Universiteit Leiden

Opleiding Informatica

Finding Business Processes in Computer Usage Data

using Frequent Sequence Mining

Name:	Jeroen van den Heuvel
Date:	25/08/2017
1st supervisor:	Walter Kosters (LIACS)
2nd supervisor:	Wojtek Kowalczyk (LIACS)
3rd supervisor:	Simon Hall (Accenture)

MASTER'S THESIS

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

Abstract

For companies, an analysis of the business processes being performed by employees is very valuable information that can be used to drive the automatization of processes using Robotic Process Automation (RPA). This information is not always easy to obtain. We propose an end-to-end solution for collecting data, analyzing the data and presenting the results. First we collect computer usage data by continuously monitoring users for a few days. In this data set, our goal is to find the business processes the users were executing. To achieve this we look for frequently appearing patterns using Frequent Sequence Mining (FSM) algorithms. We extended existing methods to allow for variation in the matching sequences. The result of such an analysis is a huge list of frequent sequences, some more interesting than others. We use sorting and clustering to highlight the more interesting results, and present the results in a graphical user interface. Our experiments on a synthetic data set and a real-world data set indicate that we can successfully find business processes using these methods.

Contents

1 Introduction

For companies, it is very interesting to know what business processes their employees perform and how much time they spend on each of them. Apart from the value this empirical insight offers, there are many use cases for this data. One is the prioritization of processes[1] for Robotic Process Automation (RPA), based on objective data that is gathered by recording users' actions in the background. That is currently the task of a business analyst, who collects more subjective data through surveys, interviews and time and motion studies. Another application is helping slower users speed up their processes by teaching them a faster alternative. Both free up valuable time and effort, and allow for increased worker productivity.

This thesis presents a case study of Accenture, a digital consultancy firm with a large presence in the RPA market, both as a middleman and as a vendor. The goal of Accenture with this research is twofold: to increase the productivity of in-house production teams, and to offer compelling services to clients.

In our context, a business process is a well-defined task that can consist of multiple subtasks. Examples of business processes are handling an order and shipping a product. Because we only look at computer usage data, we limit ourselves to business processes that are at least partially performed on a computer.

We install a monitoring tool on the computers of end users to gather a broad range of data, including, but not limited to mouse clicks and keystrokes. Patterns that repeat often could be a business process. The analysis shows which processes occur frequently, and how frequently they occur in that data set. We also determine how much time users are spending on each business process.

An example of a recorded sequence is

```
1 switch to outlook, title: template - message (html)
2 mouse click
3 scene change
4 type example@domain.com, tab (x3)
5 mouse click
6 switch to outlook, title: insert file
7 type filename.doc, enter
8 mouse click
```

In the sequence that corresponds to the example above, the user sends an email message from a template, with an attachment. This sequence is quite specific, and if it occurs frequently enough, then it is an interesting business process. In practice, the data above would be a small subsequence of a much larger list of events. Our goal then, is to find re-occurring patterns of events from a data set. Frequent Sequence Mining (FSM) methods are particularly suited to this kind of problem.

The FSM family of data mining techniques is aimed at finding patterns in sequential data. FSM techniques have their origins in frequent itemset mining (FIM). A common schoolbook example of FIM is finding shopping items that are frequently bought together. Nowadays applications of FSM algorithms exist in many research fields. The problem statement in FSM research is always a variation on the following theme: which frequent sequences appear in the dataset, and how often do they appear? In text mining this information can be used to find contiguous words called n-grams, with applications in spam detection [11], trend identification [12], document classification [5] and more. In DNA mining, applications of FSM include protein function prediction [27] and gene expression analysis [1]. For telecommunications, some of the applications are finding group patterns [28], or predicting customer behaviour [4]. Other applications include network intrusion detection [29] and predicting failures of computer systems [25]. More applications can be found in [8].

The main contributions of this thesis are a practical use case of applying FSM to a business problem, including a novel definition of the interestingness of sequences. We propose an end-to-end solution, from

[1] Depending on the context "process" means business process, or computer process.

data collection to data analysis to the presentation of the results. In Section 2 we go over existing research in the topics data collection and analysis. We then discuss requirements and the problem statements in Sections 3 and 4. In Section 5 we describe how we monitor low-level user-interface (UI) interaction. In Section 6 we propose an implementation of the Generalized Sequential Pattern (GSP) algorithm, that can mine monitoring data for noisy business processes. Section 7 details how we filter and present interesting subsequences to users. Our proposed methods are tested and verified in Section 8. Finally, section 9 contains the conclusion and future work.

This work is a master's thesis for the Leiden Institute of Advanced Computer Science (LIACS) from Leiden University and Accenture's AI group. The supervisors are Dr. W.A. Kosters and Dr. W.J. Kowalczyk (both affiliated with LIACS), and S.P. Hall from Accenture.

2 Literature Review

Computer usage data has been analyzed in various studies. In [7], monitoring data of elderly computer users was analyzed to detect indicators of dementia. The researchers recorded various low-level and high-level events: keyboard strokes, mouse clicks and mouse movement, clipboard updates, file change events, window events such as minimizations and focus changes. The researchers used this monitoring tool to log the execution of four pre-defined tasks. Analysis of this data (using an undocumented method) suggested that users afflicted with dementia could be identified basis on pause durations and key press durations.

In [19], database audit logs are mined for intrusion detection. For all users, a profile is constructed that describes their usual behaviour. As new database queries come in, they are compared with the known profile of the user who made the query. If a pattern deviates too much, it is flagged as a possible intrusion. The advantage of such a system is that it can find intrusions without knowing what intrusions look like.

Mining web browsing data is performed on logs of network activity, possibly extended by Intentional Browsing Data (IBD). Monitoring with IBD offers more detailed data by including web-browser actions such as scrolling or saving a page, and even chat or search actions [23].

Tan et al. [22] and Soares et al. [18] applied FSM techniques to web page access data. Tan et al. propose a fast algorithm that can find frequent sequences after only two database passes. While this technique is an order of magnitude faster than traditional Apriori-based methods, it can only find maximal frequent subsequences. In a web server case study, Soares et al. used the prefix-span algorithm to find frequent subsequences in web access logs. More research on web mining can be found in [17], a literature survey on the topic.

The above examples of research work with pre-defined tasks, meaning the start and end points of the processes are known. Our data lacks that structure, and finding the executed tasks is our objective. This means finding the start and end points of reoccurring sequences is part of the problem. Such issues are more common in bioinformatics research.

The application of FSM on DNA data is characterized by (1) small alphabets, (2) lots of noise and (3) sequences that can be millions of items long. In this context, finding common patterns is a big challenge. The number of frequent sequences explodes due to these three properties. To keep the running time of any FSM algorithm manageable, smart tricks are required.

Wang et al. tackle this issue in [26] and propose a two-phased solution. In the first phase, the algorithm searches for base segments: frequent, contiguous patterns of a certain length. The second phase then searches for frequent sequences of such base segments. To deal with noise, any number of unmatched characters is allowed between the matched base segments. The information obtained in phase one greatly speeds up the search in phase two. Additionally, sequences of data that did not appear in any segment are replaced with an ϵ item. These ideas result in a much faster execution time, at the cost of losing a subset of frequent subsequences.

Another approach that applies to dealing with large datasets is parallelization. Metemi et al. implemented a parallelized Aho-Corasick algorithm to address the multiple pattern matching problem [13]. They propose a machine learning approach to estimate the required work of a sub-task, and use that to divide the work between a host machine and a compute device. The result is a more efficient use of the available cores.

Finally, one can take samples of a large data set to estimate frequencies. Syed et al. researched finding maximally discriminative patterns of a certain length, between data that was divided in positive and negative examples [20]. They used Locality Sensitive Hashing (LSH) to quickly classify sequences as probably similar, where similar means up to a maximum Hamming distance. With LSH, a signature derived from the full item is compared, making classification much faster. In addition, Syed et al. are looking for approximate matches, thus this kills two birds with one stone.

To help identify interesting patterns among the results generated by an FSM algorithm, many researchers have developed interestingness measures. There are various survey articles that provide lists of such measures [6, 9, 10, 21]. One example of an interestingness measure is the number of occurrences of a sequence. Many other measures exist, even in the subdomain of measures that are applicable to strings [2]. Which combination of measures to use is a very problem specific issue, and the quality of a solution is inherently subjective.

3 Requirements Discussion

In this research, there are three stakeholders: the first group consists of the end users who are subject to the monitoring and whose data is analyzed. Second is the group of data analysts, whose requirements deal with the feasibility of the analysis. The third group consists of the business stakeholders, who use the outcome of the analysis to make decisions.

3.1 End user

The end user has two concerns. One concern is his or her[2] privacy. Although the end user consents to being monitored, he cares about his privacy. Recorded data might include private usage, or even login credentials, which might be a security risk to the user or to his company. Many companies also have strict data protection policies in place. Therefore he wants control over his data, and he does not want data capture that is redundant to the analysis.

The second concern is the effect of the monitoring on the user's normal workflow. The monitoring should not cause the system to slow down in any way, or change the way the user is used to interact with the computer.

3.2 Data analyst

The data analyst shares the end user's concern for non-interfering monitoring software. But in contrast to the end user, he wants as much data as possible. While superfluous data does encumber the analysis, the data analyst can always use preprocessing and pruning to reduce the data to a more manageable size. The data analyst also wants the data to be an accurate representation of the users' activity. Notably the ordering of events must be preserved.

3.3 Business stakeholder

The business stakeholder is mainly concerned with the interpretation of the analysis. He has three concerns. First is the time the analysis takes. The algorithmic analysis should not take more time than a

[2] We only use the male pronoun "his" for the remainder of the thesis, for readability.

traditional analysis would take. Secondly, suppose that the solution to this problem would result in a large amount of sequences, how does he find interesting sequences? He wants results that can easily be digested. Thirdly, suppose the preprocessing done by the data analyst causes a loss of detail in the data, how does he recognize the business process behind a sequence of events? He would like to see examples of a sequence found in the analysis.

4 Problem Statements

The problem statements in this section have been formulated with the requirements discussion from Section 3 in mind.

(1) Develop a monitoring tool and run it to collect data continuously from users, such that

- the resulting data is rich enough to accurately reflect the workflow of users, and to distinguish different processes,
- it does not interfere with the normal workflow of the users, and
- the user can prevent sensitive data from being shared.

(2) Given a sequence data set D, let Σ be the alphabet that contains all the unique items appearing in the sequences of D. Given a start alphabet $\Sigma' \subseteq \Sigma$, a minimum support threshold $\sigma > 0$, a maximum total gap $\tau \geq 0$, find all subsequences x, such that $x = yz$, with $y \in \Sigma', z \in \Sigma^*$, that are frequent and comply with the total gap constraint.

(3) Given a list of frequent subsequences, present the results of the analysis to non-technical users. This expands to three sub-problems statements:

- sort the list by interestingness,
- group similar sequences, and
- present example recordings of sequences.

4.1 Definitions

To explain the problem statement number 2, we need to define several concepts. Note that some of these definitions might be a slight variation on definitions used in other publications.

Sequence data set. A sequence data set is a set of one or more recorded sequences, $D = \{S_1, S_2, \ldots, S_p\}$. Each sequence is an ordered list of events $S = e_1 e_2 \ldots e_q$, annotated with a user identifier. Each sequence represents all data generated by the associated user, and the sequences are unbound in length.

Subsequence. Sequence A is a subsequence of sequence B, iff A can be obtained from B by deleting zero or more elements. Hence the order of the original sequence is maintained. This definition is useful for dealing with noisy sequences. Conversely B is a supersequence of A.

Frequency. The frequency or support of a subsequence is equal to its number of occurrences in D. A subsequence is frequent if it appears at least as often as the value of the minimum support parameter σ. Overlapping occurrences do not count towards the frequency.

Apriori Property. Any substring of a frequent subsequence is also frequent[3]. This implies the reverse: any supersequence of an infrequent subsequence is also infrequent. It follows that the Apriori Property lets us prune large parts of the search space when we are looking for frequent sequences. For the value of σ, this implies: the higher the minimum support, the more sequences are infrequent, the earlier we can stop searching a certain subspace and the quicker an algorithm is done. On the other hand, subsequences that appear less often than σ cannot be found at all.

[3]Different definitions apply to different domains. In for example itemset mining the definition is: any subset of a frequent itemset is also frequent.

Note that the definition of frequency is tied to the definition of the Apriori Property. Others have used a definition of frequency that involves the number of possible locations for matches, but this can break the Apriori Property if gap constraints are considered [15].

Total gap constraint. To facilitate the matching of subsequences rather than substrings, we use the total gap limit τ. The total gap constraint changes the rule for matching a subsequence with a possible occurrence. Subsequence $A = a_1 \dots a_n$ matches with sequence $B = e_1 \dots e_m$, iff $B = a_1(\Sigma \setminus a_2)^* a_2 \dots (\Sigma \setminus a_n)^* a_n$ and $m \leq n + \tau$. Note that B has to start with a_1 and end with a_n. For example, subsequence $Acure$ matches with $Accenture$ in the dataset, iff the maximum total gap is at least four. In this case $cent$ would be considered noise.

This definition differs from the consecutive gap constraint that is often used in the literature. This gap value is applied between every consecutive event [14, 3]. We use the total gap because (1) noise in our data is unevenly distributed within matches and we only want to allow noise as necessary and (2) it does not break the Apriori Property.

To see that the total gap constraint does not break the Apriori Property, suppose sequence A matches with sequence B, with $g \leq \tau$ events marked as gaps. Then any substring of A can also be found in B, with g' gap elements, where $g' \leq g \leq \tau$. This holds for every match, therefore the frequency of a substring of A is at least the frequency of A.

5 Monitoring Data

We gather data by continuously monitoring knowledge worker's computer usage. The continuous nature means that we do not make assumptions or choices regarding when the user performs interesting or uninteresting processes. Rather, we monitor all usage for a set amount of time, which gives us a broad scope for candidate processes. This enables us to find processes of which the user was not explicitly aware.

Remember from Section 4.1 that a data set consists of one or multiple sequences. All events are split into sequences based on a user identifier. The user identifier is tied to the machine on which the events were recorded. Other than this, a timestamp is saved with each event.

Additionally, it would be possible to split a user's sequence when he has been inactive. This is only sensible when we can avoid splitting the underlying business processes with certainty. We cannot meet this requirement, because the user might pick up his task after any period of inactivity. Furthermore we cannot distinguish between data missing due to manual deletion or a computer being in sleep mode. Using any measure of inactivity could therefore lead to false negatives.

A look at the risk of misclassification around possible cut-off points also supports this decision. A false positive is not a big problem. Sequences that are uninteresting likely get buried during our sorting process. Furthermore, we found that the amount of time required to handle the additional sequences is negligible. A false negative is a bigger issue. If one match of an interesting subsequence is missed, its children and parents also miss a match. It might even cause a sequence's support to drop below σ, eliminating it from the list of results.

5.1 Raw data collection

For our monitoring purpose, we extended an existing system that is used to record short macros. Our software monitors seven different types of events, as detailed in Table 1. For every logged event, we also save event-specific information, as described by the cells in the table's second column.

The keystroke hook is triggered whenever a key is pressed. Every key press is captured by the keyboard hook. The mouse button is one of the following values: left, right, double left, double right, middle.

Event type	Event specific properties
Keystroke	Key name
Mouse click	Mouse button Mouse coordinates Screenshot
Mouse scroll	Mouse coordinates Direction
Window focus change	New process title
Window title change	New window title
Clipboard update	Clipboard value (text)
Scene change	Screenshot

Table 1: Recorded events and properties, by type.

The mouse x and y coordinates are measured in pixels, relative to the display in which the mouse is positioned. If M, N are the largest values for the attached displays' respective heights and widths, then $0 \leq x < M$ and $0 \leq y < N$. The most upper left x and y coordinates are (0,0).

The clipboard hook is triggered on any update to the clipboard, independently of how it was set. If the data in the clipboard is a text string, we also store it in the event. If the data is binary we ignore the data.

We also detect when the screen images changes significantly, this is called scene change detection. we take screenshots periodically with an interval of one second. Every time a screenshot is taken, we hand greyscale images of the latest two screenshots to a simple algorithm. We score the difference of a frame I_1 with its preceding frame I_2 with the Sum of Absolute Differences (SAD)

$$d(I_1, I_2) = \frac{1}{MN} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} |I_1(x, y) - I_2(x, y)|, \tag{1}$$

where $0 \leq I(x, y) \leq 1$ is the grayscale value of pixel (x, y) in image I. This score is normalized by the height M and width N of the image, which makes the formula is agnostic to image size. If this score is larger than a fixed threshold, we log a scene change. The threshold has been empirically set to 0.02, to detect changes as big as a pop up window, and to ignore changes as small as typed text or a notification box.

In this implementation, detection around the threshold is not very consistent. Whether or not an event is created depends on the contrast with the previous screenshot and timing. Still, we found that the majority SAD values for scene change events is much larger than the threshold. We accept that sequences become slightly more noisy because of this implementation.

5.2 Data consolidation and pre-processing

The raw data as we capture it can be improved before the analysis. For many of the seven event types we use some type of preprocessing.

The goal of our preprocessing operations is twofold. First we want to further specify events that belong to the same type, but are without doubt distinctive. Secondly, we want to combine the information of multiple events to fewer events, when possible. As sequences get more concise, less gaps are needed for matches, and the analysis takes less time.

5.2.1 Keyboard logger

All contiguous key strokes are processed as a string. The preprocessing then consists of three phases: consolidating, splitting and classification. We consolidate backspace key strokes. For n backspace key

strokes, we remove the preceding n key strokes, and the backspaces themselves. We are not able to recognize more complex edits to a written string, such as when the user moves the text cursor.

We split the consolidated string on the following key strokes: $enter$, tab and alt+tab, because each has a distinct functionality in most environments, separate from typing. The separators become their own item. For example, the following series of key strokes: "name tab address tab name@domain.ext $enter$" is split into six separate type events. Splitting is useful when variations of the same business process are being matched. One such variation could using a mouse click instead of a tab key. While the separators themselves would not be matched, the text in between separators would correspond.

The split string is then classified as one of the following: email address, number or other. We have chosen these classes to add information, while making sure that misclassifications are highly unlikely. Together with the three special key strokes, six elements are added to Σ: $\{t_1, \ldots, t_6\}$.

5.2.2 Mouse clicks

Of the different type of mouse clicks, clicks with the left mouse button are the most common, and can account for 40% of the events in a data set. This high a frequency warrants further specification to avoid false matches. To subdivide these mouse clicks, we use the background colour of the area that was clicked. To get the background colour, we examine a range of pixels around the clicked pixel[4]. We then take the colour that occurs most frequently in this range. This lets us ignore outlier colours, which often appear when buttons have text on them. To map the colour to the event's index value, we take the colour's hash value modulo 93. Therefore we add $\{m_1, \ldots, m_{93}\}$ to Σ.

That this operation results in the background colour of the button requires three assumptions. First, buttons have the same background colour whenever they are clicked. Secondly, the entirety of the button has the same background colour. Lastly, the most prevalent colour of the button is the same as the most prevalent colour in the rectangle[5]. Though these assumptions might not always hold perfectly, the added information outweighs the slight chance of a misclassified button background colour.

Because clicks of either of the other mouse buttons are relatively infrequent in our sample data sets, classifying them according to the mouse button is enough. Therefore we add one item to Σ for the remaining four classes that we collect as raw data: $\{m_{94}, \ldots, m_{97}\}$, respectively mouse right, mouse double left, mouse double right[6] and mouse middle.

5.2.3 Mouse scrolls

We ignore repeated mouse scroll events. The general usage pattern for scrolling the mouse wheel is a repeated number of scrolls in the same direction. This gives rise to many variations of the same scrolling action, where the number of scrolls differs slightly. To make these variations equivalent, we ignore the number of scrolls altogether.

Furthermore, we do not distinguish between scrolling up or scrolling down, because users are prone to overshoot when scrolling to a desired view. Therefore we add only one item to Σ: $\{m_{98}\}$.

5.2.4 Windows process change

For the process change we can make the events more detailed, without losing generality. Any user action is specific to a context, and by including the title of the computer process, we incorporate that information. Therefore we add the following process change events p to Σ: $\{p_1, \ldots, p_k\}$, where k is the number of unique processes that had focus at some point. In practice, $7 \leq k \leq 20$, depending on the number of different users, and how many events they recorded.

[4] A square of 21×21 pixels, centring around the pixel that was clicked.

[5] This will not always hold when clicking the edge area of a button, or when the buttons are round.

[6] Double right mouse clicks generally do not occur.

From this viewpoint, it does not seem logical to allow multiple subsequent ps. We remove any p that is immediately followed by another p.

5.2.5 Window title change

Window title changes are somewhat more difficult, because applications often handle their window title differently. On the one hand, this could be valuable information for general purpose software such as internet browsers, to distinguish use cases. On the other hand, some applications change their title liberally based on for example, which user is logged in. In the latter case, this would obstruct matching processes executed by different users.

Therefore we decided not to use the updated title itself as the basis for an event. Instead we treat window title change events as if they were process change events. Though this solution is not powerful as a discriminant, it is a safe way to detect changes in the general functionality of an application.

5.2.6 Clipboard update

We have chosen not to identify the contents of the clipboard, as we do with a text type event. Instead, we only log the occurrence of a clipboard update. This is for two reasons. First, clipboard updates are generally much less frequent than text type events, thus there is less need to split clipboard updates to different classes. Secondly, the copying of values itself might be a (sub)process. Note that this item does not capture clipboard paste actions, for technical reasons[7]. These would generally be captured by either a t_3 or by $m_{94}m_1$ instead.[8] The clipboard update adds one item to Σ: $\{c\}$.

5.2.7 Scene change

Scene changes are currently not modified, and we only log the occurrence of a scene change. So that contributes one item to Σ: $\{s\}$.

To sum up, we use the data of six of the seven sources, and preprocessing turns this into at most $k + 106$ unique events for a data set. The alphabet Σ then contains $\{c, m_1, \ldots, m_{98}, p_1, \ldots, p_k, s, t_1, \ldots, t_6\}$. Some items (including many of the left-mouse-button clicks) may not occur in a data set. As an example, if we use this preprocessing scheme on the recording from Section 1, we get the following sequence: "$p_1m_1st_1t_1t_5t_5t_5m_1p_2t_3t_4m_1$".

As noted in the problem statement in Section 4, we use a separate start alphabet $\Sigma' \subseteq \Sigma$. To enforce the idea that all actions are specific to a p event, all subsequences must start with a p event. Σ' consists of $\{p_1, \ldots, p_k\}$

6 Data Analysis with GSP

In this section we consider the Generalized Sequential Pattern (GSP) algorithm, to mine a data set for frequent sequences. The basic GSP algorithm (Algorithm 1) is based on the Apriori Property, which is explained in Section 4.1. The general idea of the algorithm is to generate frequent sequences in a breadth first manner, starting from sequences of length one (a 1-sequence). The support of 1-sequences is calculated, and sequences that are not frequent get pruned. The remaining sequences are used to generate candidates 2-sequences, using Σ. The parent sequences can then be used to prune the candidates. Suppose the issue is whether or not the sequence aab is frequent: then both aa and ab need to be frequent. This information is available if the search for 2-sequences is finished.

[7] The Windows event hook API is not triggered when reading the value of the clipboard.

[8] t_3 means a "type other" event, from $ctrl$+v, $m_{94}m_1$ evaluates to a right mouse click (to open a context menu) followed by a left mouse click over a white background colour (to select paste).

The downside of the GSP algorithm is related to the explosion of the number of frequent sequences. This is problematic from both a time and a space perspective, because the number of possible sequences grows exponentially with the length of the sequence. Furthermore, every level of sequences requires a pass over the data set. With respect to memory usage it is also expensive, because all frequent subsequences of length n must be known to find those of length $n + 1$.

This exponential growth can be countered by increasing the minimum support value σ, which enables much earlier pruning. It will naturally also reduce the amount of results.

Algorithm 1: GSP (breadth-first)

```
1  i = 1
2  F_1 = {e ∈ Σ | e is frequent}
3  out = empty list of sequences
4  while F_i ≠ ∅ do
5      forall seq ∈ F_i do
6          children = extend(seq, Σ)
7          children = prune(children, F_i)
8          forall child ∈ children do
9              if child.support ≥ σ then
10                 out.add(seq)
11                 F_{i+1}.add(child)
12             end
13         end
14     end
15     i = i + 1
16 end
17 return out
```

Algorithm 2: GSP (modified)

```
1
2  F = {e ∈ Σ' | e is frequent}
3  out = empty list of sequences
4  while F ≠ ∅ do
5
6      seq = F.dequeue()
7      children = findChildren(seq)
8      forall child ∈ children do
9          if child.support ≥ σ then
10             out.add(seq)
11             F.enqueue(child)
12         end
13
14     end
15
16 end
17 return out
```

6.1 Speed and memory improvements

We made a few changes to the GSP algorithm, with the goal of expediting the data analysis. The modified GSP algorithm is displayed in Algorithm 2. First, we only consider sequences that start with an event $e \in \Sigma'$ (line 2). A sample data set shows that only about 25% of the logged events is a process (or window) change, which greatly reduces the number of candidate processes. Because the process change events are well spread across the data, it also concentrates the search in areas that are significantly different from one another.

Secondly, the matches we find (as described in Section 6.2) are stored for each sequence. For each occurrence, we store the indices (sequence index and event index) that define where the match's last event appears in the data set. This is useful when generating the children of a sequence. Because we remember the indices of of the parent's occurrences, we do not need to pass over the entire data set to find the child occurrences.

Only expanding subsequences that start with a process change event has several consequences. One consequence is that we lose the pruning mechanism described in the first paragraph of Section 6. One can no longer be sure that a subsequence is infrequent based on the frequency of its right parent, because right parents that do not start with a process change event are not evaluated.

Another consequence is that, by abandoning the level by level approach, the search is no longer strictly breadth first. Search for a 5-sequence can commence right after its left parent 4-sequence has found all matches. The result is seemingly paradoxical: the modified GSP algorithm is less memory hungry, because it stores more information.

The advantages of the modified GSP algorithm become apparant when the data set is large (or τ is high). In these cases the number of frequent sequences grows much faster in Algorithm 1 than in Algorithm 2.

For example the analyses on any of the the experiments of Section 8 were sped up by about two orders of magnitude with the modified GSP algorithm.

6.2 Noisy matching

The subsequences that we search for are generated by processes that are not deterministic. This is because user interfaces are often designed to be forgiving, and the execution of a task can typically be achieved with different actions, and in different orders. Furthermore, the users who interact with these user interfaces tend to experiment, forget and make mistakes when it comes to executing a process. These appear in the sequence database as permutations, insertions or in limited cases, even deletions. Hence the data is permeated with noise, even in the interesting regions that contain frequent subsequences.

To add noisy matching to GSP, we implement the total gap constraint as described in Section 4.1. Other than the indices that describe where match occurs in the data set, we also store the number of gaps g required to make the match. When finding the children of a sequence (Algorithm 3), we use the remaining noise budget, $\tau - g$ to determine how far we need to look ahead from the index i.

Algorithm 3: $findChildren(parent)$

```
1  Children = empty set of sequences
2  forall match ∈ parent.Matches do
3      forall event ∈ S[match.i ... match.i + τ − match.g] do
4          if event is duplicate of previous event or event is not frequent then
5              continue
6          end
7          child = parent + event
8          Children[child].addMatch(i, g) //find new i, g based on match gap and letter position
9      end
10 end
11 return children
```

Notice from Algorithm 3 that we skip an $event$ when the same $event$ has already appeared in the range of S defined by the noise budget. We do this to prevent overlapping matches. For example, suppose $\tau = 1$ and $D = \{abb\}$, then ab can be found twice. As per the definition of matching with the total gap constrain in Section 4.1, we only consider the strict prefix ab a match.

Overlap can also occur when $\tau = 0$. We use the renewal model to avoid duplicate counting of overlapping substrings. As Regnier explains: "In a chain of overlapping occurrences, the first occurrence is always valid. Another occurrence is valid iff it does not overlap on the left with a valid occurrence" [16]. For example, aa is found only once in aaa, as the leftmost occurrence.

There are advantages to this approach of noise handling. First, we can match sequences with up to τ insertions. Other operations such as substitution or deletion can be matched by describing them in terms of insertions from the shorter sequence. Secondly, by considering noise during support calculation, we avoid prematurely pruning interesting sequences. One downside is the disproportionally high cost of transpositions of two adjacent events: two insertions. Another disadvantage is an explosion in the number of results.

7 Data Representation

Our analysis of monitoring data produces a very large amount of results. Some research enjoys the luxury of only considering maximal frequent subsequences[9]. In our application a subsequence of maximal

[9] A frequent sequence s is maximal if there exists no frequent supersequence of s.

length is not likely the most interesting subsequence. We have to consider all frequent subsequences potentially interesting business process.

We are able to reduce the set of results by using $\Sigma' \subseteq \Sigma$. In the classical situation where $\Sigma' = \Sigma$, the Apriori Property causes an explosion of the amount of results. For every frequent subsequence of length n, the left and right parents (of length $n-1$) are also frequent. By choosing Σ' such that its items only appear in a fraction of D, we mitigate this problem. This generally restricts the generation of results to left parents, and eliminates many derivative subsequences.

Still, the amount of frequent subsequences makes identifying relevant information difficult. To make the results easily digestible for users, we have implemented a two part solution. We first sort, then cluster the results and present consolidated data to the user.

7.1 Sorting

With the sorting operation, we intend to highlight subsequences based on interestingness. Much research has been done on interestingness metrics [2, 21, 24]. We take some ideas from the literature, but mostly use custom metrics that are specific to our problem.

The qualities that make a sequence interesting are: length, frequency, noisiness of the occurrences and specificity. Length and frequency are obvious measures. Specificity warrants an additional explanation. A sequence is more interesting if it is more detailed and its events are less common, all other things being equal. More detailed sequences are more likely to exclude matches with similar events, but that do not refer to the same business process. To see why noisiness is important, note that matches for a sequence are generally easier to find as τ increases. As occurrences use more of their noise budget, they become less specific and less interesting.

We consider a metric based on the statistical probability that the matches of subsequence s appear in D. We use the Bernoulli model: the probability is calculated with respect to a random distribution of the same events.

To see how unexpected a subsequence is we first calculate the prior probability $p(s, g)$ of an occurrence of s with g gap events appearing in a sequence of length $|s| + g$ in Equation 2. The product on the right calculates the probability of each of the events of s to appear in a sequence of length $|s|$. We multiply this by the binomial coefficient on the left, which is the number possible ways in which you can place g wildcards into s[10]. We also multiply by $g!$, the number of possible permutations of the wildcards.

$$p(s, g) = \binom{|s| + g - 2}{g} g! \prod_{j=1}^{|s|} \frac{n_S(s_j)}{|S|}, \tag{2}$$

where $n_S(s_j)$ is the frequency of event s_j in sequence S. Now we can calculate the probability that the occurrences of s appear in D. We solve the recurrence relation in Equation 3. The probability of string s appearing exactly σ times in sequence S, given gap values $g_1, \ldots, g_\sigma$ (one for each occurrence), is given by

$$P(\sigma, |S|) = \begin{cases} 1, & \text{if } \sigma = 0 \\ 0, & \text{if } |s| \cdot \sigma + \Sigma_{i=1}^{\sigma} g_\sigma > |S| \\ P(\sigma, |S| - 1) + & \\ \quad p(s, g_\sigma) \cdot P(\sigma - 1, |S| - |s| - g_\sigma) - & \\ \quad p(s, g_\sigma) \cdot P(\sigma - 1, |S| - |s| - g_\sigma) \cdot P(\sigma, |S| - |s| - g_\sigma), & \text{otherwise} \end{cases} \tag{3}$$

[10] We do not use actual wildcards. Our definition of matching with the total gap constraint from Section 4.1 forbids noise elements that could have been a matching element. Therefore we slightly overestimate the probability. We accept this inaccuracy for ease of calculation (and notation).

In this equation we abstract from: possible overlap in matches, a decreasing p_s after a match to account for the changed distribution of events in S. Furthermore, we treat D as one long sequence, by using $S = D_1 \dots D_{|D|}$, where D_i is sequence i of data set D.

Though the abstractions mean Equation 3 is not entirely accurate, it does not have to be. A sorting under this probability metric balances the four qualities well. The downside of this method is of course the time complexity, which is $O(|S| \cdot |s|)$ in our dynamic programming implementation. Combined with the huge amount of results the GSP algorithm produces, calculating this metric takes a considerable amount of time for large data sets, or when τ is high.

As a quicker alternative, we also developed a heuristic metric, defined as

$$heuristic(s) = \big(length(s) + nRareEvents(s)\big) \cdot \sqrt{support(s) - noisePenalty(s)}, \tag{4}$$

where $nRareEvents(s)$ is the number of events in s that are less frequent in S than they would be if they were uniformly distributed in S. The $noisePenalty(s)$ is a linearly scaled penalty that is 0 when the occurrences of s do not contain noise, and $0.5 \cdot support(s)$ when all occurrences contain τ noise. We use the square root to normalize the unbounded frequency with respect to the shorter length of a sequence.

Though this metric also takes into account all four interesting qualities, it too has balance issues. The normalization has the undesired effect the sorting is biased against very frequent sequences. Moreover, $noisePenalty(s)$ does not generalize well for high τ. Furthermore, tuning this metric is a trial and error process, which also makes it high maintenance.

To sum up, we propose two metrics. The simpler $heuristic$ metric is useful to quickly get a somewhat unbalanced sorted list. If the data set is small enough[11], use $P(\sigma, |S|)$ to get a sorted list that corresponds more closely with the subsequences' interestingness. We leave the choice to the user.

7.2 Clustering

The goal with the clustering operation is to make browsing the results easy by aggregating some sequences. We take a sorted list of subsequences as input, and output a list of groups (or clusters), where each group is a list of subsequences. Ideally, each cluster represents a single business process, and the sequences within the cluster then represent the variety of executions of that process.

An obvious solution to this problem can be a variant of the popular Expectation-Maximization (EM) algorithm. Such algorithms cluster any number of objects to a predefined number of classes. EM algorithms are usually applied in an n-dimensional vector space, in which the mean value of a class is defined. This is not the case in the space of subsequences, thus an EM approach would be problematic.

Instead we use a simple clustering approach based on the Jaccard similarity. Two subsequences can belong to the same group if their Jaccard similarity is above a threshold j_{min}. Suppose M_i is the set of matches for a subsequence s_i, then the Jaccard similarity of two sequences s_1, s_2 is given by

$$J(s_1, s_2) = \frac{|M_1 \cap M_2|}{|M_1 \cup M_2|}, \tag{5}$$

where $|M_1 \cap M_2|$ is the number of matches that overlap at least partially, $|M_1 \cup M_2|$ is defined analogously. A smaller j_{min} corresponds with fewer clusters. We empirically tuned the value to 0.4.

Pseudo code for simple clustering is given in Algorithm 4. We iterate over the sorted list of frequent subsequences. The centre or root of each group is defined by the first element that is added to it. If a

[11] Calculating the values of the frequent sequences for one of the experiments (experiment on Bob's data, see Section 8) took five hours. As the number is frequent sequences increases with a larger data set or different settings, the time required can quickly become infeasible.

Algorithm 4: simple clustering

```
1  Groups = empty list of groups
2  forall seq ∈ sortedSequences do
3      forall group ∈ Groups do
4          found = false
5          if J(seq, group.root) > j_min then
6              group.Add(seq)
7              group.root = seq
8              found = true
9              break
10         end
11     end
12     if !found then
13         Groups.Add(new group(seq))
14     end
15 end
16 return Groups
```

subsequence is similar to the root of any of the existing groups, add it to that group. Otherwise, create a new group with the subsequence as root.

The simple clustering algorithm is not very expensive, its time complexity is linear in the length of the input data. We also preserve the initial ordering of sequences within clusters, which is useful for the final Section of data representation, the user interface.

7.3 User interface

To make browsing the analysis results easy, we built a user interface (UI). The tool (called the Accenture Process Analyzer) allows one to use the methods from Sections 6 and 7. The main feature of the Process Analyzer is the ability to display events of a sequence in detail, but also offer a more global view for increased browsing speed.

For this section, refer to Figure 1 for a screenshot of the Process Analyzer. A typical use case works as follows: the user loads one or multiple files that were generated by the monitoring application (step 1). Then one chooses values for the two parameters: minimal support (min #occurrences in the UI) and τ (max noise in the UI) (step 2). After the analysis finishes, the rest of the UI is populated. Two UI elements called expanders show information about the loaded data and general information about the analysis (step 3). Finally, there is one expander for each cluster of sequences. By default, each sequence is indicated by the signature only, but by selecting a sequence the legend is shown (step 4). To further inspect the contents of a sequence, one can use the html documentation (step 5).

The html documentation is available for each of the occurrences in a sequence, and shows the raw recording of an occurrence as described Section 5.1. This includes the exact keystrokes and the screenshots that accompany the screen change events and mouse clicks. The information in the html documentation can be used to verify whether the sequence actually indicates a business process or not.

The utility of the clustering approach now becomes apparent. Each of the clusters should correspond with one potential business process. Its sequences are then variations of the process. If the inspection of a few sequences reveals that a cluster is uninteresting, one can collapse the expander to hide all sequences belonging to a cluster. This enables the user to quickly browse the results to locate interesting sequences.

Finally, once one has found a sequence that corresponds to a business process, one can generate an automation script. This script can be edited and run by Accenture's in-house RPA software, the Process Studio. The script generated by the Process Analyzer only uses the raw recording of one of the occur-

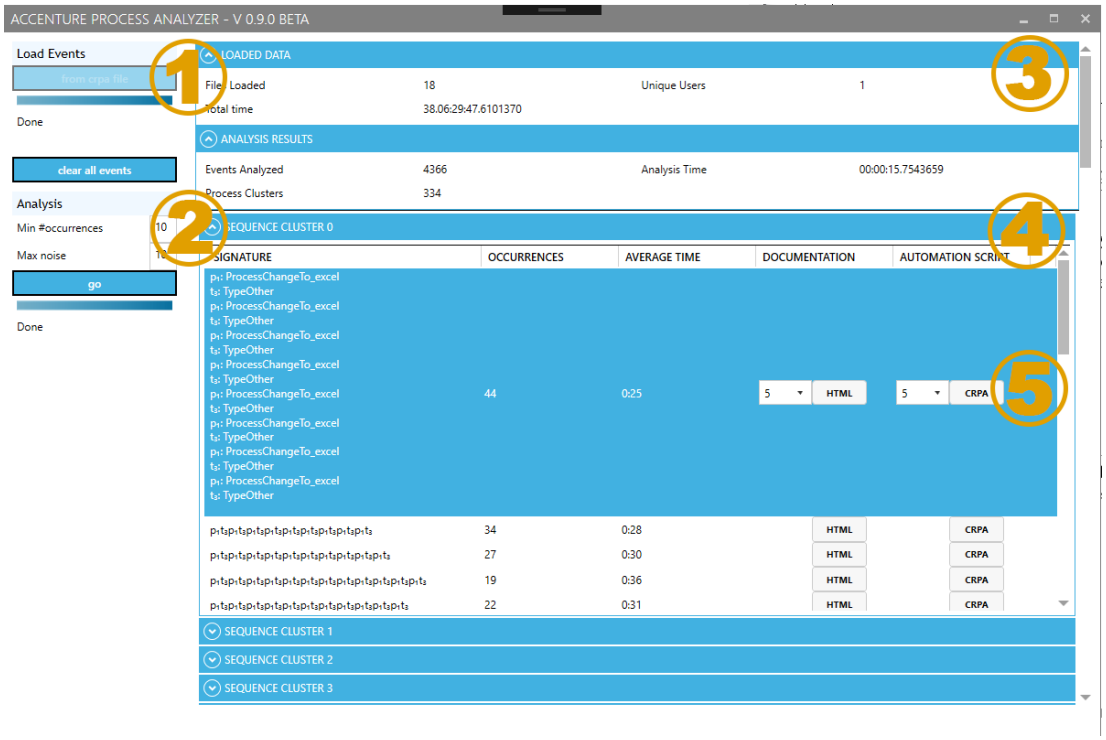

Figure 1: Screenshot of the Accenture Process Analyzer, the numbers indicate the steps involved in a typical use case.

rences to generate the script. Therefore it is a very basic script, but useful as a starting point for process automation nonetheless.

8 Experiments

For our experiments, we are interested in the following questions. How good is our GSP at finding occurrences a data set? Can business processes be found in their entirety? How does τ influence this? These questions are answered in Section 8.1. The experiments are performed on a synthetic dataset.

Furthermore, we want to know how well our algorithm performs in a real-world situation. How does it handle large data sets? How good is GSP as a process discovery tool? Are interesting subsequences properly prioritized over uninteresting sequences? For answers to such questions, consult Section 8.2.

8.1 Recall ratio

Ideally we want to find all occurrences from head to tail for all interesting sequences. In practice this is problematic. Therefore we investigate to what extent we reach this goal, using the recall ratio. In general the recall ratio is given by $tp/(tp + fn)$, where tp is the number of true positives, and fn is the number of false negatives. Because we are interested in both the fraction of occurrences and the length of the occurrences found, we use two recall ratios.

One is the support recall, given by: support of s / the number of executions of business process b in data set D. The other is the length recall: suppose s has a set of matches M, then the length recall is $\sum_{i=1}^{|M|} |M_i|/|\bar{b}|$, where $|\bar{b}|$ is the average length of b, in events, and $|M_i|$ is the length of a match, which can include up to τ non-matching events.

8.1.1 Experimental setup

To find the recall rates, we need a data set D in which the set of business processes B is known, along with their average length in events and their number of appearances in D. This requirement can only be fulfilled by using a synthetic data set.

The synthetic data set contains 15,259 events and consists of 8 hours and 9 minutes of data in total, over ten episodes. We filled the synthetic data set with many executions of two business processes. Both processes would be triggered by an email message for a license request. First, values are copied from the email message to the argument box of our RPA script software[12]. This makes up b_1, and is repeated two to four times. Then we run the RPA script and b_2 follows: navigate to an online portal; fill in the values in the form; download the license file; open an email template; fill the recipient and attachment values. The data contains 493 occurrences of b_1, and b_2 is repeated 153 times, typically in the form of $b_1^r b_2$, where r, the number of repetitions of b_1, is typically 2–4.

We devised B to give b_1 and b_2 very different noise profiles, to show how GSP deals with different levels of noise. The selection of values when copying in b_1 can be done in many ways. Often (but not always) we selected fields with a triple click, which does not always succeed and fails when the data is not in the correct format. Hence there is a large intra-process variety. Examples of b_2 show on the contrary very little variety, because they were performed by a script.

We perform our GSP algorithm on this synthetic data set. The frequent subsequences are sorted by the $product$ metric, and clustered using the simple clustering algorithm. Afterwards, we manually search the results for subsequences that correspond with b_1 or b_2. Because τ greatly affects both recall ratios, we repeated the experiment with different values for τ: 3, 5, and 7. We choose $\sigma = 50$.

8.1.2 Results

The relation between the support ratio, length ratio and τ can be seen in Figure 2. Among frequent sequences there is always a trade-off between support and length. A more specific (longer) subsequence generally matches with fewer occurrences; to capture a larger fraction of occurrences, length is sacrificed. This trade-off is due to the variety among process executions, and is clear even in the less noisy b_2 process.

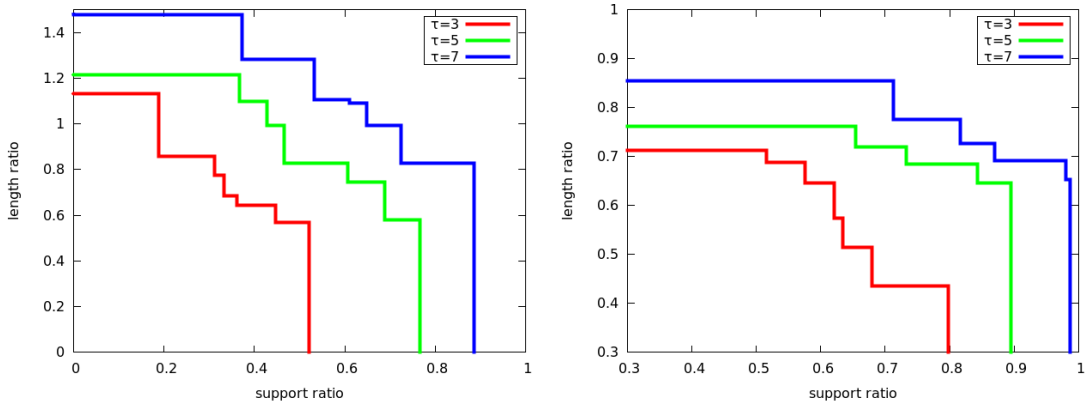

Figure 2: Plots of the Pareto fronts of the support ratio on the horizontal axis versus the length ratio on the vertical axis. The figures relate to the license argument process b_1 (left), and the generate license process b_2 (right).

While it is true that raising τ improved both recall ratios, this has drawbacks that are not apparent from Figure 2. First, the execution time grows exponentially with τ. Secondly, sequences that do not correspond to a business process are more likely to appear in the results. We investigate this effect in

[12] Accenture develops RPA software in-house under the name Accenture Robotics Platform (ARP)

Section 8.2. Finally, sequences that do correspond to a business process are more likely to be prepended or appended with non-corresponding events.

Process b_1 is one example of an incorrectly expanded sequence. In Figure 2 this happens when a data point has a length ratio greater than 1. Because b_1 is often directly followed by another b_1, some subsequences span multiple b_1s. This effect can be seen whenever multiple business processes follow each other in the data.

For $\tau = 7$, the best support ratios we found were 88% and 99% for b_1 and b_2 respectively. There are two important caveats to this result. On the one hand, subsequences with better recall ratios can always be found by increasing τ[13]. On the other hand, this data set was specifically engineered to contain very few events that do not belong to either b_1 or b_2. Furthermore, it is smaller than a real-world data set. Therefore based on this experiment, we cannot recommend a certain value for τ.

8.2 Real-world performance

For our real-world use case, we continuously monitored two users (Alice and Bob) over a few days. They were selected because we suspected that their activities include repetitive processes. Alice and Bob provided us respectively two and four working days' worth of monitoring data, which amounts to 4,384 and 32,558 events, respectively. They were instructed to perform their tasks as they normally would. This makes the data set bigger, and much noisier than the synthetic data set from Section 8.1.1.

8.2.1 Experimental setup

As opposed to the data set in the previous section, we do not know what processes Alice and Bob[14] perform during the recordings. Hence a quantitative analysis that gives us support ratios or length ratios is problematic. Instead, we verify our results with Alice and Bob directly. We ask them a few questions about selected sequences we think are interesting. First, we ask them to identify the processes on the basis of html documentation examples. This tells us: whether the processes we find are part of a longer process; how well the clustering works; whether the various occurrences of a process we find actually refer to the same process. Finally, we ask about any business processes that we might have missed. We decide not to ask about the number of occurrences of the processes, because we deem any answer to this question unreliable.

We are also interested in the performance of our sorting and clustering methods. Ideally, if we find n interesting sequences, we want the first n clusters to represent them. To what extend do we reach this goal? And related: how well do we filter uninteresting sequences? This should lead to a few points of improvements for the sorting and clustering methods.

Unfortunately, Alice and Bob were not performing the same business processes. Therefore we cannot analyse how much larger τ needs to be to adjust to find a generic pattern that captures the patterns of multiple users. Alice and Bob contributed respectively two and four days' worth of data. We used $\tau = 10$, which is higher than the τ used for the analysis on the synthetic data set, because we expect more noise in these data sets. We use $\sigma = 10$ (Alice), and $\sigma = 20$ (Bob), and the heuristic sorting metric.

8.2.2 Results

We have made a selection of the results of the analysis of Alice's and Bob's data, which can be found in Table 2. For each of the first five clusters of sequences, the top sequence is included. For readabilty, we use the superscript to denote repetitions. Note that this is only a fraction of the results. In total the

[13] Suppose $\tau = \infty$, then there would be a subsequence with $lengthratio = supportratio$ = 1, for any reasonable business process. Of course one would not be able to find such a sequence, because this makes almost all sequences frequent.

[14] Alice and Bob are not their real names.

analysis produces over 300 clusters for over 250,000 sequences (Alice), and almost 5,000 clusters for over 40,000 sequences (Bob)[15].

User	Cluster	Process Signature	Support	Avg time
Alice	1	$(p_7t_3)^8$	44	25 s
Alice	2	$(p_7t_3)^{10}$	15	39 s
Alice	2	$(p_7t_3)^4$	88	16 s
Alice	3	p_7c^{11}	42	5 s
Alice	4	$p_7t_3ccp_7c^9t_3cc$	14	8 s
Alice	5	$(p_7t_3)^2t_4(p_7t_3)^2$	37	15 s
Alice	6	$p_7t_3t_4(p_7t_3)^7$	10	22 s
Alice	7	$(p_7t_3)^3ccp_7c^9$	10	13 s
Alice	11	$p_7sp_7p_7$	99	7 s
Alice	12	$p_7t_3cct_3p_7t_3c^3$	16	9 s
Bob	1	$p_2m_1^3$	535	29 s
Bob	2	$p_7t_3c^5$	137	7 s
Bob	3	$p_2m_1^5$	191	32 s
Bob	4	p_7^3	687	9s
Bob	5	$p_7t_3c^5t_3ct_3ct_3cc$	45	11 s
Bob	5	$p_7t_3ccp_7t_3ct_3cc$	56	8 s
Bob	6	$p_8m_1^3$	349	19 s
Bob	40	$(p_{17}t_5p_8)^4$	23	15 s

Table 2: A selection of frequent sequences from the analysis of Alice's and Bob's data. Legend in order of appearance: p_7: process change to Excel, t_3: type other, c: update clipboard, t_4: type enter, p_2: process change to Outlook, m_1: mouse click (left button, white background colour), p_{17}: process change to Internet Explorer, p_8: process change to Word.

Upon validating our results with Alice and Bob, we found that there is still room for improvement.

Alice identified a few interesting business processes based on the sequences we found. Sequences in cluster 1 mostly refer to editing the human resource Excel file (HR file). Those in cluster 2 deal mostly with editing the remaining work Excel file (work file). Clusters 3 and 4 have many sequences that involve copying large amounts of values from the work file to the HR file. Cluster 5 contains sequences that refer to Alice using the Excel's find functionality to find the correct team member before changing his number of worked hours. Bob's main business process was: (1) downloading an updated version of a roll-on/roll-off HR Excel file, (2) copying various values relating to one entry to another Excel file, and (3) filling in the same information through a web interface. Of these we found part (2) as a business process. We found another process where Bob would request a co-worker access to client computers. Bob also identified one process (the creation of a new work item) that did not appear in our discussion. Upon further inspection, we found several of these among the generic sequences in cluster 6.

Many uninteresting sequences still appear in the results. The problem is significant in the analysis of Bob's data set. Four of the first six clusters are all uninteresting, because they capture general Outlook usage, or are too general to capture specific activity. Another example of this is $p_7sp_7p_7$, in cluster 11 (Alice).

The fact that two sequences in different clusters are so similar indicates a similar problem: there will be many false positives because the signatures are too generic. Notice that all 15 occurrences of $(p_7t_3)^{10}$ appear in $(p_7t_3)^8$, because the second sequence is a substring of the first sequence. In many other sequences[16] we found that not all occurrences refer to the same activity. To make the sequences more consistent, and to reduce the frequency of generic sequences, we should further specify the window title change events.

[15]There is a large difference in the sequence-cluster ratio between Alice and Bob. Though we do not research this further, it could be an indication of a different amount of variety in their work. At least it shows that different data sets can have very different properties.

[16]Specifically, this concerns the following sequences: $(p_7t_3)^8$, $(p_7t_3)^2t_4(p_7t_3)^2$ and $p_7t_3ccp_7c^9t_3cc$ (Alice).

The clustering approach produces mixed results. On the one hand we received confirmation that we correctly put the first sequences of clusters 1 and 2 into different buckets (Alice). Though these clusters have very similar sequence signatures, the Excel files being edited were mostly different. One the other hand, the first sequences of multiple clusters refer to the same business process. This applies to clusters 3 and 4 (Alice) and clusters 2 and 5 (Bob). The clustering can still be improved.

9 Conclusion and future work

In this thesis we addressed the problem of finding and presenting interesting business processes from a computer usage data set. We proposed an end-to-end solution to this problem: from data collection, to data analysis, to the presentation of results.

There are a few issues remaining in our solution. Sequences are not always specific enough to prevent misclassifications. The heuristic sorting metric does not always filter uninteresting sequences. The sorting metric based on the statistical probability of the occurrences appearing in the data is better balanced, but has a problematic time complexity. The few misclassification of clusters we encountered were likely the result of lacking specificity in the sequences; the general approach clustering seems to work.

Despite the issues outlined above, most occurrences of sequences referred to the same general activity. More importantly, we were able to extract multiple interesting business processes from the data sets Alice and Bob provided. We confirmed this with two experiments. One by comparing the results of our analysis with a synthetic, labelled data set. Another using real-world data generated by two project management team members, and validating the analysis results with them. Therefore the sorting and clustering approach fulfilled its basic purpose.

We expect that the analysis of other data sets will reveal new issues. Before we could find any interesting sequences in our real-world data set, we needed to further refine our preprocessing. Each of the data sets we used had very different characteristics, making it likely that new data sets will contain unexpected features that thwart the analysis. Therefore a few more iterations of collecting data, analysing data and tuning are necessary before the pipeline performs well on new data sets.

9.1 Future work

Variations in process executions are the largest source of problems for the analysis. As we see it, there are two paths for improvement. One is to decrease the amount of noise per sequence. We could try to generalize the low-level information (such as mouse clicks), and extract high-level information from the sequences (such as look up customer number in Excel file). Another path is to make the analysis more noise agnostic. If our sorting metric could be used on a large data set and generate balanced results with respect to noise, then we could set τ arbitrarily high. This would be a more general solution to finding frequent sequences while allowing noise.

There is one feature we have been unable to test thus far. In theory our method of analysis is compatible with data that is recorded by multiple users who perform the same business processes. However, we expect that our current methods are unable to match executions of different users to the same sequence, when their usage is subject to different habits. Performance for the analysis of data recorded by multiple users would likely improve upon implementing the suggested improvements in the previous paragraph.

Though speed has not been an issue for the data sets we used, it might become an issue in the future if the data sets drastically grow. The time complexity of the analysis algorithm is exponential in the worst case after all. In this case we can easily parallelize the GSP algorithm. Because finding matches is only dependent on the matches of a parent sequence in our implementation, we expect that a speedup linear in the amount of processor cores should be feasible.

Acknowledgements

Thanks to Walter Kosters, Wojtek Kowalczyk and Simon Hall for providing guidance and direction. Also thanks to Sanne Vrijenhoek for giving advice and corrections. Finally, we thank Alice and Bob for collecting data.

References

[1] ALVES, R., RODRIGUEZ-BAENA, D. S., AND AGUILAR-RUIZ, J. S. Gene association analysis: A survey of frequent pattern mining from gene expression data. *Briefings in Bioinformatics 11* (2010), 210.

[2] BAENA-GARCIA, M., AND MORALES-BUENO, R. Mining interestingness measures for string pattern mining. *Knowledge-Based Systems 25* (2012), 45–50.

[3] BEEDKAR, K., AND GEMULLA, R. DESQ: Frequent sequence mining with subsequence constraints. In *IEEE 16th International Conference on Data Mining (ICDM)* (2016), pp. 793–798.

[4] EICHINGER, F., NAUCK, D. D., AND KLAWONN, F. Sequence mining for customer behaviour predictions in telecommunications. In *Workshop on Practical Data Mining: Applications, Experiences and Challenges* (2006), pp. 3–10.

[5] GARBONI, C., MASSEGLIA, F., AND TROUSSE, B. Sequential pattern mining for structure-based XML document classification. In *Advances in XML Information Retrieval and Evaluation: 4th International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX 2005)* (2006), LNCS 3977, Springer, pp. 458–468.

[6] GENG, L., AND HAMILTON, H. J. Choosing the right lens: Finding what is interesting in data mining. In *Quality Measures in Data Mining*. Springer, 2007, pp. 3–24.

[7] GLEDSON, A., ASFIANDY, D., MELLOR, J., BA-DHFARI, T. O. F., STRINGER, G., COUTH, S., BURNS, A., LEROI, I., ZENG, X., KEANE, J., BULL, C., RAYSON, P., SUTCLIFFE, A., AND SAWYER, P. Combining mouse and keyboard events with higher level desktop actions to detect mild cognitive impairment. In *IEEE International Conference on Healthcare Informatics (ICHI)* (2016), pp. 139–145.

[8] GUPTA, M., AND HAN, J. Applications for pattern discovery using sequential data mining. In *Pattern Discovery Using Sequence Data Mining: Applications and Studies*. IGI Global, 2012.

[9] HILDERMAN, R., AND HAMILTON, H. J. *Knowledge Discovery and Measures of Interest*. Springer, 2001.

[10] HUYNH, X.-H., GUILLET, F., BLANCHARD, J., KUNTZ, P., BRIAND, H., AND GRAS, R. A graph-based clustering approach to evaluate interestingness measures: a tool and a comparative study. In *Quality Measures in Data Mining*. Springer, 2007, pp. 25–50.

[11] KANT, R., SENGAMEDU, S. H., AND KUMAR, K. S. Comment spam detection by sequence mining. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining (WSDM 2012)* (2012), ACM, pp. 183–192.

[12] LENT, B., AGRAWAL, R., AND SRIKANT, R. Discovering trends in text databases. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining* (1997), pp. 227–230.

[13] MEMETI, S., AND PLLANA, S. A machine learning approach for accelerating DNA sequence analysis. *The International Journal of High Performance Computing Applications 30* (2016), 1–17.

[14] MILIARAKI, I., BERBERICH, K., GEMULLA, R., AND ZOUPANOS, S. Mind the gap: Large-scale frequent sequence mining. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD 2013)* (2013), ACM, pp. 797–808.

[15] MIN, F., WU, Y., AND WU, X. The apriori property of sequence pattern mining with wildcard gaps. *International Journal of Functional Informatics and Personalised Medicine 4* (2012), 15–31.

[16] REGNIER, M. A unified approach to word occurrences probabilities. *Discrete Applied Mathematics 104* (2000), 259–280.

[17] SINGH, B., AND SINGH, H. K. Web data mining research: A survey. In *IEEE International Conference on Computational Intelligence and Computing Research* (2010), pp. 1–10.

[18] SOARES, C., DE GRAAF, E., KOK, J., AND KOSTERS, W. Sequence mining on web access logs: A case study. In *18th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2006)* (2006), pp. 291–298.

[19] SRIVASTAVA, A., SURAL, S., AND MAJUMDAR, A. K. Database intrusion detection using weighted sequence mining. *Journal of Computers 1* (2006), 8–17.

[20] SYED, Z., INDYK, P., AND GUTTAG, J. Learning approximate sequential patterns for classification. *The Journal of Machine Learning Research 10* (2009), 1913–1936.

[21] TAN, P.-N., KUMAR, V., AND SRIVASTAVA, J. Selecting the right interestingness measure for association patterns. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2002)* (2002), ACM, pp. 32–41.

[22] TAN, X., YAO, M., AND ZHANG, J. Mining maximal frequent access sequences based on improved wap-tree. In *Sixth International Conference on Intelligent Systems Design and Applications* (2006), pp. 616–620.

[23] TAO, Y.-H., HONG, T.-P., AND SU, Y.-M. Web usage mining with intentional browsing data. *Expert Systems With Applications 34* (2008), 1893–1904.

[24] VELLAISAMY, K., AND LI, J. Bayesian approaches to ranking sequential patterns interestingness. In *9th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2006)* (2006), LNCS 4099, Springer, pp. 241–250.

[25] VILALTA, R., APTE, C. V., HELLERSTEIN, J. L., MA, S., AND WEISS, S. M. Predictive algorithms in the management of computer systems. *IBM Systems Journal 41* (2002), 461–474.

[26] WANG, K., XU, Y., AND YU, J. X. Scalable sequential pattern mining for biological sequences. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management* (2004), ACM, pp. 178–187.

[27] WANG, M., SHANG, X.-Q., AND LI, Z.-H. Sequential pattern mining for protein function prediction. In *Advanced Data Mining and Applications: 4th International Conference (ADMA 2008)* (2008), LNCS 5139, Springer, pp. 652–658.

[28] WANG, Y., LIM, E.-P., AND HWANG, S.-Y. Efficient mining of group patterns from user movement data. *Data & Knowledge Engineering 57* (2006), 240–282.

[29] WUU, L.-C., HUNG, C.-H., AND CHEN, S.-F. Building intrusion pattern miner for Snort network intrusion detection system. *Journal of Systems and Software 80* (2007), 1699–1715.