Interactive Sequence Mining

Providing Valuable Insight into Application Usage by Exploring Usage Patterns in Application Traces

Rutger Kerkhoff

Supervisor
prof. dr. A.P.J.M. Siebes

A thesis presented for the degree of
Master of Science

Universiteit Utrecht

Computing Science
Utrecht University
Netherlands
February 1, 2016

Interactive Sequence Mining

Providing Valuable Insight into Application Usage by Exploring Usage Patterns in Application Traces

Rutger Kerkhoff

Abstract

Understanding how people use a program is vital for proper development and support of any application. Numerous ways to gain this insight exist, but those often require user feedback or further adjustments to the program. In this thesis we run multiple pattern mining techniques on existing program traces to reveal patterns in application usage. This process is illustrated and verified by taking existing log files of the application TeleDIA which is currently used in production in various healthcare institutions. To make numerous algorithms available to users new to the field of pattern mining we created an interactive tool. The tool supports the user by providing sensible default parameters and inferring others from the dataset. To further support the user the program provides an interactive graph-view that visualises patterns. The view also allows the user to interactively grow the patterns to specific interesting sequences he is interested in, which are then used as a highly-effective filter on all pattern suggestions. With this method we tackle part of the ever-present problem of pattern explosion in pattern mining. Experiments show we are able to mine almost all patterns present in the trace files. Next to quantitative results we interviewed key users to confirm that the mined patterns are interesting, and in some cases novel. This is supported by the fact that changes were made to TeleDIA, changes based on insight gained using our interactive sequence mining prototype.

Contents

1 Introduction

1.1 Problem description

We all use software daily; from simply reading email and browsing Facebook to using a complicated stock brokering system. With the rise of smartphones, software has been even more intertwined with our daily activities. The ubiquity of software makes that everyone can identify different kinds of software. Some software helps us efficiently perform a task, while others take painstakingly many interactions before getting anything done. For some tasks this might be impossible to avoid, but often this is due to the fact that we use the software in a different way than intended by its creators [22].

Problems or inefficiencies can arise because of this mismatch, and may go unnoticed. This can partly be explained by the fact that developers do not always have sufficient domain knowledge or have to deal with changing requirements [7]. Providing insight into how users interact with the software can help build towards a better match between intend and actual usage, and ultimately towards an increase in software quality [16].

In this thesis the web-application TeleDIA will be examined. TeleDIA is an application with a central server and is accessed by its users via a browser and the HTTP protocol. The application was created to streamline the process of patient diagnosis for examinations. There is an expected mismatch between intended usage by TeleDIA developers and the way the system is used by their users. The application was developed with a certain workflow in mind, but it is unclear if this still fits daily practice. On the other hand the intended workflow could be efficient but the training they give their users might not be adequate.

To solve this problem the actual usage is modelled. The data available for analysis is a clickstream, which consists of requests made to the application server. To model the usage from these traces, algorithmic data analysis techniques can be applied.

The process in the application is not linear: depending on the choices made different input fields are presented, replacing part of the screen. However, on a high level the process is strictly defined. Regardless of the choices made on a single page, there is a fixed set of buttons to finish or proceed in a task. Different groups of users can be identified in the application which will each perform different, sometimes overlapping, tasks.

The intended goal of the thesis is to answer the question: How can valuable insight into application usage be provided by modelling interaction using application traces?

1.2 Research questions

Specifying what needs to be modelled exactly is required to further specify the problem. There are many possible paths to take through the TeleDIA. To grasp general usage it is not interesting to model how a single user works with the application, but rather how the application is used by the majority. This however does not mean that a single users flow can be ignored, as in some cases it can provide valuable insight into more fine grained usage. Three kinds of usage patterns can be identified that, for our needs, together adequately model how an application is used []. They will be listed in the next paragraph and be elaborated upon in the paragraphs thereafter.

The first kind of patterns in the application consists of actions that are shown by the majority of the users, general usage. More specifically, the general usage consist of patterns that are general and reliable, possibly novel and surprising. Secondly, possible unintended usage in the form of exceptions to the general patterns will be identified. Together with the general patterns these will describe *what* happens in the application. Lastly the co-occurrence of these patterns, to model *when* patterns occur in a session, will need to be identified. The descriptive pattern terms are used as defined by Geng and Hamilton [14].

The first kind of patterns that will receive focus are those that model the general application usage. Specific actions, which are supposed to vary often, will be ignored. The value of these patterns lies in the fact that this represents the majority of the users, and should closely follow the intended use.

The unintended usage will be defined as usage that does not agree with the frequent episodes, but does describe a similar use case or sub-pattern in the application. For this a similarity measure will have to be defined. A sequence will be deemed similar if the steps deviate only slightly and the sequence achieves the same goal. We suspect that most deviations from a pattern are based on significant events, encoding alternate workflows. While some might be valid, and again confirm intended usage, other patterns can show erroneous or inefficient paths. The fact that they are still similar to intended patterns is required to filter out completely different use cases and isolate probable errors.

The associated patterns will give a higher level view of the usage. It will show what a user typically does in a session by correlating the patterns describing use cases identified in the previous two steps. These correlations will also be checked against intended usage.

A common problem with mining patterns is that, depending on parameter settings, either too few or to many patterns are mined. While frequency and other constraints can be used to filter these, the results are still too overwhelming for a user to grasp. To tackle this problem we will take a user-centred approach. The knowledge a user already has about an application is invaluable and can be used as a highly effective filter []. Note that this means that we do not aim to exhaustively model all the traces, just the set required for insight.

What remains are the rare patterns which do not show any similarity to other patterns. While these can still be valuable, the logs unfortunately contain inadequate information to directly infer any conclusion from them. Complementing the log files with the knowledge of the user can reveal them, but will depend heavily on user input.

The last problem that needs to be tackled is the complexity of the pattern generating software. The analysis performed is preferably repeated regularly and preferable no data mining expertise is required for this. The application should be usable by developers. While some might have in-depth knowledge about patterns, most do not. To make the application accessible the usual pattern mining parameters that have to be defined will be provided. To provide these automatically, the running time, space requirement and usability of the results need to be controlled. Ultimately achieving parameterless pattern mining, for our domain.

To formalise; the main research question and sub-questions are listed below.

How can valuable insight into application usage be provided by modelling usage patterns in application traces?

(a) How can insight into general application usage be provided?

(b) How can similar patterns be identified?

(c) How can patterns be associated?

(d) How can these patterns be used in an user centred mining application?

(e) Can we achieve parameterless mining for program traces?

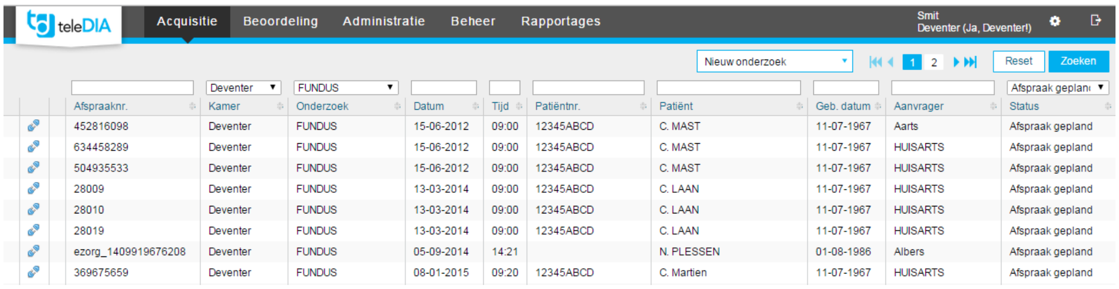
To evaluate the results both synthetic test data and the TeleDIA logs will be used as input. The synthetic test data will show the correctness and performance of the algorithms. The TeleDIA logs will show the real-life relevance of the results found by the algorithm. Due to the fact that interestingness for patterns is subjective the test performed on the TeleDIA logs is empirical. The identified interesting patterns will be reviewed by key users. The key users should be able to identify mismatches from both the TeleDIA developers point of view and the users' their viewpoint.

1.3 TeleDIA

In this section the application that initiated this thesis is discussed. Next to showing practical usability of the results this case will be used for examples to clarify theory throughout this report.

TeleDIA is an application developed to streamline the process of patient diagnosis. We can identify four steps in the process that are also present in the software: *acquisition*, *assessment*, *administration* and *feedback*.

1.3.1 Workflows



Afspraaknr.	Kamer	Onderzoek	Datum	Tijd	Patiëntnr.	Patiënt	Geb. datum	Aanvrager	Status
	Deventer	FUNDUS							Afspraak geplan
452816098	Deventer	FUNDUS	15-06-2012	09:00	12345ABCD	C. MAST	11-07-1967	Aarts	Afspraak gepland
634458289	Deventer	FUNDUS	15-06-2012	09:00	12345ABCD	C. MAST	11-07-1967	HUISARTS	Afspraak gepland
504935533	Deventer	FUNDUS	15-06-2012	09:00	12345ABCD	C. MAST	11-07-1967	HUISARTS	Afspraak gepland
28009	Deventer	FUNDUS	13-03-2014	09:00	12345ABCD	C. LAAN	11-07-1967	HUISARTS	Afspraak gepland
28010	Deventer	FUNDUS	13-03-2014	09:00	12345ABCD	C. LAAN	11-07-1967	HUISARTS	Afspraak gepland
28019	Deventer	FUNDUS	13-03-2014	09:00	12345ABCD	C. LAAN	11-07-1967	HUISARTS	Afspraak gepland
ezorg_1409919676208	Deventer	FUNDUS	05-09-2014	14:21		N. PLESSEN	01-08-1986	Albers	Afspraak gepland
369675659	Deventer	FUNDUS	08-01-2015	09:20	12345ABCD	C. Martien	11-07-1967	HUISARTS	Afspraak gepland

Figure 1: A screenshot of TeleDIA, showing the list of patient scheduled for *acquisition*

Acquisition consists of collecting patient personal data, possible medical history and actual test results in the system. The TeleDIA software connects with various other computer programs and physical devices to collect data needed for assessments. A few of the connections include, but are not limited to, getting ECG-, spirometry- and fundus photography-data. Most of these results are stored within the system, in a few cases the data is stored in a third party database.

The next step in the diagnostic process is *assessment*. A specialist can log in from any remote location and directly access all the collected data. All of this information can be viewed in the browser used to connect to the system. Like in acquisition, it is possible to launch external applications to analyse specific results. The specialist will input his assessments, possibly consult colleagues within the system, and finishes by finalising the assessment.

The *administration* step consist of checking reports and handling so called 'no shows'. For this use case TeleDIA shows an overview of all planned appointments and current cases. The view also provides an overview of progress and possible external assessments. Administration also allows a user to instruct the system to send messages to other systems.

In the last step, *feedback*, a physician can view the assessment and case data and discuss this with the patient. Again the user can view the collected data within TeleDIA. External tools are not available in this step.

The application is completely web based, which means administrative procedures and settings are also configured through a browser, and also show up in the log data.

1.3.2 TeleDIA logs

The fact that TeleDIA is a client-server application using the HTTP protocol makes it very similar to a traditional webserver. Like a traditional web server we have a central log file which logs all requests made to the server. A lot of work has already been done in this area, see related work Section 2.4 for an overview. For a sample of a log file please see Appendix A.

Where TeleDIA differs from an website is that it visually and functionally represents a more classical administration program. A lot of AJAX-requests (Asynchronous javascript and XML requests) are implemented to prevent the browser from refreshing often. This results in a log file that is lot more elaborate, with information as specific as what button has been clicked or in some cases even if a character has been typed.

A problem that we encounter is that not all lines have a similar structure, and not all variables are always included. Another peculiarity in the data is that some fields and actions generate multiple requests, even though they are triggered by a single user action. The solution for this and all of the other problems specifically related to interpreting the logfile can be found in Section 3.

One thing to keep in mind is that the software is tailored for the specific customers, healthcare facilities. Each customer has its own diagnostic process, which is captured in TeleDIA. While there are definitely similarities there exist many differences as well, this makes direct comparison of instances problematic and each instance and thus logfile unique.

1.3.3 Patterns in TeleDIA

A pattern in the case of TeleDIA is a set of sequential user actions. For example, with a user at the patient overview table, the user selects a patient. Fills in various fields and checkboxes, some of these trigger a new request, others do not. These fields and check-boxes concern the specific symptoms of a patient, and are thus highly likely to vary. A user can start an external program by clicking a button in TeleDIA, use the program, and then load the results with another click in TeleDIA. In this case we do not have data about the external program. Finally, if the user has filled the necessary fields, he can click on the 'finish' button. In this example the log will contain at least 6 requests, most likely more. In the above example it is clear that we can have wildly different traces. However, not all checkboxes and fields trigger requests, only those that later influence the process. Thus while we might have patient specific variation, this will likely be still the same for larger groups of patients.

A common problem in pattern mining is the question 'what entails a sequence?'. We have a couple of options, from using whole TeleDIA-sessions, to manually identifying steps that indicate the start and end of use cases. Note that to find correlated patterns we always have to consider the sessions as a variable.

Another known problem is that some pages are encoded by multiple different URLs. This can hide patterns as infrequent in the logs if the spread is too high. During preprocessing we will attempt to mitigate this problem.

When performing sequence mining algorithms we often employ a window, as mentioned in the related work Section 2.2. What size of window should we use, and should we enforce a minimal length? More about this in Section 4.1.1.

A final peculiarity about the data is that the diagnostic process heavily involves human interaction, and there are a multitude of reasons a user can take a lot more or less time for some specific step. We will have to keep this in mind when defining minimum and maximum sequence lengths.

1.3.4 Analysis goals

From the view of TeleDIA developers we want to achieve several practical goals. As a start they want the general and most frequent patterns to be easily generated and viewed. Deviating patterns should be visible, and for every action they want to know the frequency.

Both general and specific patterns are deemed interesting. Specific checkboxes and field depending on the symptoms of a patient should be visible if required, but not prevent the system from suggesting the general overlying pattern.

Furthermore it is suspected that administration procedures can be optimized. Due to the lack of insight into the actual usage, the trade-off between possible gain and needed investment is often unclear. Most administrative procedures only happen sporadically, there should however be a way to surface them.

To effectively use TeleDIA training is provided for the users. Training is needed because the application can perform quite complex and critical functions. Not all individuals are directly trained by experts, but rather a small group of a company acquiring the software is trained. This group will in turn train the rest of the users. It is within the realm of possibilities that an error in training occurs, and that they can affect a relatively large group of people if this happens early on. To tackle this use case it, should be possible to validate expected patterns against the log file.

Finally there was an interest voiced in specific use cases. Specific pattern statistics for these use cases can be used to predict impact of changes to the code base.

2 Related work

The basis of this research is derived from various theories from the field of pattern mining. In this section various related work is compared. We start with the origin of the field and how our problem is present from the start in Section 2.1. In Section 2.2 we will zoom in on various frequent sequence mining methods. Section 2.3 discusses alternatives to mining on frequency. Related working concerning application log mining is described in Section 2.4. Finally existing work on interactive mining of patterns is compared in Section 2.5.

2.1 Pattern mining

The field was introduced by Agrawal et al. [1], by discussing transaction mining on itemsets. We will concern ourselves with events in sequences, instead of itemsets in transactions. The added time element is elaborately explained by Manilla et al. [26]. The main topic we will discuss has been ever present. However, as often as it is discussed it is overlooked: What are the patterns that the result set should contain?

Traditionally, as in the above mentioned papers, the result set contained frequent patterns. Simply take the most occurring sequences and add them to the result set. This leads to another problem: Next to defining a criteria we need to define at what level this criteria becomes relevant. Usually this is handled by setting parameters, in the frequent pattern case a support threshold. This approach, next to requiring domain knowledge, is error prone, and can lead to missing true patterns or finding sporadic patterns as explained by Keogh et al. [23]. Next to identifying the problem Keogh et. all try to solve this by using techniques based on Kolmogorov complexity, an approach we also take. Where they concern themselves with clustering, classification and anomaly detection, we will pursue a complete model of the database.

Various other ways of tackling the previously defined problems have been proposed. From a pattern mining point of view the non-redundant closed patterns are defined by Pasquier et al. [27]. In the paper the authors improve efficiency of the algorithms by only considering the closed itemset lattice. In practice this means that completely redundant itemsets are not evaluated. This works especially well on dense datasets. The even more succinct maximal patterns as described by Zaki et al. [41] can improve efficiency even more. However, as opposed to closed patterns, maximal patterns are not lossless: information about frequency is lost to increase performance. We will only use closed patterns to obtain lossless results. Another approach was published by Knobbe et al. [24], their pattern team approach consider patterns together, instead of trying to quantify their individual interestingness. Pattern team discovery is done as a second step, a way to filter a large result set. We will provide different ways of filtering the results, but our method could be extend to include one or more of the heuristic pattern team definitions.

Restricting the result set, for example by filtering early in the process, is often performed by applying constraints to the result set. Note that this is often done to increase performance, not improve the result set. Pei et al. [29] describe this well. They allow for both monotone and anti-monotone constraints, giving the ability to highly specify the result set. Bonchi et al. [3] extend this to a pre-processing approach. Their algorithm, named ExAnte, aims to reduce the search space and the input dataset based on constraints. Both approaches can be highly efficient and provide interesting result sets. Due to efficiency it would be possible to iteratively run and adjust the parameters and get the request information. They are however, too involved. Our mining algorithms will be run by possibly inexperienced dataminers, who will not be able to correctly define parameters not to mention correct constraints.

2.2 Frequent sequence mining

In sequence mining we see similar approaches to restrict the result set. Though not all methods translate directly; events in a sequence have an order as opposed to being just a set. Work has been done on this, for example Tatti et al. [36] developed a new method for closed strict episodes. We can also identify a few methods specific to sequences, like using various windows as implemented by Hirate et al. [20]. The algorithms we include are build upon the same principles as portrait in those papers.

There are numerous other ways of restricting the number of results. The interested reader is referred to Garofalakis et al. [13], describing how to apply regular expressions constraints to sequential pattern mining. This could be a useful addition to our approach, giving more control to the expert user. However, inexperienced users are our main target, and we shift our focus to less involved methods. Han et al. [17] provide an algorithm for mining top-k patterns without minimum support. While removing the need for minimum support Han et al. require a minimum k to be defined as well as minimum length, replacing one parameter with two others. It can be interesting to test what novice users prefer, for our approach we suspect the single parameter is preferred and thus leave this for future research. As a final note on the different kind of patterns we refer to multidimensional frequent patterns and patterns containing valued items, for example as defined by Pino et al [32] respectively Fournier et al. [12]. These two methods allow to use more data then just the events themselves. This allows for new analysis, new quality measures and thus new ways of filtering. While we do not implement them in our approach, it could easily be extended to support them. This is one of the most promising extensions, as it allows to use extra information in the analysis and goes beyond sequence mining.

2.3 Sequence mining variations

To cover a larger area than just frequent patterns we borrow an algorithm from DNA sequence alignment. Sequence alignment is a closely related field; DNA can be seen as a very long sequence of a restricted set of events. We implement a Smith-Waterman [34] like algorithm. While this is often too slow for DNA sequence comparison it suffices for our relatively short sequences. More advanced ideas about combining frequent patterns and similarity is explained by Laxman et al. [25]. They model frequent patterns with a specific type of hidden Markov models (HMM), an Episode Generating HMM. While this seems similar to our approach, this method again results in giving a limited set of answers to the pattern mining problem.

A different family of pattern mining algorithms is based on minimum description length (MDL). Instead of counting frequency or applying other constraints, the MDL approach works towards compressing the database. This was introduced by Vreeken et al. [38] with their KRIMP algorithm. KRIMP compresses transaction databases, whereas we consider sequences. We therefore will use an implementation based on KRIMP, the SQS algorithm. SQS, Summarising event seQuenceS, was proposed by Tatti et al. [37]. Next to providing a way to use SQS as a filter the paper also introduces a parameterless approach for sequence databases, again maximising the compression. We will use the latter of the two algorithms.

Lastly we touch upon the subject of associated pattern mining, in our case sequential rules. We base our solution on the algorithm presented by Agrawal et al. [2] to generate all rules based on frequent itemsets. These itemsets will be found using Zaki et al. their CHARM algorithm to find closed itemsets [40]. As a novelty we do not find rules for our itemsets, but rather between our previously found closed sequential patterns.

2.4 Existing log mining applications

Pattern mining has a lot of practical applications, one relatively large group of applications concerns web log mining. This is often concerned with predicting what page will be visited next [21, 31, 4] or clustering pages [35]. Other times the goal is one more general of website structure optimisation [6]. As our test case TeleDIA is a web application, the similarity is evident. Both the previous research and this thesis deal with click streams of users, and attempt to model them. A subtle difference is that an analysis is usually more coarse than our approach. We will analyse detail down to single buttons as opposed to only requested of pages. A bigger difference is that they mine towards specific goals, predicting a next page, improving structure efficiency, and we will attempt to provide a model for the whole system.

Another application is mining user interfaces (UI mining), more about this can be found in the paper by El-Ramly et al. [9]. El-Ramly et al. concerned themselves with recreating an old UI. Another use they also explored is extracting use case models [10] from the found patterns. While we will not explicitly mine towards these goals, it is likely that mined patterns will be used to improve the TeleDIA UI and further specify use cases.

Finally our work is related to the field of process mining. For example Chan et al. [5] developed a similarity neighbourhood algorithm taking into account, amongst others, order constraints and frequency. They used the mined data to create new processes. We will employ a few ideas seen in process mining, like the importance of the time factor. Unlike Chan et al, our focus is on modelling, not to create, the processes in the application under analysis.

2.5 Interactive data mining

A different solution was proposed by Hipp et al. [19]. They focus on the idea that the amount of returned results is not actually the problem, but the fact that they can not be filtered or recomputed quickly. To solve this they present an approach with an initial almost unconstrained mining run, and later use the results of that to answer further question. This is important as pattern mining should, in their view, allow to gain more insight into the data without changing to hypothesis testing. An excellent explanation of this problem can be found in an article by de Bie [8]. Next to an explanation de Bie also suggests a framework to formalise subjectiveness. The ideas expressed in the last two papers closely relate to the topic of this research: we do not aim to filter and present a succinct and interesting result set, but rather allow for exploratory mining. Key differences are that both Hipp and de Bie require a high level of understanding from the user, that the user can formulate his own wishes. We will focus on an even more general approach.

While this specifically, to the extend of our knowledge, has not been done before, there have been similar ideas. Goethals et al. [15] at the Advanced Database Research and Modelling (ADReM) in Antwerp have developed an application called MIME (Making Interactive Mining Easy) to interactively mine itemsets, this was extended[1] to support classification trees by Pauwels et al. [28]. The general approach is the same; like Goethals et al. we will provide the user with an interface in which he can manually grow his model. In growing his model the user is supported by various mining algorithms and presented with up to date statistics according to the choices made. The obvious difference is that they do not support sequences, which is the focus of our application. This introduces new challenges like how to deal with gaps, repeating patterns, multiple occurrences etcetera. A less significant difference is that in their application statistics have a more prominent role, whereas we focus on the actual patterns.

[1]At the moment of writing only the original MIME software is available on *http://adrem.ua.ac.be/mime*

3 Datasets

The data we have at our disposal are the standard TeleDIA logs. They contain all kind of requests to the system. The entries that we are interested in are all triggered by requests. Other entries include messages from various system TeleDIA connects to, nightly processing and other messages not trigged by user actions. They have to be filtered out. Requests are generated by various UI elements: buttons, table rows, links, text fields, radio buttons and drop down menus. Along with the element the log contains a timestamp, session ID, available and used server memory, the time it took to process the request and various other technical variables. There are more than 30 variables saved. A sample of the log file can be found in Appendix A.

The patterns we can and will find depend for a part on the preprocessing steps. What we include and exclude but also how we represent the data to our input influences the mining algorithms. In the following subsections the choices made in the preprocessing step are described and motivated.

3.1 Events

The obvious candidate for identifying an event is the URL. A webserver with the HTTP protocol does not usually have a state, and TeleDIA is no different. Every request the user must send all information required to handle that request. This is largely done based on a URL (uniform resource locator), it tells the server what to do. The first problem we encounter when analysing the URLs is that because of the underlying structure different URLs can encode the same action. This problem can be fixed by identifying these URLs. Most of the time it can be done automatically, i.e. if only a prefix differs, sometimes this has to be done manually. A case that happens often in TeleDIA is that, when a user clicks on a table, the row number is included in the URL. While it might be interesting to see if all user only use a specific row, this creates too much variety to successfully identify patterns. As they all encode the same action, open a table row, the end of the url identifying this part will be stripped.

The next case is that a single URL can encode multiple actions. Based on the user input the server might respond differently, and follow a different path in the code utilizing different functionality. The server does this based on added information in the request. We tackle this by talking to the developers and analysing the log file. This resulted in using more of the logged data, *componentClass* and *componentPath*, to find exactly what functionality (component) is called. Note that these are also not unique, but when combined they are.

Finally we have some uninteresting URLs, when the user is typing in a certain text field this can produce a request every letter. This will quickly obfuscate larger patterns, and increase the runtime of the mining process. This was solved by manually identifying these URLs and adding them to a list to filter, only the first request is kept. This problem sometimes occurred for every letter, sometimes twice for a click. Note that it is undesired to filter all duplicate request, as it can indicate an erroneous path or not responding UI element.

3.2 Sequences

While some algorithms can work on a long single sequence, most work on multiple sequences. One way to split into multiple sequences is that we identify each use case as a single sequence. This however hides inter-usecase correlations, and might split up erroneous paths. In practice the mining algorithms are able to work on longer sequences and we do not have to split the sequences in such small parts. We opted to split on an identifier already in the log data, the *session* variable. The session variable is used by the application to identify logged-in users.

The session data is also used to identify different users using the application for the traces. The events occur interleaved in the log file, and we use the session ID to identify distinct sequences. Splitting on session gives us a natural and efficient way to split the log file into transactions.

Note that we assume patterns between users are merely a coincidence. This assumption does not strictly hold for TeleDIA: after an acquisition an evaluation will always occur, these sequences are directly cause and effect. However, this is done by design and embedded in the core of the application.

3.3 Resulting entries

After filtering irrelevant information based on the observations in the previous subsections, mostly leaving out information that is intended for debugging the application, we are left with the variables as listed in Table 1.

Variable	Description	Example
time	Timestamp recorded by the server, can define an absolute ordering.	2015-06-01 06:25:13,151
URL	The complete request URL, not always uniquely defining an action	http://teledia.example.nl/
componentPath	The exact path of a component, not always uniquely defining an action	studieWerklijstPanel:filterForm :filterSelector
componentClass	The exact path of the class that was called, solely not defining an action, but combined with the two above it does.	nl.topicuszorg.teleDIS.web.login .LoginPage
duration	How long it took to process the request in milliseconds	19
memory	The memory available, maximally used and currently used by the server	maxmem=1050M, total=1050M, used=248M

Table 1: Relevant variables

The *duration* and *memory* variables do not describe the path a user takes nor contain information needed to identify a session. However, they can contain valuable information for further analysis. We therefore keep them available, just for presentation. It is possible to use this information as extra dimensional data for the mining process, for example finding patterns that take an excessive amount of memory. However, as initial analysis showed no promise (see Section 8.2), we did not focus on this.

4 Methods

In this section both the underlying algorithms (Section 4.1) and the actual implemented application (Section 4.2) will be discussed. Even though we do discuss values for parameters, they are not definitive. All parameters are adjustable by the user in the final application.

4.1 Algorithms

As defined in the introduction various kind of patterns are required. Rather than finding one algorithm that does all, various well known algorithms can be combined to achieve all goals. The core of the application therefore consists of the various sequence mining algorithms. As they are all largely based on previous work this section is mainly concerned with choices made and deviations from the original implementations.

One important consideration when mining sequences is if we allow gaps. Especially in our case allowing gaps can result in impossible sequences, there is a certain order in which pages must be traversed. When allowing gaps it can also happen that we find too general patterns which are uninteresting, for example: "log in" followed by "log out". Taking these two problems into account advocates for disallowing gaps in patterns. However, we do not want to miss a pattern if a few steps are different every time, a likely situation when filling in a form. We'll have to allow gaps, but to a certain extend.

4.1.1 Frequent patterns

General application usage is what the majority of the user does. For this classical frequent patterns are a good fit, with a cut-off below a minimum support. We choose to use closed patterns to not overwhelm the user with redundant information. For added flexibility Philippe-Fournier [12] algorithm was chosen. The algorithm gives us the ability to define the sizes of gaps in patterns and sizes of patterns themselves, next to minimum support. The algorithm is based on work by Hirate and Yama [20] which is in turned based on the PrefixSpan algorithm [30]. This means that instead of traditional candidate generation the algorithm uses projected databases, the original authors describe it well: *Sequence databases are recursively projected into a set of smaller projected databases based on the current sequential pattern(s), and sequential patterns are grown in each projected databases by exploring only locally frequent fragments.* Our chosen algorithm also uses backscan pruning, as defined and described by Wang et al [39]. The pruning is done by growing a sequence with an item in front and after using the projected database to see if it is still frequent. If it is, with equal support, it can be pruned as being a closed item. In practice this algorithm is very fast.

Frequent patterns are considered relevant if their frequency, or so called support, is above a certain threshold. The support is defined as the number of database transactions, sessions in our case, contain the pattern. This minimum frequency threshold varies for every dataset, and can therefore not always be easily defined. It is a tradeoff between mining speed and size of the result set, but the amount of patterns a user can comprehend is also a factor here. We found that for our datasets a support of 10 respectively 5 percent work well. The difference can be explained by how the application is used, in one case this is a much more uniform usage than the other. A problem that arises is that it is very likely this value can change for future log files, we therefore attempt to automatically mine this value. This is explained in Section 4.1.5.

An advantage of the Philippe-Fournier algorithm compared to the classical sequence mining algorithms is that we can define four parameters for the windows: Minimum- and maximum time between two consecutive events, and minimum- and maximum total time. The first two parameters are the most interesting, as they define if we do or do not allow for gaps. We opted to make the default values *minimum* = 1 and *maximum* = 1, as we found that allowing gaps quickly caused an explosion in the number of unrealistic paths. Disallowing the gaps did result in incomplete paths, but it however successfully captured the parts around the gaps. A user with domain knowledge can easily put these back together if required. The maximum length is set to 100, to have a default behaviour of no maximum length, as we do not wish to miss an unexpected long pattern. A reason to reduce this would be if the running time becomes to long, this can significantly reduce the search space if large gaps are allowed. The minimum window length is not set at one, but rather at 2. This means at least 3 events are included in a pattern. This prevents the enormous amount of insignificant correlated item-pairs to show up, and rather surface the more complete sequences.

A final note about the implementation of the algorithm that it finds a pattern at most once in a sequence. This has rather large implications, as it is not uncommon in the logs for a pattern to occur more then once in a sequence. However, as we defined in our research question, we are looking for general patterns. This means that if a single user repeats a set of actions a lot this is not considered interesting. By only counting a pattern once in a sequence, and thus a user session, we prevent this from influencing the results. Note that if a user logs out and in again this is counted as a new session. Combining these is a different problem altogether. An empirical analysis done on combined sessions showed this does not have any significant impact.

4.1.2 Similar patterns

Similar patterns are introduced to tackle the problem of mining *unintended usage*. Small deviations can point towards inefficient patterns, or possibly wrong and unintended usage. Where we ignored gaps altogether when mining for frequent sequences we will allow them for similar patterns. We match a part of the sequence exactly and allow for gaps or mismatches to find deviations. To define similarity we leverage sequence alignment algorithms from the area of biological data mining. Nucleotide or protein sequences alignment to be specific. Our problem is slightly different, we do not have the problem of length as is common in nucleic acid sequences. With our relatively short sequences we can do an exhaustive match using a dynamic programming approach. We implemented the Smith-Waterman [34] algorithm.

Instead of point mutations in a nucleic sequence our events mutate. While ideally we assign a match score between all possible events, this is infeasible for our case. Both because we have not much to base this on and also because our number of events is a lot larger then possible nucleotides encoding DNA sequences. For DNA sequences a common approach is to base a match score on a substitution matrix, for example the BLOSSUM62 matrix [18], which is defined for every pair. We opted to count a fixed match score. Future work could include a matrix of events based on which events occur in a page together, this would however be very specific to the application.

As with the match score we also define a fixed mismatch and gap score. Fixed in the sense that they are the same for all combinations of events. They will be adjustable in the program interface.

Ideally all unintended usage is mined and presented to the user. However, as it is suspected this usage will not always show up in the patterns, it is insufficient to align the previously mined patterns for comparison. Aligning the traces itself will give an enormous amount of results and is of little use for the user. Instead we leverage the interactive element of our solution and let the user define what sequence to match. This sequence is in turn compared to the whole database. Similar patterns are returned with support, providing the user immediate insight if it is worth investigating. The interactive element should make it easy to quickly test patterns and deviations thereof, adjusting it frequently with the newly computed patterns. This can create a thorough understanding of the usage around a specific pattern which can easily be based on a use case.

As we are looking for similar sequences compared to a specific sequence, and we have no definition of *correct sequence*, we allow gaps in both the created pattern and the sequences in the database. The dynamic programming approach returns the specific subsequence that matched the defined pattern, which is ideal in our case.

While a session in the database can be rather long, the defined sequence is mostly likely short, and thus the runtime is dominated by the database. In practice this is negligible for various patterns.

On a final note we discuss the parameters previously mentioned. The default values are based on reason and practical tests. The reasoning is as follows: A lot of events are specific to a specific path, even a single occurrence of such an event indicates the same use case. We therefore want to deem sequences similar if they only have a few events in common. Furthermore an extra event is less of a problem then a complete different action. Therefore the gap penalty, the same as an extra event in another sequence, has a default of -1. The mismatch penalty is -2. To allow for numerous deviations for every match the match score has been set at 4. Finally the threshold is defined as 8. This reasoning requires at least 3 events to match, with possibly a gap and a mismatch. For more deviations it should also have another exact match. In practice this works well. Having the options easily accessible makes it possible for a user to define this more liberal or strict if required, without needing an expensive mining run.

Note that we only find similar patterns in between two defined events, the algorithm will never end with a gap or mismatch, and therefore can not find similar endings. While this might seem as a disadvantage, this prevents completely unrelated matches with sequences that happen after the one under consideration.

4.1.3 Associated patterns

With the algorithms in the previous two sections we already largely model *what* happens. Next we want to look at *when* they happen. For this we can mine associated rules on our patterns. We first identify frequent patterns, and then we find frequent rules. The set of frequent co-occurring patterns does not suffice as we want to mine only significant correlation with a certain confidence. For this we employ another algorithm from the SPMF library, a closed association rules algorithm. We start with mining frequent patterns, using CHARM [40]. CHARM uses a vertical database format, i.e. keeps track of items[2] with a list of the transaction[3] ids they occur in. It then goes on to compute a prefix-tree utilizing both the items and transaction ids, they define it as the *Itemset-Tidset Search Tree*. The tree is computed in a depth first manner. When mining closed patterns the algorithm can skip several levels of the tree, and experiments show it is several orders of magnitude faster then other closed association rules algorithms [40].

The association rules are then computed using a fast algorithm as defined by Agrawal et al in [2] which is based on the classical association rules algorithm described in earlier work [1]. The difference being that in the latter only a single item can be in the antecedent, and in the former multiple items are allowed. In our application we do not expect multiple items in an antecedent, a correlation between more then two patterns in the database, but if it occurs it must not be missed. An optimisation included in the new version concerns only checking consequents consisting of already found frequent consequents.

Using patterns as items allows us to zoom out, forget about order. It however does not allow us to take similar patterns into account. The application could be extended by allowing for specific patterns to be added before the mining, either manually or automatically.

[2]These items represent previously found frequent patterns

[3]A transaction is synonym to a sequence

For this algorithm we define both support and confidence. We leverage support as usual to drastically reduce the amount of possible combinations. High support of an itemset however does not indicate a high correlation. Consider a pattern that occurs in every sequence, it will have a high support together with another pattern that occurs above the threshold. To tackle this we will use the confidence measure. Most easily explained in an equation:

$$confidence(A \rightarrow B) = support(A \cup B)/support(A)$$

The *confidence* of a set is the support of the complete itemsetdivided by the support of the precedent. This will prevent the earlier described uninteresting rule from showing up. However, the rule the other way around is still possible. This is desired behaviour, if we adjust the previous example to the first pattern occurring in half of the patterns then $A \rightarrow B$ is still not interesting, but if confidence of $B \rightarrow A$ is high the correlation can be interesting.

By default the support is set rather low while the confidence is just below half. In our dataset there are quite some patterns that happen in every session, and they tend to hide the interesting results. It is not possible to, a priori, say what patterns are interesting and which are not. This could be tackled by hiding uninteresting items from the database or blacklisting patterns. We opted for a different solution making use of the interactiveness of our solution. By creating a pattern in the application and selecting it the large list of patterns can quickly be filtered. We therefore prefer to have a few uninteresting patterns that are easily filtered, instead of missing out interesting patterns.

We list both from the precedent and antecedent and do not distinguish in this. While it might be useful to know if one happens before the other, it most often does not. In the cases that it does it is likely the domain expert can deduce it from the actual steps in the pattern.

4.1.4 Summarising Sequences

While frequent and associated patterns already produce a large set of patterns, they both rely heavily on the *support* measure. This is no coincidence, as it is an anti-monotone constraint which is both intuitive and highly effective for pruning. Many other measures can be defined, which can be pushed into a mining algorithm, they however consider each pattern on its own. Completely different approaches are possible, we opted for the *Summarising Event Sequences* by Tati et al. [37]. The algorithm they define, *SQS*, is build upon the idea that sequences who describe the database well together are interesting. Describing the database is taken as compressing the database. The set of patterns that makes the database the smallest without loss of information must be the most interesting set. For the details of the algorithm we refer to the previously mentioned paper, a short description of its functioning will follow here.

For compressing the database the minimum description length (MDL) principle is used, two part MDL to be exact. This means that both the database and compressing patterns are encoded separately. The patterns are encoded in the so called *code table*. This table holds a unique identifier for a pattern, the events in the pattern, a gap identifier and a non-gap identifier. Note that all singletons are also in the table. The database is encoded as pattern stream and a gap stream, from which the original database can be rebuild. Note that we do not allow overlapping occurrences of patterns, just gaps in singletons. This also means that if a pattern occurs it is not by definition encoded by the exact pattern.

For our use we do not need to explicitly compute the encoding size or the pattern streams, the set of patterns suffices. As the size of a pattern identifier directly depends on the chance of it occurring using the *optimal universal code for integers* enables us to compute this an encoding from the usage.

While this already reduces the computation time somewhat, the complete problem of finding the unique best set of patterns is practically infeasible. Testing all possible combinations of all possible patterns would be required. We can not build a solution incrementally because it is not possible to say if a pattern that is good at compressing the database combines well with other patterns. To tackle this problem the authors describe two solutions. First they use a seed set of patterns and select the best combination of those. While this is an efficient approach it disregards a large part of the search space. We could choose the frequent patterns as input for the algorithm, but this would be nothing more than a filter.

The other solution the authors suggest is a heuristic approach, which we opted to implement. This approach does grow the pattern table incrementally. It start outs with only the singletons and continues by estimating how well the combinations of two singletons as a pattern will compress the database. The best combination is chosen and added to the codetable. This combination step is iteratively repeated, every time for all the combinations of the current codetable, with all singeltons. Note that computations can not be reused, as each time the code table is different and the estimated score will differ. Each time a pattern is added there is a step that tests each pattern to be pruned, while it might have been a good pattern to add early on, its usage in the database can be replaced by a different pattern.

When combining patterns we allow gaps, because of the natural limiting of returned patterns and the negative impact of overlap this does not cause the often seen pattern explosion. However, sometimes erroneously gaps are estimated. To prevent this from happing, we test a pattern with the gaps filled, after adding a pattern.

One practical problem encountered is that, especially at the start, patterns can get the same estimated score. We have no way of knowing what pattern will be better later, and being a heuristic this might still not even be the correct choice. So we can pick one at random. A user however likes to have consistent results, if he restarts the program with the same log he expects to see the same patterns. For this we sort on other, irrelevant, pattern data to get an absolute order. In this case we use item ids and pattern size.

The patterns are presented to the user sorted on compression size, best describing patterns first. They are not listed with how many times they are used in the encoding but with their actual occurrence count, which is computed in a final pass. This makes it consistent with the other pattern suggestions in the application, and the compression importance can be deduced from the order.

The patterns provides an new perspective and in practice work quite well. The computation time is however still rather long, especially for big databases. Another perk is that the algorithm does not have parameters to be defined, and is thus suitable for novice users. However, the exact encoding chosen can be seen as an implicit parameter, experimenting with different encodings can possibly improve results and is future work.

4.1.5 Inferring parameters

One of our aims is to make the program usable by users without any experience in pattern mining. A problem with traditional algorithms is that they all require one or more parameters to be set, before the user is presented with a list of results. The wrong parameters can result in not finding any patterns, or, in the worst case, make the program run without terminating. Providing a manual with the explanation of the parameters in layman's terms can guide the user, but we do not consider this very user-friendly. We will therefore attempt to provide the parameters for the user.

The application we are analysing is not being used uniformly by all customers, and the resulting log files therefore also show different characteristics. The differences can be quite significant, an in-depth explanation is provided in Section 5.1. For the similar and associated patterns we fill the parameters based on the reasoning explained in Section 4.1.2 respectively 4.1.3. Unfortunately, for the frequent patterns we can not perform such a reasoning.

To tackle this problem we can leverage the parameterless algorithm for summarising sequences as defined in Section 4.1.4. While the algorithm optimises towards a different goal, it will use more patterns for a large heterogeneous database than a small one. We can analyse this resulting set of patterns to make an educated guess on a suitable minimum support. To make this guess the first approach was to simply take the minimum support from the least occurring pattern of SQS. A quick analysis however showed that SQS tends to return long patterns with low support. Using this low support as minimum support for frequent patterns would result in the pattern explosion we are trying to avoid[4].

We found that we can get a good estimate by taking the support of the least supported pattern belonging to the top 30% of most supported patterns returned by SQS. This can be explained by the distribution found in Section 5.1, as SQS does not encode an event twice it is expected that the 20% of frequent events account for most patterns. The other patterns will contain the less frequent events, which will have a lower support. Because the algorithm is rather fast compared to SQS the support was slightly decreased to that of the top 30%. We will show in Section 6.6 that for our datasets this approach gives us a estimate in the right order of magnitude, which can be fine tuned by the user if required.

4.2 The interface

To be able to use the core algorithms in an efficient and intuitive way a set of other tools will be implemented. In this section these tools will be briefly described and motivated.

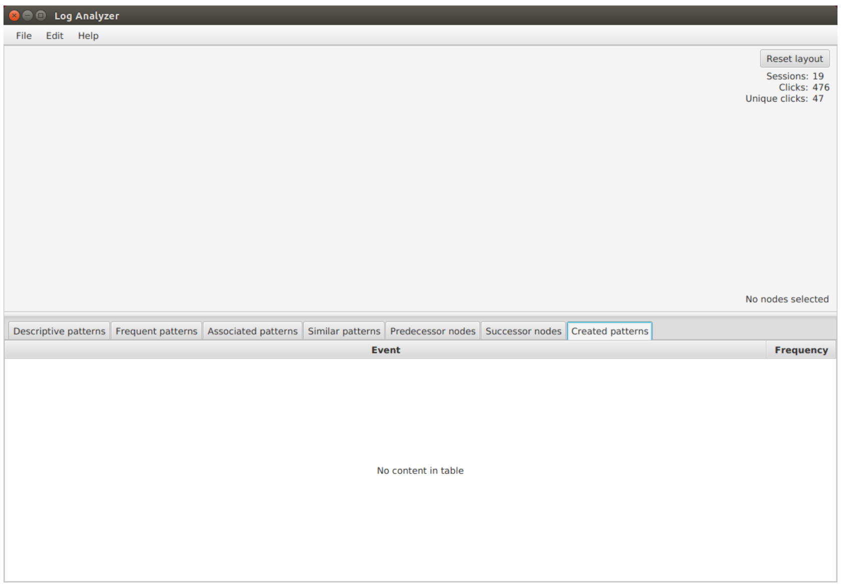
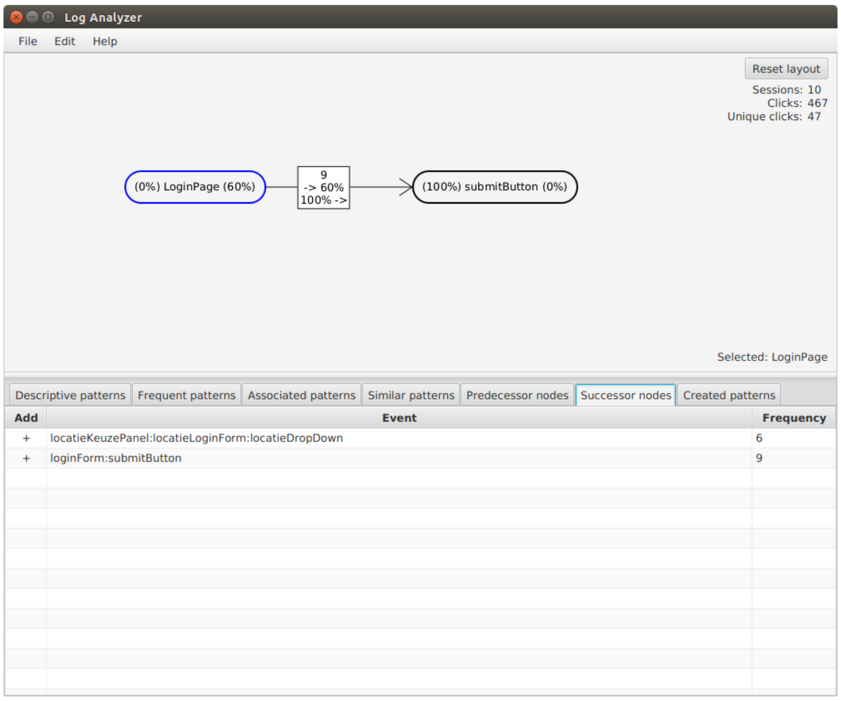


Figure 2: A screenshot of the main interface

First of all the main interface, shown in Figure 2. The main interface was inspired by the

[4]This explosion does not occur for summarising patterns as it does not encode events twice and also takes into account the frequency of which an event occurs.

extended version of MIME for classification trees [28]. The interface consists of two main parts. The graph (originally tree) in the top part and patterns (rules) below are a clear commonality between the two. A difference is that the MIME interface allows more choice in algorithms and measures, where we limited ourselves to a few algorithms. This could still be extended but is beyond our scope.



Add	Event	Frequency
+	locatieKeuzePanel:locatieLoginForm:locatieDropDown	6
+	loginForm:submitButton	9

Figure 3: A selected node with its two successors shown

Next to the pattern suggestions the ability to manual grow the graph in a specific direction has a prominent place in the interface. The search space of the log is enormous, a user can not comprehend it at once, and in contrast to building classification trees we do not wish to build an exhaustive classifier. After selecting a node both predecessors and successors are available quickly, in one click. All the other nodes can be added in two clicks. A selected node and its successors are shown in Figure 3.

The graph is annotated with frequency, both relative and absolute an example label can be seen in Figure 4. This important part of the graph provides much needed context. Sporadic paths can be easily recognized and dealt with accordingly. The frequency label shows, in order: absolute-, relative forward- and relative backward support. Relative forward and backward are based upon the currently created graph. The nodes also have two relative frequencies listed, summing the incoming respectively outgoing edges. This gives a quick overview of the *completeness* of the created patterns.

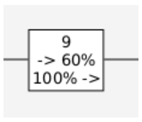
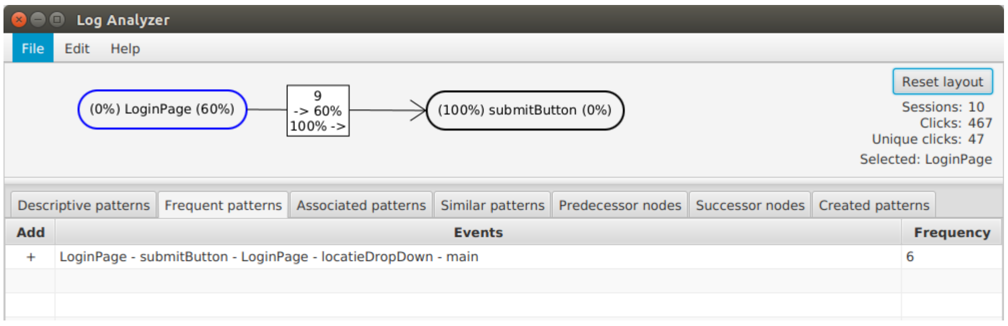


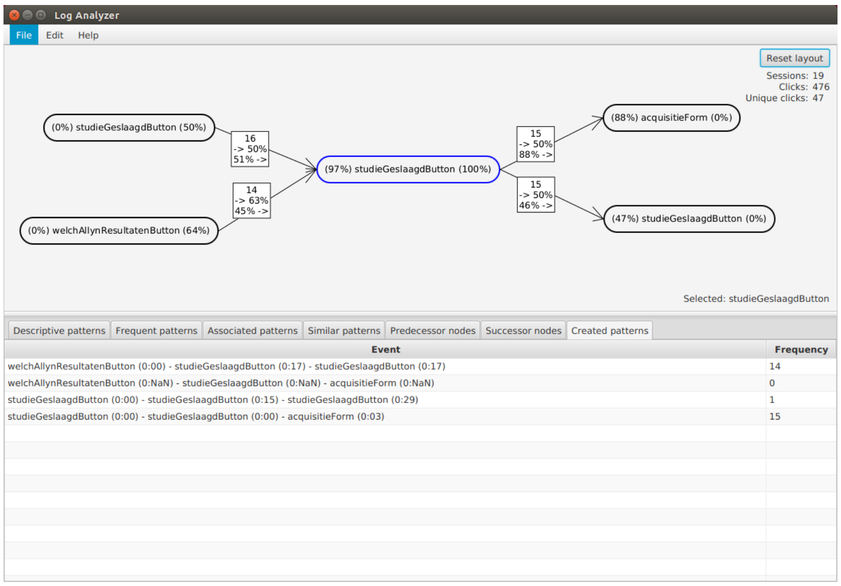
Figure 4: Frequency

Another aid the user has in filtering the suggestions is selecting nodes, this will show only the patterns directly related to the nodes from the various algorithms. This is one of the most powerful features, and allows the user to filter all suggestions to the specific set he is currently interested in. An example is shown in Figure 5.



Add	Events	Frequency
+	LoginPage - submitButton - LoginPage - locatieDropDown - main	6

Figure 5: The graph with suggestions filtered on selection



Event	Frequency
welchAllynResultatenButton (0:00) - studieGeslaagdButton (0:17) - studieGeslaagdButton (0:17)	14
welchAllynResultatenButton (0:NaN) - studieGeslaagdButton (0:NaN) - acquisitieForm (0:NaN)	0
studieGeslaagdButton (0:00) - studieGeslaagdButton (0:15) - studieGeslaagdButton (0:29)	1
studieGeslaagdButton (0:00) - studieGeslaagdButton (0:00) - acquisitieForm (0:03)	15

Figure 6: The graph with created patterns listed

While the graph is an intuitive way to view and build a pattern, once it becomes complex it can become unclear what pattern accounts for how much of the usage. For this there is an overview with a more classical view of the patterns, shown in Figure 6, a simple list. These only include the maximal patterns from the graph, meaning they start at nodes without parents, and end in nodes without children.

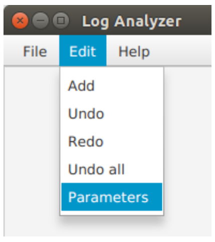
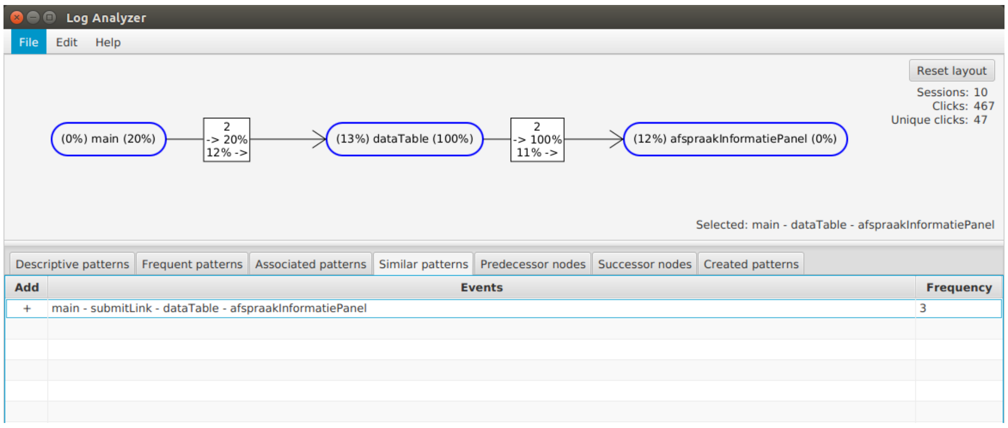


Figure 7: Undo / redo

While we focus on sequential user actions and the patterns therein, there is more to understanding application usage. Both on a technical and an interactive level. For this we can annotate the nodes with extra information. We have chosen to include the average relative times between actions, these are visible in the graph overview pane in Figure 6. This information was included because it was specifically requested, but it also shows the possibilities available. This is easily extended or replace with for example application memory usage.

To make the application directly usable for the key users we also included a parser, both for specific TeleDIA log files and generic item sequences. The common undo and redo user actions are also included, to encourage the user to try out different configurations without fear of losing a promising pattern. This is shown in Figure 7. The other actions, selection, dragging etc. are also implemented as in many common applications.

An interesting problem that surfaced is graph layout. Laying out nodes is not a trivial task, and beyond the scope of this application. While libraries are available they often required an extra layer of complexity or changed the interaction possibilities. Graph consistency was also preferred, and thus moving existing nodes to create graphs more pleasant to the eyes was not an option. We therefore opted for a simple heuristic placement, at an offset from a neighbouring node, depending on the node its size. Usually this works relatively well, and in the other cases the user can easily drag the nodes to a more suitable location.



Add	Events	Frequency
+	main - submitLink - dataTable - afspraakInformatiePanel	3

Figure 8: A pattern selected with a similar pattern suggested

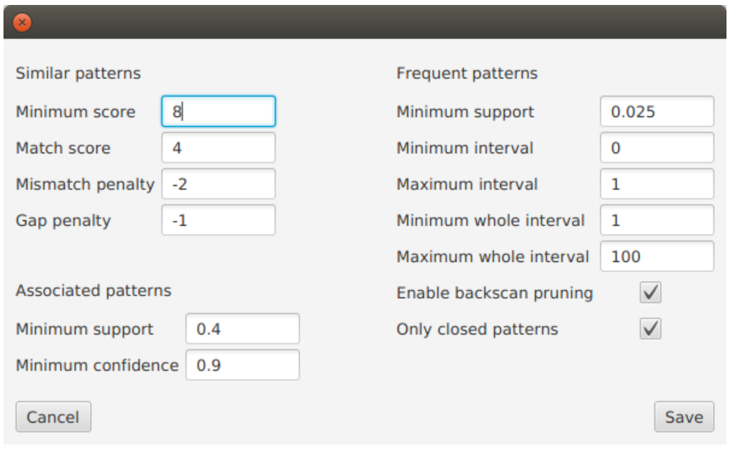
A note to make about the graph is that we do not allow for loops to occur. While it is definitely something that happens in our patterns, repetitive action, this increased program complexity. Even worse, the frequency annotations were not unambiguously defined any more. The patterns are suggested and can still be created, but are unrolled when displayed.

As an advanced feature a dialog to configure the algorithms is available, to allow compensating for different log files or choose a different trade-off between speed and number of patterns. A screenshot can be seen in Figure 9.

For a more thorough explanation of how the application should be used the reader is referred to Appendix G, which contains the user manual.

Lastly we discuss a non-interactive feature. Users do not like to wait, but when loading a log file and the algorithms are run the application can freeze for tens of minutes. And while mining is and will stay an expensive task, we can mitigate some of the waiting time by saving the results. After an expensive mining run the found frequent and summarising patterns are saved to a file. This file with patterns is saved in the same directory and with a similar name as the original log file. A next time the file is loaded these patterns can be loaded almost instantly. We save a file with summarising patterns and one with frequent patterns. Note that if the user changes the parameters for the frequent pattern algorithm the computations are repeated and the frequent pattern file is overwritten.

A final note is that the program was written in the Java programming (version 8) language, making it cross platform. The newest Java technologies were used to create a pleasing interface with easy possibilities of extensions. An added bonus was the availability of a library with an


Similar patterns

Minimum score	8
Match score	4
Mismatch penalty	-2
Gap penalty	-1

Associated patterns

Minimum support	0.4
Minimum confidence	0.9

Frequent patterns

Minimum support	0.025
Minimum interval	0
Maximum interval	1
Minimum whole interval	1
Maximum whole interval	100
Enable backscan pruning	✓
Only closed patterns	✓

Figure 9: The parameters dialog

enormous amount of mining algorithms [11]. We implemented only a few, but it is trivial to extend the application with more algorithms of the library. The application is a prototype, but it is written with clean code and documentation. It should be possible to improve upon and finalize the application into a stable one.

5 Evaluation methods

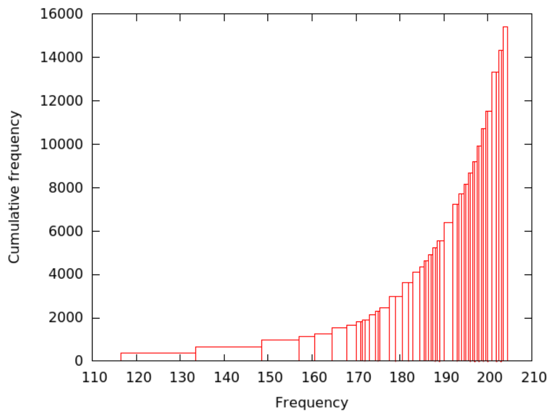
To test the validity of our approach we set up a series of experiments. The goal of this research is to find *interesting* sequences. Capturing the subjective element can not be done directly, and thus this evaluation will not be exhaustive. We *can* test if the various algorithms combined find a set of interesting sequences that we manually identified in a dataset. We define measures to quantify the quality of our approach in Section 5.3. To get a set of interesting sequences our main test will be performed on sequences injected into real world data, as described in Section 5.2. Finally, to subjectively test the performance of the application, we perform a series of tests with key users. Key users are those actively working on the TeleDIA application. This is described in Section 5.5 and will indicate the quality of the prototype overall. We start out with describing our base datasets in Section 5.1.

5.1 Original database

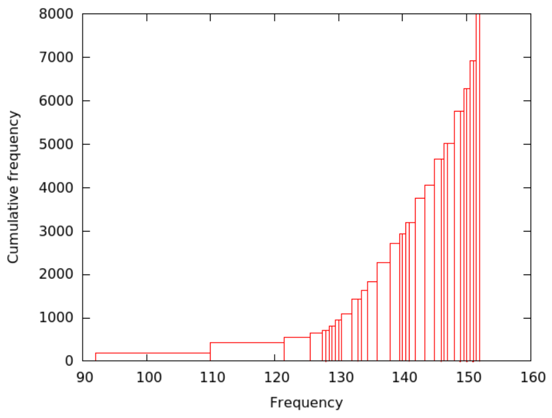
The domain under consideration deals with traces of how humans interact with an application. Because of the complex characteristics of this domain it is infeasible to create a completely representative synthetic database. We therefore resort to using a real world trace file as a base. We will start our analysis by running our algorithms on the original trace file and use the found results as a base line.

We have two datasets at our disposal, from two different TeleDIA instances. As described in the introduction (Section 1.3) TeleDIA is a flexible application and is used differently by different customers. We therefore have two unique datasets, which in turn contain two different sets of interesting sequences.

The first dataset contains 15385 events, in some other literature called items. In those events we count 204 unique events, with about 20% of them accounting for 80% of all event occurrences. See Figure 10a for a cumulative frequency graph of the events. Those 15385 events are spread over 675 sessions. The median session length is 9, with the longest session having 354 events, and the shortest only 1. On average a session has 22 actions.



(a) Dataset A



(b) Dataset B

Figure 10: Cumulative frequency

For the second database we calculate the same characteristics. See Figure 10b for the cumulative frequency graph. A comparison of the characteristics of the two databases can be found in Table 2.

	A	B	A'	B'
Total events	15385	7993	14450	3077
Sessions	675	380	620	159
Unique events	204	152	202	115
Median $\|S\|$	9	8	11	7
Average $\|S\|$	22	21	23	19
Maximum $\|S\|$	354	177	354	169
Minimum $\|S\|$	1	1	1	1
Timespan	8.5 days	16.5 days	7 days	7 days

Table 2: Original database characteristics. Note that $|S|$ is the length of S and can be computed for all $S \in D$

To get consistent results and get a relatively low run time, allowing for numerous tests, we took one week of data from both datasets. See the characteristics in the table in column A' and B'. Note that this includes a weekend, where the system is used less. Both weeks are regular representative weeks, without any holiday or other event that could influence the usage.

5.2 Test paramaters

After performing our initial runs for a baseline we will continue by injecting sequences into the dataset. Statistics from the baseline are shown in Table 3. We make sure all injected sequences are disjunct, meaning they do not overlap. Sequences will consist of a random sample of events already *existing* in the current database to form a consecutive series. We will inject them into existing transactions, keeping the support of existing patterns the same. When injecting patterns with gaps we include random extra events.

	A'	B'
Number of frequent sequences	331	236
Number of summarising sequences	64	32
Length of longest frequent sequence	28	14
Length of longest summarising sequence	24	7
Support of maximum supported frequent sequence	375	90
Support of maximum supported summarising sequence	728	115
Relative automatically computed support in percent	0.015	0.025
Absolute automatically computed support	10	4

Table 3: Baseline run statistics

Injecting sequences will be done while varying four parameters: *number of sequences*, *sequence length*, *number of gaps* and *support in database.* A brief explanation follows.

We start with the *support* parameter. As you can see in the previous section most events happen sporadically. The goal of this research is to model complete usage, and we are still interested in those events. We are however not immediately interested in unique user paths, so we do not wish to find sequences that happen only once. We will vary the *support parameter* from a single occurrence to the maximum support found for a sequence in the original database. As you can see in Table 3 this is 728 for dataset A'. For computational reasons we will not take increments of one in this case.

Secondly we will consider the *sequence* length parameter. We will vary this from the minimum length, two, till the maximum length of a mined sequence in the original database. We choose these values to keep the sequence as realistic as possible. We will consider all possible integer values in this range.

Next we vary the number of gaps. We start out with zero gaps, and will increase this number, one at a time, until we have as many events in the sequence as gaps. Note that these gaps will be placed randomly, and can result in both a singular large gap or many small gaps.

Lastly we consider the number of sequences to inject in our dataset. We start with injecting a single sequence, and keep increasing until we have the original number of summarising sequences in our dataset. As this will be a rather large range of values we will increase with increments of ten. Note that in the originally mined frequent sequences we still have quite some redundancy, our injected sequences will be random and most likely not overlap significantly. We therefore take the number of summarising sequences, which should better represent non-overlapping sequences, instead of the number frequent sequences.

All of these parameters will be varied individually. The parameters that are not varied will be kept at their default value (see Table 4). After analysing the individual results we will select varying combinations for further exploration. We take this approach to reduce the search space.

	Tested parameter	Number of patterns	length	gaps	support
	Number of patterns	1-64	5	0	11
A’	Length	1	2-28	0	11
	Gaps	1	28	0-28	11
	Support	1	5	0	1-375
	Number of patterns	1-32	5	0	5
B’	Length	1	2-14	0	5
	Gaps	1	14	0-14	5
	Support	1	5	0	1-90

Table 4: Initial test parameters

The last parameter we consider does not concern the injection of sequences into the dataset. We will vary the the *used algorithms*. The frequent, associated and summarising sequences can always be found. For the similar patterns we will 'select' the first and the last node and half of the intermediate nodes randomly selected, and see if we find the original sequence again. Varying the used algorithms can provide insight into how the algorithms augment each other.

5.3 Measures

To quantify our results we define a set of performance measures. The measures mainly concern the injected patterns. Note that we do not always wish to find an injected sequence, as short infrequent sequences should be considered noise. We keep this in mind when evaluating our results. We also define the well-known *precision* and *recall* measures for our context to summarise the measures.

Measure 5.3.1. *Sub-sequences:* The number of injected sequences not exactly present in the result set, but with a sub-sequence of the injected sequence present.

The first is computed by comparing the set of injected sequences with the result set and counting the number of injected sequences which *only* have a sub-sequence present. If multiple sub-sequences are present we take the longest. If there are both an exact match and a subsequence it is *not* counted for this measure.

Measure 5.3.2. *Equal sequences:* The number of injected sequences found exactly in the result set.

As with the first measure we look at the longest sequence containing an injected sequence. If a super-sequence is found we do *not* count the pattern as an exact match.

Measure 5.3.3. *Super-sequences:* All injected sequences contained by a larger sequence in the result set.

If an injected sequence is present as a sub-sequence of a sequence found in the result set we count it as a super-sequence.

Measure 5.3.4. *Missed:* All injected sequences that are not present as either sub-sequences, equal sequences or super-sequences in the result set.

Sequences that we are not able to identify but did inject are counted in this measure. We only count it if it is not present in any form.

Lastly we keep track of patterns that do not concern the injected sequences. We keep track of how many of the patterns mined on the original database are still available after injecting. We merely compute this to keep track of information we might lose, a possible indication of how reliable the results are.

When counting we consider both the number of sequences and their injected support, if we find the sequence but with lower support this is reflected in the measures. This also means a single sequence can count towards multiple measures.

The measures can be computed using the set of patterns returned by the algorithm mining frequent patterns, the set returned by the summarising sequences algorithms and using the two sets combined. As on the single set we will try to get the best possible match on the combined set. This will provide us with insight into the individual performance of the algorithms and if they possibly supplement each other.

5.4 Procedure

Automated tests are performed to generate a sufficient set of tests. Each tests will be repeated six times, to minimise the impact of outliers. Outliers can, for example, be the effect of injecting a pattern with characteristics which make it easier or harder to find. After varying the individual parameters initially a first analysis will be done to identify interesting values for the parameters. A second round of testing will be done by varying parameters simultaneously by setting their interesting values.

Finally we are also in the position to directly compare the different mining algorithm on returned pattern quality. We will attempt to identify a preferred algorithm in our domain or conclude the combination is required.

5.5 Qualitative tests

To capture the subjective element we perform two qualitative tests. The first one evaluates the quality of the returned sequences and the second evaluates the interactive solution as a whole.

5.5.1 Sequence quality

To evaluate the quality of the individual sequences we run the program on our datasets at default parameters. The mined sequences will be analysed one by one with a key user. For each sequence we will mark if the sequence is *new*, *correct*, *incomplete*, *irrelevant/incorrect* or *redundant*.

New sequences are sequences the key user did not know about and finds interesting. *Correct* sequences are known information confirmed by the application. *Incomplete* sequences are defined as sequences that are part of an interesting pattern but missing events. *Irrelevant or incorrect* sequences are sequences which do, according to the key user, not contain any information, or even contradictory information. The *redundant* sequences are (perceived as) duplicate patterns.

5.5.2 Program quality

As a last test we will evaluate the designed application as a whole. We will attempt to capture the quality perceived by the key users. For this we employ a questionnaire as defined by Pu et al. [33] for recommender systems. One can see the application as a sequence recommender. The questionnaire also concerns the application as a whole. The wording in the questionnaire is slightly adjusted to better suit our domain. The full questionnaire can be found in Appendix F.

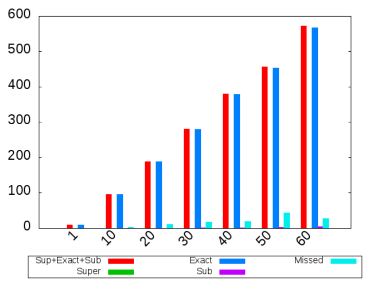
We ask the user to use the application in an exploratory fashion and in a hypothesis testing way. For the latter, to guide the user in the program, we first ask him to define a set of goals, sequences or cases the key user wants to analyse.

This test was done with limited help to prevent biased results. The help that was provided is in the form of the general application documentation, present in Appendix G.

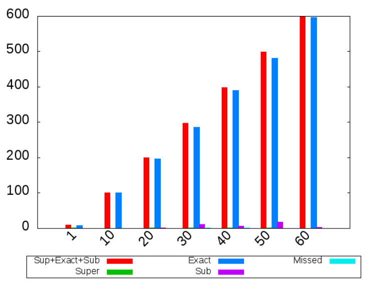
6 Experiments

The experiments are all run as described in Section 5. All of the results are averaged over 6 runs. In the first four sections we vary a single parameter, keeping the others at their defaults. For the detailed values and motivation see Section 5.2. The graphs in this section all have the parameter that is being varied on the X-axis, and for each value at most 5 visible bars. The first bar shows the sum of the number of injected patterns we found, either a super-pattern, exact pattern or a sub-pattern. The next three bars show the individual numbers for these three kind of patterns (Measures 5.3.1 - 5.3.3). The last bar shows the number of completely missed patterns (Measure 5.3.4). As described in Section 5.3 these individual measures are disjunctive and sum to the total number of injected patterns. The Y-axis always shows the number of injected patterns. The counts on this axis are individual occurrences of the sequences, for example: injecting 3 patterns 5 times means we get at most 15 exact matches, and thus the value 15 on the Y-axis. To prevent clutter the labels have been omitted in the graphs. The only exception is in Section 6.6, in this section we consider the computed support. Instead of the number of patterns we have the minimum support threshold on the Y-axis, which is labelled.

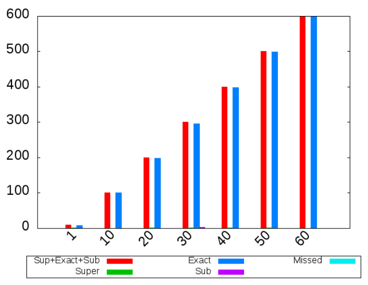
6.1 Varying number of patterns



(a) Frequent patterns

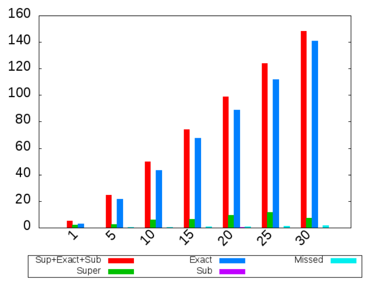


(b) Summarising patterns

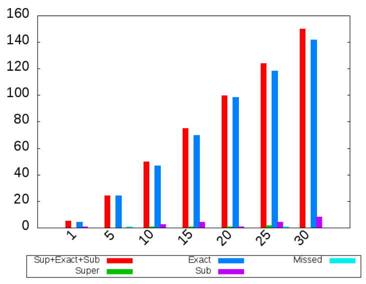


(c) Combined

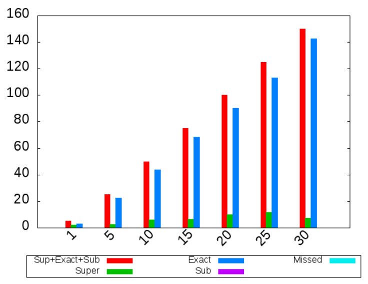
Figure 11: Varying number of injected patterns on dataset A'



(a) Frequent patterns



(b) Summarising patterns



(c) Combined

Figure 12: Varying number of injected patterns on dataset B'

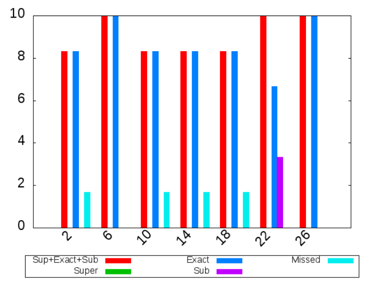
We start by analysing the influence of the number of patterns injected. We can see that both on dataset A' and dataset B' the set of mined *frequent patterns* (sub-figures *a*) contain almost all injected patterns exactly. Only a few patterns are missed, most likely because in such a case two patterns are injected into the same transaction, a situation the frequent patterns are designed to miss. To confirm this suspicion a few randomly chosen cases where this problem surfaced were chosen and analysed. The injected sequences were manually located in the resulting database, and they where indeed found to be in a single transaction in those cases. The frequency of this matched exactly the number of missed sequences. On dataset A' this occurs more often than on dataset B', most likely due to different rounding of the minimum support parameter in the algorithm. This most likely resulted in another slack pattern in dataset B', an injection above minimum support. This allows missing a single injection without completely losing the pattern.

The *summarising patterns* (sub-figures *b*) are also able to identify almost all injections. In some cases it however finds a subsequence, this is most likely because of the heuristic approach. Another possible explanation, involving the frequency of the missed events, shows more in another test and is discussed in Section 6.2.

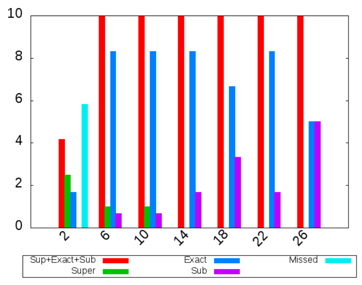
A difference we notice between frequent and summarising patterns is that the former also find super-sequences. This is due to noise that combines with patterns to become a bigger, frequent pattern. More injections increase the odds of this occurring. We do not observe this for summarising sequences. This can be explained by the fact that adding another sequence to the codetable increases the size of this table. If the new encoding doesn't reduce the database with more then than the code table is increased, the new sequence will not be added.

The graphs *combining the two sets* (sub-figure *c*) show no misses and less subsequences, indicating that it are not the same patterns the algorithms have problems with identifying.

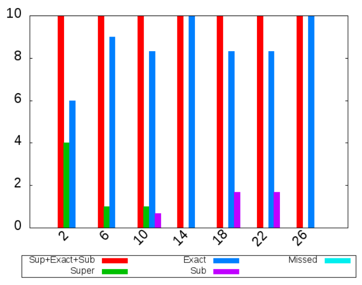
6.2 Varying length



(a) Frequent patterns



(b) Summarising patterns

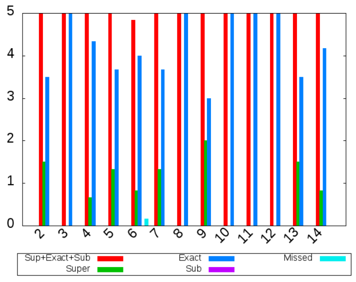


(c) Combined

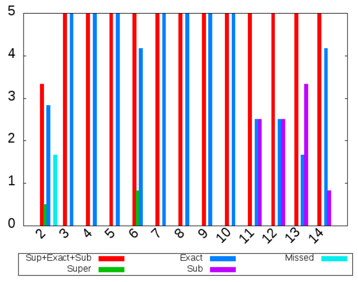
Figure 13: Varying length of injected patterns on dataset A'

The next experiments tests the influence of varying the length of the injected sequence. The frequent pattern algorithm shows similar results for both dataset A' and B' compared to the previous section. In the result-set most patterns are found as exact matches. Some are even found as super sequences on dataset B', and some are missed on dataset A'. The same reasoning applies here as in the previous section.

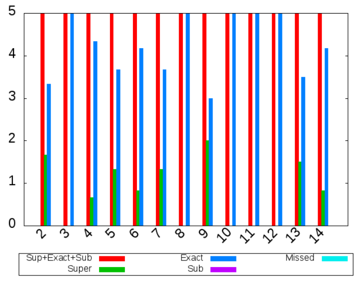
We see more differences between the result-sets of the summarising sequences algorithm for dataset A' and B'. The summarising sequences algorithm has difficulty with very short sequences, of size two. This is because the gain of encoding these is small, especially if support is not high enough. The very long sequences are more often found as subsequences. This is somewhat surprising, as once a pattern is partly found it should be grown by the heuristic. Closer inspection of the mined sequences shows that almost the complete sequence is found, only missing a single item somewhere. This item was to be found frequent itself. A possible explanation is that



(a) Frequent patterns



(b) Summarising patterns



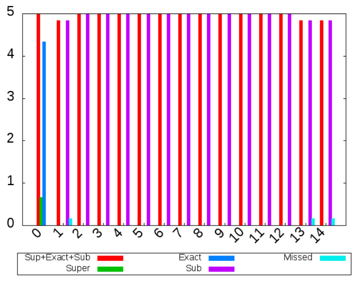
(c) Combined

Figure 14: Varying length of injected patterns on dataset B'

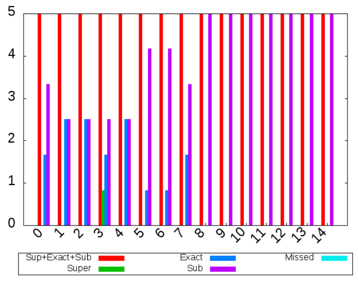
including this item in the pattern, and in the pattern-stream, does not include enough gain compared to including it in the gap-stream. Another possible explanation is the occurrence of a coding bug in an edge case, as the problem does not occur in most of the test-runs. Inspection of the code was however inconclusive, a pattern is always tested without gaps, after adding a pattern based on a gapped estimate to the code table.

The combining graphs show that all injections where covered by at least one of the algorithms. Because on dataset B' the frequent patterns found more exact results the combined set of dataset B' only shows exact and super-sequences, whereas on A' in some cases only sub-sequences are recovered.

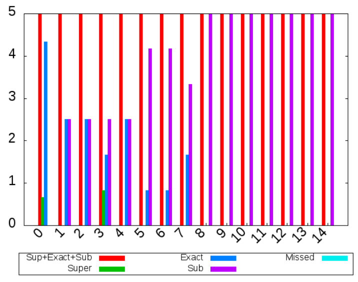
6.3 Varying gaps



(a) Frequent patterns



(b) Summarising patterns



(c) Combined

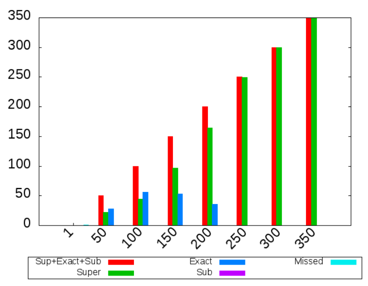
Figure 15: Varying gaps of injected patterns on dataset B'

Varying the amount of gaps in the injected sequences results in the graphs shown in Figure 15. The results on dataset A' are similar and can be found in Appendix B. As expected the frequent patterns, which do not allow for any gaps with the chosen configuration, only find the patterns when no gaps are present. Most of the injected patterns are found in that case, except for some runs when a supersequence or miss is selected. This behaviour is identical to the injection of a single sequence as seen in Section 6.1.

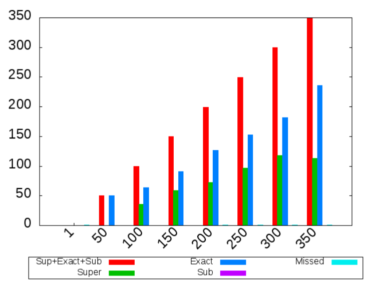
Summarising sequences also show comparable results on sets A' and B'. On both datasets not all patterns are recovered, and often sub-sequences are found. When the number of gaps is low however we see that about half of all injections are found exactly. Keep in mind that this is an average over 6 tests, and closer inspection showed that in some cases all injections where found, where in others not a single exact match was made. In all cases all injected patterns do have some part that is found. Next to most likely having a low support the length parameter plays a role here. To allow for more gaps to be injected while still retaining some of the pattern this parameter has been set at 28. This makes it possible that the heuristic does not even consider the whole pattern, or does not find enough improvement compared to a encoding a subsequence. For follow-up tests with shorter injected patterns see Section 6.7.3.

The combined graphs have the most in common with the summarising patterns, except for when no gaps are included, then the frequent patterns clearly perform better.

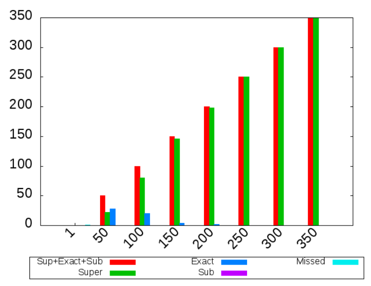
6.4 Varying support



(a) Frequent patterns

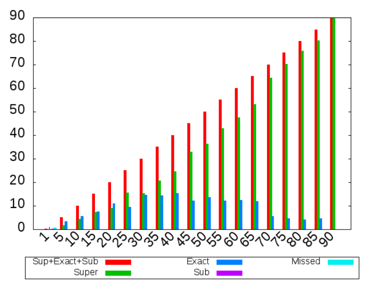


(b) Summarising patterns

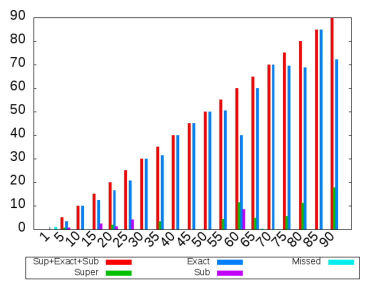


(c) Combined

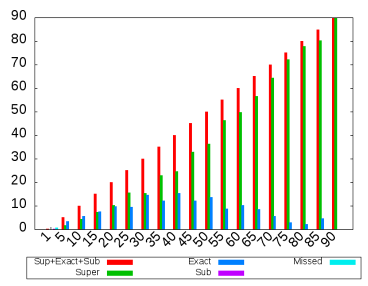
Figure 16: Varying support of injected patterns on dataset A’



(a) Frequent patterns



(b) Summarising patterns



(c) Combined

Figure 17: Varying support of injected patterns on dataset B’

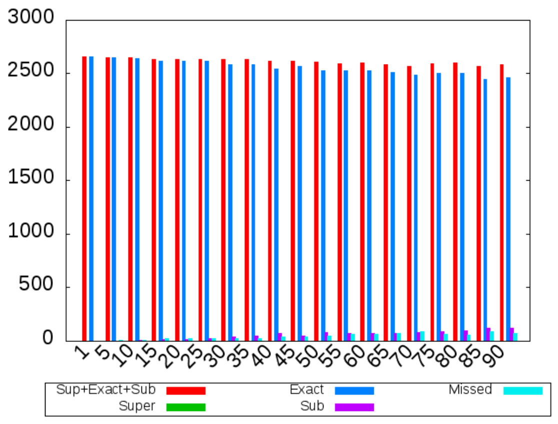
Varying support shows good results for both algorithms on both datasets. In Figure 16 we see the graphs for dataset A’, dataset B’ is shown in Figure 17.

For the frequent patterns we only miss sequences in the case that they are below minimum support, as is expected. Interestingly, when support is increased almost only supersequences are found. The random injection of events makes the events in front and after a sequence also random, but because 80% of the actual events are caused by 20% of possible events it is likely some pre- or post-events are similar in random cases. This chance increases when more of the same patterns are injected, as can be seen in the graph.

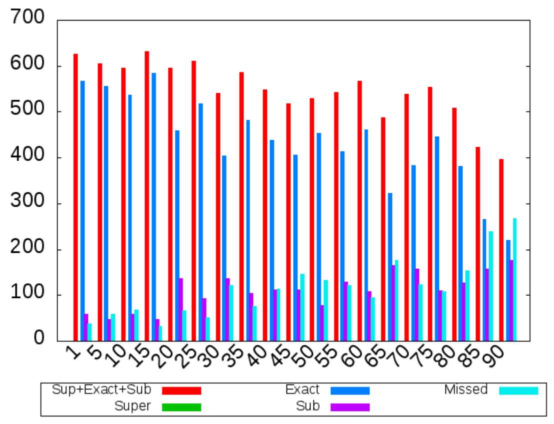
For the summarising sequences we do not see similar behaviour. Interestingly it finds almost only exact patterns. Sometimes a few subsequences can be observed, sometimes a few supersequences, but most of the time the match is exact. Clearly the algorithm is less susceptible to noise in the dataset. On dataset A' a few more supersequences are found, this is because this dataset includes a lot more transactions with *only* the startpage event, when increasing injections the injected pattern with this event appended is also picked up.

The combined graph is similar to the frequent graph almost exactly the same. However, in this case it does not mean that it performed better, this graph is a result of how the measures where defined. One can argue that exact matches are in fact better.

6.5 Original patterns after injection



(a) Frequent patterns



(b) Summarising patterns

Figure 18: Original patterns after injecting varying support on dataset B'

A factor not considered in the previous sections, when testing the various parameters, is if the originally mined patterns are still present. The graphs in Figure 18 are representative for all tests when numerous sequences are injected. We observe that the frequent patterns only miss a few pattern occurrences when more sequences are injected, this is completely the result of injected sequences breaking up original sequences. One thing to note is that we do not start new transactions, in our case sessions, when injecting patterns. The minimum support thus does not change and all originally mined patterns will still be above minimum support, if not broken up.

The summarising patterns algorithm misses more of the original patterns. A first note to make is that originally the set of summarising patterns only returned about a quarter the number of patterns the frequent pattern algorithm did. For this reason broken up patterns are more visible in the graph. We see however more variation as opposed to a linear decrease. As the heuristic used in summarising patterns depends on the frequency of the events and the order in which they occur, an injection can completely change this. This can result in a pattern in the original set being excluded and a (similar) pattern being included. Another factor in the algorithm is the number of patterns used, every added pattern requires more bits to encode, and at some point adding a pattern does not outweigh the negative effect of encoding it. Still more than half of all the original patterns remain.

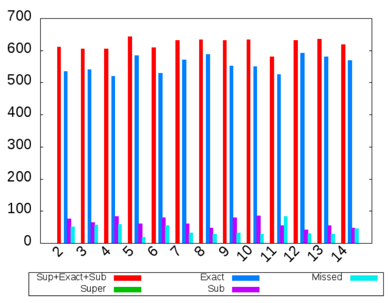
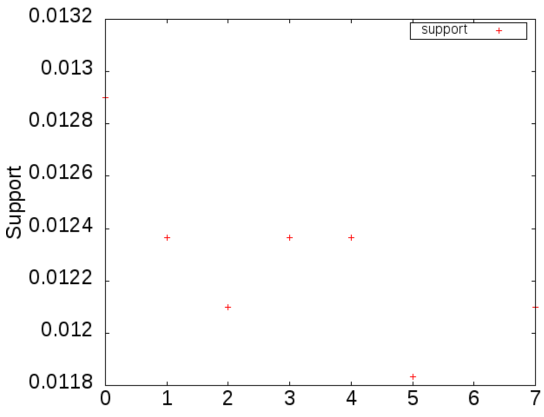


Figure 19: Original summarising patterns when varying length on dataset B'

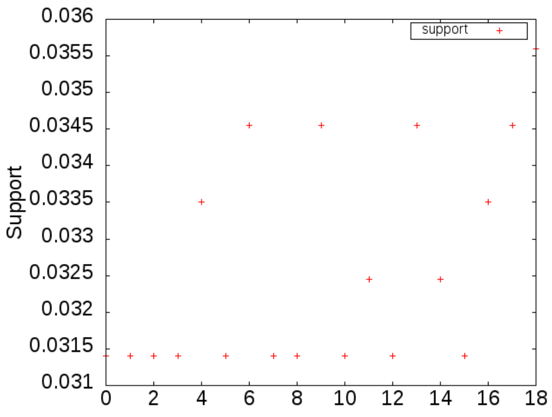
Figure 19 shows the original summarising patterns when only a few patterns are injected. The graph shows the results while varying the length of the injections on dataset B'. In the graph we can see that a lot of the original patterns are retained. While there are still more patterns lost than of the original frequent patterns, the performance is stable. Overall the effect of patterns breaking up is more visible on summarising patterns as the algorithm is more sensitive to the actual occurrences of the injections.

6.6 Computed support

In all our tests the heuristically computed minimum support seemed sufficient. While this is no proof it will always work, our tests show that the variance is low and negligible. For dataset A' the standard deviation was 0.00133 over all tests, and for dataset B' this was 0.00294. These values are negligible, in the datasets they will not trigger a change in the actual number of patterns required. The computed support obtained while performing the test of injections with increasing support (Section 6.1) can be seen in Figure 20. Statistics over all runs are available in Table 5.



(a) Dataset A'



(b) Dataset B'

Figure 20: Heuristicly computed minimum support when varying number of patterns

Dataset	Minimum	Maximum	Average	Standard deviation
A'	0.0097	0.0177	0.0133	0.00133
B'	0.0314	0.0440	0.0325	0.00294

Table 5: Computed support statistics over all tests

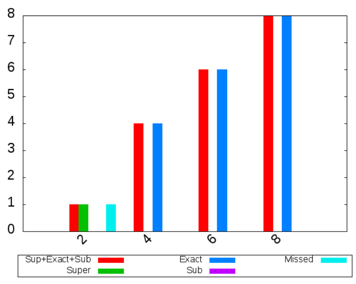
6.7 Varying various

Overall frequent pattern seems to recover almost all injected patterns. Because we can make no statements about the quality of the patterns without inspecting every single one manually, we can not say if any pattern recovered is better or worse. We will simply focus on finding all patterns. The quality of the patterns is further discussed in Section 7.1.

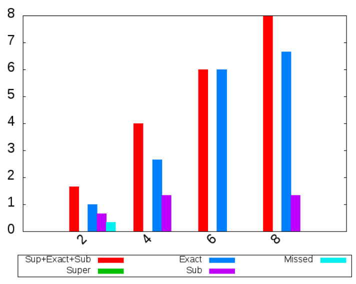
We will continue our tests by selecting areas where the algorithms did not perform well. We know that below the minimum support of the frequent patterns we do not find any frequent patterns, we will test if we can cover those with summarising patterns. The other situation is where gaps are present. We will start by increasing support, and see if relatively more patterns are found. Another possible way to improve, based on the previous tests, is to reduce the length of the patterns, both with default and increased support. As frequent patterns will not be able to find gapped injections with the current configuration, these tests will also focus on summarising sequences.

6.7.1 Below minimum support

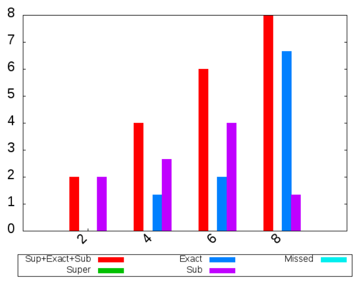
The following experiment will cover injecting sequences below the computed minimum support. We also vary the length as we suspect this can positively influence the ability to find patterns, encoding long sequences compresses more.



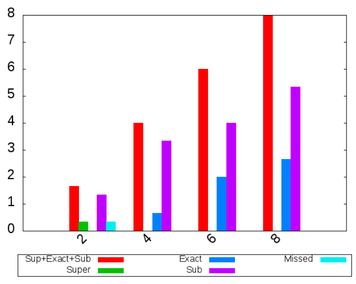
(a) injected length 3



(b) injected length 10



(c) injected length 18



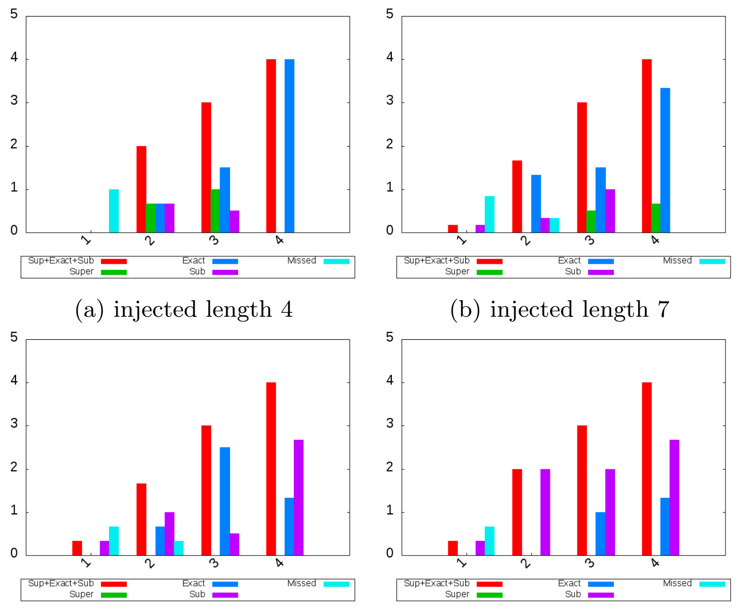
(d) injected length 28

Figure 21: Varying support below minimum support on dataset A', algorithm summarising patterns

In both Figure 21 and Figure 22 we observe that when we have short sequences we find exact and superpatterns. Increasing the sequence length shows more sub-sequences in the result set. As expected the frequent patterns algorithm finds nothing exact below minimum support, see Appendix C.

Summarising patterns can find patterns, or parts thereof, from a support of two. This is fundamental to the algorithm as at least one time a pattern has to be included in the code table, the overhead is not compensated if it can not be used more then once. We omitted the single injection from the graphs for A'. Note that in some cases a pattern will be matched, this is purely a coincidence with the patterns already present in the database. This can be seen in the graphs for B'. Note that a support of 4 is already frequent in B', on this dataset we hope to detect sequences at support 2 and 3.

Increasing the length increases the chance a subpattern naturally occurs in the data, which can bee seen in frequent patterns (Appendix C). In the two tests with the longest sequences, in



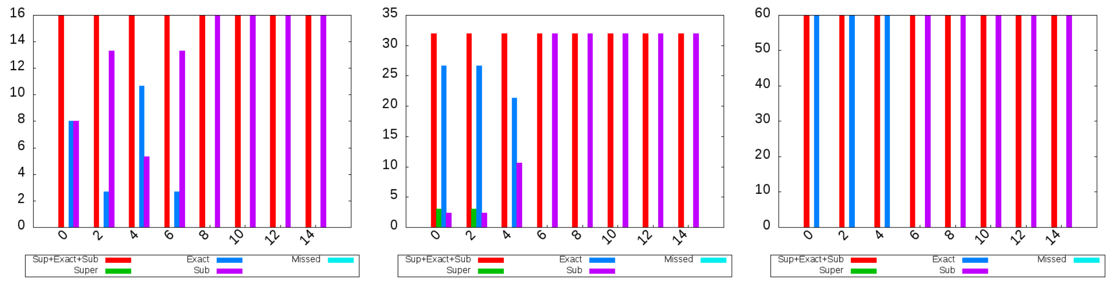
(a) injected length 4 (b) injected length 7

(c) injected length 10 (d) injected length 14

Figure 22: Varying support below minimum support on dataset B', algorithm summarising patterns

A' length 18 and 28 in B' length 10 and 14, we start to find subpatterns. Therefore this isn't necessarily anything new for summarising with long patterns. However, the set of summarising patterns is significantly larger and thus covers new information. Surprisingly, if pattern length is increased, less exact patterns are found. Most likely this happens as a result of a sub-pattern being encoded with more benefit. We find short low support patterns in some cases, and longer patterns usually show a subsequence. On both dataset A' and B' this is the case. Note that the behaviour described in Section 6.2, discussing pattern length, is also part of the reason we find sub-patterns in long injections.

6.7.2 Gaps: higher support



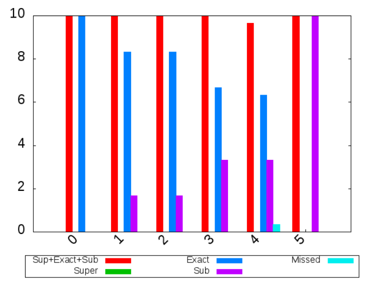
(a) injected support 16 (b) injected support 32 (c) injected support 60

Figure 23: Varying gaps on dataset B', length 14, algorithm summarising patterns

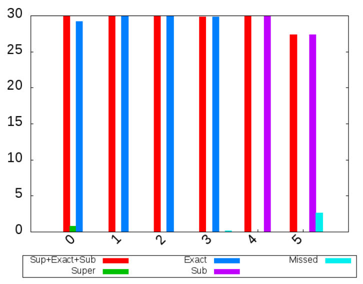
We find a lot more gapped patterns when support is increased. For dataset B' this is shown in Figure 23. Dataset A' has similar results, the graphs for A' are included in Appendix D. This improvement continues when we increase support even more. However, when we increase the amount of gaps we still only find subpatterns in the best case. Intuitively this can be understood, as the long-split up sequences easily encode a few still long enough frequent sequences when split up. With more gaps it is more likely similar splits are made. In this case it seems more beneficial to encode two patterns as opposed to including a long gap-stream.

In both datasets we see that exact patterns aren't found after about 1/3 of the pattern becomes a gap. This limit seems to hold when increasing support. Note that in both cases we consider very long sequences. Close inspection often revealed long matched patterns, with all but a few events of the injection included.

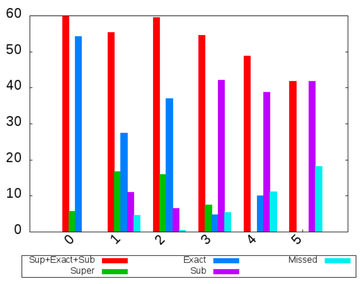
6.7.3 Gaps: shorter sequences



(a) injected support 10

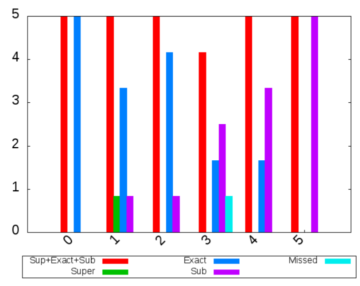


(b) injected support 30

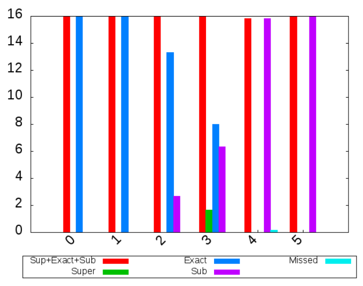


(c) injected support 60

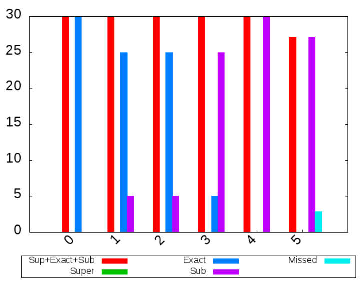
Figure 24: Varying gaps on dataset A', length 5, algorithm summarising patterns



(a) injected support 5



(b) injected support 16



(c) injected support 30

Figure 25: Varying gaps on dataset B', length 5, algorithm summarising patterns

When decreasing sequence length we directly observe an improvement in the number of exact matches found. Even with low support the patterns are exactly matched. Increasing the support in this case seems to decreases the performance of the algorithms. One explanation is that when we get a high support and many randomly chosen gap events we introduce a lot of noise. Take for example the case of dataset A', with a sequence injected 60 times and each sequence having 5 gaps. This already results in 300 events randomly injected events.

When support gets high, the subsequences of a pattern, separated by gaps in some injections, become a good candidate to encode themselves. It is clear that on dataset A' even higher support makes it turn into super and subsequences, accompanied with a few misses.

On dataset B' the effect can also be observed, we already match a pattern with 4 gaps in some cases at the lowest support, of 5. Increasing support makes it more reliably detect patterns with 1 or 2 gaps, but decreases the ability to match more gaps. In a short sequence there are only so many places a gap can be placed, and the subsequence become frequent themselves.

In these tests, with sequence length 5, exact matches on both datasets seem to be limited around 3 to 4 gaps.

7 Qualitative experiments

In this section we will discuss the qualitative test results. Note that the group of interviewed key users is relatively small, namely three, so we do not claim to have conclusive results. We can however already describe some interesting findings.

7.1 Pattern quality

We tested pattern quality as described in section 5.5.2. The patterns where listed using the *Logan application*. The test was only done on the larger of the two databases, A, with default parameters. The results are shown in Table 6 and Table 7 for summarising- respectively frequent patterns.

Kind	Number	Percentage
New	3	4.6%
Correct	24	36.9%
Incomplete	16	24.6%
Redundant	11	16.9%
Irrelevant	11	16.9%
Total	65	

Table 6: Summarising patterns quality

Kind	Number	Percentage
New	7	4.4%
Correct	52	32.2%
Incomplete	7	4.4%
Redundant	20	12.4%
Irrelevant	75	46.6%
Total	161	

Table 7: Frequent pattern quality

A first observation is that for summarising patterns the largest group consists of *correct* patterns, and for frequent patterns this is the second largest group. This is a first indication that the patterns mined are in fact interesting. Upon closer inspection we see that both also returned new patterns, triggering the key user to further inquire. In both cases this is only a small percentage, but this is expected, as upon further inquiry with they key user these were not optimal paths, which were not supposed to even occur in the data.

These two groups, together with the *incomplete* patterns, are the most interesting patterns to the key user. Incomplete belongs to this group as even though the patterns are not complete, they do contain useful information. Ideally these patterns include further steps, the algorithm fails to recognize the larger pattern, it can still be a useful start for further manual pattern expansion.

The redundant and irrelevant patterns are the ones a user does not deem interesting at all, and would rather not see. Especially the frequent patterns include a lot of these, over 50%. Upon further inspection the irrelevant patterns are mostly patterns concerning logging in and general sequences which contain mostly technical events. Note that these technical events might be interesting to a programmer, but are of little meaning to the key user evaluating. The redundant patterns are sub-sequences of other returned sequences, either exact or with small meaningless deviations.

The summarising patterns algorithm has more than 50% of relevant patterns and is clearly more concise. It did however return less than half the amount the frequent patterns returned, and most likely missed a few. This is supported by the experiments in the previous sections.

In practice the key user was able to quickly see which patterns are and which are not relevant, and able to ignore the irrelevant patterns. It would however be more user-friendly if those were not shown at all.

Because the summarising algorithm returned more relevant patterns this was the preferred view of the key-user, even if the frequent pattern returned more correct and new patterns in the absolute sense.

7.2 Program quality

The final tests we perform, as described in Section 5.5.2, covers program quality. The latest version of the prototype, along with the user manual and questionnaire, was given to our three key-users. All three used the application until they felt confident they got a good understanding of the application, and filled out the questionnaire. The answers are discussed in the following paragraphs, in the order the questions appear on the questionnaire.

The first set of answers showed that the patterns are deemed interesting. They matched the persons' interest and were reported to be, on some occasions, novel. The diversity was also appreciated, with the drawback that quite some of the found patterns are similar.

The question regarding the explanation of the patterns was answered quite varied. The variance is explained by the different levels of experience of the users. For those that had more knowledge about the area of pattern mining the descriptive names were good enough. Those who did not have the knowledge found the program lacking in this area. This is an indication that the program does not explain the kinds of patterns well.

When it came to the interface adequacy the users were neutral. The basic functionality is present in a way that can be understood easily, and the interface is clear enough. However, the overall attractiveness of the program is something that can be improved. This can include both visual aspects and locations of interface elements the user wants to interact with.

All of the respondents agreed on the fact that the program was easy to use. Finding a sequence to analyse was easy with the tools provided. Next to the suggested patterns the tools of manual pattern creation are needed for this. This result is an indication that the growing of patterns by picking neighbours, available after selecting a node, is indeed an intuitive approach.

The patterns scored well when it came to perceived usefulness. Note that this concerns the suggested sequences, not the one the user creates. The patterns where both useful as a starting point, especially in the case of novel patterns, but also to quickly grow a bigger sequence from specific, selected, nodes.

The questions concerning the attitude towards the overall program were answered in a positive manner. All respondents liked the program, and found the recommended patterns convincing. This is an interesting result, as some of the respondents answered in an earlier question that they did not understand why a pattern was suggested. The fact that they are still convinced of the patterns is most likely due to some patterns confirming what they suspected. The users are more neutral when it comes to expectations of liking the suggested patterns. Most likely this is an effect of having numerous similar patterns suggested, only one of those is usually interesting.

The behavioural intentions show that the users will definitely use the program again, and even inform colleagues about the program. Most of the respondents will act on the information gathered with the program. The program created the insight the users wanted to get from the log file and the users are overall satisfied.

8 Conclusion

Interviews with the application users showed that they are enthusiastic about the program. The user centred interactive pattern mining approach combining the strengths of various algorithms is a valuable tool for general application developers. Using the tool they gained valuable insight into the actual usage as performed by the real-world users. This is supported by the fact that they have already acted on information obtained with the program: the layout of a page in TeleDIA was adjusted. This was done because the patterns showed that the order of how this page was filled in was not in line with the location of the fields and buttons, resulting in users having to scroll down and up multiple times.

The synthetic experiments show that the implemented algorithms are able to recover all patterns in the database, patterns with a large variety in characteristics. Frequent patterns still seem to be the most complete, however overzealous at times. Summarising patterns returned a set that was not only smaller, the set also described the fabricated database more exactly as it was less influenced by noise. Furthermore the summarising patterns were perceived by the key users as a concise representation of the user actions stored in the analysed log files. The algorithms together cover all of the database well.

Not only were all patterns in the trace files recovered, we also achieved creating an application that was usable by novices in the field of pattern mining. Using both reasoning and an automated heuristic to supply various parameters made the algorithms accessible. The interactive nature of the application provided the user with an intuitive way to filter the enormous sets of patterns and focus on specific areas. This is important as finding all patterns is not an end-goal, with the right parameters one can always construct an algorithm finding these exact patterns. What the user really requires is relevant patterns, a subjective measure that changes over time. The questionnaire showed that the program achieves this goal.

Supplying more than just patterns was key in the application. Simple successors and predecessors and more advanced sequence alignment algorithms both helped the user enormously to explorer the search space.

Another aspect of understanding usage is getting a grasp on what use cases are performed by a single user in a session. By performing another mining run on the mined patterns we are able to uncover correlations between the patterns themselves.

Our main research question: *How can valuable insight into application usage be provided by modelling usage patterns in application traces?*, is answered by the resulting program. In this research we identified all the patterns in the log file, presented them to the user in an interactive application and, according to the users, created valuable insights. The unique way of making pattern mining interactive was received well and is intuitive enough for users without any experience in the field of pattern mining. The only goal that is partly reached is parameterless mining, while we do supply all parameters it is likely that this will not work in all situations. To truly reach parameterless mining more work is required.

Next to answering the main questions the research provides various other insights. First of all concerning the sensitivity of the frequent patterns. With a minimum support too low it is likely spurious patterns are picked up in the form of frequent patterns combined with noise. Simply increasing the minimum support will not suffice, as various other patterns would then also be lost. Summarising patterns did not have this problem. Secondly, the importance of representation. It is an area we did not focus on, but one that became more relevant the more crystallized our ideas became. Users can benefit greatly from visual cues and comprehend information faster when properly displayed. With small adjustments like shortening names and adding redundant percentage displays the key users became a lot more self reliant.

As a final note the users showed a great interest in general, gaining more insight into the actions of TeleDIA users proved to be valuable. Both by word of mouth and after attending a demonstration various other application developers inquired how the mining application can be adjusted for their log files, indicating that it is indeed a widespread problem.

8.1 Future work

While we were able to take important steps the main problem still persists. Filtering what is relevant from a result-set is hard. Further directions of research should focus on including pre-processing in the interactive process, to better capture what the specific user is interested in. For our application simply filtering the more technical and irrelevant actions could be a large improvement. This does however require manual intervention and, due to the knowledge required to decide if an action is irrelevant, is error prone. Another problem that can arise is that which is irrelevant now might become relevant later.

Furthermore, performance became an issue. This was solved for our case by simply running the expensive mining run once and storing the patterns on disk. When the application is closed and launched again it suffices to just load the patterns from file. But in this case changing the parameters becomes problematic, as in that case computations have to be done again. Smarter representations of the database, allowing quick filtering before running the algorithms, can be the dramatic improvement we require. For example; a structure where every represented node contains a pointer to all traces in the database was a considered improvement.

An obvious improvement could be the inclusion of more algorithms and more measures. For example one could add more fuzzy mining algorithms, that can work on a less exact log, if required. The extra algorithms can also consider extra meta-data, for example as multi-dimensional pattern mining, or as added constraints during the process.

Another direction further research can take is to uncover differences between log files. In our case we could load log files from different application instances and run algorithms mining the differences between the files.

Related to this is the real time adding of log-entries. This introduces all kinds of new problems, most importantly: *how long is a pattern relevant?*. In our current approach we implicitly assume the user loading the log file selects a relevant timespan.

Finally we suspect a lot can be gained by focussing more on the representation of the results. Not only the way the data is displayed, but also how the user can interact with it can be expanded to make it easier for a user to extract the required information. This is supported by the results of our questionnaire. Various other visual cues should be tested, both for displaying and interacting with the data.

8.2 Alternative methods

The chosen method of solving the problem is not the only possible approach. In this section alternative methods will be discussed, with their advantages and disadvantages, and why they were not selected. We start with discussing process mining in Section 8.2.1. In Section 8.2.2 we describe a classic approach which was initially attempted, and while the approach was not sufficient it inspired the final solution. In Section 8.2.3 multidimensional mining is discussed, which turned out to be a future direction.

8.2.1 Process mining

As mentioned in the related work Section 2.4 the research in this thesis has similarities with process mining. Process mining is, as the name suggested, data mining applied to processes. Usually this is done on bigger, real world, processes. For example, a whole chain from product development to distribution can be modelled, exposing at various levels the underlying processes. This model can then be used to identify bottlenecks and extract other kinds of valuable information. Process mining can range from information on an organisational level to, at the other end of the spectrum, a single users' workflow. Most of the process mining techniques are able to handle logs of human action, which is often more varied and ambiguous.

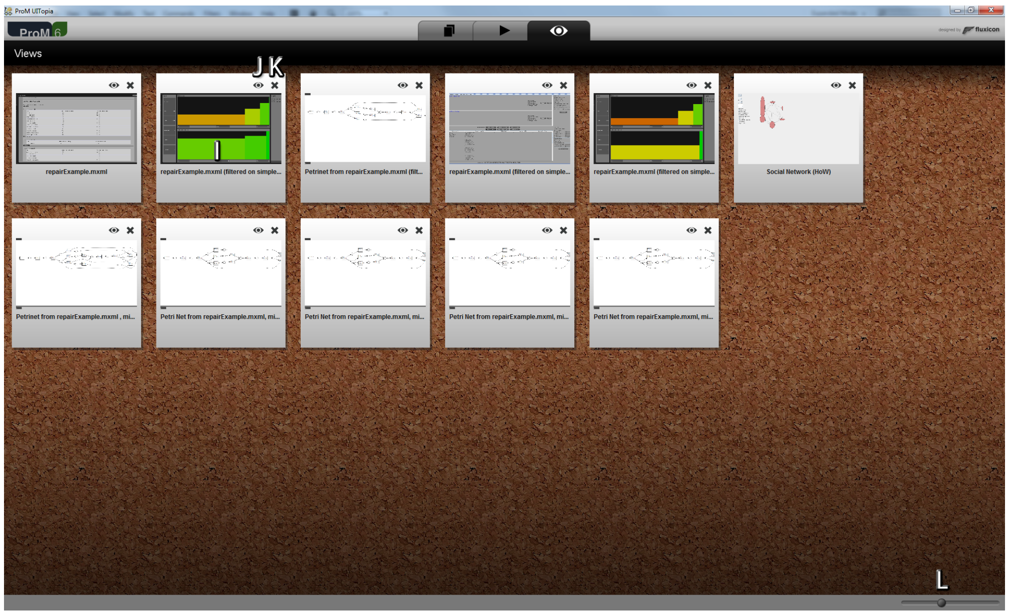

ProM UITopia
ProM 6
Views
repairExample.mxml
repairExample.mxml (filtered on simple...
Petrinet from repairExample.mxml (filt...
repairExample.mxml (filtered on simple...
repairExample.mxml (filtered on simple...
Social Network (HoW)
Petrinet from repairExample.mxml , mi...
Petri Net from repairExample.mxml, mi...
Petri Net from repairExample.mxml, mi...
Petri Net from repairExample.mxml, mi...
Petri Net from repairExample.mxml, mi...
designed by fluxicon

Figure 26: A screenshot of ProM tools

A popular and open source process mining tool is the ProM framework[5] shown in Figure 26.[6] This general purpose framework supplies a varied set of process mining tools and algorithms. At first sight this framework seems to support a lot of mining options suitable for our application. The biggest problem with this application is that it requires quite some knowledge from the user. It would be possible to do an analysis of current log files with the application, but the developers would not be able to repeat this in the future. They would require training, and be able to at least identify the relevant options from the enormous amount of options ProM supplies. For example, the software supports showing social relations between various users in the process, but that is something that is not supported by our dataset.

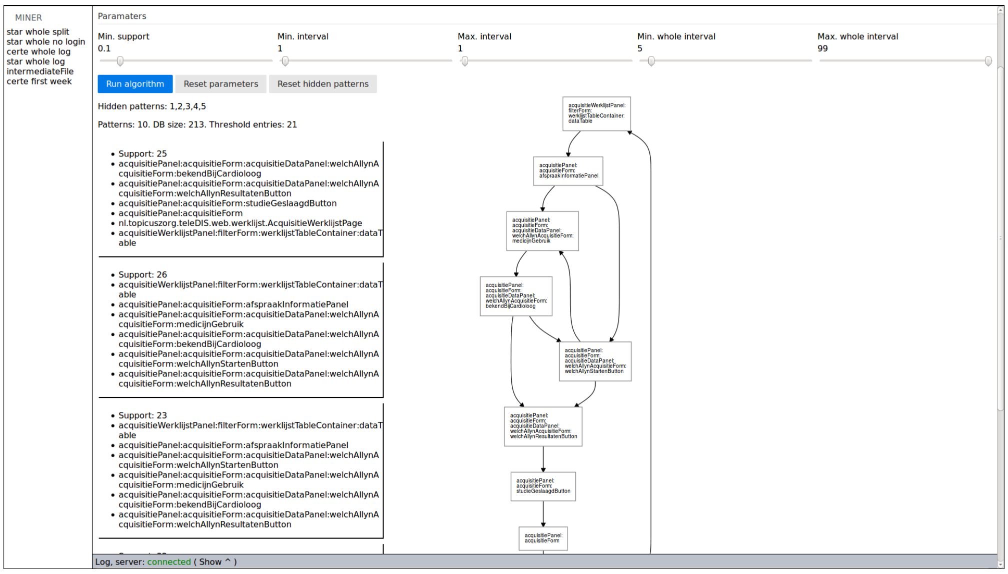
A problem that all process mining tools handle is that there is no formal description of the process available. For us however this is not the case. We have an application that logs factual data. The fuzzy or heuristic methods of calculating the underlying process are not relevant for our case.

Process mining is a powerful method that has proven itself for business. However, the plethora of cases it can handle adds unnecessary complexity for our goal. This is partly due to the fact that process mining originates from business intelligence, and therefore has a different focus. We aim to create a more simple tool for application developers and product owners, which only provides the information they require. Note that it is within the realm of possibilities to transform the implemented algorithms to be usable with ProM, as it they are both written in Java.

[5]http://www.promtools.org/doku.php
[6]Screenshot source: http://www.promtools.org/doku.php?id=gettingstarted:prom6ui

8.2.2 Initial tool

Ideally a user just runs a program and is presented with the information he wants, without any further interaction. This thought triggered the development of the first prototype. The datasets were analysed and various algorithms were tested. With the help of key users the most relevant results were identified. An application was created that ran the algorithms against a log file, computing these relevant results, and listed the found patterns. Listed both as a simple list and a graphical network. An example is shown in Figure 27.



MINER
star whole split
star whole no login
certe whole log
star whole log
intermediateFile
certe first week

Paramaters

Min. support 0.1
Min. interval 1
Max. interval 1
Min. whole interval 5
Max. whole interval 99

Hidden patterns: 1,2,3,4,5

Patterns: 10. DB size: 213. Threshold entries: 21

- Support: 25
- acquisitiePanel:acquisitieForm:acquisitieDataPanel:welchAllynAcquisitieForm:bekendBijCardioloog
- acquisitiePanel:acquisitieForm:acquisitieDataPanel:welchAllynAcquisitieForm:welchAllynResultatenButton
- acquisitiePanel:acquisitieForm:studieGeslaagdButton
- acquisitiePanel:acquisitieForm
- nl.topicuszorg.teleDIS.web.werklijst.AcquisitieWerklijstPage
- acquisitieWerklijstPanel:filterForm:werklijstTableContainer:dataTable

- Support: 26
- acquisitieWerklijstPanel:filterForm:werklijstTableContainer:dataTable
- acquisitiePanel:acquisitieForm:afspraakInformatiePanel
- acquisitiePanel:acquisitieForm:acquisitieDataPanel:welchAllynAcquisitieForm:medicijnGebruik
- acquisitiePanel:acquisitieForm:acquisitieDataPanel:welchAllynAcquisitieForm:bekendBijCardioloog
- acquisitiePanel:acquisitieForm:acquisitieDataPanel:welchAllynAcquisitieForm:welchAllynStartenButton
- acquisitiePanel:acquisitieForm:acquisitieDataPanel:welchAllynAcquisitieForm:welchAllynResultatenButton

- Support: 23
- acquisitieWerklijstPanel:filterForm:werklijstTableContainer:dataTable
- acquisitiePanel:acquisitieForm:afspraakInformatiePanel
- acquisitiePanel:acquisitieForm:acquisitieDataPanel:welchAllynAcquisitieForm:welchAllynStartenButton
- acquisitiePanel:acquisitieForm:acquisitieDataPanel:welchAllynAcquisitieForm:medicijnGebruik
- acquisitiePanel:acquisitieForm:acquisitieDataPanel:welchAllynAcquisitieForm:bekendBijCardioloog
- acquisitiePanel:acquisitieForm:acquisitieDataPanel:welchAllynAcquisitieForm:welchAllynResultatenButton

acquisitieWerklijstPanel: filterForm: werklijstTableContainer: dataTable
acquisitiePanel: acquisitieForm: afspraakInformatiePanel
acquisitiePanel: acquisitieForm: acquisitieDataPanel: welchAllynAcquisitieForm: medicijnGebruik
acquisitiePanel: acquisitieForm: acquisitieDataPanel: welchAllynAcquisitieForm: bekendBijCardioloog
acquisitiePanel: acquisitieForm: acquisitieDataPanel: welchAllynAcquisitieForm: welchAllynStartenButton
acquisitiePanel: acquisitieForm: acquisitieDataPanel: welchAllynAcquisitieForm: welchAllynResultatenButton
acquisitiePanel: acquisitieForm: studieGeslaagdButton
acquisitiePanel: acquisitieForm

Log, server: connected (Show ^)

Figure 27: A screenshot of the first prototype

The prototype provided the key users with initial valuable insight about how the application is being used. However, it also created more questions: why wasn't a certain pattern included? How often do they deviate from a path? How long does the whole process take? The main problem was that the results were static. Iteratively adjusting the parameters solved part of this problem, showing some alternates, but this became too intensive and was a hit and miss process. It also required intimate knowledge of the algorithms and how the parameters influence the outcome, knowledge the users do not have.

The biggest problem turned out to be the initial wish, a static overview of the log. Once the user was aware of a pattern he did not wish to see this any more, and wanted to focus on a different part of the log file. A clustering algorithm was implemented to group relevant patterns together, but this was merely a symptom fix and did not solve the underlying problem.

While the first approach allowed for initial insight, it did not give the user the control he needed. As mentioned in related work Section 2, it is hard to define interesting patterns a-priori. One possible solution to filter the interesting patterns is to include the knowledge of the user. Therefore the initial program was abandoned and the final approach was taken, where the latter is especially superior to the initial approach on the level of interaction.

8.2.3 Multidimensional

Instead of an interactive approach to get more relevant results we could also attempt to use more data. The log files contain more information than we currently use, and possibly information can be added in the future. Including or identifying additional information specific for our cause could make it possible to determine what pattern is interesting and what is not.

One example could be to log what kind of user is logged in, at what location and possible how long he has been using the application. Incorporating this information into the mining process can create natural groups of patterns specific to use cases. This can, for example, surface patterns taken by administrators, which do happen very infrequently in the whole dataset, but are suspected to be inefficient. It could also show, of all users with a *acquisition* role who perform similar patterns, differences between locations. Possibly this would automatically pinpoint users that require extra training.

The biggest drawback of this approach is that first the extra information has to be identified. For this an initial understanding of the domain and application is required. While we speculate a few use cases in the previous paragraph, they might not actually be relevant. Combined with the fact that the data was not available yet let us to our chosen approach. With the knowledge gathered using this research it might be possible to identify the relevant extra parameters, and expand the application to include this multi-dimensional information. This could be an interesting direction for future work.

9 Bibliography

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

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

[3] Francesco Bonchi, Fosca Giannotti, Alessio Mazzanti, and Dino Pedreschi. Exante: Anticipated data reduction in constrained pattern mining. In *Knowledge Discovery in Databases: PKDD 2003*, pages 59–70. Springer, 2003.

[4] Jose Borges and Mark Levene. Data mining of user navigation patterns. In *Web usage analysis and user profiling*, pages 92–112. Springer, 2000.

[5] Nguyen Ngoc Chan, Karn Yongsiriwit, Walid Gaaloul, and Jan Mendling. Mining event logs to assist the development of executable process variants. In *Advanced Information Systems Engineering*, pages 548–563. Springer, 2014.

[6] Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava. Web mining: Information and pattern discovery on the world wide web. In *Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on*, pages 558–567. IEEE, 1997.

[7] Bill Curtis, Herb Krasner, and Neil Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.

[8] Tijl De Bie. Subjective interestingness in exploratory data mining. In *Advances in Intelligent Data Analysis XII*, pages 19–31. Springer, 2013.

[9] Mohammad El-Ramly and Eleni Stroulia. Mining software usage data. In *Proceedings of 1st International Workshop on Mining Software Repositories (MSR'04)*, pages 64–8, 2004.

[10] Mohammad El-Ramly, Eleni Stroulia, and Paul Sorenson. Mining system-user interaction traces for use case models. In *Program Comprehension, 2002. Proceedings. 10th International Workshop on*, pages 21–29. IEEE, 2002.

[11] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu., and V. S. Tseng. SPMF: a Java Open-Source Pattern Mining Library. *Journal of Machine Learning Research (JMLR)*, 15:3389–3393, 2014.

[12] Philippe Fournier-Viger, Roger Nkambou, and Engelbert Mephu Nguifo. A knowledge discovery framework for learning task models from user interactions in intelligent tutoring systems. In *MICAI 2008: Advances in Artificial Intelligence*, pages 765–778. Springer, 2008.

[13] Minos N Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Spirit: Sequential pattern mining with regular expression constraints. In *VLDB*, volume 99, pages 7–10, 1999.

[14] Liqiang Geng and Howard J Hamilton. Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)*, 38(3):9, 2006.

[15] Bart Goethals, Sandy Moens, and Jilles Vreeken. Mime: a framework for interactive visual pattern mining. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 757–760. ACM, 2011.

[16] Jonathan Grudin. Why cscw applications fail: problems in the design and evaluationof organizational interfaces. In *Proceedings of the 1988 ACM conference on Computer-supported cooperative work*, pages 85–93. ACM, 1988.

[17] Jiawei Han, Jianyong Wang, Ying Lu, and Petre Tzvetkov. Mining top-k frequent closed patterns without minimum support. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 211–218. IEEE, 2002.

[18] Steven Henikoff and Jorja G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

[19] Jochen Hipp and Ulrich Güntzer. Is pushing constraints deeply into the mining algorithms really what we want?: an alternative approach for association rule mining. *ACM SIGKDD Explorations Newsletter*, 4(1):50–55, 2002.

[20] Yu Hirate and Hayato Yamana. Generalized sequential pattern mining with item intervals. *Journal of computers*, 1(3):51–60, 2006.

[21] Renáta Iváncsy and István Vajk. Frequent pattern mining in web log data. *Acta Polytechnica Hungarica*, 3(1):77–90, 2006.

[22] Erik Kamsties, Daniel M Berry, Barbara Paech, E Kamsties, DM Berry, and B Paech. Detecting ambiguities in requirements documents using inspections. In *Proceedings of the first workshop on inspection in software engineering (WISE'01)*, pages 68–80. Citeseer, 2001.

[23] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. Towards parameter-free data mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206–215. ACM, 2004.

[24] Arno J Knobbe and Eric KY Ho. Pattern teams. In *Knowledge Discovery in Databases: PKDD 2006*, pages 577–584. Springer, 2006.

[25] Srivatsan Laxman. Discovering frequent episodes: fast algorithms, connections with hmms and generalizations. 2006.

[26] Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data mining and knowledge discovery*, 1(3):259–289, 1997.

[27] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Efficient mining of association rules using closed itemset lattices. *Information systems*, 24(1):25–46, 1999.

[28] Stephen Pauwels, Sandy Moens, and Bart Goethals. Interactive and manual construction of classification trees. *BENELEARN 2014*, page 81.

[29] Jian Pei, Jiawei Han, and Laks VS Lakshmanan. Pushing convertible constraints in frequent itemset mining. *Data Mining and Knowledge Discovery*, 8(3):227–252, 2004.

[30] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *Knowledge and Data Engineering, IEEE Transactions on*, 16(11):1424–1440, 2004.

[31] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, and Hua Zhu. Mining access patterns efficiently from web logs. In *Knowledge Discovery and Data Mining. Current Issues and New Applications*, pages 396–407. Springer, 2000.

[32] Helen Pinto, Jiawei Han, Jian Pei, Ke Wang, Qiming Chen, and Umeshwar Dayal. Multi-dimensional sequential pattern mining. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 81–88. ACM, 2001.

[33] Pearl Pu, Li Chen, and Rong Hu. A user-centric evaluation framework for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 157–164. ACM, 2011.

[34] Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[35] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web usage mining: Discovery and applications of usage patterns from web data. *ACM SIGKDD Explorations Newsletter*, 1(2):12–23, 2000.

[36] Nikolaj Tatti and Boris Cule. Mining closed strict episodes. *Data Mining and Knowledge Discovery*, 25(1):34–66, 2012.

[37] Nikolaj Tatti and Jilles Vreeken. The long and the short of it: summarising event sequences with serial episodes. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 462–470. ACM, 2012.

[38] Jilles Vreeken, Matthijs Van Leeuwen, and Arno Siebes. Krimp: mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

[39] Jianyong Wang and Jiawei Han. Bide: Efficient mining of frequent closed sequences. In *Data Engineering, 2004. Proceedings. 20th International Conference on*, pages 79–90. IEEE, 2004.

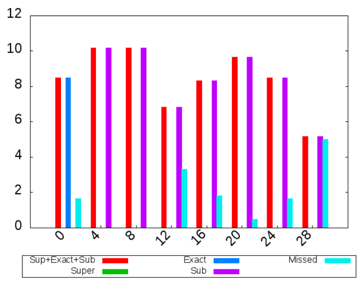
[40] Mohammed Javeed Zaki and Ching-Jiu Hsiao. Charm: An efficient algorithm for closed itemset mining. In *SDM*, volume 2, pages 457–473. SIAM, 2002.

[41] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, Wei Li, et al. New algorithms for fast discovery of association rules. In *KDD*, volume 97, pages 283–286, 1997.

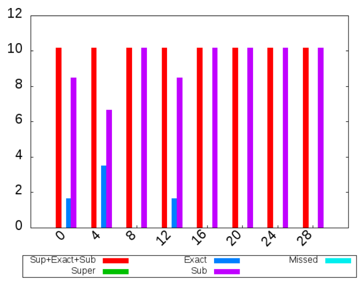
A Example log line

```
04-06-2015 09:42:04 INFO [] - RequestLogger              - startTime
    ="2015-06-04 07:42:04,124",duration=10,url="http://teledia.
    voorbeeld.nl/wicket/page?9-1.IBehaviorListener.0-acquisitiePanel-
    acquisitieForm-studieGeslaagdButton&_=1433403724105",event={
    handler=ListenerInterfaceRequestHandler, data={pageClass=nl.
    topicuszorg.teleDIS.web.acquisitie.acquisitiewerklijst.
    AcquisitiePage, pageId=9,pageParameters={},renderCount=1,
    componentClass=nl.topicuszorg.teleDIS.web.acquisitie.
    acquisitiewerklijst.RegulierAcquisitieForm$2, componentPath=
    acquisitiePanel:acquisitieForm:studieGeslaagdButton, behaviorIndex
    =0,behaviorClass=nl.topicuszorg.teleDIS.web.acquisitie.
    acquisitiewerklijst.RegulierAcquisitieForm$3, interfaceName=
    IBehaviorListener, interfaceMethod=onRequest}},response={handler=
    AjaxRequestHandler, data={pageClass=nl.topicuszorg.teleDIS.web.
    acquisitie.acquisitiewerklijst.AcquisitiePage, pageId=9,
    pageParameters={},renderCount=1}},sessionid="
    jr7htfdh79dk1szx9z7h1lhal",sessionsize=-1,sessionstart="2015-06-04
     06:12:37,779",requests=159,totaltime=12718,activerequests=0,
    maxmem=1030M, total=1030M, used=302M
```

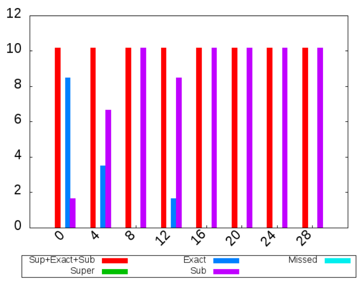
B Varying gaps



(a) Frequent patterns



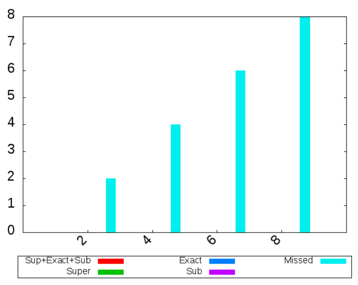
(b) Summarising patterns



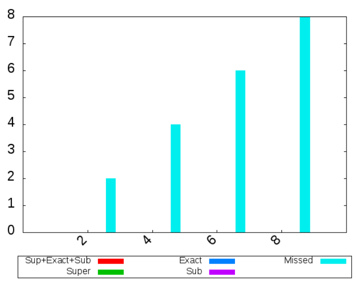
(c) Combined

Figure 28: Varying gaps of injected patterns on dataset A'

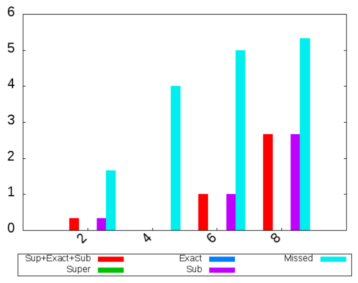
C Below minimum support



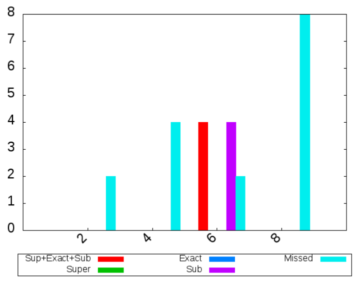
(a) injected length 3



(b) injected length 10

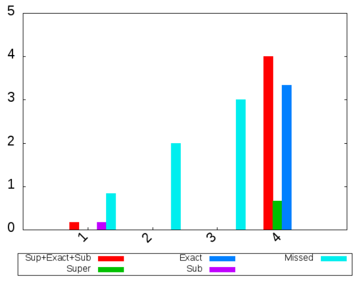


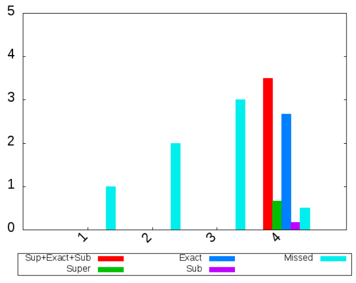
(c) injected length 18

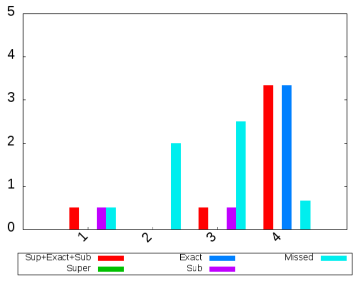


(d) injected length 28

Figure 29: Varying support below minimum support on dataset A', frequent patterns


(a) injected length 3


(b) injected length 10


(c) injected length 18

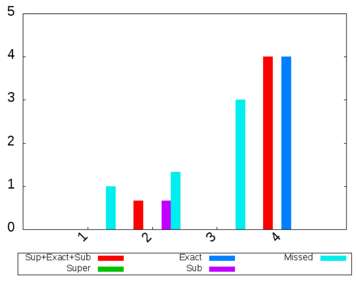
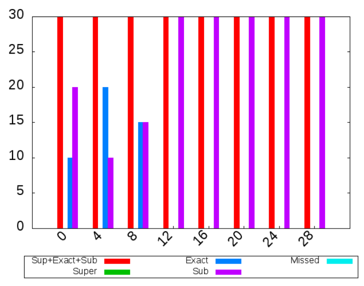
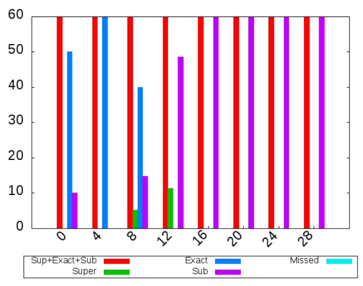

(d) injected length 28

Figure 30: Varying support below minimum support on dataset B', frequent patterns

D Gaps: higher support


(a) injected support 30


(b) injected support 60

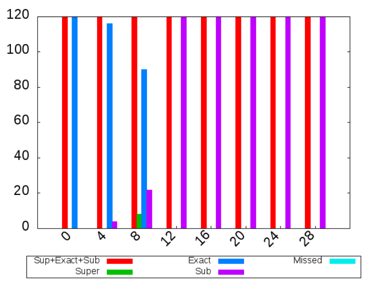
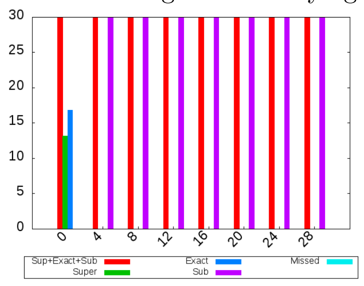
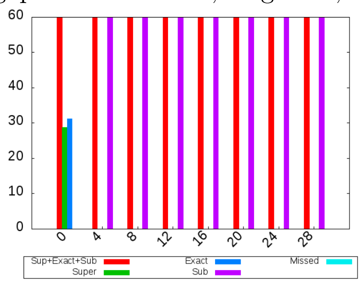

(c) injected support 120

Figure 31: Varying gaps on dataset A', length 28, summarising patterns


(a) injected support 30


(b) injected support 60

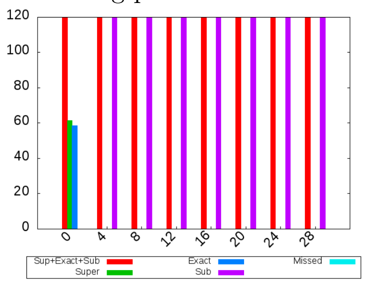
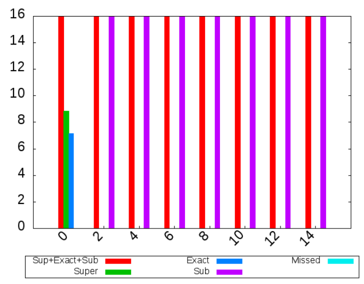
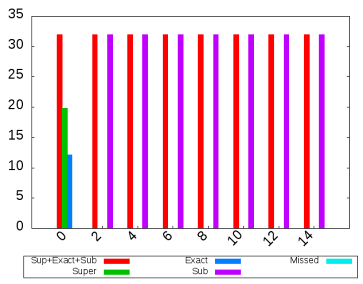

(c) injected support 120

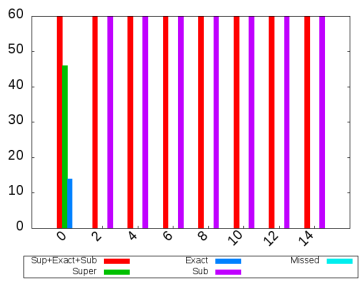
Figure 32: Varying gaps on dataset A', length 28, frequent patterns



(a) injected support 30



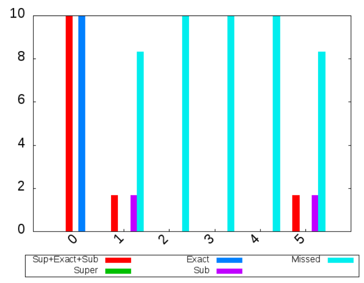
(b) injected support 60



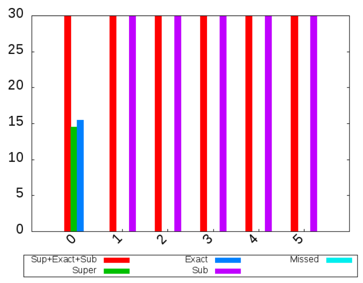
(c) injected support 120

Figure 33: Varying gaps on dataset B', length 28, frequent patterns

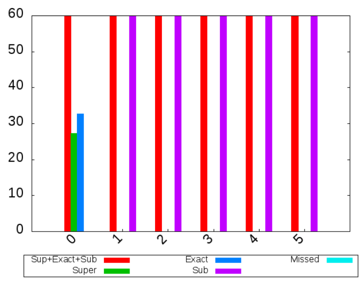
E Gaps: shorter sequences



(a) injected support 10

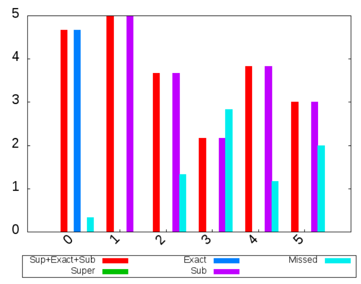


(b) injected support 30

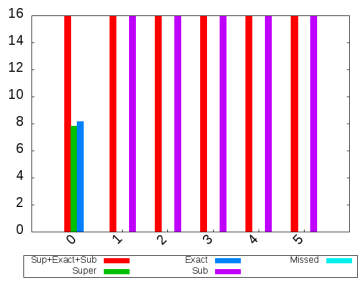


(c) injected support 60

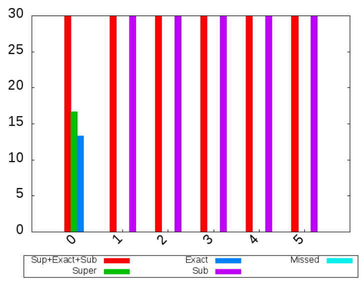
Figure 34: Varying gaps on dataset A', length 5, frequent patterns



(a) injected support 10



(b) injected support 30



(c) injected support 60

Figure 35: Varying gaps on dataset B', length 5, frequent patterns

F Questionnaire

Quality of recommended patterns:

1. *The patterns recommended to me matched my interests*

___ Strongly Agree ___ Agree ___ Neither agree nor disagree ___ Disagree ___ Strongly Disagree

2. *The patterns recommended to me are novel and interesting*

___ Strongly Agree ___ Agree ___ Neither agree nor disagree ___ Disagree ___ Strongly Disagree

3. *The patterns recommended to me are diverse*

___ Strongly Agree ___ Agree ___ Neither agree nor disagree ___ Disagree ___ Strongly Disagree

4. *The patterns recommended to me are similar to each other*

___ Strongly Agree ___ Agree ___ Neither agree nor disagree ___ Disagree ___ Strongly Disagree

Interaction adequacy

5. *The program explains why the patterns are recommended to me*

___ Strongly Agree ___ Agree ___ Neither agree nor disagree ___ Disagree ___ Strongly Disagree

Interface adequacy

6. *The layout of the program interface is attractive and adequate*

___ Strongly Agree ___ Agree ___ Neither agree nor disagree ___ Disagree ___ Strongly Disagree

Perceived ease of use

7. *Finding a sequence to analyse with the help of the program is easy*

___ Strongly Agree ___ Agree ___ Neither agree nor disagree ___ Disagree ___ Strongly Disagree

Perceived usefulness

8. *The recommended patterns effectively helped me find the ideal sequence*

___ Strongly Agree ___ Agree ___ Neither agree nor disagree ___ Disagree ___ Strongly Disagree

9. *I feel supported to find what I like with the help of the program*

___ Strongly Agree ___ Agree ___ Neither agree nor disagree ___ Disagree ___ Strongly Disagree

Attitudes

10. *Overall, I am satisfied with the program*

___ Strongly Agree ___ Agree ___ Neither agree nor disagree ___ Disagree ___ Strongly Disagree

11. *I am convinced of the patterns recommended to me*

___ Strongly Agree ___ Agree ___ Neither agree nor disagree ___ Disagree ___ Strongly Disagree

12. *I am confident I will like the patterns recommended to me*

___ Strongly Agree ___ Agree ___ Neither agree nor disagree ___ Disagree ___ Strongly Disagree

Behavioural intentions

13. *I will use this program again*

___ Strongly Agree ___ Agree ___ Neither agree nor disagree ___ Disagree ___ Strongly Disagree

14. *I will tell my colleagues about this program*

___ Strongly Agree ___ Agree ___ Neither agree nor disagree ___ Disagree ___ Strongly Disagree

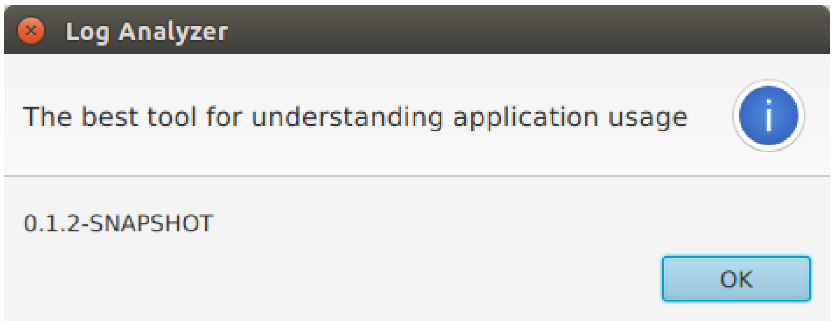
15. *I will act on the sequences recommended, given the opportunity*

___ Strongly Agree ___ Agree ___ Neither agree nor disagree ___ Disagree ___ Strongly Disagree

G User manual

User manual: Log Analyser (Logan)

January 26, 2016



The best tool for understanding application usage

0.1.2-SNAPSHOT

Introduction

This document is intended for the users of the application *Log Analyser (Logan)*. It will describe how to use the application. For the inner workings and further development please see the *Design* document.

The application *Logan* is tailored to provide insight into usage patterns of an application by analysing a log file. While other types of sequential logs may be possible to be loaded and analysed, these will not be explicitly supported.

A *pattern* is a sequence of events. An *event* may be a click of a user, other user input or the (automatic) loading of a page. Both are included, one can argue that loading of a page is usually triggered by user action. However, what page is loaded can differ, for example in the case of incorrect form input.

The pattern will be visualised as a *directed graph.* A directed graph is a set of *nodes*, which in our case correspond to events, connected by *directed edges.* A directed edge indicates a transition from one node to the other, in other words; indicates that an event follows the other. The edges are visualised using arrows.

The graph is annotated by various *measures.* These measures concern absolute and relative frequency of the patterns, nodes and edges as they occur in the log file.

The *algorithms* to automatically mine the data vary from traditional sequence mining algorithms and minimum description length based mining methods to sequence alignment methods. The algorithms extract patterns that can help the user with understanding the program usage. Patterns are mined with certain characteristics, for example with high frequency. The mined patterns can then be used to interactively expand the graph.

In Section 1 the various UI elements will be explained. The various ways of interacting with the application will be described in Section 2. Finally Section 3 will describe the algorithms, what kind of patterns they mine and how their parameters can be configured.

Contents

1 Interface

When the application is started the user is immediately presented with the main interface, which can be seen in Screenshot 1.

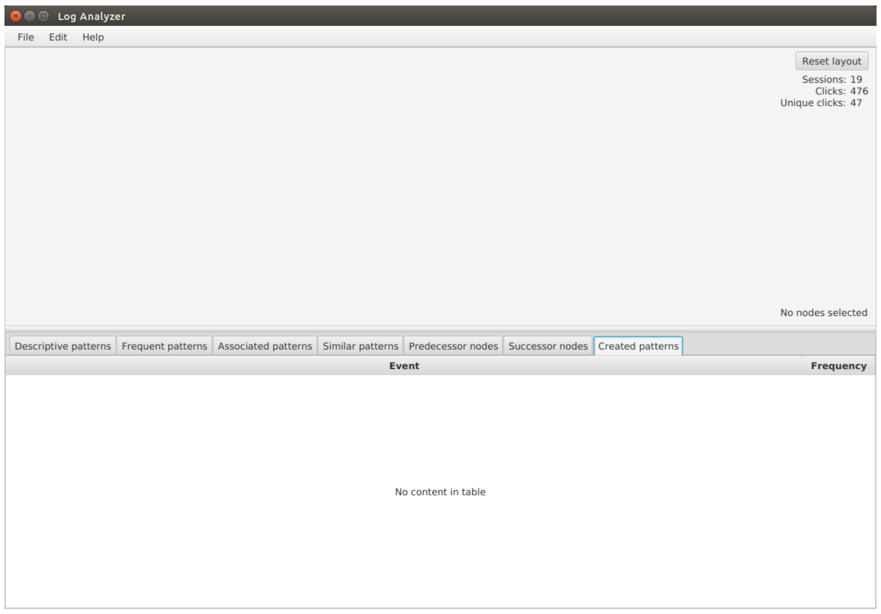


Image 1: The main interface

In the main interface we identify two main parts: The *graph area* and the *pattern tables*. The graph area, the top half of the program, is where interaction is possible and patterns can be manipulated. The pattern tables, in the bottom half, house patterns suggestions to grow the graph. It has seven different tabs, each with a different table.

At the very top we have a *toolbar*. From the toolbar the user can load a log file, add manual items, undo/redo and update the parameters. How this can be done will be explained in Section 2.

1.1 Graph area

When the user loads a file the *Session*, *Clicks* and *Unique clicks* fields in the top right will be populated. The session value shows the number of sessions present in the loaded log file. Usually a session is created every time a user logs into an application (TeleDIA), and thus links the actions taken sequentially by a user. The clicks value indicates the total sum of events in our database, taken by all users combined. The unique clicks indicate how many distinct events occur in our log file. This can be translated to buttons, pages etc.

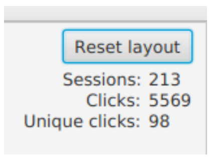
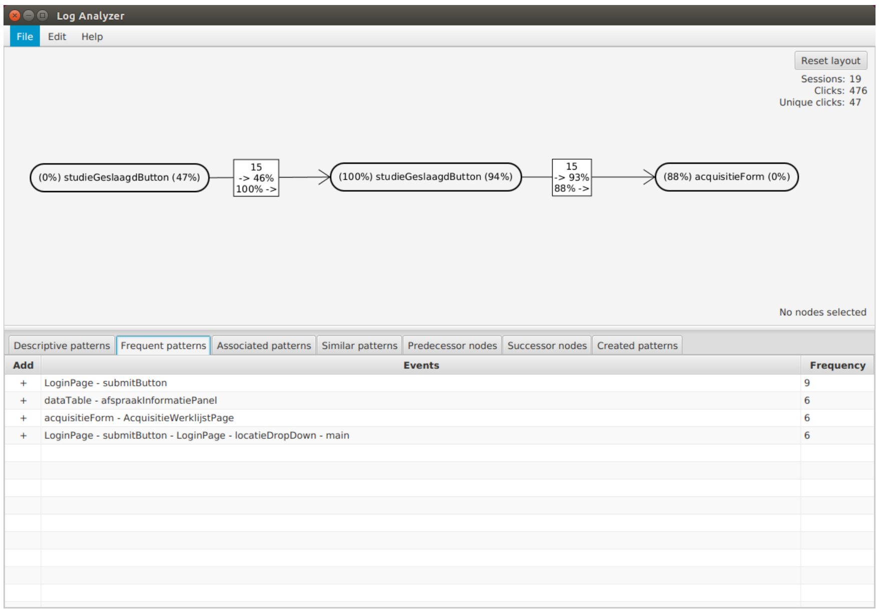


Image 2: Log statistics

1.2 Pattern information

Once the user adds a few nodes and a pattern occurs, the necessary steps will be explained in the next section, more information will be visible. The application will then look similar to Screenshot 3.



Sessions: 19
Clicks: 476
Unique clicks: 47

No nodes selected

Add	Events	Frequency
+	LoginPage - submitButton	9
+	dataTable - afspraakInformatiePanel	6
+	acquisitieForm - AcquisitieWerklijstPage	6
+	LoginPage - submitButton - LoginPage - locatieDropDown - main	6

Image 3: The main interface with some patterns

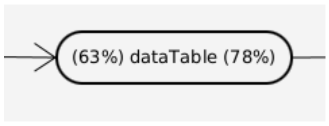


Image 4: Node statistics

Next to a node two values can be seen, both are percentages. See Screenshot 4. The percentage on the left indicates how many of all predecessors, constrained by the current graph, are present in the created graph. Being constrained by the graph means that only predecessors are shown which occur before the current node and all added successors in the graph. The percentage on the right indicates how many of all successors are present in the graph, constrained by the nodes its predecessors. More intuitively: The percentage indicates how complete the current representation is. A low percentage means that common successors/predecessors for the current pattern are left out.

An edge is annotated with three values, see Screenshot 5. All these values are constrained by the graph. The top value is absolute, how many times this edge occurs in the data (in the sequence of the created pattern). The second value indicates the percentage of times this edges is chosen compared to all other possible edges (again constrained by the graph) from the origin its point of view. The second value indicates this for the destination its point of view. Note that the sum of all outgoing edges equals the percentage described in the previous paragraph, on the right of the node. The sum of all incoming edges equals the value on the left.

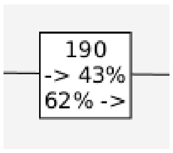
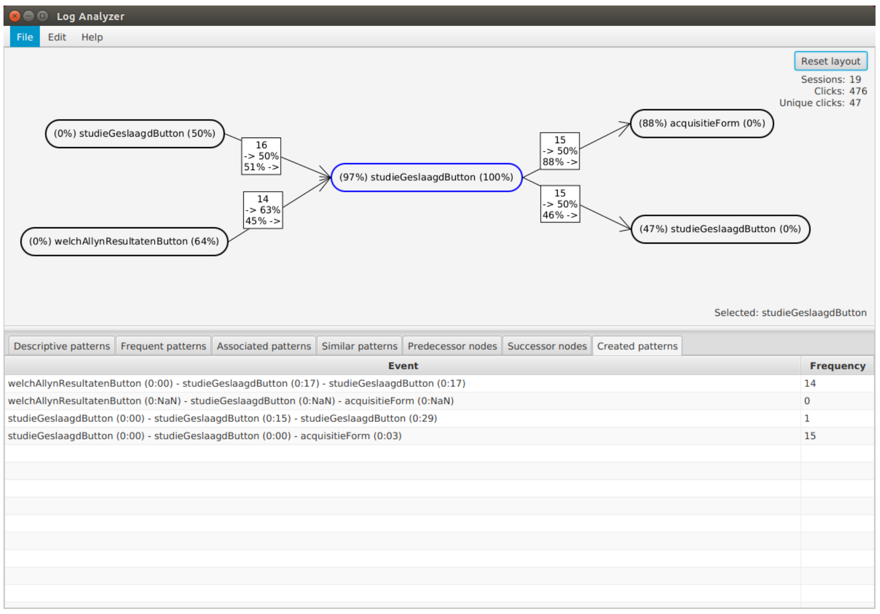


Image 5: Edge statistics

1.3 Pattern tables

In large graphs the patterns accounting for the values can become overwhelming, and for this the *Created patterns* tab was introduced. It is located in the pattern table area of the application. The tab is shown in Screenshot 6.



Event	Frequency
welchAllynResultatenButton (0:00) - studieGeslaagdButton (0:17) - studieGeslaagdButton (0:17)	14
welchAllynResultatenButton (0:NaN) - studieGeslaagdButton (0:NaN) - acquisitieForm (0:NaN)	0
studieGeslaagdButton (0:00) - studieGeslaagdButton (0:15) - studieGeslaagdButton (0:29)	1
studieGeslaagdButton (0:00) - studieGeslaagdButton (0:00) - acquisitieForm (0:03)	15

Image 6: The created patterns tab

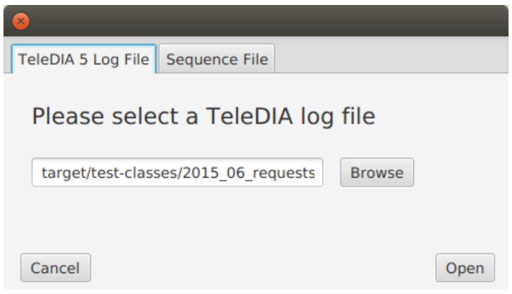
In the created pattern table we see the patterns created. A pattern consists of all consecutive steps that can be taken in the graph while traversing connected edges. These patterns start in the nodes without parents, and end in the nodes without children. Behind every nodename we see a time in parentheses. This is the relative time for the pattern in minutes followed by seconds. The first node always occurs at time 0. For the other nodes the time is the average time that has passed between clicking the previous node and the node itself, computed from all occurrences of the pattern in the database. The value in the last column is the frequency, how many times the exact pattern occurs in the loaded log. If the pattern does not occur its frequency will be 0, and the time will show NaN (Not a Number).

2 Interaction

Logan provides numerous ways to interact with a log file. This section will describe the various ways in a somewhat chronological order. It is expected that the user has some preference and will deviate from the order shown here.

2.1 Loading a log file

To load a log file, chose *File* and select *Load*. This brings up a dialog with two options. The first is to load a TeleDIA log file, and the second a general sequence file. The TeleDIA file can be an unadjusted log file directly from the application. The sequence file is as defined for the *HirateYamana* algorithm of the *SPMF library* [1]. Note that this is a file with just numbers. To link the numbers to a name a second file should be provide with comma separated number-name pairs. In the case of a sequence file this second file is required.



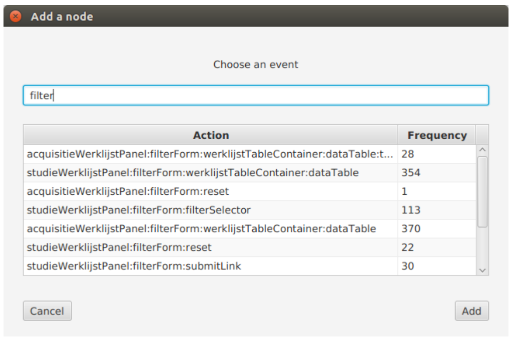
Please select a TeleDIA log file

target/test-classes/2015_06_requests

Image 7: The load log file dialog

After the logs are loaded the application will start the mining of patterns. On larger log files this can take quite some time, and the application will be unresponsive while the mining is performed. Once the mining is done the patterns will be saved as *logname.sqs* and *logname.frequent*, where logname is the name of the chosen log. The next time the log file is loaded the patterns will be taken from these files and will be done in seconds.

2.2 Adding nodes



Choose an event

filter

Action	Frequency
acquisitieWerklijstPanel:filterForm:werklijstTableContainer:dataTable:t...	28
studieWerklijstPanel:filterForm:werklijstTableContainer:dataTable	354
acquisitieWerklijstPanel:filterForm:reset	1
studieWerklijstPanel:filterForm:filterSelector	113
acquisitieWerklijstPanel:filterForm:werklijstTableContainer:dataTable	370
studieWerklijstPanel:filterForm:reset	22
studieWerklijstPanel:filterForm:submitLink	30

Image 8: The add node dialog

To start analysis the user can opt to add a single node. To do this the user selects *Edit* in the toolbar and click *Add*. This will bring up a window with all unique actions in the database. Every event has a listed frequency, how many times it occurs in the database (Figure 8). The list can be filtered by typing in the text field. A node can be added either by selecting it and clicking add, double clicking an action name, or typing its exact name and clicking add.

2.3 Interacting with nodes and edges

Once a node is added to the graph only its short name is visible. To view the full name simply hover the mouse anywhere on the node, the name will appear as a tooltip.

Added nodes can be moved around, this is done by a click on the node, holding the mouse button and dragging. Releasing the mouse drops it at the current position. Instead of moving the node the whole graph can also be moved. This is achieved in a similar fashion, by clicking anywhere but a node and dragging the mouse.

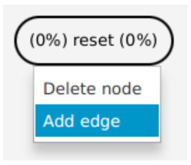


Image 9: Node right-click

The *Reset layout* button on the top right (shown in Screenshot 2) does not reset individual node positions, but moves the whole graph to the top left of the screen.

A right click on a node (Screenshot 9) brings up two options: *Delete node*, which simply deletes the nodes and all edges connected to it, and *Add edge*. To create a new edge click *Add edge* and then select the destination node by left clicking on it. The edge will always start at the node from which the add edge action was initiated.

An edge can be deleted by right clicking on it.

2.4 Selecting nodes

To filter the various tables (and get similar- and associated pattern suggestion) nodes can be selected. This is done by left clicking the node. The node will turn blue when selected and added to the list of nodes in the bottom right. Multiple nodes can be selected, and they will show up *in graph order* on the bottom right. Graph order is the order obtained by following the edges in the direction of the arrows. Selecting nodes for which no order is available, no forward edges from one to the other, is undefined. However, the order shown on the bottom right is the order used throughout the rest of the program.

To select all nodes double-click anywhere but on the nodes and edges. To deselect all nodes a triple click is implemented.

2.5 Adding predecessors and successors

When a single node is selected the two tabs in the graph area called *predecessors* and *successors* list the predecessors and successors of this node. Note that these are only non-empty when a single node is selected. They simply show all nodes that either occur before or after the selected node. The frequency is the absolute number of times this happens in the dataset. These are not filtered based on what is shown in the graph.

2.6 Undoing actions

All actions can be undone and afterwards redone as in most common computer software. Both the *Undo* and *Redo* items are available under the *edit* menu in the *toolbar*. The shortcuts *CTRL+Z* (undo) and *CTRL+Y* (redo) are also implemented. From the toolbar the user can also choose *Edit* and then *Undo all* to undo all actions and end up with a fresh start.

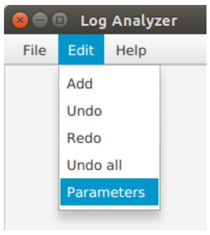


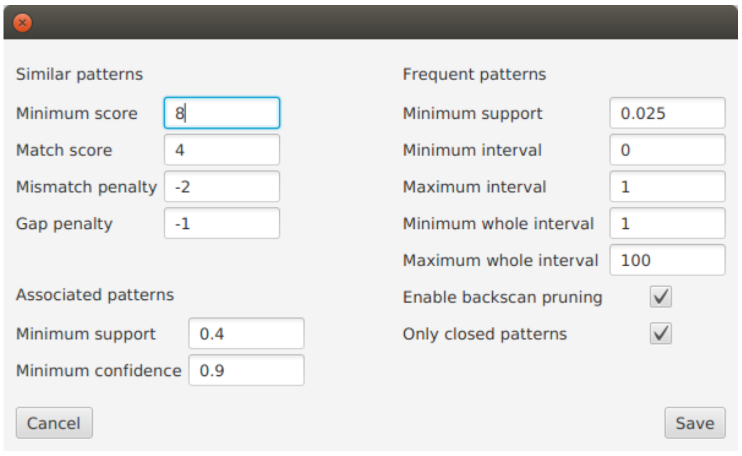
Image 10: Edit menu

2.7 Computed patterns

Next we discuss the most powerful feature, adding computed patterns. This is done from the pattern table area. After selecting a tab with suggestions a pattern can be added by clicking on the plus sign (+) in front of it, or double clicking on the chosen pattern in the *events* column. The patterns can be sorted on frequency or event name, descending and ascending, by clicking on the respective header.

The computation of similar patterns can take a few seconds on a large dataset, so this computation is only performed when the tab is opened. To quickly select a few nodes without having to wait on the intermediate similarity computations you can *select any other tab.*

Finally the algorithm parameters can be edited. There are three groups of parameters, for the *Frequent patterns*, *Similar patterns* and *associated patterns.* The exact meaning of the parameters is explained in the next section. The default values should work well. The format and sign of the parameters should stay the same, i.e. no fractions where whole numbers are required. The program will provide an error if a mistake is made providing new parameters. When the parameters are adjusted the patterns will be recomputed, and the saved pattern file (see Section 2.1) will be updated.


Similar patterns

Minimum score	8
Match score	4
Mismatch penalty	-2
Gap penalty	-1

Associated patterns

Minimum support	0.4
Minimum confidence	0.9

Frequent patterns

Minimum support	0.025
Minimum interval	0
Maximum interval	1
Minimum whole interval	1
Maximum whole interval	100
Enable backscan pruning	✓
Only closed patterns	✓

Image 11: The add node dialog

3 Algorithms

The pattern area with the various tabs houses the pattern suggestions. We discuss them from left to right.

3.1 Descriptive patterns

First we have the so called *descriptive patterns.* The general idea behind the patterns is that they, as a group, describe the whole database well. A way to imagine this is that each pattern found is replaced in the database with a unique number, and in this the size of the database is reduced. The patterns are chosen to create the smallest database possible. Usually this results in patterns that are either long or frequent, preferably a combination of the two. The algorithm has no parameters that can be configured. The frequency column shows how often they exactly occur, without gaps, in the database, multiple occurrences in a single session are counted.

3.2 Frequent patterns

The second tab houses the *frequent patterns.* These patterns are the sequences that happen most often in the database. They are counted at most once per session. If a sequence is frequent above the *minimum support threshold* it is included in this tab. The frequency column shows the number of session this sequence occurs in, possibly with gaps.

The algorithm has six parameters. First the previously mentioned *minimum support.* This is computed using a heuristic based on the parameterless run of the previously described descriptive pattern algorithm. Decreasing this will show more patterns, but can also drastically increase runtime. The *minimum-* and *maximum interval* deal with the time between two consecutive actions. Default they are at 1, meaning no gaps are allowed. Increasing the minimum will force gaps between every action, increasing the maximum makes them optional. The *minimum-* and *maximum whole interval* deal with the complete length of the pattern. Increasing the minimum will hide short patterns, while decreasing the maximum will hide long patterns. Default they are set at 2 respectively 100. The checkbox *only closed patterns* will hide redundant patterns, when checked. Redundant patterns are strict subpatterns of other patterns with equal frequency. If the frequency is higher they will not be counted redundant. The last parameter, *backscan pruning* is an optimisation which should be left checked.

3.3 Associated patterns

The *associated patterns* is empty by default, until nodes are selected. The tab will show patterns that occur in the same sessions as the selected pattern occur. These patterns are any of the patterns found in the previously described frequent pattern algorithm. The two parameters here are the support and confidence. The support in this case is similar to the other algorithms, with the exception that the support is calculated for the combination of patterns, i.e. the number of times the patterns occur together. The confidence is a measure capturing the relation between how many times this combination occurs as opposed to how many times the patterns occur by themselves. For example, a pattern which occurs in every session will likely have a high support with various patterns, but all of them at a low confidence, as most of the time the pattern occurs outside of the combination. Note that such a pattern can still be included in the results, because looking from the the view of the other pattern the confidence might be high enough. Both confidence and support are parameters that can be changed. A confidence of 0 will list all found combinations.

3.4 Similar patterns

The *similar patterns* is also empty at the start. Whenever a few nodes are selected it will test the selected pattern for similar patterns in the database. This is done by exactly matching two events, and possibly more in between. Mismatches and gaps are penalized. The penalties can be configured. Next to the penalties a match score can be supplied, and a minimal match score for a pattern to be deemed similar. The default minimal score parameter (8) requires at least two matches (match score 4), more if gaps (-1) or mismatches (-2) are in between.

The support shown in this table is the number of times the suggested similar pattern occurs in the database. In practice similar patterns seems most effective when selecting the first and last node of the pattern under interest, and a few nodes in between.

4 Notes and issues

4.1 Undocumented features

These are tricks that work in the current application but may break in the future. Usually to work around an issue.

- When changing parameters -0 is seen as valid negative number. Useful when you do not want to include a mismatch or gap penalty for the similar patterns algorithm
- Creating a loop, which would cause a crash, or other erroneous actions (for example adding an already existing edge) are prevented, however no real feedback is provided (only in the log file).
- In the graph view the pattern counts are non-overlapping, while the successors and predecessors allow overlap. So A-A-A in the database for the pattern A-A is count 1 in graphview, count 2 in successors-predecessors for A.

4.2 Bugs

- Immediately after loading a file the algorithms are run (if no pattern files are available). This will cause the program to be unresponsive, and can take a long time. No feedback is provided in the application, but it is shown in the log. (Note: in the case of big files this time grows rapidly, if it takes longer then an hour its unlikely it'll complete in time. In this case split the log).

- Loading another file without closing the application adds double undo/redo listeners (so double undo red actions).
- When a pattern is included twice in the graph the frequency percentages can be counted double (and go beyond 100%). If percentages seem odd reduce the graph complexity.
- Loading a sequence file instead of a log file disables the time statistics in the created pattern view, they will always be 0:00.
- Loading a Logan unsupported *sequence file* (but supported in the SPMF library) can parse correctly but cause a crash. This definitely occurs when a SPMF file with time stamps is loaded.
- Clicking in an empty table row causes an error (does not crash the program)
- Previously placed nodes with coordinates of 0,0 (default) might jump in some cases, because of autoplacement, when children/parents are added.

References

[1] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu., and V. S. Tseng. SPMF: a Java Open-Source Pattern Mining Library. *Journal of Machine Learning Research (JMLR)*, 15:3389–3393, 2014.