of case attribution. The method proposed in this thesis tries to do this using event embeddings which are introduced in the following section.

2.3 Event Embeddings

The method presented in this paper is fundamentally based on the concept of *event embeddings*, which are themselves based on the natural language processing architecture *word2vec*. The word2vec architecture allows the learning of abstract representations of words and their relations, so called *embeddings*. This concept was first proposed in 2013 by Mikolov et. al. in [17] and [18]. The underlying idea of word2vec is to encode the relations between words in a body of text using a shallow neural network. The resulting word embeddings are represented by vectors. The more similar the vectors of two words are according to the cosine similarity measure, the more semantically similar the words are. The technique therefore allows to capture the semantic meaning of the words, based on the way they are used in the sentences of a body of text.

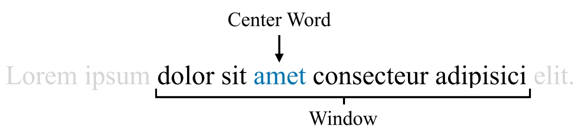


Figure 2.1: An example sentence from a body of text. The window has a size of five and the center word is marked in blue. The two words in front of and after the center word are the context words.

During the training of the two-layer neural network, a sliding window of a specified odd size is used in order to iterate over the sentences. An example for this can be found in Figure 2.1. The word in the middle of this window is called the *center word*. The words in the window before and after the center word are called *context words*.

There are two different approaches to the word2vec architecture; continuous bag-of-words (CBOW) or skip-grams. The main differences between the two approaches are the input and output layers of the network. While in CBOW the frequencies of the context words are used in order to predict the center word, in the skip-gram model the center word is used to predict the context words. The order of the context words is not considered in CBOW. However, the skip-gram model does weigh the context words that are closer to the center word more heavily than those that are further away. A representation of the CBOW architecture can be found in Figure 2.2.

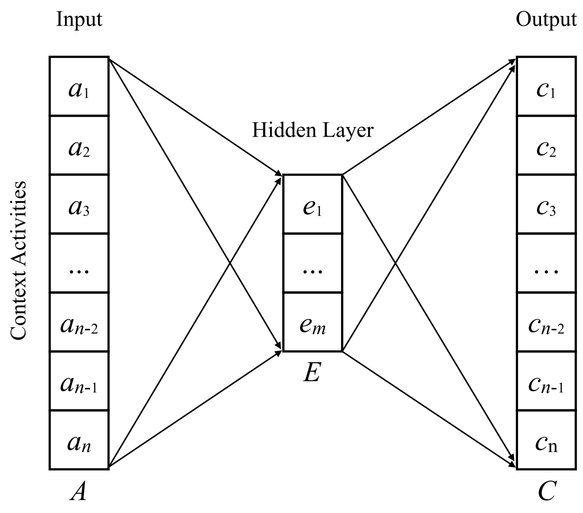


Figure 2.2: A graphical representation of the concept behind the event2vec architecture. The vector A of size n counts how often every activity occurs in the considered window. E is the vector representing the event embedding of size m where $m << n$ and vector C is a one-hot encoding of the center activity in the ideal case.

Both approaches produce an embedding of the context word in the form of a vector. The advantage of this architecture is that the size of the resulting embedding vectors can be freely determined through the size that is used for the hidden layer. Using this architecture, it is therefore possible to reduce the dimension of the input vector ($|V|$) considerably compared to the output embedding ($|E|$). Additionally, the word embeddings also capture information about the context in which a word is frequently used. As mentioned before, the more similar the vectors of two words, the closer the words are in meaning. In addition to this, the embeddings can also be used in order to predict the center word based on a set of given context words. Because of this versatility, the word2vec architecture is today also widely used in areas other than natural language processing, such as biology [19], medicine [20], or process mining [21].

In the context of process mining, the body of text under consideration is substituted by the event log. In event embeddings, activities and traces take the role of words and sentences in word embeddings. Using this definition, the principle behind word2vec can easily be applied to event data too. Instead of the vocabulary V there is the set of all possible activities A. During learning, each activity is associated with its embedding vector E, which is the output of the hidden layer. The output layer of the network C ideally represents a one-hot encoding of A, in which only the desired center activity is mapped to one. Analogous to the word embeddings, event embeddings also capture information about the relations between the different activities. This enables the possibility to find activities that are similar to each other and allows to predict the most likely center activity based on a set of context activities. These properties of event embeddings are used by the proposed method in order to predict the boundaries between cases, by only using the sequence of activities in the interaction log. As mentioned before, this capability is not only important in the context of process mining, but also in related fields such as robotic process automation which is introduced in more detail in the next section.

2.4 Robotic Process Automation

Repetitive tasks that need to be performed manually are part of many business processes. Those tasks are often relatively simple, yet time intensive and hard to automate. They often involve multiple information systems, from different vendors, such as email clients or bookkeeping software. The automation of such repeating sequences of manual actions has the potential to reduce the workload of employees and therefore cost. However, this is not a straightforward task, as the involved systems are often not built with automation in mind. Even in cases in which they are, the involvement of a wide range of different systems throughout the process introduces compatibility issues in between them, which again hinders automation efforts. Automating such tasks is therefore only practical in situations in which the task is simple and clearly defined, has only few variants and does not change over time. These restrictions are severely limiting the potential areas of application of repetitive task automation.

The field of *Robotic Process Automation* (RPA) is concerned with unlocking automation capabilities for complex tasks that are involving multiple different software systems. [22] The main idea of RPA is to not rely on software interfaces that are offered by the systems, but to replicate the actual interactions of the user with the device in order to automate the task. This is done through the use of so-called *software robots (bots)*. The bots start by observing the user while they perform the task under consideration. Based on the observed user interactions, the bots are then able to replicate the actions and perform the task independently. Instead of utilizing some kind of provided API, the user interactions are recreated by the bots using the same means as the user (e.g., through clicks and mouse movements).
The field of RPA can be divided into several sub-fields that are concerned with different phases of the automation process. For example, the identification of promising target tasks, the recording and interpretation of the user interactions, or the final execution of the learned behaviour by the bots. The results of this thesis are especially relevant in the context of task identification and recording analysis.

Chapter 3

Related Work

While the proposed method is applied in the context of UI logs in this paper, it fundamentally aims to solve the underlying event-case attribution problem of process mining. As introduced before, the ability to assign a process instance to every event in the log is a basic prerequisite for most process mining operations [12]. The correlation problem is therefore repeatedly faced when trying to apply process mining techniques to other areas than the traditional, well described business process. In her work on applying process mining to the interactions of users with the interface of a CT scanner in a clinical context, J. Reindler mentions the definition of what constitutes as a case and which events belong to a case as a *major challenge* [23].
Missing case information is also a challenge for Marrella et al. in their work on measuring the learnability of software systems [24]. As input, their method receives a set of unsegmented UI logs, which cannot be used directly for their task. In order to segment the log, they use a set of predefined start and end events and split the log accordingly. They find that this is not ideal because not all types of cases can be discovered. The approach may therefore only be applied to a limited set of software systems, which reduces its overall utility [24].
Jlaiaty et al. faced the attribution problem in their work on discovering business process instances based on email logs [8]. Their approach uses a combination of different metadata attributes such as timestamps, sender/receiver information and named entities in the email. Based on these they calculate a metric that is used in order to identify the process instances. According to their results, there are however edge cases that cannot be easily identified using this approach [8].
IoT sensors produce large amounts of data that in some applications may be considered as event data. This data also does not usually contain case information. This circumstance is explored by Janssen et al. in [25]. They consider the need for *sensor case slicing* in order to segment the long traces that are generated by each sensor into smaller cases that are based on different activities. In order to do so, they split the long traces according to a fixed length that is application dependent. Their goal is to find the optimal sub-trace length, such that after splitting, each case contains only a single activity. One major limitation to the approach that is mentioned by the authors, is the use of only a single constant length for all of the different considered activities. This is problematic because the different activities may have varying lengths [25].

More general approaches to the event-case correlation problem that are not related to a specific application domain also exist. An early work in the area by Ferreira et al. is concerned with identifying the different sources of events in an unlabeled event log [26]. They use a first-order Markov model in order to create a process model of the observed process. In this context, the method also discovers the underlying case identifiers. It can therefore also be applied to the correlation problem. While the method generally provides good results according to their findings, they mention that it does not perform well when it is applied to processes containing loops. This is caused by the fact that only the last state is considered in a first-order Markov model [26].
Processes containing loops pose a challenge for many approaches to the correlation problem. The *Correlation Miner* proposed by Pourmirza et al. in [10] is a process discovery algorithm that also suffers from this problem. They set up a quadratic programming problem in which all directly-follows relations between events in the log are considered. Using a time based heuristic, they try to find the best assignment between the case boundaries and these directly-follows relations. The method produces good results for acyclic processes, but has a rather high time complexity [10].
The method proposed by Bayomie et al. in [27] is able to also handle processes containing loops. They use background information about the process such as a heuristic based on the activity durations and a process model. This information is used in conjunction with a decision tree in order to generate a number of segmented logs with associated scores. From this the best log segmentation is selected. This method again heavily relies on timestamps and is rather slow, reporting runtimes of about 45 minutes for a log with only 1,000 cases [27].
Typically, not all systems involved in a process are *process-aware*, meaning that they have no understanding of the underlying process and its instances and have no means to record events. An investigation on how such systems can be integrated into process mining workflows was performed by Pérez-Castillo et al. in [7]. In this context, they also focus on the correlation of events and cases and propose an approach that uses additional *correlation attributes* of events that are selected by process experts (e.g. patient identifier, medical condition, gender) in order to assign events to cases. While the approach provides good results in their case study, it can only be applied when additional attributes are available. It also has a quadratic runtime, which is not ideal for large event logs [7].
A more generally applicable approach is proposed by Bayomie et al. in [28]. In this work they try to improve on their earlier work [27] by lifting the requirement for an existing timestamp heuristic. The only additional input required beside the event log is a model of the process under observation. They use simulated annealing in order find the case assignment that is optimal in regards to how well aligned the cases are to the model and how high the variance of activity execution time is in between cases. Their results show excellent performance on an array of different logs, outperforming the previous state-of-the-art approach [27]. While the runtime of their approach has improved significantly over the previous best approach, it still lies in the order of hours [28].

In addition to process mining, the event-case attribution problem and other strongly related problems have been described as relevant issues in other areas. One such area is robotic process automation. In their case study about the applicability of RPA in different business processes, Aguirre et al. conclude that RPA is only viable for stable processes, because of the manual modeling that is required [29]. This manual effort may be avoided by employing automated systems in the early stages of RPA. The development of such

systems is one of RPAs main challenges according to Syed et al [30]. Reliable case identifiers are an important part of building such systems. Gao et al. mention this as a problem in their work on automated RPA learning, where they assume a case identifier as given and mention its absence as a future research opportunity [31]. Furthermore Montero et al. identify missing case information as a challenge for automated testing in RPA [32].
Existing approaches to the problem in the RPA field often assume the existence of unique start and end events in order to segment the log. Jimenez-Ramirez et al. use this approach in [33] in order to segment the log crated by a back-office system with rather clear entry and exit points. Similarly, Linn et al. rely on known activity starts and ends in their work on desktop activity mining in the context of RPA [34]. They require the observed user to manually mark the start and endpoints of the current activity. This approach is not feasible for longer term deployments, as they mention in their case study.
A recent work by Volodymyr et al. is trying to address these issues by neither requiring specific start and end events nor an existing model of the process [35]. In their work they aim to identify tasks that are suitable for automation using RPA. Because of the nature of this problem, they cannot assume existing information about the task. They perform the segmentation based on the control flow graph of the log. They first discover the back edges in the graph. Based on these, possible start and end events are identified. The log is then segmented based on the identified events. While the method does not require start and end events as input, it still relies on them for the final segmentation step. This makes it susceptible to situations in which there are multiple different start events [35].

A problem very similar to the event-case attribution problem is session identification in the context of web usage mining. In order to study user behaviour on the web, each recorded event first has to be associated with the user that triggered the event [36]. This sessioning is often performed based on the timestamps of the events. There exist simple threshold based approaches (e.g. a fixed timeout of 30 minutes) and more refined approaches that use additional knowledge about the timings. One such approach is proposed by Mobasher et al. in [37], where they determine the sessions based on a comparison between the time a user is expected to spend on a page with the actual time the user spent on the page. While the proposed method significantly outperforms more simple threshold based approaches, it still suffers from the fundamental flaw that the resulting sessions are biased based on the chosen tolerances. This makes it harder to identify outlier behaviour.

The foundation of the method proposed in this paper is build on the previous work of Lakhani et al. who used the word2vec CBOW model [17, 18] in order to reconstruct traces with missing events [21]. Using traces that are known to be correct, they trained the word2vec model with a window size of one. During prediction they generate pairs of context words that are used to predict the center word, which is then selected for reconstruction if its probability is higher than a certain threshold. They were able to achieve a 90% reconstruction accuracy on a real dataset [21].
This highlights the potential of the word2vec architecture as an abstraction approach for event data, particularly for traces. The proposed method builds upon these findings and applies a similar word2vec abstraction in the context of the event-case attribution problem. This is the first application of the word2vec architecture to this problem in the context of process mining.
There is however existing work in the field of process mining that uses embeddings as abstract representations of event logs, for example for anomaly detection [38] or conformance checking [39]. Koninck et al. explore various different neural embedding strategies

for event data in [40] and come to the conclusion that there is a strong potential for the application of neural embeddings in the process mining context. In [41] Barbon et al. compare different embedding approaches such as trace replay, alignments and word embeddings. According to their results, word2vec is a suitable embedding approach and is especially suited for prediction tasks. It is however one of the slower evaluated approaches regarding learning time.

In conclusion, the importance of the event-case attribution problem in different research areas and applications is underlined by the existing work in the field. Previous approaches often rely on predefined start and end events, timestamp heuristics or additional attributes that are only available in specific use cases. Furthermore, they are often not able to handle processes containing loops and have runtime characteristics that are not suited for large event logs. Few of the approaches focus on the special characteristics of UI logs, like the inherit non parallelism, which may be exploited to achieve better results. The method proposed in this paper tries to advance the area of case attribution by considering the peculiarities of UI data, acting mostly timestamp independent, handling loops up to a certain length and enabling comparatively fast execution times after the initial training of the model. The proposed approach will be introduced in detail in the next chapter.

Chapter 4

Approach

As introduced before, an unsegmented UI log cannot be used for process mining directly, since most process mining algorithms require the presence of case information. In order to enable process mining applications for such unsegmented log data, case attribution has to be performed. This means, turning the unsegmented log into a segmented one by associating each event or interaction of the log with a concrete case.
This association is performed by the proposed method in a novel way based on underlying word2vec models that try to abstract the known information about the process under consideration. However, these models have to be trained before they can be used for this task. Since the unsegmented input log cannot be used for training, because of the missing case information, an artificial training log is generated in the initial step of the method. The models that are trained using the generated log can then be used in order to calculate a case boundary likelihood score. Using the scores, the events or interactions of the contiguous input log can finally be segmented into distinct cases.

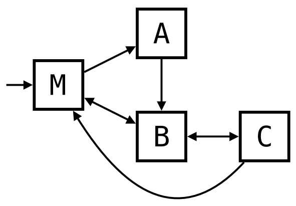


Figure 4.1: The DFG of the exemplary process that is used throughout this chapter.

In this chapter, the concepts that were introduced above are presented and discussed in more detail. A comprehensive overview over the different phases of the proposed case attribution approach is given. Additionally, peculiarities of the developed reference implementation are discussed. Throughout the chapter, a simple exemplary interaction log is used in order to explain the core concepts of the proposed method. The directly-follows graph of the example process, consisting of the four actions M, A, B and C, can be seen in Figure 4.1. While M is representing a main menu and acts as the entry point to the process, the remaining actions represent different features of the underlying software system. The following three unsegmented user traces are considered as part of the example: ⟨M, A, M, B, C⟩, ⟨M, B, C, M⟩, ⟨M, A, B, C⟩.

4.1 Overview

Before introducing and discussing the proposed method and its successive steps in detail, here a short overview over the different main phases of the proposed case attribution approach is given.

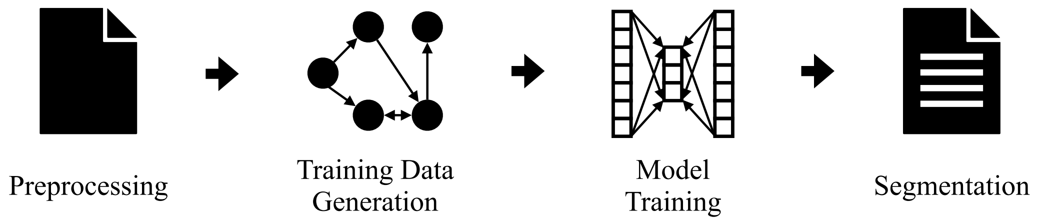


Figure 4.2: An overview over the four different main phases of the proposed method.

Preprocessing The given user interface log must adhere to some minimal requirements before the proposed method can be applied. The log passes through multiple filtering stages in order to ensure that is a suitable input for the method.

Training Data Generation Based on a transition system that is build from the input log, a set of artificial training traces is generated. The given DFG of the process is used in order to filter the transition system and ensure its consistency with the real process.

Model Training The traces of the generated training log are connected and separated by artificial end actions. Based on this set of connected traces, different word2vec models are trained using varying sets of parameters. The trained models capture the relations in between the actions of the log.

Scoring & Segmentation The trained models are used in order to estimate the likelihood of a case boundary at all positions in the input traces. Based on the combined scores of all models, candidates for split points are identified and checked for their consistency. The input log is finally segmented based on the identified split points.

4.2 Preprocessing

The proposed method is designed in order to infer case information in a given unsegmented UI log. The raw input data therefore consists of an unsegmented UI log $L = (I, \sigma)$ as described in Section 2.2. Each user interaction $i \in I$ therefore consists of a *case identifier* $\sigma(i)$ which is initially unknown, a *user identifier* $u \in U$, an action identifier $a \in A$ and a timestamp $t \in \hat{T}$. In order to ensure the consistency of the transition system that is the basis of the training log generation, a DFG $G = (V_G, E_G)$ containing information about the allowed directly-follows relations in the process is needed as an additional input.
The method requires that each user interaction in the input log is associated with a unique user identifier u. However, this might not always be the case. For example, in the MOQO application that is considered in the conducted case study, users are initially required to log in to their accounts. Actions that happen before this login can therefore not be associated with the user. Since a combination of interactions from different users might

violate the assumption that there is no parallelism between the actions, interactions with an unknown user identifier u have to be filtered out from the original input log.
The set of possible actions A will later built up the vocabulary of the word2vec model. As introduced before, the word2vec architecture is originating in the area of natural language processing. In this context, common so-called *stopwords* are filtered from the input before the vocabulary is built. Stopwords are words that are very frequent in the considered language and do not provide information about the semantics of the considered sentence. The same concept can be applied to the different types of recorded actions in A. It therefore needs to be defined which subset of the recorded actions is considered and which actions are discarded. For example, in the MOQO application there exist multiple user interfaces that are reused in different contexts. In such cases, the UI log will contain log entries for the main interface, as well as the reused interface. Since both of these actions are related to the same user interaction and will always occur together, they should be grouped and reduced into a single action. Additionally, actions that occur randomly or very frequently throughout the recording (e.g., notifications) and are not directly related to the process under observation, might be filtered from the log. Moreover, the abstraction level used for the actions has to be determined. For example, one might consider raw interaction data such as click positions or mouse movements, interactions with UI elements such as buttons, or the screens/pages a user visits. Different abstraction levels are ideal for different use cases, depending on the available data, the system under observation and the goal of the analysis. In the conducted case study, a screen abstraction is used.
Since the proposed method acts mostly independent of timestamps, the resolution of the timestamps $\hat{T}$ is not as important. What is however important, is that the user interactions have a unique ordering. It has to be guaranteed, that an interaction that occurred after some other interaction in the log also did so in the real process. Otherwise the method might not be able to discover a segmentation that reflects reality.

The reference implementation of the proposed method was realized using the Python programming language. Additionally, the scientific computing libraries *pandas* [42] for data importing and preprocessing and *numpy* [43] for vector computations and random number generation were used. The underlying word2vec models are realized using the *gensim* [44] library, which is used for model training and prediction tasks. The datasets used for testing and evaluation are provided in a basic CSV format. The raw data is initially imported into a dataframe using pandas. The provided DFG G is encoded as an adjacency list in the form of a dictionary that defines the directly following actions for each action of the log. The different filtering operations during preprocessing are performed using pandas and are applied directly to the dataframe. The preprocessed dataset in the form of a pandas dataframe is the input for the following training data generation step of the method.

4.3 Training Data Generation

After preprocessing, in this step of the method, a set of artificial traces is generated that will be used as the training data for the word2vec models. Since every interaction is assumed to be associated with one concrete user, we know that a potential case can only ever contain interactions from a single user. Because of this, the preprocessed interaction log L' from the previous phase is split into several sub-logs L_u initially. Each of the sub-logs L_u then contains all of the user interactions that are associated with a specific user

$u \in U$. Solving the case attribution problem for the complete input log L' then corresponds to solving it for each of the sub logs and combining the segmentation functions. In the reference implementation, the interactions in the dataframe are grouped based on the user property after filtering. During grouping, the sequence of actions of each user is reduced to a list of actions. Each of these lists represents the long connected trace of a specific user.
For generating the training data, all of the user sub-logs are however considered in the following. The final training log is meant to represent the real UI log as closely as possible, in order to transfer the maximum amount of information about the underlying process. The input log can not be used for training the models directly as information about the real case boundaries is required for the model training. In contrast to the input log, the training log will contain this case information, as it is known at the time of generation.
The artificial traces that make up the training set are generated using a transition system that is created based on the interactions in the input log. Previous work in the field of process mining, e.g. [15] shows that the control flow information contained in a log can be modeled well using transitions systems. In order to transform the UI log into a transition system, a window size and event representation have to be chosen. These parameters depend on the process under observation and have to be set accordingly. Initially a transition system $TS_u = (S_u, E_u, T_u, i)$ is created for each sub-log L_u where for the set of activities E_u it holds that $E_u = A$ where A is the set of actions and i is an artificial initial state that is shared among all of the transition systems. The sets of states S_u and transitions T_u are built based on the sequence of user interactions contained in the sub-log L_u and the chosen abstraction. In order to obtain the transition system of the whole input log, the individual transition systems are combined, leading to the transition system $TS = (S, A, T, i)$ with:

$$\begin{aligned} S &= \bigcup_{u \in U} S_u \\ T &= \bigcup_{u \in U} T_u \end{aligned} \tag{4.1}$$

While creating the transition system, information about the frequency of each transition in the log is also collected. We define a weighting function ω for the transitions $t \in T$ where:

$$\omega(t) = \# \text{ of occurrences of } t \text{ in } L \tag{4.2}$$

This heuristic information is used during the later generation phase in order to estimate the probability of different transition sequences. Additionally, a preprocessing step might be introduced during which infrequent transitions with $\omega(t) < \epsilon$ are removed from the transition system for some threshold ϵ. This can prevent irregular transitions that occur infrequently from being over represented in the training set.

$$\begin{tikzcd}
 & & \texttt{(M,A)} \arrow[r, "1"] \arrow[d, "1"] & \texttt{(A,B)} \arrow[d, "1"] & \\
\texttt{(i)} \arrow[r, "3"] & \texttt{(M)} \arrow[ur, "2"] \arrow[dr, "1"] & \texttt{(A,M)} \arrow[d, "1"] & \texttt{(B,C)} \arrow[r, "1"] & \texttt{(C,M)} \\
 & & \texttt{(M,B)} \arrow[ur, "1"] & &
\end{tikzcd}$$

Figure 4.3: The transition system that is obtained from the three exemplary traces.

Considering the three traces from the exemplary process, the frequency annotated transition system that can be found in Figure 4.3 is obtained by following the described procedure using a past sequence abstraction and a window size of two.

In the reference implementation, a past sequence abstraction with a window size of four is used. The size of the window was chosen in order to strike a balance between the accuracy of the transition system and the required computational effort. A larger window size results in a significant increase in the number of states of the transition system, in turn quadratically increasing the number of paths and therefore also the computation time.
A state of the transition system is encoded as a Python tuple of strings representing the sequence of actions. A transition consists of a tuple of the origin and target state. The states of the transition system are created by sequentially iterating over the long connected trace for each user. A short term memory containing the last observed actions of the trace represents the actions in the window. The state tuple is created from these actions. A transition is obtained by creating a tuple containing the last state and the current state. An example for this, based on a trace from the exemplary log can be seen in Figure 4.4. In order to store the frequency of states and transitions, the tuples are used as the keys for two dictionaries containing the corresponding counts. Whenever a state or transition is encountered, the dictionary count is increased.

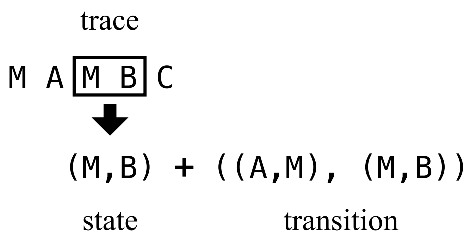


Figure 4.4: A schematic depiction of the generation of the states and transitions of the transition system.

In contrast to transition systems that are created based on logs that are segmented, the obtained transition system will most probably contain states that are not reachable and transitions that are not possible according to the real process. This is caused by the fact that normally, the used state abstraction (e.g., past set abstraction) is applied on a case-by-case basis. Since there is no case information in this situation, the abstraction is applied to the whole sequence of interactions that is associated with a specific user. This inherently leads to the situation that consecutive interactions that belong to different cases will be included as an unwanted transition in the transition system. As can also be seen in Figure 4.4, an example for this is the trace ⟨M, A, M, B, C⟩, which introduces a state (A, M) that is not allowed according to the provided DFG. It is therefore important to remove such illegal transitions from the system. However, this will likely lead to a number of states that are no longer reachable by any of the remaining transitions. In the reference implementation, such unreachable states and the corresponding transitions are also removed from the transition system in order to speed up the computation. The effect of this on the eexemplary transition system can be seen in Figure 4.5.

This removal is performed based on the DFG G that is provided as an additional input alongside the UI log. The DFG might be created by process experts that try to manually assess the validity of the succession of events, or created automatically based on existing

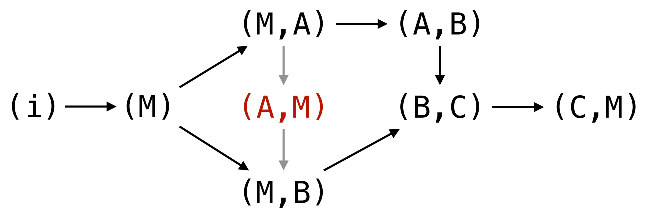


Figure 4.5: The upper light gray transition does not have a corresponding edge in the provided DFG and is therefore illegal. Since the red state can consequently no longer be reached, it is also removed from the transition system. The same is true for the lower gray transition.

information about the process. For example an automated analysis of the source code of the system that is recording the events might be possible. The complexity of this task is depending heavily on the abstraction level that is chosen for the user interactions (e.g. it is easier for screen successions than for single button clicks) and the type of software system.
Based on the edges E_G of G, the validity of all transitions in T can be verified. However, this comparison is not straightforward, because an edge in G corresponds to the succession of two actions, while a transition in T signifies a change of the state of the process. These are two independent events that can not be compared to each other directly. The type of comparison between edges and transitions that is conducted, is dependent on the kind of state abstraction that is used during the creation of the transition system. In the exemplary transition system, a state is of the form: (M,A) and a transition of the form: ((M, A), B, (A, B)). The transition signifies that the state (A,B) can be reached from the state (M,A), by performing the action B. This shows that only the two last actions of the state tuples are relevant for the comparison with the DFG. The exemplary transition therefore translates to the edge (A,B) in the DFG. Using this definition, for the reduced transition system $TS' = (S', A, T', i)$ and the DFG $G = (V, E)$, it holds that:

$$\begin{aligned} T' &= \{((..., a_1), a_2, (..., a_1, a_2)) \in T \mid \exists\, (a_1, a_2) \in E\} \\ S' &= \bigcup_{(s_1,a,s_2)\in T'} \{s_1, s_2\} \end{aligned} \tag{4.3}$$

In order to further speed up the following computations, the representation of the transition system in the reference implementation is changed to a dictionary that encodes the adjacency list of the states. In an additional optimization step, a dictionary is computed, that for each action contains all the states of the transition system that correspond to this action.
The reduced transition system TS' now functions as the foundation for the following generation of the artificial training log. The main idea behind the generation phase is to randomly select a path between two states in the transition system and generate corresponding events that portray that path, transition by transition. The procedure that is used in order to select paths in the transition system is hereby critical and determines how closely the final training log resembles the real process. The randomly selected paths are chosen based on a probability measure that tries to estimate the likelihood of each path. In order to do so, the weighting function ω that was created alongside the transition system is used. The likelihood of a path is calculated by combining the probabilities of all

search (BFS) is performed starting at this state, by continuously expanding the reachable states of each considered state, based on the adjacency list that was computed previously. Since loops are possible using this approach, an infinite number of paths may be discovered. In order to prevent this, the expansion stops when the likelihood of the extended path falls below a chosen threshold. In order to not disadvantage longer paths disproportionately over shorter paths, a factor is introduced that takes the length of the path into account. When there are no states left to expand, the list of all paths is returned.
One of the paths is now randomly chosen based on the distribution of the probabilities of the paths (line 5). As previously discussed, each state of the chosen path can be translated into a corresponding user action of A. For example, the state (M, A) from the exemplary transition system translates to the action A. Different paths between the states of the transition system may translate to the same sequence of actions, an example of this can can be seen in Figure 4.6. For the generation of the artificial traces, the underlying states are however not important. Such redundant paths are therefore reduced to their corresponding sequence of actions, which speeds up computation.
The obtained sequence of actions is translated into a sequence of individual events by creating a mock event for each state of the path (lines 7-10). The activity of each event is the action that corresponds to the considered state. Beginning with the start event, the timestamps of the created events are repeatedly increased by a fixed period of time (line 9). This ensures that the events in the log can be sorted correctly. The chosen time gap in between the events is therefore not of importance here.

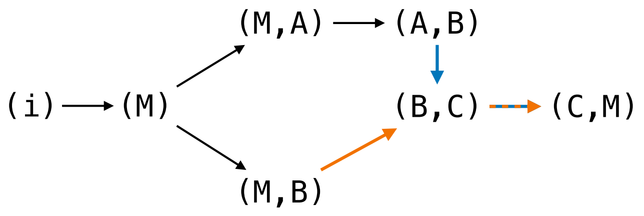

$$\begin{tikzcd}
 & & \text{(M,A)} \arrow[r] & \text{(A,B)} \arrow[d] & \\
\text{(i)} \arrow[r] & \text{(M)} \arrow[ur] \arrow[dr] & & \text{(B,C)} \arrow[r] & \text{(C,M)} \\
 & & \text{(M,B)} \arrow[ur] & &
\end{tikzcd}$$

Figure 4.6: The blue and orange paths in the transition system do not share the same state sequence, since they do not have the same start point. Both of the paths are however translated to the same sequence of actions: B, C, M. Only this action sequence is relevant for generating the artificial traces.

The procedure described above is executed repeatedly until the desired number of cases is generated (line 3). In order to differentiate between the events of different cases, a consecutive increasing case identifier is assigned to all of the events that were generated from the same path (line 12). Events with different case identifiers are therefore ensured to actually belong to different cases. The resulting events are collected in the final segmented training log L_T which is the output of this step of the proposed method. In contrast to the input log, the training log is segmented and therefore contains reliable case information that can be used for the training of the word2vec models. The size of the training log L_T can be freely chosen and only depends on the number of cases that are generated. In general a larger log is better for training, however the generation of an artificial case using the proposed procedure is computationally demanding (this is explored in more detail in chapter 5). Therefore, a balance between log size and time effort has to be found for every application.
Despite the already introduced optimization efforts, the number of paths that are visited during the path generation procedure remains significant. In order to reduce the required

computation time, a caching system is introduced. Every action sequence and its frequency is added to this cache when it is first encountered. When the same start state is chosen during a later iteration, the already computed paths for this state are loaded from the cache, instead of performing the expensive path computations again. This reduces the number of required path calculations significantly, especially for later iterations, as the probability that the chosen start state is already in the cache increases. Experiments with this system showed that it can reduce the number of required path calculations by up to 90%, thereby greatly accelerating the log generation.

The presented procedure for the generation of the training log is only one of many possible approaches to artificial log generation. In general, the method proposed in this paper requires a base log that contains correct case information as a foundation for the training of the models. How this training log is obtained is not as important and may also be dependent on the type of process that is under consideration. Other process abstraction models than transitions systems might provide better results for process data with other characteristics than UI logs. In cases in which a normative model of the process, created by domain experts, such as a BPMN model, process tree, or petri net exists, play-out can be used in order to obtain an accurate training log in a straightforward way.

4.4 Model Training

The training log that is obtained in the previous step of the method is now used in order to train the underlying models. As introduced before, the core component of the proposed method consists of a range of different word2vec models that are used in order to detect the boundaries between cases in the input log. When applied in the context of natural language processing, the input of a word2vec model is a corpus of sentences which consist of words. Instead of sentences built from words, we here consider traces made up of actions. We therefore do not consider the overall interactions, but only the corresponding ordered sequence of actions $\langle a_1, ..., a_n \rangle$. For example the trace ⟨M, B, C, M⟩ from the exemplary process.
The training log L_T that was generated in the previous phase of the method is the basis for the model training. L_T however cannot be used directly for this purpose, as it does not have the format of sequences of actions that was described above. Instead of the long connected traces of actions, that make up the input logs L_u, the training log consists of individual events. In order to obtain the required connected traces from the training log, events belonging to multiple different cases are combined. This is done in the reference implementation by randomly shuffling the traces in L_T and connecting all pairs of subsequent traces. An example of this, based on the exemplary process can be seen in Figure 4.7. Instead of only pairs of traces, different numbers of traces are connected in order to more accurately reflect the input log, in which a user may have performed any amount of cases.

For each of the selected cases, the corresponding trace e.g., ⟨M, B, C, M⟩ is created from the associated events. After the last event of each case, an artificial end activity a_{end} is inserted. For the example trace this results in: ⟨M, B, C, M, a_{end}⟩ Finally, the augmented traces of the selected cases are joined, resulting in a single long trace, for example: ⟨M, B, C, M, a_{end}, M, A, B, C, a_{end}⟩. This trace is then inserted into the structured training set L'_T. The random sampling and joining of the traces that was described above is performed

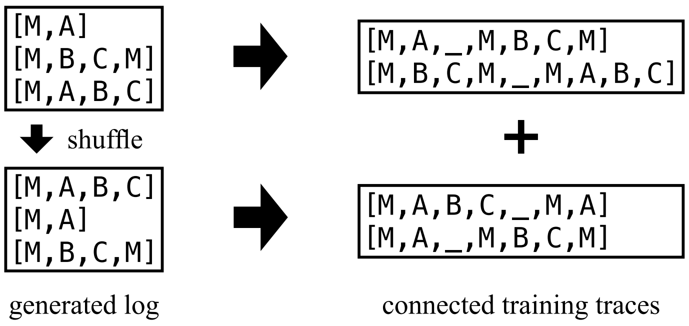


Figure 4.7: The procedure used in order to generate the input for training the models.

repeatedly until the desired number of traces has been generated.
The goal of this sampling is to create traces that have the same structure as the traces of the input log, long connected traces, consisting of several cases for each user. The critical difference between the input log and the structured training log L'_T is, that in the training log, the actions belonging to different cases are separated by the artificial end action. During training, the word2vec models will be learning about the relations between this end action and the other actions of the log.

As described in Section 2.3, the word2vec models consist of three different layers. The input layer, the output layer and a single hidden layer. During training, the model is provided with the structured training set L'_T. A sliding window is used in order to separately iterate over each trace. The action that occupies the central position of the sliding window is called the *center action*. The surrounding actions that lie in the window are the so-called *context actions*. The proposed method uses the CBOW variant of word2vec architecture. As introduced before, this means that the context actions are used as the input of the network in order to predict the center action. The input vector A therefore represents an encoding of the frequency with which the actions occur in the sliding window. A is multiplied with the weight matrix $W_{A,E}$ and the sigmoid activation function is applied. The resulting embedding vector E is multiplied with the weight matrix $W_{E,C}$ and the sigmoid function is applied once more. The final output vector C contains the likelihood of all actions in the vocabulary, given the context words encoded in A.
As mentioned before, the error that is made in the output layer is used during training in order to adjust the weights in the weight matrices $W_{A,E}$ and $W_{E,C}$ using the back propagation algorithm. The resulting forward and backward steps are repeated for all positions of the sliding window and for all traces in L'_T. When training is complete, the weight matrices will be tuned in such a way, that the model is able to predict the most likely center action for the given context actions, based on the relations in L'_T. An example activation of this network can be found in Figure 4.8.

There are different parameters influencing the characteristics of a word2vec model. The size of the sliding window has been mentioned before. This parameter influences how much of the context surrounding an action is considered. While a small window size will be able to portray frequent variants accurately, longer relationships between the actions are not captured. A larger window size allows the model to comprehend structures such as loops up to a certain length. However, models with larger window sizes are susceptible to noise and generally require larger training sets.
Another important parameter is the dimension of the embedding vector. It can be freely

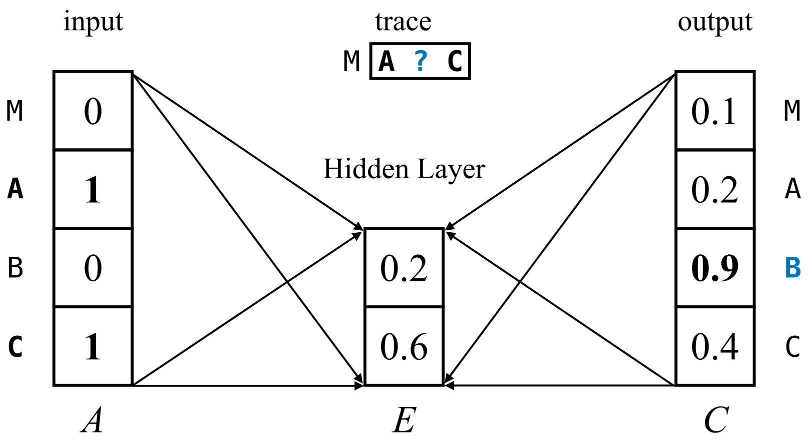


Figure 4.8: An example activation of the shallow neural network based on the trace ⟨M, A, ?, C⟩ from the exemplary process. The active window is indicated by the box, B is the center action that is to be predicted, while A and C are the context actions. The predicted center action is marked in blue.

chosen, but mainly depends on the size of the vocabulary. Compared to the field of natural language processing, where the vocabulary frequently consists of all words of a language, it will generally be significantly smaller in the context of process mining. Increasing the embedding dimension results in larger weight matrices, which in turn increases the amount of required training data and the number of training iterations. Patel et al. have shown that there is a lower bound for the embedding dimension that depends on the concrete application [45]. This should therefore be exceeded. Because the model training will in most cases take a considerable amount of time, the models are trained only once and subsequently saved to disk for later reuse in the reference implementation.
In general, there is no optimal set of parameters for all different kinds of logs. Because different parameter combinations result in models with varying strengths and weaknesses, multiple word2vec models based on different parameter sets are trained. The training phase of the method therefore results in k different word2vec models $M_1, ..., M_k$. During the scoring phase, the outputs of the models are combined in order to achieve a result that overall is significantly better than that of any of the standalone models.

4.5 Scoring

The k models that were created in the previous phase can be used in order to assess the probability of a certain action at any given position in a trace. Because of the artificial end action a_{end}, the likelihood of a case boundary, given a set of context actions $a_1, ..., a_n$ can also be calculated. In order to do so for every position in the trace, a sliding window is used again. For a single model, the likelihood of a case boundary at a certain position, is calculated based on the embedding vectors of the context actions. The mean e_{mean} of these embedding vectors $e_1, ..., e_n$ is calculated in the following way:

$$e_{mean} = \frac{1}{n} \cdot \sum_{i=1}^{n} e_i \tag{4.6}$$

The obtained vector e_{mean} is multiplied with the weight matrix $W_{E,C}$ and the sigmoid function is applied. The output vector C_{mean} contains the likelihood of all actions given

the context actions, including that of a_{end}. The possible values range from zero to one, where values close to one signify that the corresponding action is probable. In the reference implementation, this likelihood is calculated using the `predict_output_word` function provided by gensim.
We denote the likelihood of an action a according to the model M_i, given the context actions $a_1, ..., a_n$, as $l_i(a, (a_1, ..., a_n))$. The combined likelihood of a case boundary at a given position in the connected trace considering all k models is calculated in the following way:

$$l(a_{end}, (a_1, ..., a_n)) = \sum_{i=1}^{k} w_i \cdot l_i(a_{end}, (a_1, ..., a_n)) \tag{4.7}$$

Where w is a weight vector that is used in order to weigh the impact of different models on the final score differently. It holds that $\sum_{i=1}^{k} w_i = 1$. The likelihood function defined in this section is used in the next step of the method in order to assess the probability of a case boundary at every position in the connected trace.

4.6 Segmentation

For the final segmentation phase in which the actual case attribution is performed, the connected traces of all users are considered separately. Based on the scoring method that was described in the prior section, the likelihood of a case boundary at a given position in the connected trace can be estimated. For a connected trace $\langle a_1, ..., a_n \rangle$ of length n, we denote the obtained score at a position i as s_i. Plotting the sequence of scores $\langle s_1, ..., s_n \rangle$ results in a graph, that will be the basis for segmenting the input trace. An example for such a graph based on a connected trace from the exemplary process can be seen in Figure 4.9.

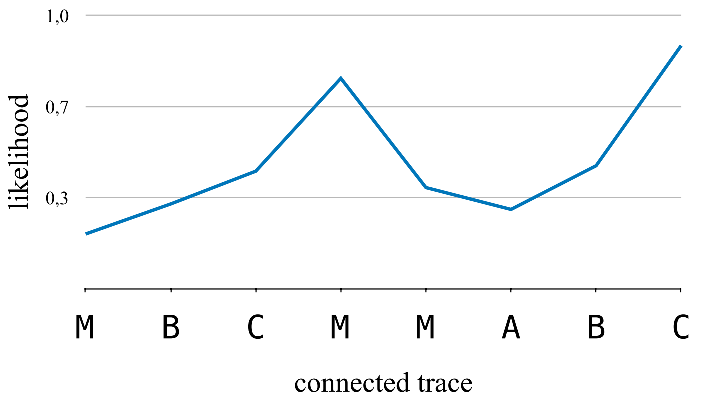


Figure 4.9: An example for the scoring graph that is the basis for the segmentation. A higher score signifies that a case boundary is more likely at the following the position. Prominent peaks are strong indicators for case boundaries.

Different approaches to converting the score graph into a segmentation of the log are possible. One approach that showed a good performance during the implementation of the proposed method is described in the following. The main idea behind this approach is to focus on areas in the graph in which sudden changes in the likelihood occur. These locations can be described as prominent *peaks* in the score graph. Applying this to the example graph in Figure 4.9, a case boundary would be identified after the C action. Another peak can be observed in between the two M actions, it is less prominent but does also correspond to a case boundary in reality. In order to obtain a good segmentation, it is important to be able to distinguish between different kinds of peaks. We identify a prominent peak at position i in the sequence of scores $\langle s_1, ..., s_n \rangle$ when the following conditions hold:

$$\begin{aligned} s_i &> b_1 \cdot s_{i-1} \\ s_i &> b_2 \cdot s_{i+1} \\ s_i &> \frac{b_3}{k} \cdot \sum_{j=(i-k-1)}^{i-1} s_j \end{aligned} \tag{4.8}$$

Where $b_1, b_2, b_3 \in [1, \infty)$ are hyper parameters that influence the sensitivity of the segmentation algorithm. The first two inequalities ensure the peak property of the considered position compared to the direct predecessor and successor. The parameters b_1 and b_2 can be used in order to influence the required prominence of the peak. Usually it will hold that $b_2 \geq b_1$ as a large drop after the peak is a more important feature of the peak than the rise that leads up to the maximum.
The third inequality is introduced in order to ensure that not only the relation to the neighbouring scores, but also the absolute score value is considered. In order to do so, the score at the examined position is compared to the mean of the k preceding scores. This ensures that the peak is actually protruding significantly from its larger surroundings. The additional parameter b_3 is used in order to control the magnitude of the required protrusion.

Based on this, for each user u and the corresponding sequence of scores $\langle s_1, ..., s_n \rangle$, all positions that adhere to the criteria introduced above are identified. A position that meets all three conditions is added to the set of candidate split points $P_{\text{candidate}}$. In a last step, all of the identified candidate split points are checked for their plausibility. We know that a trace has to consist of at least one action. Candidate split points that directly follow each other are therefore not plausible. If such a situation is discovered for two candidate split points, only one of the positions is added to the final set of split points P. This set is used in order to perform the actual segmentation of the user logs L_u, in which all interactions that lie in between two case boundaries are assigned the same case identifier.

In the reference implementation, the segmented dataframes of the different users are joined and a `case:concept:name` column that holds the computed case identifiers is inserted. The joined dataframe is then exported into a CSV file and saved to disk. The dataset that results from the described segmentation implementation will be identical to the input dataset, except for the additional case identifier column. The segmented output interaction log can now be used for further analysis of the underlying process using conventional process mining software and techniques. This is explored in the conducted case study being discussed in Section 5.2 in the following chapter. The chapter also contains a comparison between the introduced method and two other case attribution approaches.

Chapter 5

Evaluation

5.1 Artificial Test Cases

In this section, the quality of the log segmentations that can be obtained using the proposed method is evaluated under different conditions using a set of different test logs. The resulting segmented logs are compared against the true case information in the input log and against segmentations that are produced by two different baseline segmentation approaches. In the following, the generation of the test logs, the adjustable log characteristics, the baseline approaches used for comparison and the evaluation metrics are introduced.

5.1.1 Test Data

In order to be able to adequately assess the quality of the segmentations that are produced by the proposed method, the computed case boundaries need to be compared to the real boundaries. This is however not possible for the overwhelming majority of real interaction logs, since they contain no case information. In general, there exist no freely available user interaction logs that contain case information. In order to be able to accurately assess the performance of the proposed method compared to other approaches and in different settings, a number of artificially generated interaction logs are therefore used for the test cases. These logs contain case information, and the characteristics of the log (e.g. size, trace length, etc.) can be varied during generation. This enables the comparison of a wide range of different types of logs and allows to observe how the log characteristics influence the performance of the proposed method.

The log generation is fundamentally based on a set of different user actions and an adjacency list, that for each action determines which actions can happen directly after it. This corresponds to a DFG that describes the underlying process. For each relation between two actions that is described by this graph, a probability is defined. The probability of all outgoing edges of a node sum up to one. In addition to this graph, for each action it is defined with which probability it is the start or end action of a task of the user. Different parameters can be used in order to influence the characteristics of the log. These parameters include the number of traces, the number of users, the most common trace length, the probability that a task is interrupted, the probability that an interaction is delayed and the time in between different traces of the same user. These parameters influence the type of log that will be generated and can be used in order to model various different situations. A trace is generated by selecting a random start action according to the given probabilities, and gradually adding actions to the trace based on the given parameters. In addition to the sequence of actions, other attributes such as timestamp and user information are also generated for each interaction. The timestamps are generated by increasing the timestamp of the previous interaction by a random interval that is drawn from a pareto distribution. The used distribution is based on an expected action duration that is defined for all actions. The user identifier that is assigned to all events of a trace is randomly chosen based on the number of users that should be present in the log according to the corresponding parameter.

In order to simulate different contexts and log types, several different artificial logs with e.g. varying size, number of users or disruption probabilities are generated and used during the comparison. All of the logs model the behaviour of a fictional public transport app that can be used to access the time schedule of a public city bus provider. It offers the functionality to view current departures at certain stops, plan a journey between a start and destination and displays all bus stops of the provider on a map. The abstraction level that is used for the interactions is that of a screen view. A screen in the context of a mobile application is an interface that takes up the majority of the available display area and mainly serves some specific functionality. In general, most of the major actions that a user may perform will lead to a transition between different screens. A screen view is recorded in the log whenever the application programmatically displays the corresponding user interface. A screen might be displayed automatically, however in the majority of cases the transition is caused by an interaction of the user. Because of this, the sequence of screens that a user visits is well suited in order to retrace the path that the user has taken through the application. The artificial logs used during the evaluation are made up of 14 of such screens. An overview over the basic structure of the mock application can be seen in Figure 5.1.

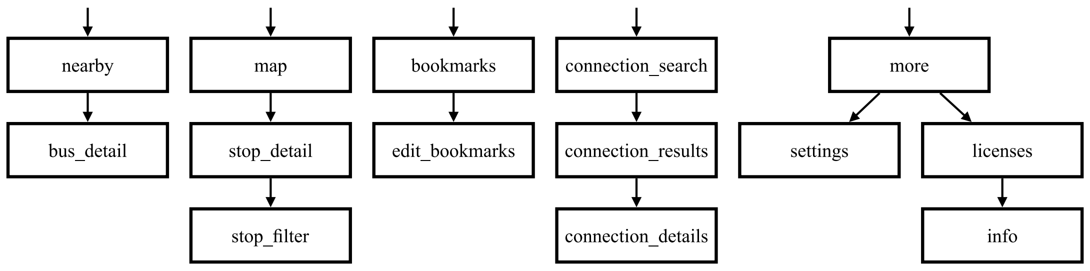


Figure 5.1: A schematic representation of the different screens (actions) that make up the activities in the artificial test log. Only the basic hierarchy is shown, there are additional allowed transitions between the screens.

The mock application has five main screens that act as an entry point and expose different types of functionality. However, there is a number of additional transitions between the screens. Most screens can be reached from different starting points within the application. The generated interactions in the test logs have the minimum amount of attributes that are required for process mining: a case identifier `case:concept:name`, an identifier for each screen `concept:name` and the timestamp at which the interaction took place `time:timestamp`. Additionally, each event is associated with a user through the `org:resource` attribute.

5.1.2 Baseline Approaches

Different baselines are used during the evaluation in order to compare the performance of the proposed method with that of more simple approaches to the case attribution problem. In the following the two baselines are introduced, one of which is based on the directly-follows relations and the log, while the other considers the timestamps between the actions. This naive but commonly used approach for case attribution is based on the time that has passed between two consecutive actions and perform the segmentation based on this. The most simple way to perform this comparison is to set a certain time threshold based on which the split points are detected. [37] If the observed duration at a certain position is larger than this threshold, a case boundary is identified and the trace is split. This approach is commonly used for case attribution and related problems such as sessioning in web mining. While this approach is able to reliably identify certain kinds of case boundaries, it cannot distinguish between two cases that occur directly after each other and cases in which actions take significantly longer than expected.

Heuristic Baseline A more refined variant of this time-based approach does not consider a static threshold for all types of actions, but rather uses a variable threshold that changes for every pair of subsequent actions. This heuristic approach is able to capture the nuances and differences between the different actions of the log. Different heuristic functions can be employed in order to determine the used thresholds. Commonly used metrics include the mean or median time between two actions in the overall interaction log. The chosen heuristic is computed for all of the directly-follows relations in the log, the value obtained for a relation acts as its threshold. In general, the median is preferable to the mean, as infrequent but large gaps between events in the unsegmented log may disproportionately influence the mean.
The heuristic approach suffers from similar problems as the simple time based threshold that were described beforehand. It is however able to model the log more closely and therefore generally performs significantly better. Because of this, it is used with the median as the heuristic function as one of the baseline approaches in the following comparisons.

DFG Based Baseline As introduced before, the proposed method is provided with a DFG of the underlying process as an additional input. The information contained in this graph is important for the generation of the training traces and therefore influences the overall performance of the proposed method. One might therefore consider to directly use the DFG for segmenting the input log, saving the further steps of the proposed method. When a user stops interacting with the software system for some time, or even quits it completely and later resumes its usage, illegal transitions may be introduced. For example the user might leave the system while displaying a certain sub menu but start again at the main menu, even if this transition is normally not possible. These kinds of case boundaries in the log can be detected by splitting the trace whenever the sequence of two subsequent events does not correspond to an edge in the DFG.
The DFG based baseline approach that is described above is also part of the conducted evaluation and will be compared against the heuristic baseline and the proposed method in the following. This is done in order to assess if the proposed method has an unfair advantage over other approaches, because it is provided with the additional information in the DFG and if this is the main contributor to the overall segmentation performance of the method.

5.1.3 Evaluation Metrics

A number of metrics are used during the evaluation in order to be able to compare the different segmentations that are generated by the proposed method and the two baseline approaches. Some basic descriptive metrics that are based on the core properties of a segmentation, such as the number of correctly identified case boundaries, are introduced in the following. The metrics are calculated based on some fundamental frequencies that can be obtained by comparing the identified case boundaries with the real boundaries. A *true positive* is a position in the connected trace that corresponds to a real case boundary and was correctly identified as such by the case attribution algorithm. The number of true positives in the overall log is denoted as TP. In a similar manner we define *true negatives* (TN) as the positions that are not corresponding to case boundaries and were identified as such, *false positives* (FP) as positions that were identified as case boundaries but are not actually boundaries and *false negatives* (FN) as real case boundaries that the algorithm was not able to identify. All of the metrics that are presented in the following can take values from a range between zero and one, where values closer to one indicate a better performance. Based on these frequencies we define the *accuracy* of a segmentation as:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

The accuracy can be used to judge the overall ability of an approach to distinguish between positions that correspond to a case boundary and those that do not. In the context of case attribution, it is important to consider the fact that positions that correspond to case boundaries are very infrequent compared to those that do not. Because of this, a high value for the accuracy can already be achieved by not predicting any case boundaries at all, which is of course not the desired behaviour. In order to obtain a more detailed impression over the specificity of the examined approaches, additional metrics are introduced. The *precision* of a segmentation result is defined in the following way:

$$precision = \frac{TP}{TP + FP} \tag{5.2}$$

It can be used in order to judge what percentage of the predicted case boundaries actually correspond to a real case boundary. An approach that is able to identify a large fraction of the real case boundaries, but produces many false positives will have a low precision. How good an approach is at reproducing the real boundaries can be judged based on the *recall*. The recall of a segmentation is close to one when the approach is able to identify most of the actual case boundaries. It is defined as:

$$recall = \frac{TP}{TP + FN} \tag{5.3}$$

A high recall can be trivially achieved by predicting a case boundary for every position of the trace. The other metrics that were introduced beforehand can be used in order to mitigate this effect. Such a segmentation will have a high recall but will produce a low precision value. In order to capture this relation between precision and recall in a single metric, the f_1-score is used. The score tries to mediate between the precision and recall metrics by penalizing segmentations that score high according to one of the metrics, but low according to the other. The f_1-score is calculated as follows:

$$f_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.4}$$

Additionally, extensions to the metrics that were defined above are introduced in order to better capture the intricacies of the case attribution problem. For the overall performance of an approach, identifying the exact position of a case boundary is not necessarily the most important task. An approach that is able to identify the general area in which a transition between two cases occurs provides a significantly higher amount of information than one that misses the boundary completely. Such an approach will at least be able to detect a number of cases that is close to the real number of cases, even tough the predicted boundaries might not be aligned exactly. In order to account for these near misses, relaxed variants of the precision, recall and f_1 metrics are introduced. We call these metrics the *weak* variants of the original ones. The difference in calculation is the way that the underlying frequencies TP, TN, FP and TP are defined. Rather than only considering the exact positions in the trace, their immediate surroundings are also taken into account. For a position in the trace that corresponds to a real case boundary, a segmentation produces a true positive if either the position itself, or the position directly before, or the position directly after it is predicted to be a case boundary. An example for this can be seen in Figure 5.2. This definition naturally weakens the requirements for the metrics, but is able to better account for the general utility that a near miss provides.

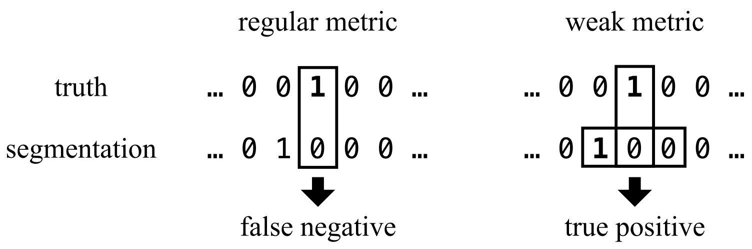


Figure 5.2: The computation conducted in a weak metric compared to that of a regular metric. The sequences encode the positions that correspond to case boundaries. The judgement changes from a false negative to a true positive when applying the weak metric.

In addition to these weak metrics, a metric that can be used in order to compare the number of cases that are produced by a segmentation with the number of cases that are actually present in the log is introduced. We call this metric the *case number deviation* (cnd) which is defined like this:

$$cnd = \frac{\#\text{ of predicted cases} - \#\text{ of real cases}}{\#\text{ of real cases}} \tag{5.5}$$

The values of the cnd do not fall in a certain range and values closer to zero, either in the positive or negative direction, are better. A perfect segmentation will have a score of exactly zero. The cnd is used in order to judge how well a segmentation can reproduce the number of cases in the log, rather than the exact boundaries.

5.1.4 Evaluated Parameters

In the following, the parameters that were considered during the evaluation are introduced and it is explained why they were selected for a closer examination. Additionally, the concrete values that were considered for each parameter are presented and discussed.

Number of Cases The number of cases generally corresponds to the size of the interaction log. In real applications, there will be contexts in which a large amount of data can be recorded and other situations in which only a limited amount of interaction data is available. In order to assess the performance of the different approaches when provided with varying amounts of data, four different interaction logs with varying amounts of traces are compared. The logs include 100, 1,000, 10,000, 50,000, and 100,000 cases. For the largest log, this means that it consists of about 800,000 events, while the smallest contains about 800. The test logs therefore cover several orders of magnitude.

Ratio of Users And Cases The ratio between users and cases in the log is especially relevant for the proposed method, since it relies on the user information in order to estimate the start and end probabilities of the different actions. In order to observe how the method reacts to a varying amount of users, four different test logs were used. The four logs have a constant amount of 10,000 cases that are related to either 100, 1,000 5,000 or 8,000 different users. Since one case is always related to exactly one user, a larger number of users results in fewer cases per user. If there is the same number of users and cases, each case is uniquely identified by the associated user identifier. A ratio of 80% was therefore chosen as the upper bound for this test parameter.

Case Length When the average case length is increased and the number of available actions remains constant, the probability for repeated actions and loops increases. Such action sequences are especially challenging for case attribution, as it is not obvious which of the repeated occurrences corresponds to a boundary. The case length therefore is an interesting target for evaluation. It can be influenced during the generation by a parameter that determines the most frequently occurring case length. Four different test logs with the most frequent case lengths 3, 8, 12 and 15 are used. Since the underlying process consists of 14 actions, this guarantees repeated actions for most of the traces in the log with the longest traces.

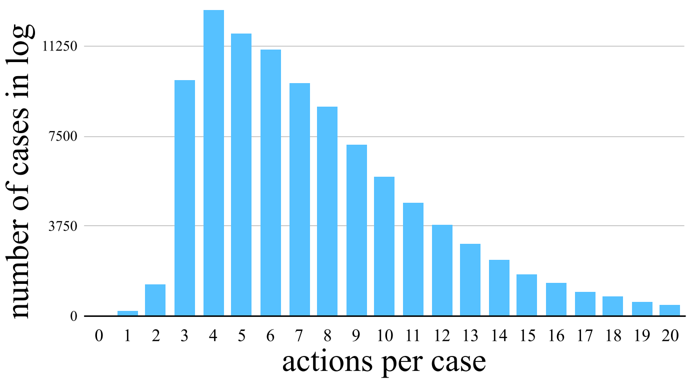


Figure 5.3: An example for the case length distribution in a test log that has a most frequent case length of four.

Fraction of Delayed Interactions We call an interaction that is in some form interrupted by an outside event, but is later resumed *delayed*. Such delays are especially relevant in a mobile context, since distractions are common. An interaction may be interrupted when the awaited bus arrives, when the user receives a call, or needs to check their calendar. In all of these situations the action duration will be longer than usual. Delayed interactions are especially challenging for time based case attribution approaches, including the heuristic approach that is considered during the evaluation. In order to investigate the impact of the frequency of delayed interactions on the evaluated methods, five different test logs are used. They differ in the probability that an interaction is delayed and cover the probabilities 0%, 1%, 10%, 50% and 80%.

Fraction of Abandoned Interactions *Abandoned* interactions are similar to delayed interactions, with the difference that the user does not return to complete their task after some time, but rather completely abandons it. Such case boundaries are difficult to detect for case attribution algorithms, because they will appear without any context and can end with actions that are not usually end actions. During the generation of the test logs, there is a certain probability that an interaction is abandoned. Five different logs with the probabilities 0%, 1%, 10%, 50% and 80% were used.

Standard Parameters In the introduced test cases, one of the log characteristics is varied, while the other parameters remain constant. For all parameters that are not currently under investigation in a certain test case, the values that are described in the following are used. These values are chosen based on the expected characteristics of the described sample process. The number of cases in each log is set too 10,000, as the conducted test showed no change in the performance of the methods for larger logs. The most common case length is set to four, since the sample application is rather simple and used for only quick interactions by the users. The probability that an interaction is delayed is set to 10%, since a mobile application is modeled in which outside distractions are relatively common. The probability that an interaction is abandoned completely is set to 1%. An additional parameter that is not considered separately during the evaluation, is the time that passes between two cases of the same user. The distribution from which this delay is drawn will be discussed in more detail in later sections.

5.1.5 Results

In this section, the evaluation results that were obtained by applying the three different case attribution approaches to the artificial test logs are presented. The presented results are grouped by the log characteristic that was varied between the different test logs.

Number of Cases For the proposed method, the number of cases in the unsegmented input log determines the amount of training data that is available for the word2vec models and might therefore significantly influence its performance. Figure 5.4 depicts the accuracy of the three evaluated methods when applied to the five differently sized test logs. We see that an increasing number of cases in the test log has no significant impact on the accuracy of the computed segmentations for the two baseline methods. This is expected generally, because these methods do not have a training phase that could be negatively impacted by a lower amount of available data. This is especially true for the DFG based baseline method. The heuristic baseline approach uses the information in the log in order to compute the median duration between the actions. A larger log means that these thresholds are based on a larger amount of directly-follows relations and may therefore be more accurate. Contrary to that, we observe that the heuristic baseline performs best for the smallest log. This may be explained by the fact that for most of the transitions, the computed threshold corresponds to the actual duration, because most transitions will not occur repeatedly in such a small log.

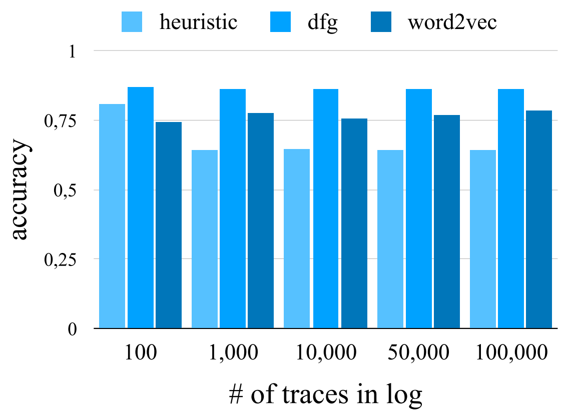


Figure 5.4: The accuracy of the test log segmentations that the were produced by the heuristic baseline, the DFG based baseline and the proposed method.

For the proposed method, a minor increase in accuracy can be observed with increasing log size. The rise in accuracy is most noticeable between the log with 100 traces and the log with 1,000 traces and flattens after that, this might suggest that the log containing 1,000 traces may already provide enough training data for the proposed method. The DFG based method has the highest accuracy for all of the test logs. The proposed method overall has the second highest accuracy and the heuristic approach the lowest.

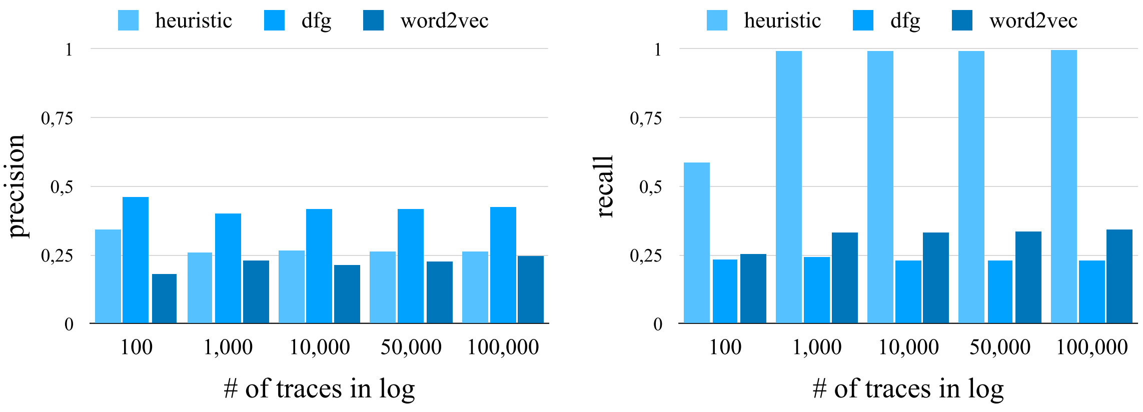


Figure 5.5: The computed precision (left) and recall (right) of the different segmentations that were produced by the methods that are under consideration, based on the five different test logs of varying sizes.

The precision and recall of the compared approaches can be found in Figure 5.5. We see that the DFG based approach has the highest precision for all logs. This is expected, as an illegal transition between two actions is a strong indicator for a case boundary. The heuristic approach has the second highest precision and the proposed method the lowest. The difference between these two methods is however much less significant. For the proposed method, the biggest increase in performance can again be observed between the logs that contain 100 and 1,00 traces. For the heuristic baseline, a stronger precision but worse recall can be observed for the smallest log compared to the larger logs. This further supports the observations that were made regarding the accuracy. For the larger logs and the DFG baseline, the size of the test log does not seem to influence the precision in a significant way.

In a similar way, no strong correlation between the log size and the recall can be observed. The recall of the heuristic approach is close to optimal and is orders of magnitude better than that of the other approaches. While the recall of the proposed method is higher than that of the DFG baseline, the difference is much less significant. The high recall of the heuristic baseline could generally be explained by a lack of precision; this is however not supported by the obtained results. Additionally, the way that the artificial traces are generated might explain the observed recall of the heuristic baseline. Two cases of the same user are separated by a random time interval drawn from a normal distribution with a mean of one hour. Since the average duration of the actions lies in the range of seconds, a large time gap in the range of tens of minutes is a strong indicator for the heuristic baseline approach. This gives it an unfair advantage over the other methods, since the log contains less tasks that occur directly after each other than expected in a real interaction log. In order to better account for this, the timestamp generation was adjusted for the following test cases by using a multimodal distribution, considering smaller and longer gaps between cases.

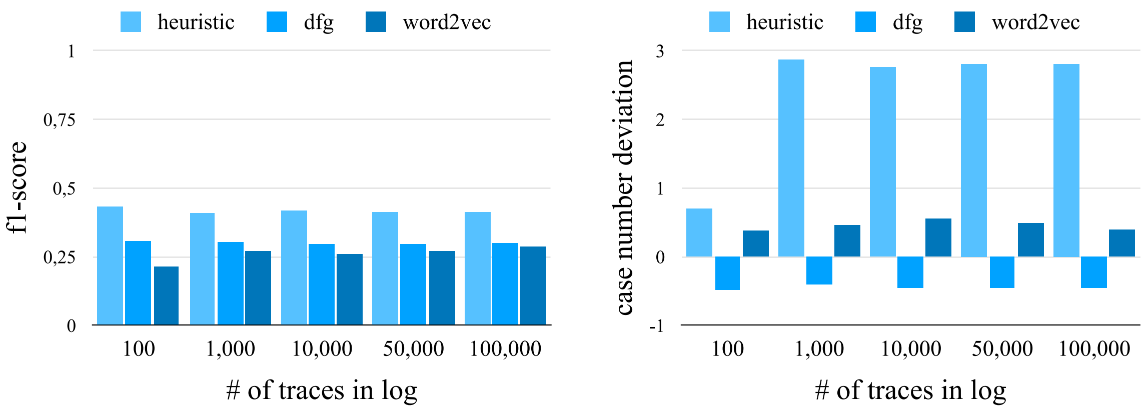


Figure 5.6: The computed f1-score (left) and cnd (right) of the different segmentations that were produced by the methods that are under consideration, based on the five different test logs with a varying sizes.

An important indicator for the quality of a segmentation is the number of identified cases compared to the original log. In order to be able to quantify this, the cnd metric was introduced. In Figure 5.6 we see that the heuristic baseline has a cnd that deviates significantly from the optimal value in most test cases. This indicates that the approach introduces many case boundaries that do not exist in reality. The average length of the traces in the log will be reduced by this, possibly causing problems during the later analysis. Again, a significantly better performance of the heuristic baseline can be observed for the smallest test log. The other methods have a cnd that is much closer to optimal. In contrast to the other approaches, the DFG baseline produces a smaller number of cases than the

original logs. This is most probably caused by the fact that illegal transitions are relatively infrequent in general. The cnd does not change significantly depending on the log size for any of the methods, only a minor increase can be observed for the proposed method. The increase is again most noticeable between the logs of size 100 and 1,000.
Overall, it can be observed that the performance of the proposed method increases between the two smallest logs and then remains mostly constant, this is also supported by the f1-score that can also be seen in Figure 5.6. This suggests that 1,000 traces provide enough information to the proposed method so that it can function correctly, while 100 traces are not sufficient.

In the presented test case, only the classic definitions of precision and recall were considered. The computed cnd values however show that there is a significant difference in the number of cases that were identified by the different approaches. The values for accuracy, precision and recall are not able to fully describe this significant difference. While the heuristic approach has the highest precision and recall, it also produces a segmentation that translates to almost three times as many cases compared to in the original log. Because of this, the weak recall, weak precision and weak f1-score will be used in all of the following comparisons.

Ratio of Users And Cases As introduced in Section 4.3, performing the case attribution for a complete log can be broken down into grouping the log based on the users and segmenting these user sub-logs individually. This is possible because one case always corresponds to exactly one user. This breakdown of the initial task applies to the proposed method as well as to the two considered baseline approaches. The ratio between the number of observed users and the number of cases in the overall log is therefore an important characteristic of an interaction log and will presumably have an influence on the performance of the different methods. Figure 5.7 shows the accuracy of the segmentations that were produced by the three evaluated approaches depending on the number of observed users.

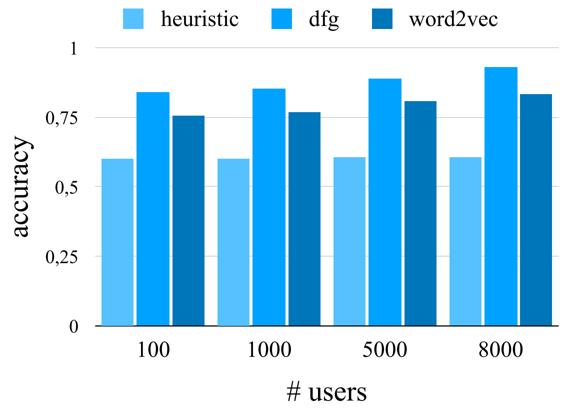


Figure 5.7: The computed accuracy of the different segmentations that were produced by the methods that are under consideration, based on the four different test logs with a varying amount of users.

We can see that the DFG based baseline method has the highest accuracy for all test logs, followed closely by the proposed method. The heuristic baseline has a significantly lower accuracy for all test cases. The accuracy rises with an increase in the number of users

for the DFG baseline and the proposed method. No significant difference in the accuracy between the different test logs can be observed for the heuristic approach. In general, an increasing performance is expected for logs with a higher number of users. This is the case, because each transition between two cases, that are related to different users, corresponds to a boundary that is predetermined for all of the methods. As the user to case ratio approaches one, the log will be trivially segmented by only considering the different users. This can also be observed clearly according to the weak recall values that are shown in Figure 5.8.
One can see, that the weak recall increases together with the number of users for all of the evaluated methods. While the increase is less prominent for the heuristic method, it is very prominent for the other approaches. The lower increase for the heuristic method may be explained by the fact, that it has a high recall in general, therefore there is overall less room for improvement. The opposite is true for the DFG based baseline, which has a low weak recall for the log with 100 users, which strongly increases for the logs with more users. In general, it can be said that the weak recall of the DFG baseline follows the fraction of users in the test log closely. The weak recall of the proposed method also increases with the number of users, however less drastically. Overall, the heuristic baseline has the highest weak recall, followed by the proposed method and the DFG based baseline.

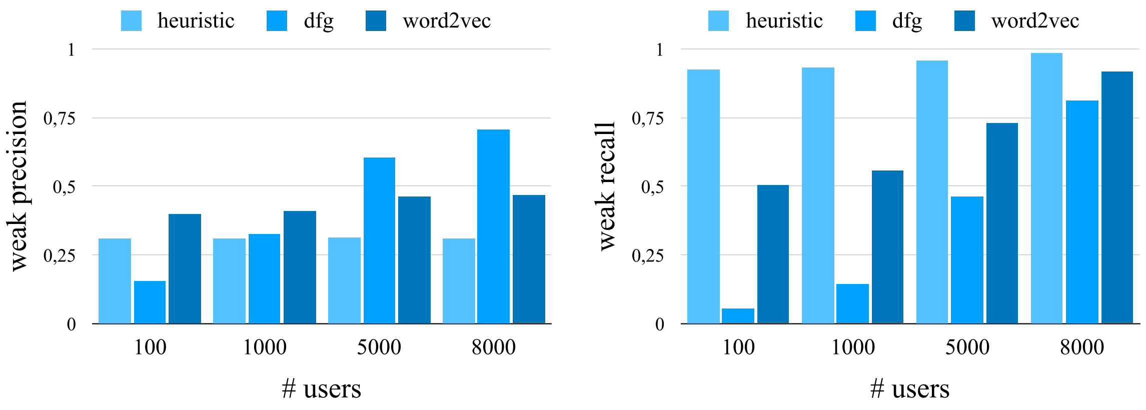


Figure 5.8: The computed weak precision (left) and weak recall (right) of the different segmentations that were produced by the methods that are under consideration, based on the four different test logs with a varying amount of users.

The weak precision shows a similar pattern as the weak recall, it is increasing with the number of users for the DFG based approach and the proposed method. The effect is however not as pronounced as with the weak recall. The heuristic baseline shows no significant changes in between the different test logs. The proposed method has the highest weak precision for the two logs with a lower amount of users, while the DFG based baseline performs better for the two logs with more users.

The results for the weak f1-scores that can be seen in Figure 5.9 do not show a clear picture. In general we see that a larger number of users is beneficial for the performance of all of the methods. As mentioned before, this is mainly caused by the fact that a high user ratio means that less case boundaries have to be identified in total. The proposed method shows the best overall performance for all but the highest ratio logs.
Given that the amount of predetermined case boundaries increases by 60% when the number of users is increased from $5,000$ to $8,000$, one would expect that the weak f1-

score increases by a similar amount. This is approximately the case for the DFG based baseline. The weak f1-score of the proposed method does however only increase by about 10%. This indicates that the gains that are caused by the predetermined case boundaries, are balanced out by a worse performance of the actual segmentation. In conclusion this means that a higher user ratio actually leads to a worse performance of the proposed method. This is unexpected and may be explained by the fact that when the amount of users increases, the number of cases per user decreases. Therefore there are less real case transitions and more transitions between different users in the training dataset of the proposed method.
In the values for the cnd that can also be found in Figure 5.9, the increase in the number of cases that is caused by the addition of the predetermined case boundaries in between the users is clearly visible. The difference between the different logs closely follows the change in the number of predetermined cases for the DFG based approach and the proposed method. The heuristic approach is again influenced less.

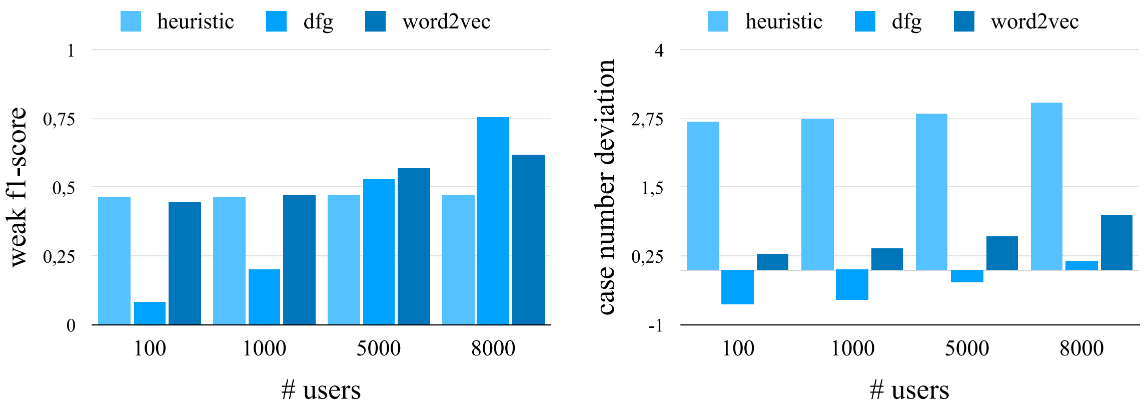


Figure 5.9: The computed weak f1-score (left) and cnd (right) of the different segmentations that were produced by the methods that are under consideration, based on the four different test logs with a varying amount of users.

Overall it can be said that the ratio between the number of users in the log and the number of cases has a significant impact on the performance of the DFG based baseline and the proposed method, while the heuristic approach is not influenced strongly. We also observed that some of the performance gains may not be caused by a better case attribution performance, but rather by the introduction of more predetermined case boundaries. In most real world applications, the user to case ratio will not be nearly as high as the 80% that are modeled by the test log with the highest ratio.

Case Length The length of the traces in a log, including the most frequent case length is an important characteristic of every process and its corresponding interaction log. The accuracy of the methods depending on the most frequent case length in the test logs was therefore evaluated and the results can be seen in Figure 5.10.
As in the previous comparisons, we see that the DFG based baseline approach has the highest overall accuracy. It is followed by the proposed method and the heuristic baseline. For the heuristic baseline we see a minor but decline in accuracy for the test logs with increasing most frequent case lengths. For the DFG based baseline approach and the proposed method, this trend is reversed. An increasing accuracy with a longer case length can be observed.

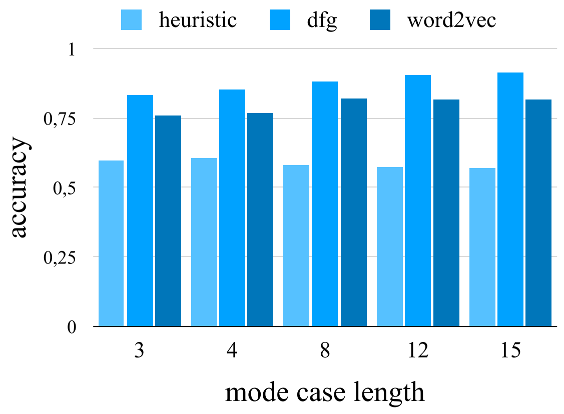


Figure 5.10: The computed accuracy of the different segmentations that were produced by the methods that are under consideration, based on the five different test logs with a varying most frequent case length.

This is somewhat surprising, because the weak f1-score which can be found in Figure 5.12 is decreasing for the test logs containing longer cases. This can be explained by the fact that the accuracy is the only used measure that also considers the true negatives. As the case length increases, the ratio between positions in the traces that correspond to case boundaries and those that do not will decrease. Since all of the considered methods are generally able to identify true negatives more reliably than the actual case boundaries, their accuracy increases when there are more true negatives. This is however not the case for the heuristic baseline. We observe that with an increasing case length, the number of potentially delayed interactions increases as well. The method therefore produces false positives more frequently.

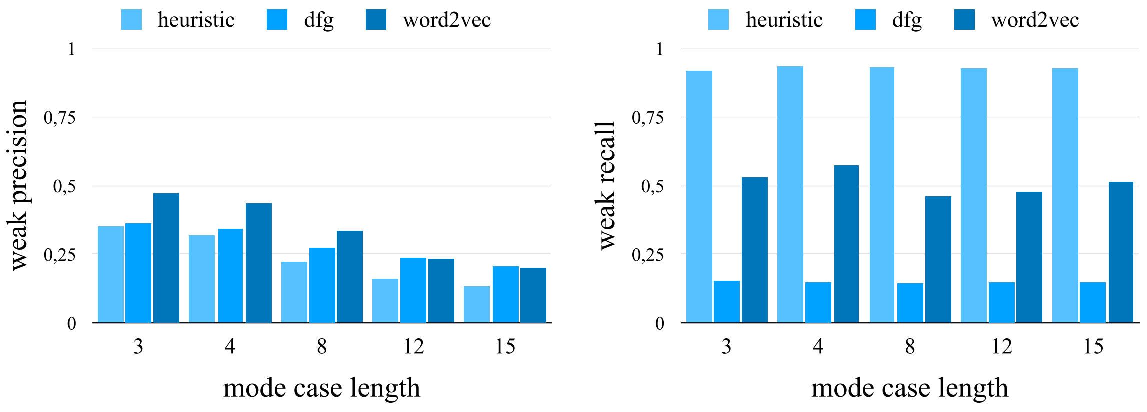


Figure 5.11: The computed weak precision (left) and weak recall (right) of the different segmentations that were produced by the methods that are under consideration, based on the five different test logs with a varying most frequent case length.

The computed weak precision and weak recall can be seen in Figure 5.11. For the weak precision we see a clear decline with an increasing case length for all of the considered methods. The decline is most prominent with the proposed method, which nevertheless has the highest weak precision in general. It is followed by the DFG based baseline and the heuristic baseline which shows the worst weak precision. The decrease in weak precision when increasing the case length for all of the methods, can be explained by the fact that longer traces will contain loops and repetitions more frequently. These are difficult constructs to identify for case attribution approaches in general and are especially challenging for the proposed method. The decline of the heuristic baseline can be explained in the same way as the decline in accuracy that was discussed before.

The results for the weak recall, also being found in Figure 5.11 show no clear trend for any of the methods. The weak recall of the heuristic baseline is again the highest and does not seem to change significantly based on the case length. The same is true for the DFG based baseline who shows the weakest performance of the considered methods when it comes to weak recall. The values for the proposed method are fluctuating between the logs and do not show a clear trend based on the case length. The fluctuations may be explained by the random nature of the test logs. Overall, it can be said that none of the methods show a significant change in the weak recall when the most frequent case length changes. This further supports the observations that are made regarding the weak precision. When the recall does not change significantly, but the length of the traces increases, the weak precision will decrease. This is expected as for longer cases, the probability for false positives increases.

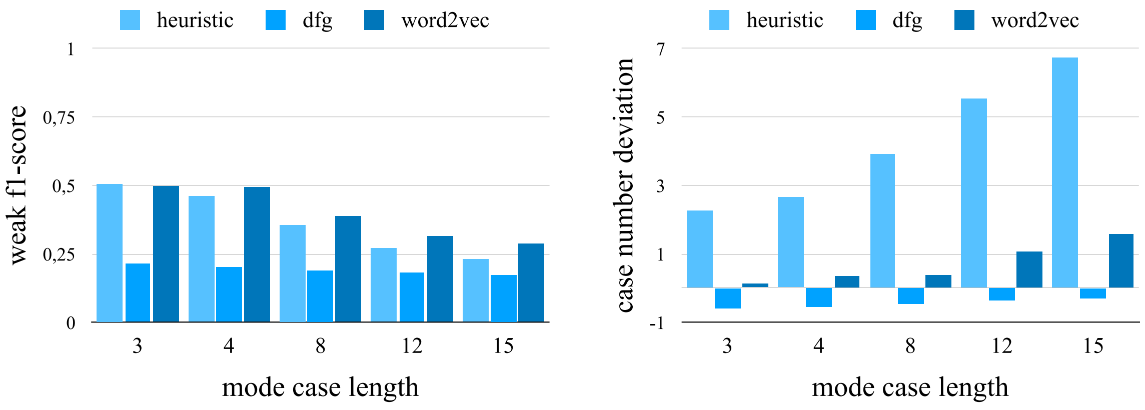


Figure 5.12: The computed weak f1-score (left) and cnd (right) of the different segmentations that were produced by the methods that are under consideration, based on the five different test logs with a varying most frequent case length.

Because of the constant nature of the weak recall that was computed in this test case, the change in the weak f1-score that is found in Figure 5.12 is mainly influenced by the changes in weak precision that were discussed before. We therefore see a downward trend of the weak f1-score with an increasing case length for all of the considered methods. With the exemption of the test log with the shortest cases, the proposed method provides the best results in general. The heuristic baseline has a weak f1-score that is comparable to that of the proposed method for the test log with a most common case length of three. For all other logs it places second. The DFG based method shows the worst performance for all of the test logs.

For the cnd in Figure 5.12, we see a significant upward trend with increasing case length for all of the methods. The heuristic baseline method again shows the worst performance, with the highest cnd values that have been observed so far in the different test cases. For the test log with the longest traces, the method produces almost 8 times as many cases compared to the original log. The other approaches show better cnd values for the different test logs. For the DFG based approach, the introduction of additional cases boundaries does bring it closer to optimal, since it does underestimate the number of cases for the test logs with shorter cases. For all of the methods it can be observed that an increase in case length, introduces the potential for more false case boundaries and therefore leads to a higher cnd.

For all of the performance metrics that were computed in this test case, it has to be noted that the same process with the same set of actions was used for the generation of all of the test logs. This is not necessarily realistic, as the most frequent case length is a characteristic of a process as well as that of an interaction log. Usually, software systems that are more complex and offer a larger amount of functionality will also correspond to longer cases. More simple software systems will analogously produce shorter cases in general. The obtained results are therefore only applicable to a limited extent, when evaluating the performance of the different methods regarding different processes with differing lengths. They are more suitable to compare the performance regarding differently sized cases, that originated from within the same process.

Fraction of Delayed Interactions Since the proposed method acts mostly timestamp independent, the occurrence of delayed interaction in the cases of the test logs should have no meaningful influence on its performance. This hypothesis is supported by the observed accuracy of the proposed method that can be seen in Figure 5.13. The DFG based baseline has the highest accuracy and does also not show any dependency on the ratio of delayed interactions.

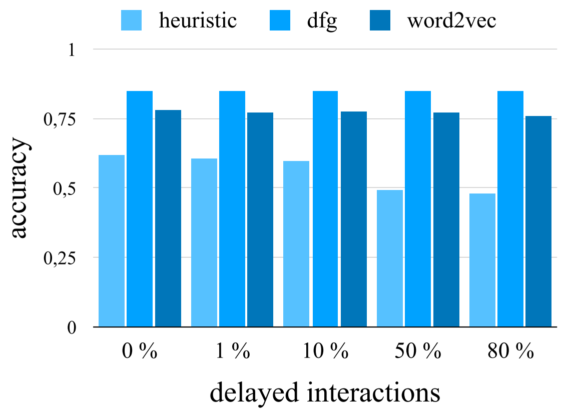


Figure 5.13: The computed accuracy of the different segmentations that were produced by the methods that are under consideration, based on the five different test logs with varying probabilities for delayed interactions.

The heuristic baseline method has the lowest accuracy for all test logs, which decreases with an increase in delayed interactions. A prominent drop in accuracy can be observed between the test logs with a delay probability of 10% and 50%. For the test log with the highest delay probability of 80%, the accuracy reduces by about 30% compared to the test log with no delayed interactions. These results for the heuristic approach are expected, as it based solely on the timestamps of the interactions. When an interaction is delayed by a significant amount, the heuristic baseline will identify a false case boundary.

While some fluctuations in the weak precision and recall of the proposed method can be observed in Figure 5.14, no clear trend depending on the number of delayed interactions can be observed. This is also true for the DFG based baseline method. The heuristic baseline again shows a downward trend in both metrics, with a more significant decline between the test logs with a delay probability of 10% and 50%. The results for the weak recall of the heuristic approach are unexpected, since increasing the amount of delayed interactions should generally have no influence on the weak recall. This might be explained by the fact that the delayed interactions will raise the used thresholds. Because of this, some case boundaries might not be able to be identified anymore. As observed previously, the proposed method has the highest weak precision.

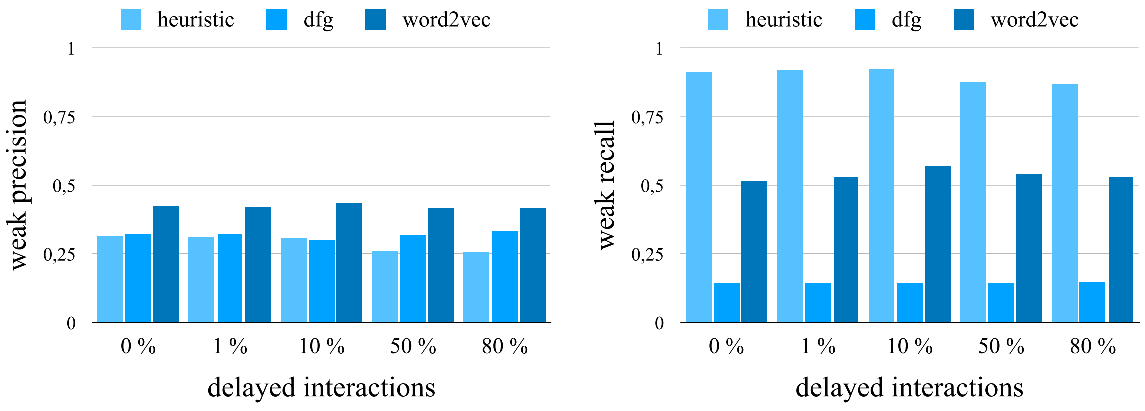


Figure 5.14: The computed weak precision (left) and weak recall (right) of the different segmentations that were produced by the methods that are under consideration, based on the five different test logs with varying probabilities for delayed interactions.

The results for the weak f1-score that can be seen in Figure 5.15 are similar to prior test cases. The decline of the heuristic method with an increasing probability of delayed interactions that could be observed for the recall and precision also translates to the weak f1-score. The proposed method again shows the best weak f1-score overall.
The cnd values for the proposed method and the DFG baseline are unchanged to prior test cases and do not show any dependency on the number of delayed interactions. The proposed method shows the best cnd performance for all of the test logs.

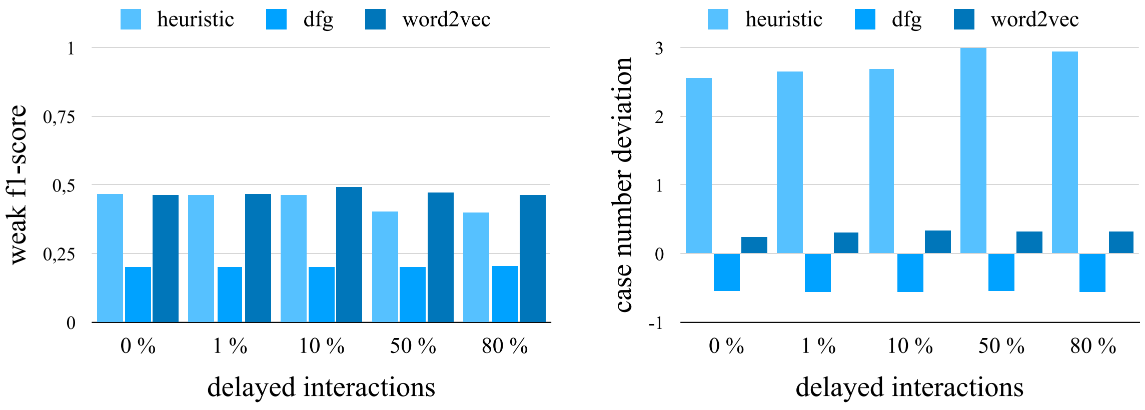


Figure 5.15: The computed weak f1-score (left) and cnd (right) of the different segmentations that were produced by the methods that are under consideration, based on the five different test logs with varying probabilities for delayed interactions.

While a negative influence of an increased probability for delayed interactions on the performance of the heuristic baseline method could be observed, the decrease is not as significant as expected. This may be explained by the fact that it generally tends to introduce a large number of case boundaries. Many of the interactions that were delayed in the test logs, may have already been identified before due to the lack of precision. As expected, no influence of the number of delayed interactions on the overall performance of the DFG based approach and the proposed method could be observed.

Fraction of Abandoned Interactions It is expected that the proposed method is relatively sensitive to the number of abandoned interactions in a log. If a user abruptly ends its usage of the software after an action that is not usually an end action, the method can be mislead into wrongly continuing the case.

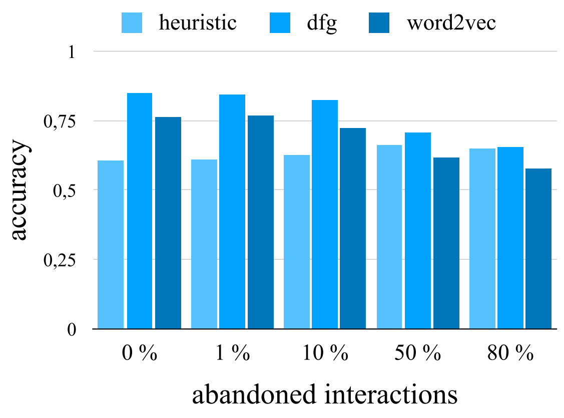


Figure 5.16: The computed accuracy of the different segmentations that were produced by the methods that are under consideration, based on the five different test logs with varying probabilities for abandoned interactions.

We can see according to Figure 5.16, that the accuracy of the segmentations that were produced by the proposed method decreases with an increasing amount of abandoned interactions. This is also the case for the DFG based baseline method. The heuristic baseline however somewhat improves in accuracy with a rising probability of abandoned interactions. Despite of the opposite correlations, the DFG based baseline has the highest accuracy for all of the evaluated test logs. For a low amount of abandoned interactions, the proposed method outperforms the heuristic baseline. This is however not the case for the logs with an abandonment probability of 50% and 80%.
The rising accuracy of the heuristic baseline can be explained by the fact that a higher probability for abandoned interactions generally leads to shorter cases. It was already shown in Figure 5.10 that an increase in case length has a negative effect on the accuracy of the heuristic baseline. In contrast to this, it was shown that for the proposed method and the DFG baseline, the most frequent case length has a positive effect on the accuracy. Since we observe a decline in accuracy in this test case, abandoned interactions seem to have a significant negative impact on the DFG based baseline and the proposed method. However, it has to be noted that small abandonment probabilities in the range of 0% to 10%, cause only a minor decline in accuracy.

The computed values for the weak precision of the methods that can be seen in Figure 5.17 are unexpected, given the numbers obtained for the accuracy. We see a strong increase in weak precision for all of the methods with a rising probability for abandoned interactions. While the heuristic baseline shows the worst weak recall for all of the test logs, the proposed method outperforms the DFG baseline for the logs with 0% and 1% abandonment probabilities, while the DFG baseline shows the best results for the remaining logs. All of the methods show precision values close to optimal for the log with a probability for abandoned interactions of 80%.

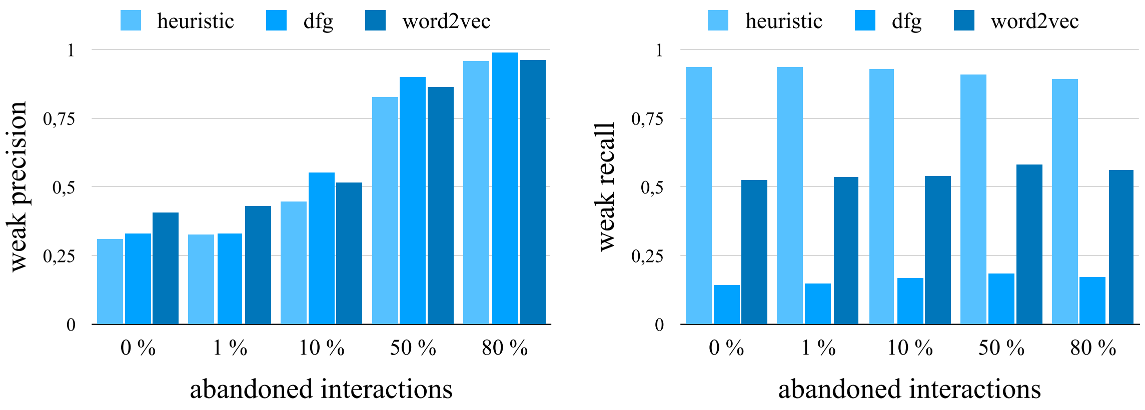


Figure 5.17: The computed weak precision (left) and weak recall (right) of the different segmentations that were produced by the methods that are under consideration, based on the five different test logs with varying probabilities for abandoned interactions.

The general upward trend and this observation can possibly be explained by the fact that the number of false positives is decreasing, instead of the number of true positives increasing. This observation is also backed by the values for the weak recall. The occurrence of less false positives may in turn be caused by fewer case boundaries being identified overall, because the shorter traces provide less possible positions for for split points. This is further reinforced by the weak recall of the methods seemingly not changing significantly based on the abandonment probability, as can also be seen in Figure 5.17. We only see a minor increase in recall for the DFG baseline and the proposed method. For the heuristic baseline, a minor decrease in the weak recall can be observed. Nevertheless, the heuristic baseline has the highest weak recall, followed by the proposed method.
The DFG baseline achieves the highest observed values for the weak recall so far, compared to the prior test cases. This is expected, as abandoned interactions will sometimes lead to illegal transitions between two actions of the log. Since the DFG based baseline is designed in order to recognize such situations, even significantly higher weak recall values than this were expected beforehand. The fact that only a small amount of the abandoned interactions will actually lead to a case boundary, translating to an illegal transition, may be the cause for this worse than expected performance.

The combination of a nearly constant weak recall and strongly increasing precision for higher abandonment probabilities, lead to a weak f1-score that increases with the abandonment probability for all of the methods, as can be seen in Figure 5.18. While the DFG based approach again has the lowest weak f1-score, the heuristic baseline has the highest. The values for the test logs with probabilities for abandoned interactions of 0% and 1% are very similar between the proposed method and the heuristic baseline.
For the cnd we see that for all of the methods, the number of cases is decreasing with an increase in abandonment probability. The effect is the strongest for the heuristic baseline

and the proposed methods has the values that are the closest to optimal for all of the test logs. The DFG based baseline follows for the logs with an abandonment probability in the range of 0% and 10%, while the heuristic approach outperforms it for the remaining logs with high probabilities. The observed decrease in the cnd is unexpected, but in conjunction with the almost constant weak recall, supports the theory that the methods identify fewer false case boundaries in the shorter cases of the logs with high abandonment probabilities. The results for the heuristic baseline are especially interesting, as an increase in the cnd was originally expected.

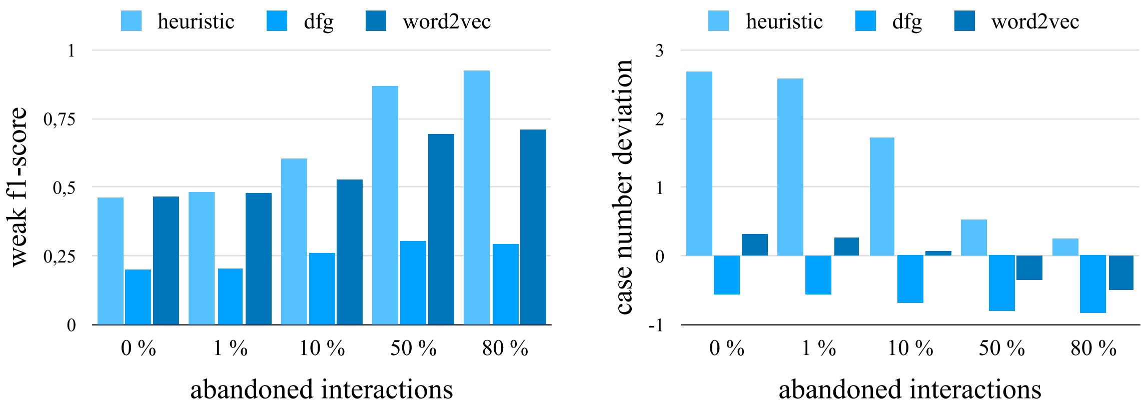


Figure 5.18: The computed weak f1-score (left) and cnd (right) of the different segmentations that were produced by the methods that are under consideration, based on the five different test logs with varying probabilities for abandoned interactions.

Overall it can be concluded, that low abandonment probabilities do not have a significant impact on the performance of the methods. For logs in which a majority of the traces is abandoned, the methods show a better performance. This is however most likely caused by the fact that the average length of the cases is reduced when a large number of cases is abandoned, this is supported by the results that were obtained for a change in the frequent case length. The proposed method seems to have a tendency of missing existing case boundaries and underestimating the amount of cases, when the probability for abandonment is high.

5.1.6 Time Analysis

In addition to the quality of the segmentations that are produced by the proposed method, the time that is required for segmenting a log is another important factor that has to be considered. Because of this, the time performance of the proposed method and the two considered baseline approaches was evaluated against differently sized test logs. The results are presented in the following.

All of the tests that were conducted regarding the time performance of the discussed methods and are presented in this section, have been conducted on a 13" MacBook Pro from 2018, equipped with a 4-core/8-thread Intel Core i7-8559U @ 2.70GHz with 16GB of LPDDR3 2133Mhz RAM and a 500GB M.2 Solid State Drive.

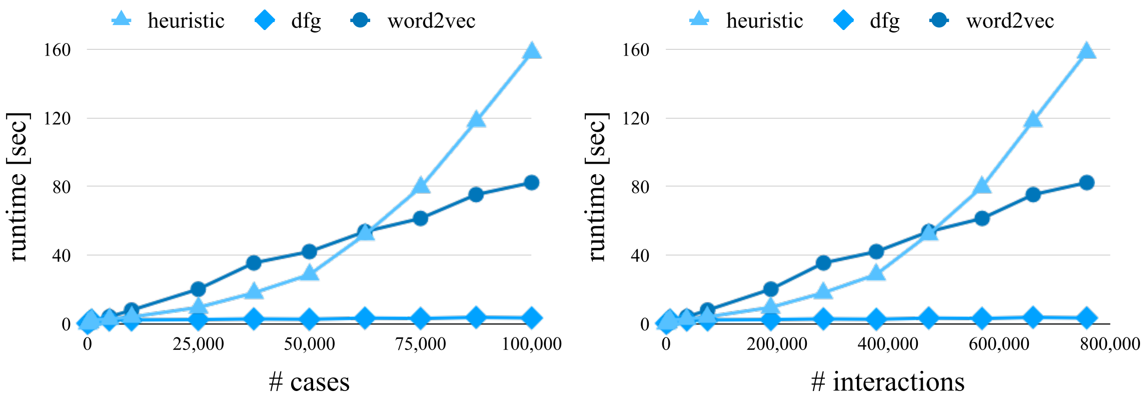


Figure 5.19: The time in seconds that the different methods needed in order to perform a segmentation of logs of varying sizes. On the left, the segmentation time depending on the number of cases can be seen. On the right, the same segmentation time depending on the number of events in the log can be seen.

In Figure 5.19, the time that the different methods needed in order to perform case attribution on the differently sized test logs can be seen. The DFG based baseline shows the best performance for all of the log sizes. We see a linear time increase between the logs with 100 and 1,000 cases, which does not seem to continue after this point. Only a minor increase can be observed for the larger logs. The DFG baseline is implemented in such a way, that it only iterates over the complete log once and assigns the case boundaries whenever an illegal transition between actions is observed. We would therefore expect a time complexity of $\mathcal{O}(n)$ where n is the number of events in the interaction log. This is confirmed by the test results for the range between 100 and $1,000$ cases. As mentioned before, this trend does however not continue for larger logs, which is unexpected. This is most likely explained by the fact that the actual iteration over the cases only takes an insignificant amount of time compared to other administrative operations, such as the creation of the result dataframe. Overall, the DFG based baseline is robust against an increasing log size and can therefore be applied to large interaction logs without limitations.

The heuristic baseline outperforms the proposed method for the range of logs with 100 and 62,500 cases. However, for larger logs it performance decreases significantly and the proposed method is able to segment the logs faster. A quadratic dependency between the size of the log and the time performance of the heuristic baseline can be observed. This observation is reinforced by the fact that a quadratic function can be fitted to the values with a coefficient of determination of 0.997, which is close to optimal. In general, the heuristic baseline does also iterate over the complete log, which introduces a linear time component. However, the median of all of the observed times for a certain directly-follows relation in the interaction log has to be computed. In order to compute the median, the lists of time differences has to be sorted. This sorting introduces a time overhead of at least $\mathcal{O}(n \log n)$. From this we obtain the overall worst case time complexity of the heuristic baseline of $\mathcal{O}(k^2 n \log n)$, for k being the number of different actions in the log and n being the number of interactions. The factor of k is introduced, since in the worst case, the sorting operation has to be conducted for all of the possible directly-follows relations, of which there are k^2. For the heuristic baseline it can therefore be said that the approach is only suitable for very large interaction logs in a limited way, because large runtimes are to be expected.

As mentioned before, the proposed method provides a better time performance than the heuristic baseline for larger logs. As can be seen in Figure 5.19, the runtime of the method increases linearly with the size of the interaction log. This is to be expected, as for each position in the connected traces, the underlying neural network of the wor2vec models is activated. The proposed method therefore has a case attribution time complexity of $\mathcal{O}(n)$, which makes it applicable to very large interaction logs.
In general, it has to be noted that none of the evaluated methods were implemented in a way that tries to optimize for the best possible time performance. It is therefore likely that the presented case attribution times could be lowered through more optimized implementations for all of the different methods.

In addition to the time that is required for the actual segmentation of the input log, the proposed method also consists of a preparation phase that needs to be considered. This phase consists of the computation of the required log statistics, the generation of the training set and the transition system and the training of the underlying word2vec models. These steps can take up a considerable amount of time depending on the log size and therefore have to be considered.
In contrast to the actual log segmentation, the preparation phase has to be executed only once for a process and can then be reused for different interaction logs from the same source. In a similar vein, a smaller sample might be taken from a very large input log, which can then be used for the preparation phase. This will reduce the computation times and can create similar results, if the sample size is large enough. In order to better understand the time requirements of the different phases of the method, a more detailed analysis of the preparation phase was conducted.

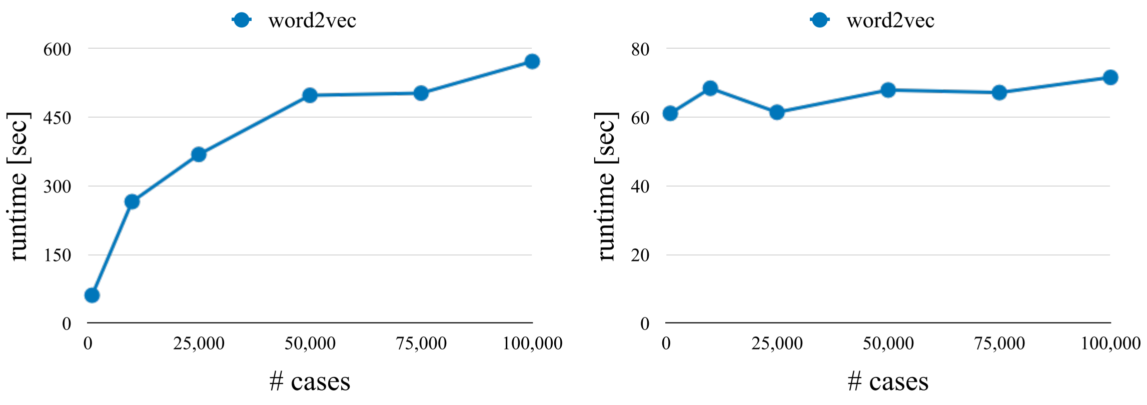


Figure 5.20: The runtime of the proposed method during the generation of the training log (left) and the time that is required for the model training (right) depending on the number of cases in the input log.

In Figure 5.20 it can be seen that the time required for the generation of the training set (left) increases quickly for small to medium sized logs, but then plateaus for larger logs. The main factor for the performance of the training set generation is the complexity of the underlying transition system. A larger log will generally contain more behaviour, which in turn will lead to a more complex transition system. More paths therefore have to be considered during the generation of the artificial traces. The plateauing for larger logs can also be explained by this, because at some point, increasing the size of the log will no longer significantly increase the amount of variants that it contains. The number

of states and transitions in the transition system will therefore stop growing, since the system already depicts all of the possible behaviour. After this point, the performance of the generation will plateau and is no longer depending on the size of the log.
For the training of the word2vec models, we see a constant required time with minor fluctuations. This indicates that there is no influence of the size of the training log on the performance of the model training. This is caused by the fact that the size of the artificial training log does not depend on the size of the input log, but can be freely chosen. Since the same sized training set was used for all of the logs, the training time did not change significantly.

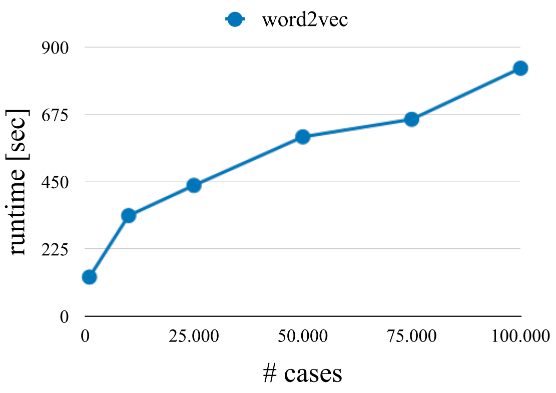


Figure 5.21: The overall runtime of the proposed method in the preparation phase depending on the number of cases in the input log.

The combined time that is required for the complete preparation phase of the proposed method, depending on the size of the input log, can be seen in Figure 5.21. The overall time is mainly influenced by the generation of the transition system, since the model training requires a constant time. Other parts of the preparation phase such as the computation of the required log statistics have a linear runtime and contribute to the overall runtime behaviour that can be seen in Figure 5.21.
In conclusion, the preparation phase consists of steps with a constant time complexity (model training), a linear time complexity (statistics) and a time complexity of $\mathcal{O}(T' + S'^2)$ for computing the paths in the underlying transition system (S', A, T', i). The size of the transition system depends on the size of the input log, but is limited by the number of variants in the log. Overall, it can therefore be said that the time performance of the preparation phase is reasonable even for larger interaction logs, especially considering that it only has to be performed once, but may be reused for multiple segmentations.

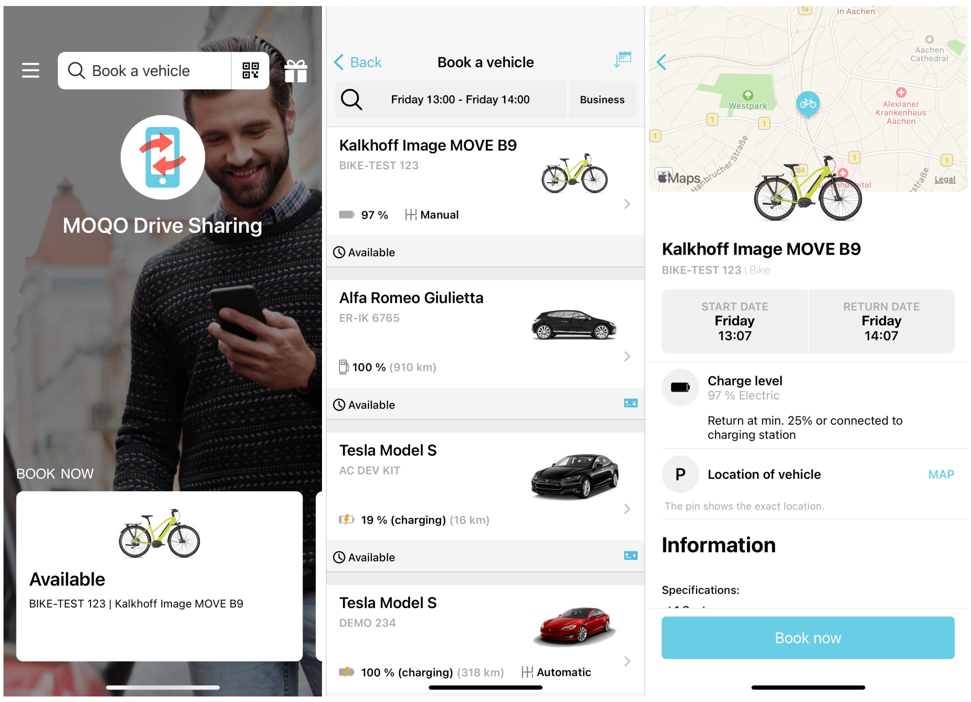
Compared to the artificial test logs that are examined in this section, the following section describes an application of the proposed case attribution method to a real interaction log obtained from a smartphone application.

5.2 Smartphone Application Case Study

In order to validate the utility of process mining workflows in the area of user behaviour analysis and to assess the quality of the segmentations that are produced by the proposed method in a real setting, a case study was conducted. The findings of this case study are presented and discussed in the following sections. In the study, the proposed method is applied to a dataset which contains real user interaction data that was collected from the mobility sharing smartphone application *MOQO*. The segmented UI log that is produced from the input log by the proposed method is used in different process mining workflows in order to analyze the underlying user behaviour. The results of this analysis are presented to experts from the user experience and mobile development team of MOQO. The experts make an assessment of these findings and try to identify critical areas regarding the actual use of the application. Based on these results, suggestions for changes and adjustments that lead to an overall better user experience in these areas are proposed.

5.2.1 Application & Interaction Data

The dataset that is used in this case study originates from user interaction recordings that were collected by the mobile application MOQO. The MOQO app is a mobility sharing application and is available for iOS and Android smartphones [46]. The app provides its users with access to different types of shared vehicles such as cars, bikes or scooters.



dashboard search results pre booking

Figure 5.22: Three different screens of the examined MOQO application. The dashboard (left) is the starting point of the application and acts as the main menu from which the majority of functionality can be reached. There are different types of dashboards (e.g. with and without a map) that are used in different context. The vehicle search results screen (middle) displays all of the vehicles that are available at the specified time. The pre booking screen (right) contains detailed information about the selected vehicle. From this screen a booking can be created.

Users of the app are most commonly using it in order to search for available vehicles, create bookings for certain vehicles at certain times and the actual usage of the vehicles, including locking and unlocking them. An important difference between MOQO and

other well known shared mobility providers such as *DriveNow*, *Car2Go*, *VOI*, etc. is, that MOQO is not providing any of the vehicles itself. It is rather offering its application and backend infrastructure as a service to interested providers who actually own and operate the vehicles.
This is an important distinction to be made, as it means that the application is used in a large number of different contexts, depending on the requirements of the actual provider of the vehicles. For example, consider a large company with a fleet of cars that are provided to its employees and a small bike rental at some tourist destination. Both of these dissimilar use cases have to be covered by the application, even tough they require particular functionalities and overall have a very different focus. This diverse set of customer requirements has over time lead to a steady increase in the number of offered features and therefore also resulted in a rise in the overall complexity of the application. This circumstance has shown to be a major and ongoing challenge for the UX design team, who has to incorporate all of these features into the application, while at the same time being restricted by the limited amount of screen area that is available on smartphones. In order to aid this design process, it is important to be able to better comprehend the way that the users are actually using the application in a real setting. For this reason and in order to be able to assist users that are facing problems in the app, a tracking functionality has been integrated into the MOQO application. Users that have agreed to this tracking are continuously sending information about the screens that they are visiting inside of the application to a central server. This server stores the recordings in the form of interaction logs. In the past, this information was only used infrequently by the development and support teams, in order to retrace the steps of users that had reported problems or bugs within the application, otherwise the recorded data has remained mainly unused.

As mentioned previously, the abstraction level that is used in the case study is again that of a screen. Analogous to the screen definition regarding the artificial test log, a screen of the app takes up the majority of the display and serves a specific function. The user can transition between different screens by interacting with the application. The two different MOQO applications for iOS and Android overall consist of 96 different screens that are tracked. Since the two apps do not share a single common code base, there are some inconsistencies in the naming of certain screens. Furthermore, some of the screens can only be accessed by users that are not yet logged in into their user account. Because the is no login information, such screens cannot be associated with a certain user and therefore cannot be considered in the context of the case study. Accounting for the aforementioned circumstances, there are 78 remaining screens that are relevant for the following analysis. The set of user interactions that is provided by MOQO for the use in this case study consists of about $990,000$ events that originate from about 12,200 different users. Since there is no explicit case information, the number of contained cases is unknown, hence the need for the preprocessing of the dataset with the proposed method. Each recorded interaction event consists of the five different properties: `timestamp`, `screen`, `user`, `team` and `os`. The `user` property corresponds to a token that uniquely identifies each logged in user. As introduced before, the app can be used to access vehicles from different vehicle providers. In order to do so, the user has to join the *team* that corresponds to that provider. A user can be a member of multiple teams. During the usage of the app, only a single team is selected at all times and the user can switch between different teams inside of the app. The `team` property is a unique identifier for the team that was selected at the time at which the interaction was recorded. The `os` property signifies the operating system that the application is executed on and can therefore be used in order to distinguish between the iOS and Android versions of the app. An exemplary excerpt of the interaction log can be found in Table 5.1.

Table 5.1: An excerpt of the user interaction data that was recorded by the MOQO application and is the basis for the conducted case study. Each interaction consists of five different properties. There is no explicitly available case information.

`timestamp`	`screen`	`user`	`team`	`os`
2021-01-25 23:00:00.939	pre_booking	b9b00	2070b	iOS
2021-01-25 23:00:03.435	tariffs	b9b00	2070b	iOS
2021-01-25 23:00:04.683	menu	3fc0c	02d1f	Android
2021-01-25 23:00:05.507	my_bookings	3fc0c	02d1f	Android
⋮	⋮	⋮	⋮	⋮

In order to provide some context about the way a typical user would experience the app, in the following an exemplary usage of the app is described:

- The user uses the search function in order to find a fitting vehicle that is available in the desired timeframe.
- When the user has selected a vehicle that suits their needs, they create a booking.
- When the time of the booking arrives, the user returns to the app in order to start the booking and unlock the vehicle with the app.
- The user can now freely access the vehicle. During the booking, the user can lock the vehicle with the app for an intermediate stop, search for nearby charging stations, or use the included digital refueling card.
- Upon vehicle return, the user uses the app to lock and return the vehicle and complete the booking.

Aside from this example, the app can also be used in order to manage the users account and payment information, for joining additional teams, reporting damages and the cleanliness of the vehicles and more. As already mentioned before, the actual usage of the application does vary drastically in between different teams and different contexts. This makes it an interesting target for various process mining workflows.

5.2.2 Methodology

In a first step, the proposed method was trained with and applied to the input log that was introduced in the prior section. For the generation of the transition system, a past sequence abstraction of size four was chosen in order to strike a balance between computational effort and the accuracy of the transition system. A DFG of the process under observation was created manually by a process expert. This was done by assessing the validity of all transitions between all pairs of screens. This DFG is provided to the method in order to remove illegal transitions from the transitions system, as introduced before. The size of the artificial training log was set to $75,000$ cases, large enough to include a large number of different variants that are part of the input log, without overly impacting the training performance. For the underlying word2vec models, two different models with

the window sizes one and three were chosen with corresponding embedding sizes of 25 and 20. These parameters were determined as optimal for this application based on multiple different test cases that were conducted with a varying set of parameters.
Based on the set of parameters that was introduced above, the proposed methods preparation phase was executed in a time of 17 minutes and 55 seconds. The subsequent segmentation of the complete input log was performed in 3 minutes and 2 seconds. Overall this results in a processing time of 19 minutes and 57 seconds for the complete preprocessing of the input log. The result of this preprocessing with the proposed method is a segmented UI log that can be used in process mining.

In order to reduce the workload of the experts from the MOQO team, the actual analysis of the segmented input log was performed by the author. The findings were then presented to and discussed with four experts from the MOQO team consisting of one UX expert, two mobile developers and the CTO. All of the participants are working directly on the application and are therefore highly familiar with it. The analysis results were presented to the participants in the form of 16 different questions. There are three types of different questions:
In questions of type I, the participants are asked a direct question about the application, in order to test their believed knowledge of the process. The answers are then compared with the insights that were obtained from the analysis and potential deviations between these views are discussed. This type of question is used in order to examine how the view that the process experts have of the process differs from reality.
In questions of type II, the participants are shown a process model that describes a part of the discussed process. The experts are asked to voice any observations they have made, based on the shown model. The discussed process model is created beforehand by the author using the process mining tool Disco [47]. It is created in such a way, that it tries to convey an unexpected or interesting observation that was made by the author during the analysis. During the discussion, the observation of the author is either validated or rejected by the statements of the participants and the implications of the observation are discussed. This type of question is used in order to assess how well process experts are able to extract new knowledge from a given process model that was prepared by a data analyst. In addition to this, the validity of the observations that are made by the author is verified and the impact that an observation has on the application is exposed.
Questions of type III consist of a discussion that is based on a factual statement that is made by the author. The statement describes some fact, or observation that was made about the application and the underlying process during the analysis. The impact of the statement on the application is discussed and possible solutions to the resulting problem are proposed by the participants. This question type is used in order to observe how the experts react to factual statements that are based on the analysis and if they are actionable enough so that solutions can be proposed.

The study was conducted online via a video conferencing system. The questions were presented to the participants using a presentation software and shared with them through the video conferencing system. The questions were asked in an open environment in which all participants were able to speak at any time. After presenting a question and clarifying all potential inquiries regarding the question, the participants were asked to write down their answers (for type I and II) and then present them to the whole group one after another. For each question, there was a set amount of time in which the participants had to give an answer. Where applicable, the answers were discussed with all of the participants after each individual had answered the question. For some questions, the correct answer was given to the participants after collecting their answers. In these cases, any potential deviations between the correct answer and the given answers were discussed in the group. In the following, the term *interaction* in the context of the app always corresponds to a sequence of screens that the user visited in order to reach a goal. An interaction with the app is therefore equivalent with a case. The different naming was chosen to make the questions more clear for the participants, who are not familiar with the concept of a case and with process mining in general.

5.2.3 Results

Q1 - Type I - What is the most frequent first screen of an interaction?
The correct answer to this question is the `station_based_map_dashboard`, which could be computed by considering the first screens for all cases that were identified by the proposed method. All of the participants were able to answer this question correctly. This is expected, as all of the participants are familiar with the application. However, the answers of the participants did not distinguish between the three different types of dashboard that exist in the app. The fact that the map based dashboard is the most frequently used type of dashboard was new and surprising for all of the participants.

Q2 - Type I - What is the most frequent last screen of an interaction?
The answer to this question can be obtained analogously to that of Q1 directly from the segmented log. In contrast to Q1, not all participants were of the same opinion regarding the answer to this question. Two participants gave the correct answer, which again is the `station_based_map_dashboard`. The other two participants chose the `booking` screen. This screen is the third most frequent case end screen following the `pre_booking` screen. After the correct answer was revealed, one participant proposed that the users may be likely to return to the dashboard after they have completed their goal. This theory can be supported with the available data. It seems that the users have an urge to *clean up* the application and return it to a neutral state before leaving it. Overall, it can be concluded that the participants have a good understanding of the frequent start and end screens of the application. However, the analysis provides more detailed information and was therefore able to discover aspects about the process that were new for the experts.

Q3 - Type I - What is the most frequent interaction with the app?
This question is asking about the most frequent case variants that are contained in the given log and the associated task of the user. Since the most frequent variants will usually be the shortest variants and a case consisting of only two generic screens cannot be interpreted as a task of the user in a meaningful way, these short variants were not considered for the answer to this question. According to the segmented log, the most common interaction of this type is, selecting a vehicle on the dashboard and checking its availability from the pre booking screen. One of the four participants did answer this question correctly. Two participants answered that searching for a vehicle on the dashboard is the most frequent interaction, which is closely related to the correct answer but does not include the availability check. The remaining participant answered, opening a booking from the list of all bookings. The results again show that the participants have a good understanding of the usage of the application, but are not able to provide details that are made visible by the log analysis.

Q4 - Type I - What is the average length of an interaction with the app?
For this question, the length of an interaction describes the number of interactions that belong to a case. The correct answer is 4.8 screens, which is rather short. The participants gave the individual answers 50, 30, 12 and 10 screens, which overall results in an average of 25.5. We see that the participants significantly overestimate the length of an average interaction with the app according to the segmented log. However, the average case length is strongly influenced by the employed case attribution method. The mismatch between the results from the log analysis and the expert opinions could therefore be caused by the segmentation that was produced by the proposed method. According to the evaluation results for the artificial test cases that were discussed in section 5.1, the proposed method does overestimate the number of cases somewhat. However, the observed deviations regarding the number of cases were overall not larger than about 50%, which does not explain the large difference between the experts expectations and the calculated value. In order to further examine this, the result was compared to that of a time based segmentation with a fixed threshold of five minutes. These case attribution techniques tend to overestimate the length of cases, as they are not able to distinguish between cases that happen directly after each other. For this reference segmentation, an average case length of 6.7 was calculated. This is comparable to the result of the proposed method and confirms the observation that the experts tend to overestimate the length of interactions significantly.

Q5 - Type I - What is the median duration of an interaction with the app?
For this question, the median duration is used instead of the average, as outliers that have case durations of several days are skewing the average disproportionately. According to the segmented log, the median case duration is 53.4 seconds. The participants gave the answers 240 seconds, 120 seconds, 90 seconds and 60 seconds, leading to an overall average of 127.5 seconds. Similar to the average length of the interactions, the participants did also overestimate their median duration. Only one participant did give an answer that was close to the calculated value. Both, the significant overestimation of the interaction length and the duration, show that the experts were not able to accurately assess the time a user needs in order to complete a task. This type of analysis is not possible using an unsegmented log and was therefore enabled by the use of the proposed method.

Q6/Q7 - Type I - How does the median interaction duration on Android and iOS compare?
As was introduced before, for each interaction it is recorded if it occurred in the Android or iOS application. This allows the comparison between the different applications during analysis. During the analysis it was discovered that the median interaction duration on iOS of 39.4 seconds is significantly shorter than the 92.9 seconds observed for the Android application. The participants were not aware of this difference, as three of the four participants thought that the interaction durations would be the same between the different operating systems and one participant thought that interactions would be shorter on Android. One of the participants argued that Android users may generally be more inclined to *play around* within the application, which may explain the observed difference. Regarding the analysis, the observed deviation could also be caused by differences in the implementation of the screen recording between the two apps. The produced segmentation may reflect cases originating from one of the apps more accurately than those from the other, because the same task of a user may translate to a different sequence of screens in the two apps.

Q8 - Type I - Given that 42% of the users use the Android app, what percentage of interactions are from Android users?
In general one would expect that the fraction of cases that originate from the Android app is similar to the number of users that are using this operating system. The conducted analysis does however show, that only 31% of cases originate from the android app, which is significantly lower than expected. The participants did not expect this uneven distribution, which is emphasized by their answers. Two participants expected a ratio of 50% and two participants answered that 60% of the cases originate from the Android app. In conjunction with the results for the median interaction time that were discussed in Q6/Q7, this means that according to the computed segmentation, Android users tend to use the app longer but overall less frequently.

Q9 - Type I - Draw your own model of the MOQO app.
The participants were asked to draw a DFG describing the most common interactions with the app. The concept of this type of graph was explained to the participants beforehand. The experts were given five minutes in order to create their models. A cleaned up representation of the resulting models can be seen in Figure 5.23 and Figure 5.24. For comparison, a process model of the segmented log was created using the process mining tool Disco [47]. The computed model, which can be seen in Figure 5.25, was configured to contain a similar amount of different screens as the expert models.

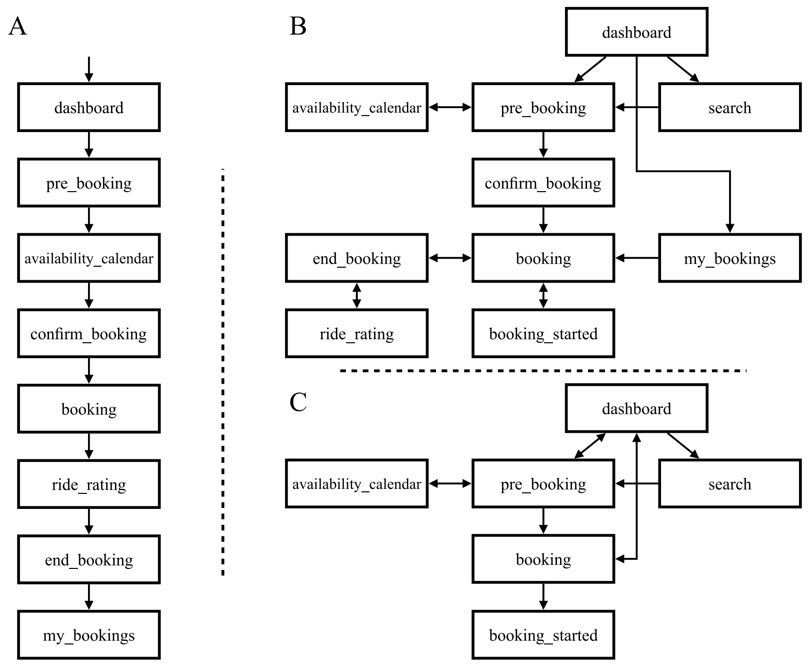


Figure 5.23: Representations of the directly-follows graphs that were created by three of the process experts as part of Q9.

Figure 5.23 shows model A, which consists of only a sequence of screens that describes a common interaction of a user with the application. It includes many of the most important screens of the app, but is not comprehensive when it comes to transitions. The `ride_rating` and `end_booking` screens are swapped and cannot occur in this order in reality. In contrast to other expert models, model A does include an entry point to the process which is set at the `dashboard`. Model B is more complex and includes more transitions between the screens. The screens and transitions are mostly correct when compared to the computed model, however some intermediate screens are missing completely. Model C is similar to model A but includes a larger amount of transitions between the screens. It also considers the possibility that a user may return to the previous screen in some cases.

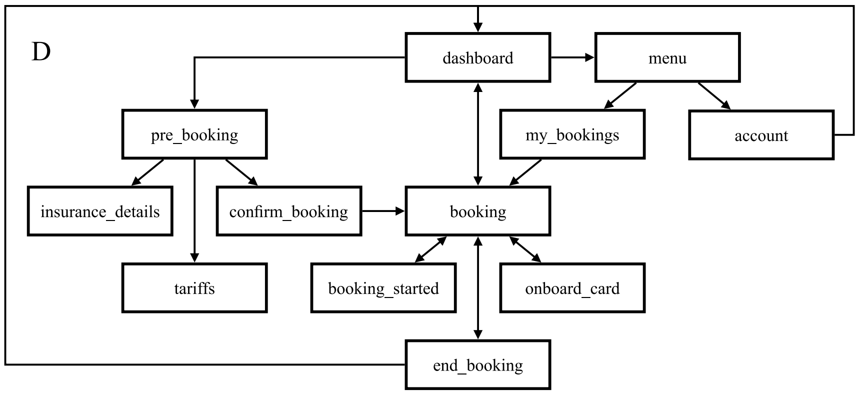


Figure 5.24: A representation of the directly-follows graph that was created by one of the process experts as part of Q9.

Model D, which can be seen in Figure 5.24, is more complex than the other models provided by the participants. It includes the largest amount of screens and also considers secondary screens such as the `tariffs` and `account` screens. These screens are not part of the most common interactions with the app according to the computed model. Similar to the other models, model D also misses some intermediate screens and does not consider the possibility to return to the previous screen in all cases.

Overall, the majority of the screens that make up the computed model are also found in the expert models. Notable exceptions to this are the `end_booking_flow` screen, the `date_range_picker` screen and the `date_time_picker` screen, which are all intermediate screens that are more of a technical nature and always occur in conjunction with their preceding and following screens. The same is true for the `search_results` screen, which was not identified by any of the experts, but has to occur in between the `search` screen and `pre_booking` screen. Overall, this shows that the experts have a good understanding of the most important screens of the application, but are not necessarily familiar with the implementation details. Regarding the computed model, this demonstrates that the model is able to extract the most common behaviour that was expected by the experts from the segmented log and that the proposed segmentation method preserves this information.
The same is true for the transitions between the different screens, with the exception of the transitions that are leading to those screens, that were not identified by the participants at all. The majority of the other transitions can also be found in the expert models. However, it is notable that the experts did only consider one directional edges in many cases and did not consider the possibility that a user might return to a screen directly after leaving it. This circumstance may be explained by the limited amount of time that was available to the participants.

The observed results regarding this question, overall show that the experts have a good understanding of the general usage of the most important parts of the app. The mobile developers (models A and B) tend to describe the interactions in a more detailed way that

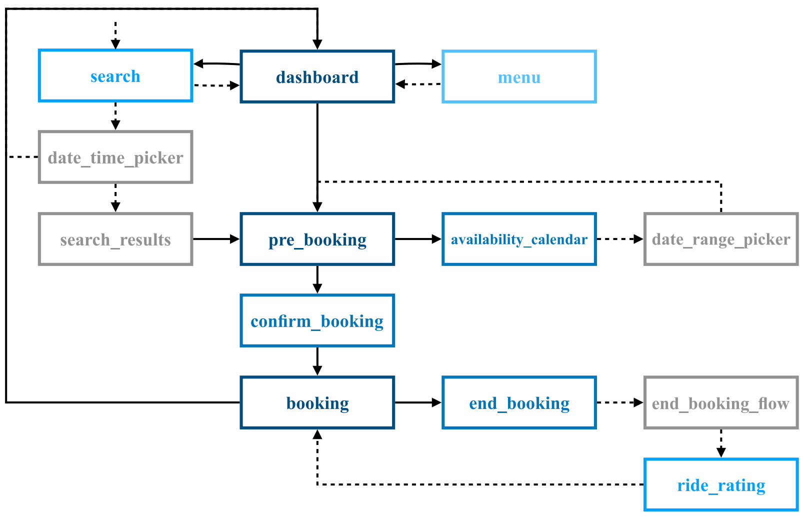


Figure 5.25: The representation of a directly-follows graph that was created using the Disco tool based on the log that was segmented using the proposed method. The model was configured in such a way, that a similar number of screens are visible compared to the models that were created by the experts. The colors indicate the agreement between the model and the expert models. Darker colors signify that a screen was included in more expert models. The dashed edges between the screens signify edges that were identified by the generated model, but are not present in the participants models.

follows the different screens more closely, while the CTO and UX expert (model C and D) provided models that capture the usage of the application in a more abstract way. The model that was created by the UX expert (D) provides the most amount of information, but is inaccurate regarding implementation details. The fact that the computed model and the expert models are overall very similar to each other shows that the proposed method is able to create a segmentation that contains cases that are able to accurately describe the real user behaviour.
One of the advantages of the models that are automatically computed based on the segmented interaction log, is that they include frequency information. This is valuable information that can be used in order to judge the importance of certain paths through the application. Manually created models do not contain this kind of additional information. Furthermore, the computed model is also describing behaviour that may differ from what the process experts initially expected. Additionally, the model is inherently able to describe not only the most common behaviour, but the complete process with all its variants and details. Models that are describing every aspect of a process are extremely difficult to create manually and require a large amount of effort and resources. This process is simplified significantly by the use of the segmented interaction log.

Q10 - Type II - Given this process model that is based on interactions ending on the *booking* screen, what are your observations?

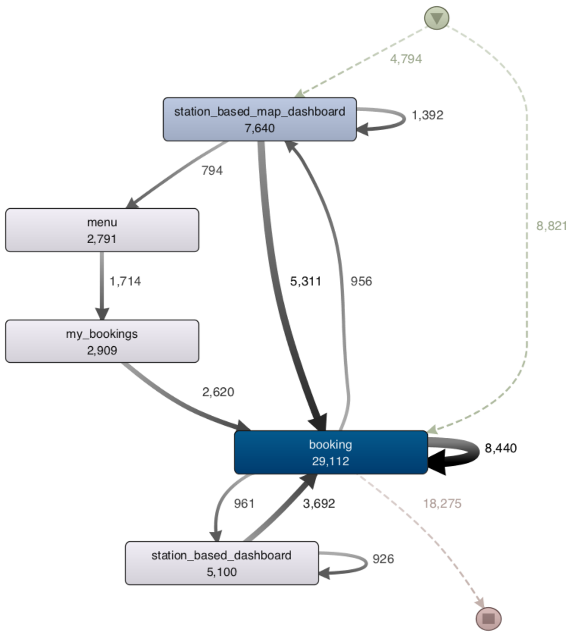


Figure 5.26: A process model that was created using the tool Disco [47] based on the segmented UI log. The model is based on the cases that are ending on the `booking` screen and only includes a subset of the screens in order to better convey the analysis findings.

Given the process model shown in Figure 5.25, one of the participants noted that the fact that the map based dashboard type is used significantly more frequently than the basic dashboard is surprising to them. This sentiment was shared by the other participants. Additionally, two of the experts were surprised by the number of users that are accessing their bookings through the list of all bookings (`my_bookings`). This latter observation was also made during the analysis of the segmented log and is the reason that this process model was presented to the experts. In general, a user that has created a booking for a vehicle can access this booking directly from all of the different types of dashboards. The fact that a large fraction of the users take a detour through the menu and booking list in order to reach the `booking` screen is therefore surprising.
This circumstance was actually already identified by one of the mobile developers some time before this evaluation, while they were manually analyzing the raw interaction recordings data. They did notice this behaviour because they repeatedly encountered the underlying pattern while working with the data for other unrelated reasons. Using the segmented user interaction log, the behaviour was however much more discoverable and supported by concrete data rather than just a vague feeling. Another observation that was not made by the participants is that the path through the booking list is more frequently taken by users that originate from the map based dashboard rather than the basic dashboard. The UX expert suspected that this may have been the case, because the *card* that can be used to access a booking from the dashboard is significantly smaller on the map based dashboard and may therefore be missed more frequently by the users. This is a concrete actionable finding of the analysis that was only made possible by the use of process mining techniques in conjunction with the proposed method.

Q11 - Type II - Given this process model that is based on interactions ending on the *search* screen, what are your observations?

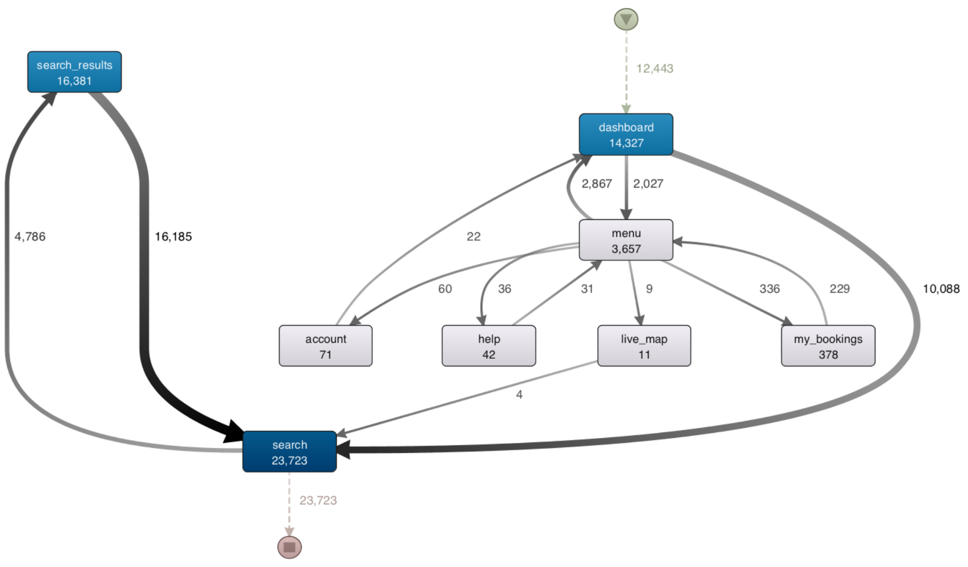


Figure 5.27: A process model created using the tool Disco [47], based on the segmented UI log. It is based on the cases that start at any dashboard and include the `search` screen. The model includes selected screens in order to convey the analysis findings.

The behaviour that was observed during the analysis was tried to be conveyed to the participants using the model that can be found in Figure 5.27. Since the model is based on all cases including the `search` screen, which start at any type of dashboard, and the `search` screen is directly reachable from the dashboards, it would be expected that no significant amount of other screens are included in the model. This is however not the case, as the `menu` screen and the various screens that are reachable from this screen are included in many of the cases that eventually lead to a search. This suggests that the users that did want to perform a search, tried to find the `search` screen in the main menu, implying that it is not presented prominently enough on the dashboards. None of the experts had this observation when they were presented the discussed model, and the observation was therefore not discussed further.
Instead of this, the focus of the participants mainly lay on the frequency information that is included in the model. Multiple experts noted inconsistencies in these numbers, for example between the `search` and `search_results` screens. Significantly more transitions happen between the `search_results` screen and the `search` screen than in the opposite direction. This is counter intuitive, as in general the search parameters will be set first and after that the fitting search results are displayed. Some of these inconsistencies may be explained by case boundaries that were wrongly identified by the proposed method. This can however not explain the large difference that can be observed here. After some discussion, one expert mentioned that there is also a way to reach the search results directly from a certain type of dashboard. This explains the observed difference. However, in this case an edge leading from the dashboard directly to the `search_results` screen should also be included in the model. After the evaluation with the experts, the corresponding model was computed again and included the missing transition. It could not be retraced how the incorrect model was initially obtained.

In addition to this, some of the participants were concerned, that a large amount of users abandon the interaction at the `search` screen, because this is a point in the application at which the user did not yet have the possibility to see any of the vehicles. This is however a misinterpretation of the shown model. In order to convey the observation that was made during analysis, all cases that included the `search` screen were cut at the last occurrence of this screen. The visualization was created based on these trimmed cases. Many of the included cases will therefore actually continue after the `search` screen, this behaviour is however not shown by the model. The confusion of the participants emphasizes the importance of providing correct visualizations and representations of the underlying process data to the experts in order to obtain the most meaningful results.

Q12 - Type I - What is the median time a user takes to book a vehicle?
The correct answer to this question is 66 seconds. This was calculated based on the median time of all cases in which a vehicle booking was confirmed. The participants gave the answers 420 seconds, 120 seconds and 120 seconds. One of the participants argued, that this time may depend on the type of dashboard that the user is using and answered 300 seconds for the basic dashboard and 120 seconds for the map based dashboard. When asked to settle on only one time, the participant gave an answer of 180 seconds. Overall this means that the experts estimated a median duration for this task of 3 minutes and 30 seconds. This again is a significant overestimation compared to the value that was obtained by analysing the real user behaviour. Again, a mismatch between the perception of the experts and the real behaviour of the users was revealed.
The hypothesis of one expert, that there are varying durations for the different dashboard types was later examined again. This more detailed analysis showed, that there indeed is a notable difference between the two types of dashboard. However, a successful booking is actually performed significantly faster on the basic dashboard (38 seconds) compared to the map based dashboard (69 seconds).

Q13 - Type II - Given this process model that is based on interactions ending on the *confirm booking* screen, what are your observations?

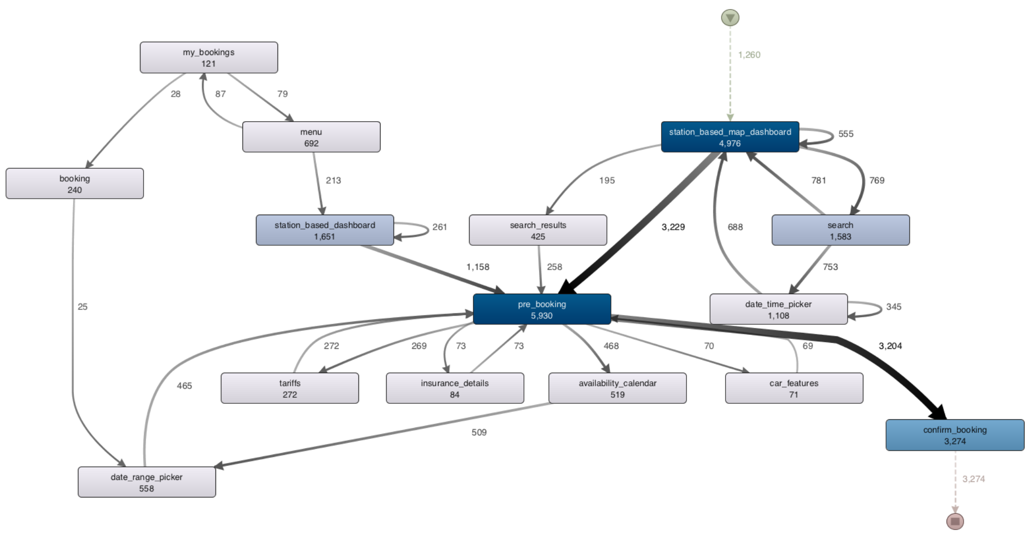


Figure 5.28: A process model created using the tool Disco [47]. It is based on cases that start at any dashboard and end on the `confirm_booking` screen.

The model that was provided to the participants for this question can be seen in Figure 5.28. The model is based on the cases that were used in order to compute the median time a user takes to confirm a booking, which was discussed in Q12. In contrast to the preceding questions of type II, there is no concrete observation that was made during the analysis regarding this model. The question is therefore more open and aims to investigate if the process experts are able to extract meaningful knowledge from a model that was not pre-selected according to a specific observation.
Several of the participants did observe that the screens that show details about the vehicles and the service, such as `tariffs`, `insurance_details` and `car_features` are seemingly used much less frequently than expected. In only about 2-10% of cases, the user visits these screens before booking a vehicle. When considering the concrete numbers, the `availability_calendar` screen (which is used to choose a timeframe for the booking) and the `tariffs` screen (which displays pricing information), are used most frequently before a booking confirmation. This suggests that time and pricing information are significantly more important to the users than information about the vehicle or about the included insurance. These findings sparked a detailed discussion about the possible reasons for the observed behaviour between the experts, which is not considered in more detail here. Nevertheless, this shows that process models that can be obtained from segmented user interaction logs are an important tool for the analysis of user behaviour and that these models provide a valuable foundation for a more detailed analysis by the process experts. Another observation regarding this model was, that a majority of the users seem to choose a vehicle directly from the dashboard cards (see an example in Figure 5.22) rather than using the search functionality. This suggests that the users are more interested in the vehicle itself, rather than looking for any available vehicle at a certain point in time.

Q14 - Type III - 2% use the *intermediate lock* before ending the booking.
The MOQO application does offer the functionality to lock certain kinds of vehicles during an active booking. This is for example possible for bicycles, which can be locked by the users during the booking whenever they are leaving the bicycle alone. To do so, the `intermediate_lock` and `intermediate_action` screens are used. During the analysis, it was found that 2% of users use this functionality in order to lock the vehicle directly before ending the booking. This noteworthy, as it is not necessary to manually lock the vehicle before returning it. All vehicles are automatically locked by the system at the end of each booking.
One expert argued that this may introduce additional technical difficulties during the vehicle return, because the system will try to lock the vehicle again. These redundant lock operations may introduce errors in the return process. Additionally, the participants suspected that this behaviour is caused by the users need to properly complete the transaction. In a similar way as to returning the key of a hotel room at checkout, instead of leaving it in the hotel room. Moreover, the wording of the buttons that are used in order to lock and return a vehicle on the `booking` screen may be encouraging this behaviour. The participants proposed that some changes to the return process may be made, in order to clarify the situation and discourage this user behaviour.

Q15 - Type III - 5% of users visit *damages* before *cleanliness*.
The application allows users to report damages to the vehicles and rate their cleanliness. During the analysis, it was observed that in some cases users are opening the `damages` screen directly before the `cleanliness` screen without performing any action on that screen. This may imply that users who want to report the cleanliness of a vehicle, are

searching for this functionality on the `damages` screen instead. This theory was rejected by the experts, who clarified that the two screens are both reachable from the `booking` screen and are displayed directly beneath each other. They suspect that the users are going trough a kind of checklist where they first check the vehicle for any damages and then report the cleanliness. However, the UX expert made the additional observation, that only a small percentage of the users seem to follow this routine, which was surprising to them. For the vehicle providers it is generally important that the users are reporting problems with the vehicles, optimally every user would do this for all of their bookings. According to the data, this is however not the case, as only a small percentage of the users are actually using both of the functionalities. The experts therefore concluded that a better communication of these functionalities is required.

Q16 - Type III - Users are confusing the *preview card* and the *booking card* on the dashboard.
As introduced before and can be seen in Figure 5.22, the basic `station_based_dashboard` screen contains preview cards that allow direct access to the corresponding vehicle from the dashboard. A very similar kind of dashboard card does also exist for active bookings of the user, offering direct access to the booked vehicle. During the analysis it was noticed, that the users sometimes opened the `pre_booking` screen from the dashboard and then opened the `booking` screen immediately after that. This suggest, that the users may be confused by the different kinds of cards on the dashboard and mistakenly choose the incorrect card to access their booking. In contrast to the previous questions, this statement is not backed by any concrete data such as a computed metric or a process model. The participants were therefore less involved in the discussion of this observation as it was not justified sufficiently from their perspective.
The experts noted that the different types of cards look very similar to each other, which may confuse the users. Adjustments may be made in the future in order to make the cards easier to visually distinguish. However, they suspected that this observation might be caused by a different behaviour. The same `booking` screen on which this observation was originally based, is used for bookings in the future, active bookings and also bookings that have already been concluded. The `booking` screen of a past booking offers the functionality to book the same vehicle again. The experts therefore assume, that a user first checks a vehicle from the dashboard card, is not satisfied with the vehicle and then opens a past booking in order to book the corresponding vehicle again. This hypothesis was investigated in more detail after the evaluation with the experts. According to the segmented interaction log, many users that follow the variant that was the basis for the original observation, open the `availability_calendar` screen from the `booking` screen. This suggests that the hypothesis of the experts is correct. A more thorough analysis would however require additional attributes in the interaction data, such as the current state (e.g., active booking) of the `booking` screen. Such attributes are however not included in the input data.

5.3 Discussion

In the following, the results that were obtained throughout the evaluation of the proposed method and are presented in the preceding sections are interpreted and discussed in more detail. Where applicable, possible improvements to the method that are based on the findings of the evaluation are proposed.

Method Evaluation The conducted evaluations on the different test cases have shown that the proposed method significantly outperforms the two baseline methods overall. While the DFG baseline generally shows a better accuracy and precision, it has a low recall that lies only slightly above that of the already predetermined case boundaries (by the transitions between users) and systematically underestimates the number of cases in the input log. It is therefore not suitable as a standalone case attribution algorithm. Nevertheless it does provide valuable additional information for the generation of the transition system in the proposed method.

The heuristic baseline overall does have a significantly better recall than both the DFG baseline and the proposed method. This is however enabled by a low precision and accuracy. Regarding the cnd, the heuristic baseline shows the worst performance throughout all test cases. In the worst case scenario, a most frequent case length of 15, the heuristic baseline identifies almost eight times as many cases as there are in the real log. While the other two methods also perform worse in this situation, the increase in cnd is much less significant. It can therefore be said that the heuristic baseline introduces many additional cases and therefore also underestimates the length of the cases significantly. In fact, for the test log with a most common case length of 15, the heuristic baseline produced a segmentation in which the most common case length is one. The heuristic baseline, as it was implemented in the performed evaluation is therefore generally not suitable for the segmentation of an unsegmented UI log that should later be used in process mining applications, because longer relations in the process are not captured in the produced segmentation. However, it has to be mentioned that there are different possible heuristics that could be used, only one of which was considered in the scope of this thesis. While the used heuristic that is based on the median duration of a relation is expected to perform better than the naive approach of a fixed time threshold, there may exist other time based heuristics that are able to produce better segmentations.

For the proposed method, the evaluation showed that it generally performs similarly or in many cases better than the considered baselines. Over all test cases, it achieves an accuracy in the range of 0.7 to 0.8, a (weak) precision in the range of 4.0 to 5.0 and a (weak) recall of around 5.0, resulting in a (weak) f1-score of about 0.5. The usage of the weak variations of the different metrics did only have a minimal impact on the scores obtained for the two baseline methods. The proposed method did however benefit significantly from the use of the weak metrics. This supports the assessment that the method is most well suited in order to discover general areas in which a case boundary is likely, rather than identifying the exact position. This is emphasized further by the fact that the cnd of the proposed method was close to optimal for most test cases and did significantly outperform the two baseline methods. Regarding the later analysis of the segmented log, the ability of an approach to approximately reconstruct the number of cases during case attribution is of great value, even if this introduces noise in the start and end events of the cases. It is also possible that the relative simplicity of the artificial test logs, caused by the low number of different actions, did disadvantage the method compared to the other approaches, because it was not able to fully employ its unique ability to capture more abstract relations between the actions in the log.
The inability of the proposed method to reliably identify the exact case boundaries may also be explained by the unclear definition of a case in the context of user interaction recordings. An example for this can be seen in Figure 5.29.

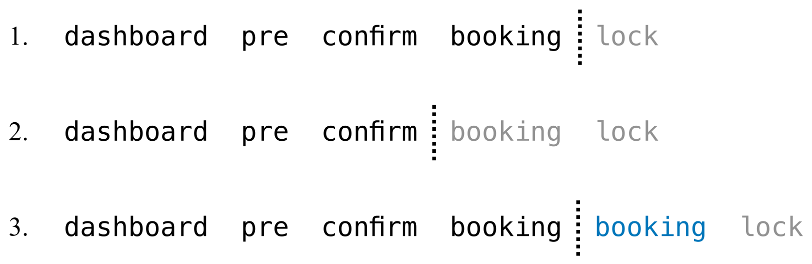

1. dashboard pre confirm booking ⋮ lock
2. dashboard pre confirm ⋮ booking lock
3. dashboard pre confirm booking ⋮ booking lock

Figure 5.29: An example sequence of interactions that can be segmented in three different ways that may all be interpreted as correct or optimal segmentations.

Looking at the example sequence of user interactions from the MOQO app, we see that there are several different options for segmenting the action sequence. For all of the three shown segmentations, it could be argued that they are somehow correct or optimal. In reality, the sequence of actions belongs to a user that booked a vehicle directly through the dashboard of the app, then used the vehicle and at a later point in time returned to the app in order to lock the vehicle. Intuitively a human would split this sequence of actions into two different cases. A first case in which the user books the vehicle and a second case in which the user locks the vehicle. The assignment of the different actions into these two cases is however not unambiguous.
According to option one, the presentation of the booking screen after the successful reservation of the vehicle is assigned to the first case. This is intuitive, because the booking screen is shown immediately after the previous action completes. However, for the second case this means that it can only be represented by a single action. This is not optimal, since it does not capture any context for this action and is also not suitable for process mining. In option two, the booking action is counted towards the second case. This produces a valid second case that can be used in process mining. The first case on the other hand, is cut short and looses an interaction that would generally by attributed to it.
The third option solves this issue by duplicating the contested interaction and assigning one copy to each case. It does however introduce additional interactions that were not part of the original log, which might be considered as unwanted behaviour. The proposed method will never introduce additional behaviour and would therefore only be able to choose between the first two options, both of which can be considered correct. Overall the example emphasizes that in the context of user interaction recordings, the location of a case boundary is not obvious in all situations. This may explain the observed difficulties of the proposed method with pinpointing the exact case boundaries.

In order to evaluate how well the proposed method is able to capture the behaviour in the input log, the action embedding vectors of the trained word2vec models are considered in the following. Figure 5.30 depicts a low dimensional representation of these embedding vectors. The model was trained with the interaction log that was the basis for the conducted case study. The different colors of the dots indicate the different areas of the application. When two actions (dots) are closer to each other in this representation, the actions are related and occur in similar contexts according to the trained model.
For a model that is able to capture the underlying relations between the actions of the log well, actions that occur during the same phase of the usage will be close to each other and will form clusters. Such clustering of different kinds of actions can be observed in figure 5.30. Especially noticeable are the clusters of actions belonging to more distinct phases

of the process, such as actions that occur before, during, or at the end of a booking. It can also be observed that the clusters of phases that are more similar to each other, are closer to each other in the diagram. For example, the cluster of actions that occur before the booking, are closer to those actions that happen during the booking and farther from the ones at the end of the booking. The overall flow of a common interaction with the application is recognizable in the diagram. This recognizable structure in the action embedding vectors suggests, that the underlying word2vec models, are able to abstract the underlying process well.

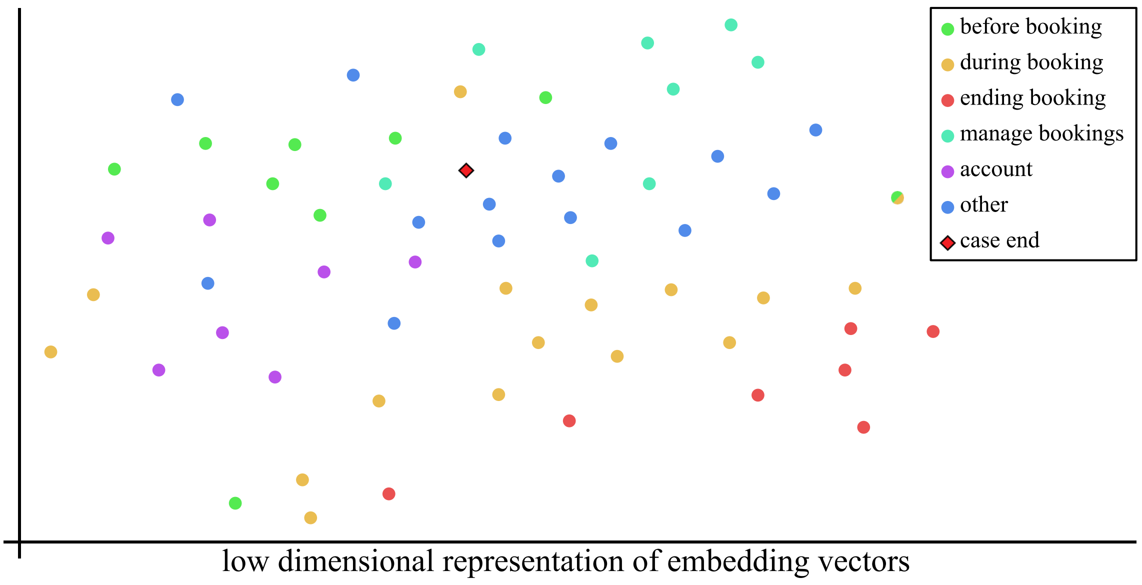


Figure 5.30: A two dimensional representation of the action embedding vectors of one word2vec model that was trained in the context of the case study. Each dot represents the relative location of an action embedding. The closer two dots are, the more similar are their corresponding embedding vectors. The different colors represent different phases of the process. The dimensional reduction is based on the t-SNE method [48].

The embedding of the artificial end action that is introduced before model training is marked in figure 5.30 with a red rhombus. We can see that it is located near the center of the graphic and shows no clear bias toward any phase of the process. This is expected, as case ends may occur in all of the different phases of the process. This however also means, that the embedding is very general and not specific at all. It has no clear relation to any of the other actions, meaning that its likeliness as a center action will never be significant compared to that of other actions. Because of this, the absolute score of the the artificial end action is not the main factor during the segmentation, but rather the change in score between different positions in the traces. The very general nature of the end event may be described as a weakness and may restrict the overall performance of the method.
One possible solution to this problem that would make the end action more specific is to introduce multiple different end actions, depending on the different process phases. This may be achieved by applying a clustering algorithm to the action embeddings, in order to discover the most prominent phases of the process. A different end action for each cluster of actions could then be introduced and the models could be trained based on this enhanced training set. Overall, this change may increase the sensitivity of the method to different types of case boundaries and therefore improve the overall performance.

The evaluation results also suggest, that the method is overall sufficiently robust against changes in the provided interaction log, such as the number of users or the percentage of delayed or abandoned interactions. This is especially true for value ranges that would be expected from real interaction logs. It was shown that the size of the input log does not have a significant impact on the performance of the method and that it is able to perform well even for small interaction logs. An increasing ratio between the number of users and the number of cases in the log is generally beneficial for the method up to a certain point. For the ratio that would be expected in an average real log of interaction recordings, the proposed method does provide good results. A significant decline of the performance of the method could be observed for increasing case lengths. This is somewhat expected, as longer cases have a higher chance of containing loops and repetitions, which are hard to recognize for any case attribution approach. As mentioned before, the test process did also have a relatively small number of different actions, the impact of the case length on the performance therefore relatively high.
Regarding the runtime of the method, it was shown that the preparation step does introduce a significant time overhead over the baseline methods. This overhead does however approach a maximum, which is why the method may still be applied to very large event logs in a reasonable amount of time. Once the preparation phase was executed, the trained method can be continuously applied to similar, but different interaction logs that originate from the same process. The actual segmentation of a given log is comparatively fast and scales linearly with the size of the input log and can therefore be executed rapidly for all kinds of log sizes.

Smartphone Application Case Study The results of the conducted case study, show the high potential of even basic process mining techniques in the context of user behaviour analysis. The evaluation of the analysis results with the process experts revealed a large amount of new and unexpected knowledge about the way the users are actually interacting with the application in a real setting.
It was made clear by the study that the process experts are able to comprehend the basic structure of the application and therefore the underlying process well. However, whenever a more detailed view of one aspect of the process was considered, the experts were not able to correctly assess the real behaviour of the users. The segmentation into distinct cases allowed a detailed analysis regarding the duration of cases and actions, which would not be possible using an unsegmented log. Based on this, it was shown that the experts repeatedly overestimated the length and duration of interactions. Concerning the modeling of the process, the experts were able to identify the structure of the most common interactions, but again lacked detail and accuracy. This is especially true when considering the transitions between different screens. The computed model was more comprehensive and included more behaviour and detail. Based on the segmented interaction log, a complete model of the overall process can be created. This requires a minimal amount of effort compared to a group of process experts manually creating such a comprehensive model.
The log analysis also enabled the comparison between different groups of users, in this case based on the used operating system, and revealed patterns that are the direct opposites to the believes of the experts. Some of the observations that were made during the analysis were already identified by the process experts previously. This suggests that the analyzed interaction log and the underlying segmentation method are suitable for abstracting the real user behaviour and conveying it to the process experts.
When the analysis results are processed, visualized and presented to the experts in the

right way, they were able to produce clear and actionable results based on the findings. For example it was shown that interactions with the app are much shorter than predicted, that the users are utilizing the bookings list much more frequently than expected, that the map dashboard is the most frequently used dashboard, that the search is less important than the dashboard suggestion cards or that users are unnecessarily locking their vehicles before returning them. Based on these findings, the experts are able to derive concrete and actionable changes to the application, with the goal of improving the overall user experience. Many of the results were completely new and unexpected to the experts and were only enabled through the use of the real dataset in conjunction with the proposed case attribution approach. The time that was required for the segmentation of the large provided interaction log and the subsequent analysis is negligible compared to the amount of information that was obtained.
However, when interpreting the results of the analysis, it needs to be considered that some of the findings may be artifacts of the segmentation itself. The results should therefore be considered with caution and whenever possible, be examined for their plausibility. In some cases, the use of additional attributes such as the current state of a booking, or the type of vehicle that is associated with certain screens would provide valuable additional context that can be used in order to validate the obtained findings. These attributes are however not available in all existing datasets and will in many cases require an adaption of the data recording process.

During the case study, only basic process mining techniques were applied to the segmented interaction log. Nevertheless, a significant amount of knowledge about the real user behaviour could be extracted. The potential of more advanced process mining approaches, such as root cause analysis, running case prediction or conformance checking regarding user behaviour analysis is significant, but was not part of the scope of the conducted case study. Overall, the experts were impressed by the findings of the analysis and were able to obtain new insights into the way their users are using the application that were not possible before. Concrete suggestions for improvements could be made and will in the future be implemented in order to improve the user experience of the application, in turn improving the customer satisfaction and lower the required support effort.

Chapter 6

Conclusion

As the performance and capabilities of modern consumer computing devices increase, the number of offered features and the complexity of the software that powers those devices raises as well. This presents the providers of modern software systems with new and ever increasing challenges regarding the user experience design of their applications. In order to address this problem, the majority of software system providers have been observing the behaviour of their users while they are interacting with the applications for some time. Large amounts of data have been generated in the form of unsegmented user interaction logs. This data could however not always be used to its full potential until now.

Summary The method that is proposed in this thesis is designed in order to split unsegmented user interaction logs into discrete cases that correspond to tasks of the user. The goal of performing this case attribution is to unlock the rich process mining ecosystem for the analysis of the user behaviour based on interaction recordings. Process mining allows for a significantly more detailed and advanced analysis of the log data compared to the basic data science techniques that are most frequently used today. However, in order to be able to apply process mining to the interaction recordings, case attribution has to be performed first. Existing approaches to this problem have long runtimes and often rely on additional interaction attributes, or the timestamps of the interactions. They can therefore not be applied to all kinds of interaction data and are not able to identify cases that directly follow each other, or have larger gaps in between the interactions. The proposed method explores a new approach to the case attribution problem based on action embeddings that is independent of additional attributes and timestamps. It is based on the generation of an artificial training log that resembles the input log and is used in order to train the underlying word2vec models. These models are applied in order to assess the likeliness of a case boundary for all positions in the log. Based on this, the split points in between cases are computed and the input log is segmented.
The detailed conducted evaluation shows that the proposed segmentation approach provides a good performance for different test logs with varying log characteristics and has an acceptable runtime, even for large interaction logs. It is generally able to identify the approximate location of case boundaries and discovers a number of cases that is close to reality. In many situations it does outperform the considered baseline approaches. In contrast to many of the existing approaches, it is not overly sensitive to deviations in the action durations and the occurrence of delayed or abandoned interactions. The evalua-

tion also showed that the underlying word2vec models are able to accurately abstract the behaviour that is contained in the input log. They are therefore well suited for being the foundation of the method. In the conducted case study, that was based on a real dataset, a user interaction log was segmented using the proposed method. The resulting log was then successfully analyzed using varying basic process mining approaches. The analysis provided the experts with concrete and actionable results that would not have been possible using traditional interaction recording analysis methods.

Limitations & Outlook While the conducted evaluation has proven the general potential of the discussed approach for case attribution, the quality of the produced segmentations may be improved further in the future. It was shown that the word2vec models are generally well suited in order to capture the relations in the log. However, as mentioned before, only a single artificial end action was introduced in the training data. Since case boundaries may appear in different contexts, the scoring of the models may be improved by considering different types of case boundaries separately. For each type of boundary, a unique end action could be inserted in between the artificial training cases. The quality of the generated traces also has a significant impact on the overall performance of the proposed case attribution method. The more closely the artificial log resembles the input log, the more accurate will be the produced segmentation. Other, more advanced log generation methods than the method that was introduced here may therefore be used for the generation of the training log.
Furthermore, the detection of the cut points based on the scores that are produced by the models could be based on a different criterion than the *peak* condition that was used here. During evaluation, it was observed that there is a considerable fraction of computed split points that are *one-offs*, meaning they are missing the real case boundary by only one position to either side. This emphasizes that the scoring generally works, but the cut point detection may not be accurate enough. Another improvement regarding these near misses could be to allow the method to duplicate certain interactions. This might be beneficial, because in certain cases, the last interaction of a case may also be the starting point for the following case. Such situations cannot be modeled adequately by the proposed method in its current state.
Regarding the field of case attribution in general, there is a notable lack of metrics that are suitable in order to judge the quality of a segmentation when no labeled ground truth data is available. Fundamental research in this area is therefore required, in order to obtain a solid benchmark that can be used in order to assess the quality of produced segmentations and in order to be able to compare competing approaches to each other.

Overall, it can be concluded that the proposed case attribution method is able to produce log segmentations that are able to meaningfully represent the underlying user behaviour. The method achieves this without the need for additional log attributes or accurate timestamp information and can be applied to large interaction logs. This makes the log data available to new application areas such as robotic process automation or process mining. When the resulting segmented log is used in process mining workflows, it can provide valuable and actionable results to the process experts that go beyond what was possible before. This emphasizes the high potential of user interaction logs, in conjunction with case attribution approaches and process mining methods, for the modern and advanced analysis of user behaviour.

Bibliography

[1] World Wide Web Consortium. Html 3.2 reference specification, 1997. URL https://www.w3.org/TR/2018/SPSD-html32-20180315/. Accessed 15.06.21.

[2] WHATWG. Html living standard, 2021. URL https://html.spec.whatwg.org/print.pdf. Accessed 15.06.21.

[3] Randy Nelson. The size of iphone's top apps has increased by 1,000% in four years, 2017. URL https://sensortower.com/blog/ios-app-size-growth. Accessed 15.06.21.

[4] Christina Bröhl, Peter Rasche, Janina Jablonski, Sabine Theis, Matthias Wille, and Alexander Mertens. Desktop pc, tablet pc, or smartphone? an analysis of use preferences in daily activities for different technology generations of a worldwide sample. *Human Aspects of IT for the Aged Population. Acceptance, Communication and Participation*, pages 3–20, 2018.

[5] Monika Dhandi and Rajesh Kumar Chakrawarti. A comprehensive study of web usage mining. In *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, pages 1–5, 2016. doi: 10.1109/CDAN.2016.7570889.

[6] Jürgen Cito, Johannes Wettinger, Lucy Ellen Lwakatare, Markus Borg, and Fei Li. Feedback from operations to software development—a devops perspective on runtime metrics and logs. In Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer, editors, *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pages 184–195, Cham, 2019. Springer International Publishing.

[7] Ricardo Pérez-Castillo, Barbara Weber, Ignacio Guzmán, Mario Piattini, and Jakob Pinggera. Assessing event correlation in non-process-aware information systems. *Software and Systems Modeling*, 13:1–23, 09 2012. doi: 10.1007/s10270-012-0285-5.

[8] Diana Jlailaty, Daniela Grigori, and Khalid Belhajjame. Business process instances discovery from email logs. In *2017 IEEE International Conference on Services Computing (SCC)*, pages 19–26, 2017. doi: 10.1109/SCC.2017.12.

[9] Dina Bayomie, Ahmed Awad, and Ehab Ezat. Correlating unlabeled events from cyclic business processes execution. In Selmin Nurcan, Pnina Soffer, Marko Bajec, and Johann Eder, editors, *Advanced Information Systems Engineering*, pages 274–289, Cham, 2016. Springer International Publishing. ISBN 978-3-319-39696-5.

[10] Shaya Pourmirza, Remco Dijkman, and Paul Grefen. Correlation miner: mining business process models and causal relations without case identifiers. *International*

Journal of Cooperative Information Systems, 26(2), June 2017. ISSN 0218-8430. doi: 10.1142/S0218843017420023.

[11] Andres Jimenez-Ramirez, Hajo A. Reijers, Irene Barba, and Carmelo Del Valle. A method to improve the early stages of the robotic process automation lifecycle. In Paolo Giorgini and Barbara Weber, editors, *Advanced Information Systems Engineering*, pages 446–461, Cham, 2019. Springer International Publishing. ISBN 978-3-030-21290-2.

[12] Wil M. P. van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose R.P., Peter van den Brand, Ronald Brandtjen, Joos C. A. M. Buijs, Andrea Burattin, Josep Carmona, Malú Castellanos, Jan Claes, Jonathan Cook, Nicola Costantini, Francisco Curbera, Ernesto Damiani, Massimiliano de Leoni, and Moe Wynn. Process mining manifesto. In *Lecture Notes in Business Information Processing*, volume 99, pages 169–194, 08 2011. doi: 10.1007/978-3-642-28108-2_19.

[13] Wil M. P. van der Aalst. *Process Mining - Data Science in Action.* Springer, 2016.

[14] Tom-Hendrik Hülsmann. Integration of mongodb in pm4py for preprocessing event data and discover process models. *Bachelor Thesis*, Sep. 2019.

[15] Wil M. P. van der Aalst, Rubin, B. F. Van Dongen, E. Kindler, and C. W. Günther. Process mining: A two-step approach using transition systems and regions. Technical report, BPM Center Report BPM-06-30, BPM Center, 2006.

[16] Volodymyr Leno, Adriano Augusto, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Artem Polyvyanyy. Identifying candidate routines for robotic process automation from unsegmented ui logs. In *2020 2nd International Conference on Process Mining (ICPM)*, pages 153–160, 2020. doi: 10.1109/ICPM49681.2020.00031.

[17] Tomas Mikolov, Kai Chen, Greg S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. URL `http://arxiv.org/abs/1301.3781`.

[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Neural and Information Processing System (NIPS)*, 2013. URL `https://arxiv.org/abs/1310.4546`.

[19] Sho Tsukiyama, Md Mehedi Hasan, Satoshi Fujii, and Hiroyuki Kurata. Lstm-phv: Prediction of human-virus protein-protein interactions by lstm with word2vec. *bioRxiv*, 2021. doi: 10.1101/2021.02.26.432975.

[20] Yu-Fang Zhang, Xiangeng Wang, Aman Chandra Kaushik, Yanyi Chu, Xiaoqi Shan, Ming-Zhu Zhao, Qin Xu, and Dong-Qing Wei. Spvec: A word2vec-inspired feature representation method for drug-target interaction prediction. *Frontiers in Chemistry*, 7:895, 2020. ISSN 2296-2646. doi: 10.3389/fchem.2019.00895. URL `https://www.frontiersin.org/article/10.3389/fchem.2019.00895`.

[21] Karuna Lakhani and Apurva Narayan. A neural word embedding approach to system trace reconstruction. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 285–291, 2019. doi: 10.1109/SMC.2019.8914322.

[22] Wil M. P. van der Aalst, Martin Bichler, and Armin Heinzl. Robotic process automation. *Business & Information Systems Engineering*, 60(4):269–272, Aug 2018. ISSN 1867-0202. doi: 10.1007/s12599-018-0542-4. URL https://doi.org/10.1007/s12599-018-0542-4.

[23] Jutta Reindler. *Siemens Healthineers: Process Mining as an Innovation Driver in Product Management*, pages 143–157. Springer International Publishing, Cham, 2020.

[24] Andrea Marrella and Tiziana Catarci. Measuring the learnability of interactive systems using a petri net based approach. In *Proceedings of the 2018 Designing Interactive Systems Conference*, DIS '18, page 1309–1319, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450351980. doi: 10.1145/3196709.3196744. URL https://doi.org/10.1145/3196709.3196744.

[25] Dominik Janssen, Felix Mannhardt, Agnes Koschmider, and Sebastiaan J. van Zelst. Process model discovery from sensor event data. In Sander Leemans and Henrik Leopold, editors, *Process Mining Workshops*, pages 69–81, Cham, 2021. Springer International Publishing. ISBN 978-3-030-72693-5.

[26] Diogo R. Ferreira and Daniel Gillblad. Discovering process models from unlabelled event logs. In Umeshwar Dayal, Johann Eder, Jana Koehler, and Hajo A. Reijers, editors, *Business Process Management*, pages 143–158, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[27] Dina Bayomie, Ahmed Awad, and Ehab Ezat. Correlating unlabeled events from cyclic business processes execution. In Selmin Nurcan, Pnina Soffer, Marko Bajec, and Johann Eder, editors, *Advanced Information Systems Engineering*, pages 274–289, Cham, 2016. Springer International Publishing. ISBN 978-3-319-39696-5.

[28] Dina Bayomie, Claudio Di Ciccio, Marcello La Rosa, and Jan Mendling. A probabilistic approach to event-case correlation for process mining. In Alberto H. F. Laender, Barbara Pernici, Ee-Peng Lim, and José Palazzo M. de Oliveira, editors, *Conceptual Modeling*, pages 136–152, Cham, 2019. Springer International Publishing. ISBN 978-3-030-33223-5.

[29] Santiago Aguirre and Alejandro Rodriguez. Automation of a business process using robotic process automation (rpa): A case study. In Juan Carlos Figueroa-García, Eduyn Ramiro López-Santana, José Luis Villa-Ramírez, and Roberto Ferro-Escobar, editors, *Applied Computer Sciences in Engineering*, pages 65–71, Cham, 2017. Springer International Publishing. ISBN 978-3-319-66963-2.

[30] Rehan Syed, Suriadi Suriadi, Michael Adams, Wasana Bandara, Sander J.J. Leemans, Chun Ouyang, Arthur H.M. ter Hofstede, Inge van de Weerd, Moe Thandar Wynn, and Hajo A. Reijers. Robotic process automation: Contemporary themes and challenges. *Computers in Industry*, 115:103162, 2020. ISSN 0166-3615. doi: https://doi.org/10.1016/j.compind.2019.103162. URL https://www.sciencedirect.com/science/article/pii/S0166361519304609.

[31] Junxiong Gao, Sebastiaan J. van Zelst, Xixi Lu, and Wil M. P. van der Aalst. Automated robotic process automation: A self-learning approach. In Hervé Panetto, Christophe Debruyne, Martin Hepp, Dave Lewis, Claudio Agostino Ardagna, and Robert Meersman, editors, *On the Move to Meaningful Internet Systems: OTM 2019*

Conferences, pages 95–112, Cham, 2019. Springer International Publishing. ISBN 978-3-030-33246-4.

[32] Jesús Chacón Montero, Andres Jimenez Ramirez, and Jose Gonzalez Enríquez. Towards a method for automated testing in robotic process automation projects. In *2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST)*, pages 42–47, 2019. doi: 10.1109/AST.2019.00012.

[33] Andres Jimenez-Ramirez, Hajo A. Reijers, Irene Barba, and Carmelo Del Valle. A method to improve the early stages of the robotic process automation lifecycle. In Paolo Giorgini and Barbara Weber, editors, *Advanced Information Systems Engineering*, pages 446–461, Cham, 2019. Springer International Publishing.

[34] Christian Linn, Phileas Zimmermann, and Dirk Werth. Desktop activity mining - a new level of detail in mining business processes. In Christian Czarnecki, Carsten Brockmann, Eldar Sultanow, Agnes Koschmider, and Annika Selzer, editors, *Workshops der INFORMATIK 2018 - Architekturen, Prozesse, Sicherheit und Nachhaltigkeit*, pages 245–258, Bonn, 2018. Köllen Druck+Verlag GmbH.

[35] Volodymyr Leno, Adriano Augusto, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Artem Polyvyanyy. Identifying candidate routines for robotic process automation from unsegmented ui logs. In *2020 2nd International Conference on Process Mining (ICPM)*, pages 153–160, 2020. doi: 10.1109/ICPM49681.2020.00031.

[36] Susan Dumais, Robin Jeffries, Daniel M. Russell, Diane Tang, and Jaime Teevan. Understanding user behavior through log data and analysis. In *Ways of Knowing in HCI*, 2014.

[37] B. Mobasher, R. Cooley, J. Srivastava, Baoyao Zhou, Siu Cheung, José M. Domenech, Koichiro Mihara, M. Terabe, K. Hashimoto, Mehdi Heydari, R. A. Helal, K. I. Ghauth, M. Bayir, I. H. Toroslu, A. Cosar, and Guven Fidan. A novel technique for sessions identification in web usage mining preprocessing. 2011.

[38] Gabriel Marques Tavares and Sylvio Barbon. Analysis of language inspired trace representation for anomaly detection. In Ladjel Bellatreche, Mária Bieliková, Omar Boussaïd, Barbara Catania, Jérôme Darmont, Elena Demidova, Fabien Duchateau, Mark Hall, Tanja Merčun, Boris Novikov, Christos Papatheodorou, Thomas Risse, Oscar Romero, Lucile Sautot, Guilaine Talens, Robert Wrembel, and Maja Žumer, editors, *ADBIS, TPDL and EDA 2020 Common Workshops and Doctoral Consortium*, pages 296–308, Cham, 2020. Springer International Publishing.

[39] Jari Peeperkorn, Seppe vanden Broucke, and Jochen De Weerdt. Conformance checking using activity and trace embeddings. In Dirk Fahland, Chiara Ghidini, Jörg Becker, and Marlon Dumas, editors, *Business Process Management Forum*, pages 105–121, Cham, 2020. Springer International Publishing.

[40] Pieter De Koninck, Seppe vanden Broucke, and Jochen De Weerdt. act2vec, trace2vec, log2vec, and model2vec: Representation learning for business processes. In Mathias Weske, Marco Montali, Ingo Weber, and Jan vom Brocke, editors, *Business Process Management*, pages 305–321, Cham, 2018. Springer International Publishing.

[41] Sylvio Barbon, Paolo Ceravolo, Ernesto Damiani, and Gabriel Tavares. Evaluating

trace encoding methods in process mining. 10 2020. ISBN 978-3-030-70649-4. doi: 10.1007/978-3-030-70650-0_11.

[42] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[43] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020. doi: 10.1038/s41586-020-2649-2.

[44] Radim Rehurek and Petr Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.

[45] Kevin Patel and Pushpak Bhattacharyya. Towards lower bounds on number of dimensions for word embeddings. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 31–36, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing. URL `https://aclanthology.org/I17-2006`.

[46] Digital Mobility Solutions. Moqo sharing app. URL `https://moqo.de/app`. Accessed 11.06.2021.

[47] Fluxicon. Disco. URL `https://fluxicon.com/disco/`. Accessed 09.08.2021.

[48] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL `http://jmlr.org/papers/v9/vandermaaten08a.html`.