UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

**Stanislav Deviatykh**

# Discovering Automatable Routines from UI Logs via Sequential Pattern Mining

**Master's Thesis (30 ECTS)**

Supervisor(s): Marlon Dumas
Volodymyr Leno

Tartu 2020

# Discovering Automatable Routines from UI Logs via Sequential Pattern Mining

**Abstract:**

Robotic Process Automation (RPA) is a rapidly evolving technology that allows us to automate non-value adding tasks (i.e., routine tasks), such as transferring data from one application to another. The automation of such tasks allows us to reduce the number of errors that occur during its execution and decrease task execution time. However, RPA intended to be used for automation routines, but not for their discovering. This thesis proposes a method for discovering routines from user interaction log by exploiting sequential pattern mining techniques for dealing with noise within the log. Since it is essential to understand which of the discovered routines are automatable, the thesis proposes a method for measuring the routine automatability index (RAI). The method is based on identifying if the routine actions are automatable by obtaining dependencies between them, more precisely, data transformations and functional dependencies. A comparative evaluation with the existing approach on synthetic and controlled-setting datasets shows that the proposed method can discover candidate routines, identify action dependencies, and measure RAI with acceptable execution time. The proposed approach has been implemented in Java, integrated with the SPMF pattern mining tool, Foofah data transformation tool, and Tane algorithm for finding functional dependencies.

**Keywords:**

Robotic Process Automation, Robotic Process Mining, Pattern Discovery, Automatable Routine, Sequential Pattern Mining

**CERCS: P170. Computer science, numerical analysis, systems, control.**

## Automatiseeritavate Rutiinide Leidmine UI Logidest Järgnevus Mustrite Kaevandamisega

**Lühikokkuvõte:**

Robotic Process Automation (RPA, Protsesside automatiseerimine robootikaga) on kiirelt arenev tehnoloogia, mis aitab meil automatiseerida mitte väärtust lisavaid ülesandeid (nt rutiinsed ülesanded), näiteks andmete transfeer ühest rakendusest teise. Selliste ülesannete automatiseerimine aitab meil vähendada ülesande läbiviimisel tekkivaid vigu ning samuti aitab kahandada ülesandele kuluvat aega. RPA on mõeldud rutiinide automatiseerimiseks, aga mitte nende leidmiseks. Seetõttu, esitleb antud uurimustöö meetodit mis aitab leida rutiine UI logidest kasutades selleks järgnevus mustrite kaevandamist, et paremini käsitleda logides olevat müra. Kõige olulisem on mõista millised rutiinidest on automatiseeritavad, ning selle mõõtmiseks pakub uurimustöö välja rutiinide automatiseeritavuse indexi (RAI). Meetodi kasutamiseks on vaja aru saada millised rutiinide ülesanded on automatiseeritavad, leides nende vahelisi sõltuvussuhteid, täpsemalt öeldes andme muudatuste ja funktsionaalsuse sõltuvusi. Võrdlus olemasoleva meetodiga, kasutades sünteetilise ja kontrollitud olukorraga andmestikke, näitas et väljapakutud meetod suudab leida vastavaid rutiine, tuvastada sõltuvussuhteid ja mõõta RAI aktsepteeritava teostusajaga. Selleks et leida funktsionaalseid sõltuvusi andmestikus, teostati uurimistöös pakutud meetodit Javaga, lisaks veel integreeritud SPMF mustrite kaevandamise tööristaga, Foofah andmete muundamise tarkvaraga ja Tane algoritmiga.

**Võtmesõnad:**

Protsesside Automatiseerimine Robootikaga, Protsesside Kaevandamine Robootikaga, Mustrite Avastamine, Automatiseeritav Rutiin, Järgnevus Mustrite Kaevandamine

**CERCS: P170. Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria).**

# Table of Contents

# 1 Introduction

The following chapter provides a brief overview of the problem of discovering and automating routines and specifies research questions under the main goal of the thesis. Also, it presents a contribution and structural review of the thesis.

## 1.1 Problem Statement

Robotic Process Automation (RPA) is a powerful technology that represents one of the emerging business process automation forms [1]. RPA improves the efficiency of the business process and increases business quality. The emergence of RPA tools is caused due to the fact that most of the business processes contain a large amount of frequent repetitive non-value adding tasks. An example of such a routine task is data transferring from one information system to another (e.g., transferring finance information from file to web application). The main goal of RPA tools is to remove the human factor from completing repetitive tasks by executing software robots (bots), that mimic user interactions with the application user interface. This approach allows us to perform routine tasks with greater accuracy and reliability. However, RPA tools provide solutions only for task automation, and it does not solve the problem of efficient routines identification.

While routine automation is an important task, the following problems can be associated with the automation process:

- At the moment, routines for automation are identified by means of video recordings and interviews with the assistance of domain knowledge experts. This process is time-consuming and resource-intensive.
- In order to automate a routine, user interactions with a software system should be recorded and collected to the user interaction log (UI log). UI logs, in raw form, consists of a long series of UI events (click events, edit events, etc.) without any indication of the points in the log where the user starts a new instance of some task and where it ends. We call such logs *unsegmented*, as opposed to *segmented* logs where the log is decomposed into traces, each of which corresponds to a clearly scoped execution of a routine. Unsegmented recording of the user interactions produces a log with a large number of records. Hence, an input for automation is too complicated.
- User interaction logs may contain some noise recorded. By noise, we mean (i) one or more events (consecutive or not) that appear in the middle of an instance of a routine but are not part of the routine; or (ii) events that appear before the start of a routine instance or after its completion. These events are characterized by the fact that they do not appear systematically, but are rare.
- Another problem arises from the fact that the implementation of the bots is an error-prone process. Therefore, testing of the bots is a time-consuming procedure.

Since routines observation by video recordings and interviews requires a lot of time, it is much efficient to generate UI logs automatically and analyze such logs in order to determine routines.

Typically, the UI log is presented by a set of sequences of actions performed by a user. Such sequences represent a task execution. Hence, sequential pattern mining techniques [2] can be used to identify frequent repetitive patterns (also known as routines) from such sequences. Sequential pattern mining is an essential topic of data mining that is used in different domains such as telecommunications, marketing, and retail [3]. Furthermore, sequential pattern mining techniques can be used to discover frequent patterns from action sequences that are present in large datasets, such as UI logs.

We say that a routine is *automatable* if every step (action) in the routine can be deterministically executed based on available data (i.e., data produced by previous actions). Figure 1 demonstrates an example of a sequence of actions repetitively performed on a spreadsheet by a user:

[select cell C1 → copy to clipboard → select cell C2 → edit cell C2]

Figure 1. Example of a sequence.

This routine may be repetitive. However, it is not automatable, unless, for every execution of this routine, the value of cell C2 after the last action of this routine can be computed from the value of cell C1 (which is the only data item produced by a previous action). If the value of C2 after the edit is equal to the value of C1, or a sub-string of C1 that can be deterministically computed, then the routine is automatable (e.g., the value of C2 is always equal to cell C1 after removing the first token in the string consisting of alphanumerical characters plus any preceding or subsequent spaces). It should be noted that the notion of "what can be deterministically computed" is defined with respect to a collection of operators, as we will see later in the thesis.

There are several limitations of the sequential pattern mining techniques in the context of discovering automatable routines from UI logs. Most such techniques operate on data in symbolic representation. In consequence, they are not intended to process complex constructs as user interaction actions. Moreover, even if the limitation of processing complex structures is overcome, sequential pattern mining is not able to determine if a discovered frequent pattern is automatable.

Another existing approach [4] that constitutes a thesis baseline allows us to discover automatable routines. However, the main limitation of the approach is the inability to deal with noise presented in the UI log and discovering only fully automatable routines (such that each routine action is automatable).

The main problem to be addressed can be formulated as follow:

***"Given a segmented UI log that may contain noise, we need to discover candidate routines for automation and determine which routine actions are automatable."***

The main research question is:

***How to discover automatable routines from UI logs?***

The other research questions are the following:

1) How to determine if the routine is automatable?
2) How to extract routines from UI log amenable for automation?

## 1.2 Contribution

This thesis proposes an approach to discover candidate routines from a User Interaction log and to evaluate the automatability of these candidate routines. The proposed solution aims to use sequential pattern mining techniques for identifying candidate routines for automation. Moreover, it exploits a combination of algorithms for discovering syntactical and semantical transformations within routine actions in order to identify if they are automatable. Finally, we have conducted qualitative and quantitative evaluations of the proposed solution in order to compare it with the baseline approach.

Chapter 2 presents the basic concepts of RPA and sequential pattern mining techniques. Chapter 3 gives an overview of the state-of-the-art of defined problems and analysis of the existing solutions. Chapter 4 reports on provided approach architecture with a detailed description of the routine identification and automatability determining processes as well as their implementation. Chapter 5 includes the description of the synthetic and controlled-setting datasets that were used in evaluation purposes and the results of quantitative and qualitative evaluations. Lastly, Chapter 6 concludes the thesis.

# 2 Background

This chapter provides an overview of the main terms and concepts about robotic process automation and sequential pattern mining techniques.

## 2.1 Robotic Process Automation

Robotic process automation is a relevantly new technology that can be defined as a set of tools for routine tasks automation that operates on structured data with deterministic outcomes [5]. The automation of repetitive tasks (so-called routines) is achievable by using RPA bots. Such software bots can interact with software applications in order to perform the transcription of user interactions with a software application. There are two types of RPA bots: attended and unattended bots [6].

The attended bots act with the possibility of user interaction. More precisely, the execution of the attended RPA bots can be triggered or terminated by the event that was performed by the user. Also, the attended RPA bots may provide or receive data that was specified by the user during the bot execution. The attended RPA bots are suitable in a situation when it is not possible to automate the entire routine.

The unattended RPA bots are executed without any user intervention. In contrast to attended bots, unattended ones are self-triggered, meaning that they are not waiting for commands from the user, and work in a continuous manner. Unattended RPA bots are suitable for executing fully deterministic routines. The example of such a routine is a process of data extractions from a spreadsheet and entering it to the browser form.

Even though both attended and unattended RPA bots allow us to automate routine tasks, they do not solve the problem of identification of candidate routines for automation. In order to address this problem, we are using Robotic Process Mining (RPM) techniques [5]. RPM is a set of tools for analyzing user interaction records in order to identify candidate routines for automation and discovering routines specifications that can be encoded to RPA executable scripts. The RPM involves three mains steps: collecting and preprocessing UI logs, identifying candidate routines for RPA from the UI log, and discovering executable RPA routines. The thesis focuses on the last two RPM phases: the discovery of candidate routines (from a segmented log) and the assessment of the automatability of a candidate routine.

## 2.2 UI log

The UI log consists of sequences of actions, each representing execution of a user task. We will refer to these sequences as *traces*. If a particular sub-sequence of actions within a trace occurs across multiple traces, these actions constitute a *routine*. Each action represents a user's interaction with various IT systems to perform the underlying task. An example of such interaction is clicking a button, copying a cell in a spreadsheet, etc. Each action is characterized by a trace id (an attribute that maps the action to particular trace), an action type (e.g., copy, edit, open), and a timestamp (an attribute that specifies when the action was executed). Besides the above characteristics, the action contains other details such as the name of the application in which the action was executed, the details of the application

element on which the action was performed (e.g., a field label, a button name, URL, Excel cell address, value of a field, etc.), called *payload*.

Table 1 shows an example of a UI log. The log contains 4 traces. Each action that is presented in the log has 7 attributes. The log describes the execution of the data transferring task from Excel to the browser. Since a sequence that consists of "Copy cell" and "Paste" actions is present in every trace, it represents a routine.

Table 1. Example of UI log.

| Case Id | Timestamp | App | Action Type | Content | Field Id | Field value |
|---|---|---|---|---|---|---|
| 1 | 13:27:50 | Excel | Copy cell | John | A1 | "John" |
| 1 | 13:27:51 | Browser | Paste | John | Name | "" |
| 2 | 13:27:52 | Excel | Copy cell | Liam | A2 | "Liam" |
| 2 | 13:27:53 | Browser | Open New tab | "" | "" | "" |
| 2 | 13:27:53 | Browser | Paste | Liam | Name | "" |
| 3 | 13:27:54 | Excel | Copy cell | James | A3 | "James" |
| 3 | 13:27:55 | Browser | Paste | James | Name | "" |
| 4 | 13:27:56 | Excel | Copy cell | Doe | B1 | "Doe" |
| 4 | 13:27:57 | Browser | Paste | Doe | Surname | "" |

As we have mentioned, the UI log may contain noise actions. Table 1 contains a noise "Open New tab" action in the second trace that appears in the middle of an instance of the determined routine and is not a part of the routine.

## 2.3 Sequential Pattern Mining

As we have mentioned, the second phase of the RPM is the identification of candidate routines from a UI log. Since the UI log consists of ordered sequences of user interaction action, the second phase aims at the identification of repetitive patterns of actions from UI log sequences. In that regard, sequential pattern mining techniques can be used.

Sequential pattern mining techniques solve the problem of discovering interesting subsequences in a set of sequences [7]. Consequently, it is possible to apply those techniques to the UI log in order to address the problem of identification of candidate routines.

Sequential pattern mining is used widely in many areas such as bioinformatics, market analysis, natural language analysis, and web analysis [7]. It concentrates on sequential data analysis and discovers sequential patterns. Specifically, the goal of sequential pattern mining is to identify subsequences in a set of sequences based on defined metrics, for example, on the length of the sequence or the sequence frequency. However, in the original problem of sequential pattern mining, *support* metric is used. The support of a sequence $S$ is the number of sequences from the input database that contain sequence $S$. The output of sequential pattern mining is frequent sequential patterns. More precisely, any pattern consists of a

sequence of symbolic items. The pattern is frequent if its support is higher then the user-specified *minimum support* (i.e., support threshold).

There are many varieties of sequential pattern mining techniques. One of them is contiguous sequence mining. The study in [8] uses this method to find the frequent contiguous sequences from biological data sequences. The algorithm of contiguous sequence mining consists of two steps:

- Find subsequences with indicated length in input sequences to construct a spanning tree. Furthermore, save the starting position of the discovered pattern.
- Filter the found patterns based on a support threshold and join sequences with the same length for the generation of a frequent candidate. Then check if a second candidate position is before the first candidate and, if so – increase the frequency counter. Repeat the step for all candidates for producing another frequent candidate.

However, the described methods tend to be not efficient if the input items contain gaps. For solving this problem, it is possible to use gapped sequence mining [9]. Mining gapped frequent subsequence patterns helps discover sequential patterns that contain items separated by a gap. The study [9] presents the Gap-BIDE Algorithm implementing gapped sequence mining.

Another type of sequential pattern mining is Periodic Pattern Mining [10]. This method is used to discover periodic patterns (frequent sequential patterns that are regularly repeating) from input sequences. Besides the pattern-finding, one of the parts of periodic pattern mining is finding periodicity for the patterns. The paper [10] presents an algorithm for discovering different types of periodic patterns, such as full and partial periodic patterns, perfect and imperfect periodic patterns, synchronous and asynchronous periodic patterns, patterns with sequence and segment periodicity and dense periodic patterns.

Overall, sequential pattern mining is computationally challenging, especially when mining long sequences. Moreover, the resulting output contains too many patterns, especially if a defined support threshold is low [11]. However, methods for mining closed sequential patterns cope with that problem. The mined sequential pattern is *closed* if it does not have subpatterns with the same support [11]. It is possible to reduce the computational time and avoid generation redundant subsequences by mining closed patterns from the UI log. Considering that real-life UI logs can represent large databases and contain long sequences, we exploit the advantages of closed sequential pattern mining.

# 3 Related work

In this section, we present the state-of-the-art on the topic of this thesis and provide a survey of previous research on discovering automatable routines from user interaction logs. Additionally, we analyze the main problems and challenges that arise while applying these techniques for discovering automatable routines. Also, we present an overview of sequential pattern mining methods and the challenges that come into play while applying sequential pattern mining for discovering candidate routines for automation.

One of the essential parts of this study is a literature review; for finding relevant papers, the following web search engines were used:

- Google Scholar
- IEEE Xplore
- Microsoft Academic

The search query was defined starting from the following keywords: “discovering automatable routines”, “automatable tasks”, “sequential pattern mining”, “event logs”, “application logs”, “discovering process”. For searching for relevant material, the following queries were used:

- “discovering automatable routines” OR “discovering automatable tasks” AND “event logs” OR “application logs”.
- “discovering process” OR “discovering tasks” OR “discovering routines”.
- “sequential pattern mining”.

After the analysis of the found works – irrelevant papers were filtered based on the following set of the inclusion criteria:

- Title and article are clear and demonstrate that the study is connected to discovering automatable routines.
- The paper contains an analysis of routines discovering algorithms.

After completing the study filtering, more studies that are relevant and cover the topic were found.

Robotic Process Automation receives increasing attention in recent years [1]. However, the problem of discovering automatable routines from user interaction logs is not well researched. The issue arises from the need to understand user behavior patterns as a sequence of user actions that can be automated. The study in [12] introduces an approach to identify sets of actions that frequently occur together in a text scenario (i.e., a textual description of user actions). This approach also identifies the action topology – the order of user actions – and allows to select the granularity of the identified action sequences. The study describes the Sequential Patterns of User Behaviour System (SPUBS) method for discovering user’s common behaviors from recorded data. SPUBS allows to identify frequent sets, discover action order, relate actions by time, and identify conditions in order to contextualize the sequence. However, textual scenarios can be challenging to interpret as user actions because they require natural language processing to transform the free (unstructured) text into structured data such as user actions suitable for analysis.

Another study [13] is focused on automatically identifying whether a task described in a textual process description is automated. It also uses textual process descriptions as input for discovering automatable tasks, which lead to the same issues described above. The approach described in this paper uses machine learning and natural language processing techniques for identifying whether a task that was described in the textual process description is automated or manual. The approach has several disadvantages. At first, it requires predefined features or documentation that do not always exist for the specific system to process. Also, for classifying the task, the method proposes to use the SVM machine learning model. As a supervised machine learning algorithm, SVM needs a manually labeled training set in order to perform classification. It means that the training set should be compiled not automatically but by domain experts or by using conduction user and stakeholder interviews, which is time-consuming and error-prone. It leads to a situation when a task that was classified as automatable is not such in reality. Furthermore, the authors of the work highlighted several limitations of the discussed approach, such as the inability to guarantee that suitable automation candidates are identified.

Discovering automatable routines from user interaction logs can be considered as a subtopic of process mining [14]. Specifically, [15] provides a review of the technique for the automated discovery of process models from event logs. Automated process discovery methods take as input event logs and produce a control-flow model of a business process. This family of techniques is similar to discovering automatable routines using user interaction logs. The study in [15] provides a review of possible metrics that show how accurate models are. The following metrics are used for evaluating automatically discovered process models:

- *Complexity*. The property that describes how complex the discovered processes model is.
- *Precision*. Ability to not produce other traces (sequences of events related to a single case) by the discovered process model.
- *Generalization*. Ability to produce traces that are not in the log but are similar to traces of the process that produced the log.
- *Fitness*. Ability to produce each trace in the log.

In addition, the study describes two main problems for automated process discovery that are relevant for discovery of automated routines as well:

- Discovery methods produce large and spaghetti-like models – in case of discovering automatable routines, the methods similarly can produce too complicated routines.
- Discovery methods produce models that either poorly fit the event log or over-generalize it.

The paper also provides a systematic review of automated process discovery methods and comparative analysis of the approaches. However, these methods do not take into account data transferring and produce process models with traces that were not present in the log. Overall, it can be used as a good starting point for discovering automatable routines from user interaction logs.

If the study described above deals with discovering models without conditions under which tasks are executed, another recent work on APD [16] provides methods for discovering process models with branching conditions. Nevertheless, this approach is not suitable for discovering automatable routines because it produces models that do not perfectly fit the log. Moreover, some actions within a routine may contain data transformations (e.g., change of the format when transferring values from source application to target application). However, the proposed approach does not consider such transformations, and it is impossible to automate routine without information about data transformations.

Another related topic is Web Usage Mining [17], which is also a sub-topic of data mining and refers to methods for discovering patterns from Web data. In Web Mining, the data is not unified. It comes from different sources and source implementations. Web usage mining is performed in three steps:

- Preprocessing. This step consists of preparing data for pattern discovery. Information in the data sources should be converted to the data abstraction. Such data abstraction is similar to the one that is used for discover-ing automatable routines from interactive user logs. At this stage, noise removal is performed by session filtering in order to generate noise-free input for the pattern discovery stage.
- Pattern discovery. [10] describes six methods that are used for pattern discovery: Statistical Analysis, Association Rules, Clustering, Classification, Sequential Patterns, and Dependency Modeling.
- Pattern analysis. At this stage, the filtering of the discovered patterns is performed. Filtering is done based on irrelevant patterns. Relevancy depends on the application for which Web Mining was done.

Some of the Web Usage Mining techniques, such as pattern discovery approaches, can be used for discovering automatable routines. However, there is not a direct way to use Web Usage Mining techniques in order to discover automatable routines from user interaction logs. In addition, unlike data that is used for Web Usage Mining, user interaction logs can be extracted not only from a Web application but from Desktop applications as well.

UI log mining is another approach that solves the problem of analyzing UI logs produced by a desktop application. One of the software systems that use UI log mining in order to discover and reuse processes that were used to solve tasks is TaskTracer [18]. TaskTracer collects data about user interaction in the application. However, the user should manually specify what task they are doing, so that each recorded action will be related to a particular task. For creating processes, TaskTracer uses a combination of machine learning algorithms, data collection, and information visualization. As a result, UI log mining tools deal with collecting noise-free event data, but they do not allow us to discover routines from the user interaction logs.

Recent work [4] introduces a well-described method to discover automatable routines from UI logs that solves the problem of activation conditions and discovering data transformations. The method consists of three steps:

- Flat-Polygons Detection. At this stage, UI logs are parsed into a deterministic acyclic finite-state automaton, and then the candidate automatable routines (flat-polygons) are extracted.
- Automatable Actions Detection. Each automatable routine candidate produced by the flat-polygon detection step is analyzed on the presence of deterministic actions (actions that are possible to reproduce by RPA tools).
- Routine Specifications Detection. At the last stage, the method extracts maximal sequences of deterministic actions from automatable routine candidates and discover the activation conditions. The output of this step is a set of tuples of activation conditions.

However, the study has the following limitation: it is not able to deal with noise. It is possible to cope with that limitation by using sequential pattern mining techniques.

A study [19] addresses the problem of analyzing UI logs in order to discover automatable routines. The study focuses on discovering routines that correspond to copying data from a source (e.g., spreadsheet) and forwarding it to the target (e.g., web form). Such routines can be interpreted as a process of transformation of one data structure into another. This process can be discovered from a set of input-output examples via data transformation techniques (e.g., Foofah [20]). Since Foofah has limitations that lead to an increase in execution time while discovering data transformations, the proposed approach aims to optimize the Foofah algorithm by grouping examples by target and input structure. Grouping examples by target refers to decomposing the transformation to the source-to-target level in order to identify relations between input and output fields. While grouping examples by target reduces the number of input fields that Foofah needs to consider, grouping by input structure concentrates on reducing the number of transformation examples given as input. The second optimization focuses on grouping the target examples into equivalence classes, where each class represents a different structural pattern. Furthermore, this step is responsible for discovering a data transformation by providing to Foofah one randomly selected transformation example from the group.

Another study [21] proposes to take advantage of the knowledge of the back-office staff. More precisely, it is possible to monitor user actions performed in the information systems in order to record UI log using a series of image-analysis algorithms. Once the UI log is created, the study presents an approach that uses it as an input of a process discovery algorithm that exploits the process mining paradigm [22]. Overall, the study is focused on identifying a part of a business process (e.g., tasks in a process) that may be amenable for automation due to them being repetitive. However, the proposed approach does not seek to discover fully automatable routines nor to generate executable scripts to automate these routines.

# 4 Contribution

In this section, we present a solution for discovering automatable routines from a UI log. The approach consists of two steps: routines identification and routine automatability assessment. The former step discovers candidate routines for automation by applying sequence pattern mining. The latter step analyzes identified candidates on the subject of amenability to automation. Figure 2 describes the high-level architecture of the solution.

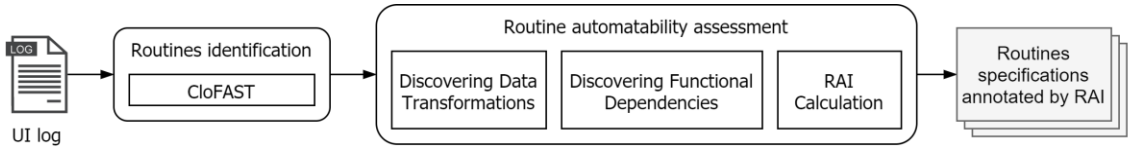


Figure 2. The architecture of solution.

The input of the proposed method is a UI log consisting of a collection of sequences of actions. The architecture of the solution consists of two modules that are executed one after the other. The first module is responsible for the identification of candidate routines for automation. This is achieved by mining frequent patterns from the given set of traces. The second module identifies automatable routines among the discovered candidates. A candidate routine is a (possibly gapped) sequence of actions, without data attributes that occur frequently. This latter module calculates a so-called routine automatability index for each candidate routine. Some candidate routines cannot be automated by a robotic process automation script, because the input parameters of some of the actions in the routine cannot be deterministically computed from the parameters of previous actions. In other words, a candidate routine is deterministic from a control-flow perspective (the sequence of actions is always the same) but not from a data perspective.

The final output of our solution is a set of routine specifications annotated by the routine automatability index. A routine specification consists of a sequence of actions that constitute a routine and a list of dependencies between routine actions.

## 4.1 Routine identification

Sequential pattern mining techniques can solve the problem of identification candidate automatable routines. There are many existing sequence pattern mining algorithms (e.g., PrefixSpan [23], CM-SPAM [24], etc.), most of them work with symbolic sequences. In sequential pattern mining, the input is a set of sequences, and each sequence is an ordered list of transactions, where each transaction is a set of symbolic literals. A symbolic literal represents a notation for a fixed value that is atomic (i.e., it cannot be decomposed). In our case, however, the sequence consists of actions that are complex objects since they are composed of multiple attributes. Therefore, we need to use a symbolic representation of sequence actions as input for sequential pattern mining technique. However, we cannot use all attributes of the action for its symbolic representation since some of the attributes correspond to the data attributes (e.g., content, field value, etc.), which values are unique for each task execution. On the other hand, some attributes contain information about the elements of the applications that were involved in the action execution. Such attributes are called context attributes and are shared among traces that contain the same routine.

Therefore, the symbolic representation of the action is the combination of its type and the values of its context attributes.

If we consider the example of the UI log from Table 1, we can notice that the context attribute *„Field Id“* for the *„Copy cell“* action contains the information about the cell address. More precisely, the column name is a context attribute value for *„Copy cell“* from each trace except the last one. The *„Copy cell“* action from the last trace has a full address as a context. For each *„Paste“* action, the attribute *„Field Id“* is also a context attribute which value equals to *„Name“* for each trace, except the last one. The context attribute of the action *„Paste“* from the last trace has a value that equals to *„Surname“*. Table 2 demonstrates the symbolic representation of each sequence from Table 1.

Table 2. Example of sequences of actions in symbolic representation.

| Sequence Id | Sequence actions |
|---|---|
| 1 | CopyCell+A Paste+Name |
| 2 | CopyCell+A Paste+Name |
| 3 | CopyCell+A Paste+Name |
| 4 | CopyCell+B1 Paste+Surname |

In order to reduce the number of potential routines to be discovered, we mine only closed patterns, and for this purpose, we selected a recently proposed algorithm called CloFAST [11]. A closed frequent sequential pattern is a pattern that is not a part of another pattern that has the same support. Since each subpattern of the frequent pattern is also frequent, mining closed frequent patterns reduce the size of the set of the found patterns and saves computational time. Furthermore, we select the CloFAST algorithm because it computes gapped sequence patterns. A gapped sequence pattern is a pattern that occurs frequently in a set of sequences, but such that the occurrences of this pattern are not necessarily contiguous. For example, if we say that "$ABC$" is a gapped pattern, it means that the occurrences of this pattern are not necessarily all made of the exact sequence "$ABC$". Occurrences of this pattern might look as follows: "$\boldsymbol{A}X\boldsymbol{BC}$", "$\boldsymbol{AB}XY\boldsymbol{C}$", "$\boldsymbol{A}X\boldsymbol{B}Y\boldsymbol{C}$", where $X$ and $Y$ are symbols that do not belong to the pattern. In our proposal, we rely on gapped patterns in order to account for the fact that UI logs may contain noise, as mentioned in Chapter 1. The recent study on the CloFAST algorithm [11] has shown that it outperforms other state-of-the-art algorithms, especially when mining long sequences. The main idea of the CloFAST algorithm is that it creates sparse id-lists that store the position of the actions, and vertical id-lists that store the position of a sequential pattern in the input sequence. The algorithm uses sparse id-lists for mining closed frequent itemsets that are used for construction closed sequence enumeration tree that enumerates the complete search of closed sequences.

The usage of algorithms for mining closed frequent sequences reduces the number of discovered patterns. However, the disadvantage of gapped sequential pattern mining algorithms such as CloFAST is that they may discover a large number of patterns, due to the fact that they allow arbitrary symbols to occur in the middle of a pattern. Many of the

discovered patterns may be irrelevant. For example, CloFAST may discover a pattern "$ABC$" even if the sequence of symbols "$ABC$" never occurs contiguously in any of the sequences. For example, if we consider a set of sequences: "$\boldsymbol{A}XY\boldsymbol{B}XY\boldsymbol{C}$", "$\boldsymbol{A}YX\boldsymbol{B}YX\boldsymbol{C}XY$", "$\boldsymbol{A}XX\boldsymbol{B}YY\boldsymbol{C}XY$", CloFAST is likely to discover the pattern "$ABC$", even though this pattern does not occur contiguously. We, therefore, need a way to determine which of the discovered patterns are likely to be candidate routines for automation. Intuitively, a pattern is a candidate routine if it occurs frequently, and the occurrences do not have too many gaps (i.e., the "noise" is infrequent). To capture this intuition, we use a cohesion metric to rank the patterns discovered by CloFAST. The cohesion metric for patterns is high when the symbols in the pattern occur contiguously in the majority of cases. Our solution applies the ranking approach that is based on the membership-based cohesion score [25]. In order to formulate the definition of the membership-cohesion score, we need to explain the concept of a minimum outlier based maximum occurrence window. For that, we will be using the following definitions:

**Definition 1.** Given a sequence $S$ and pattern $P \in S$, the occurrence window of $P$ denoted as $W_{P,S}$ is an interval $[i,k]$ within $S$ such that $S_i, S_{i+1}, .., S_k$ contains $P$.

**Definition 2.** The outlier is the action from an occurrence window $W_{P,S} \in S$ that does not belong to a pattern $P$ or which order in the window $W_{P,S}$ is not the same as in the pattern $P$.

**Definition 3.** The outlier based minimum occurrence window ${W_{P,S}}^{O}$ is an occurrence window that contains a minimum number of outliers and can be calculated as follows:

$$ {W_{P,S}}^{O} = \underset{W_{P,S}}{\operatorname{argmin}} O(W_{P,S}) \tag{1} $$

where $O$ is a function that returns the number of outliers in the occurrence window $W_{P,S}$.

**Definition 4.** A minimum outlier based maximum occurrence window $W$ is an occurrence window that contains maximum elements of the pattern $P$ while minimizing the number of outliers inside the window and can be calculated as follow:

$$ W = \underset{O(W_{P,S})}{\operatorname{argmax}} L(\underset{W_{P,S}}{\operatorname{argmin}} O(W_{P,S})) \tag{2} $$

where $L$ is a function that returns the length of the outlier based minimum occurrence window.

The pattern cohesion score $\lambda_P$ is a metric that can be calculated as a signed difference between the pattern length (the number of elements in the pattern) and the median number of outliers in the pattern minimum outlier based maximum occurrence windows:

$$ \lambda_P = |P| - M\left(\bigcup_{i=0}^{N} O(W_i)\right) \tag{3} $$

where $|P|$ denotes the length of the pattern $P$, $M$ is a function that returns the median value, $O$ is a function that returns the number of outliers, $W_i$ is a minimum outlier based maximum occurrence window from the sequence $S_i$, $i = [0..N]$, $N$ is a total number of sequences.

Figure 3 demonstrates an example of cohesion score calculation. Overall, the sequence database consists of 3 sequences, and the pattern consists of 3 actions.

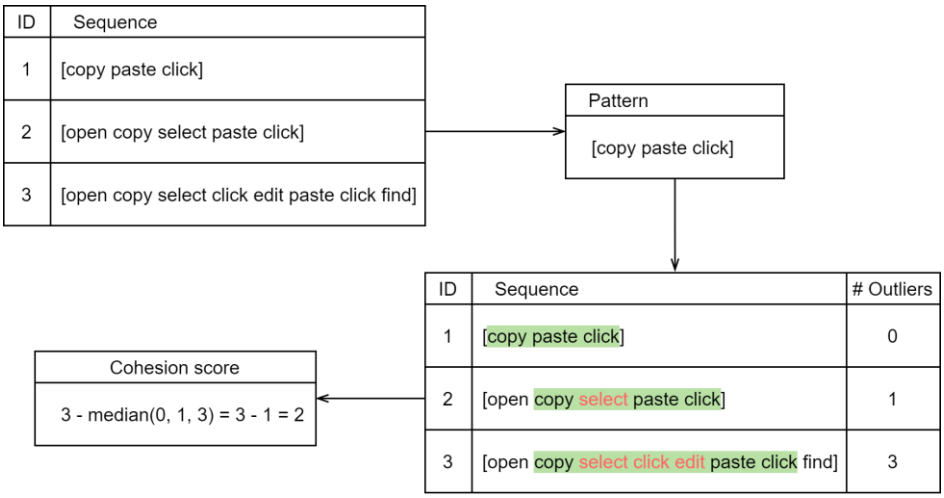


Figure 3. Cohesion score calculation example.

The first sequence minimum outlier based maximum occurrence window does not contain any outliers. However, we can observe 1 outlier (*"select"* action) in the case of the second sequence and 3 outliers in the case of the third sequence. The median outliers count equals 1, and the length of the pattern is 3, which means that the cohesion score of the specified pattern is 2.

Once we have found a cohesion score for each of the patterns, we can rank them by the value of the cohesion score in descending order. To select the most valuable patterns, we use a *cut-off score threshold*. The selection of patterns, in this case, is based on the drop-down value of the cohesion score. The *drop-down* value $\Delta$ is a percentage difference between the cohesion scores of the pattern with the highest cohesion score (the top pattern) and any pattern that is arranged after the top one in a sorted list of patterns. It can be calculated as follow:

$$\Delta = 100\% - (\lambda_P \times 100 \div \lambda_0) \quad (4)$$

where $\lambda_P$ is a value of a cohesion score for the given pattern, and $\lambda_0$ is a value of the cohesion score for the top pattern.

The cut-off score threshold is a minimum value of a drop-down interest. The main idea of using a threshold for the drop-down is filtering those patterns whose cohesion score is too low in comparison to the cohesion score of the top pattern.

Table 3 demonstrates discovered patterns in the descending order of their cohesion score. If a cut-off threshold is set to 15%, the last three patterns will be removed.

Table 3. Patterns cohesion scores example.

| Pattern Index | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Cohesion score | 30 | 29 | 26 | 5 | 6 | 2 |

Figure 4 shows the cohesion scores over pattern indices and a specified cohesion score threshold as a red line. Each pattern that is located lower than a red line will be cut off from the resulting list of patterns. The first pattern that is the top pattern has the highest cohesion score equals to 30, which means that if a cohesion score of any pattern that is arranged after the top one is less then $30 - 30 \times 0.15 = 25.5$, the pattern will be cut off from the resulting list of filtered patterns.

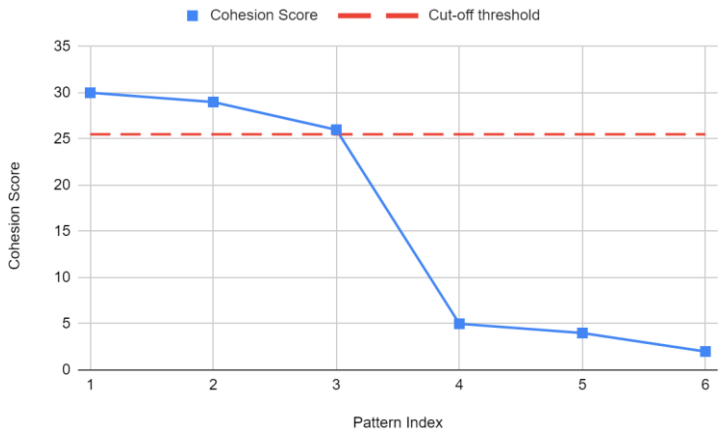


Figure 4. Cut-off cohesion score threshold.

The output of the routines identification module is a set of closed frequent sequential patterns that were mined from the UI log. The set of patterns is sorted by the cohesion score value in descending order and does not contain patterns that were cut off based on the specified value of the threshold. Each pattern represents a candidate automatable routine and is used as an input for the routine automatability assessment module.

## 4.2 Routine automatability assessment

The second module of the solution is responsible for discovering dependencies between pattern actions in order to determine if the pattern is automatable. A pattern can be fully automatable and partially automatable. The pattern is fully automatable if each action of the pattern is automatable. Otherwise, it is partially automatable. The pattern action is automatable if it is possible to deterministically compute attribute values of the action. That means that the action cannot be automated if the action attribute values are not constant and do not depend on the data that captures previously executed actions. Whether the action is automatable or not can be distinguished by analyzing a function that defines the values of the action attributes on deterministic behavior. We can examine if such a function is deterministic by analyzing dependencies within pattern actions. It should be pointed out that we consider two

types of dependencies within pattern actions: syntactical transformations (i.e., data transformations) that describe the operations over strings (e.g., split a string by a delimiter, merge two strings, etc.) and semantical transformations (i.e., functional dependencies) that describe the dependencies between attributes that uniquely determine the value of other attributes. The proposed approach determines transformations consistently, starting with identifying syntactical transformations and continue for semantical transformations.

The first submodule of the automatable assessment module refers to discovering data transformations. For that, we are using the data transformation-by-example discovery technique called Foofah [20]. Given a set of input-output pairs that describe the data in source and target formats, Foofah identifies data transformations that have to be performed to convert the data to the required target format.

Although Foofah allows us to discover complex data transformations, it is computationally inefficient. For overcoming that limitation, we are using an approach that was described in a recent study [19] that proposes two optimizations that take advantage of the information in the UI log. The approach aims to optimize the Foofah algorithm by grouping examples by target and input structure. Besides the baseline approach that proposes to analyze data transformation for each action that was executed before a given one, an optimization such as grouping examples by target allows us to analyze data transformations only for pairs of actions that consist of write and read actions. Grouping examples by input structure aims to reduce the number of transformation examples given as input by applying the tokenization technique.

In order to discover data transformations within pattern actions, we need to extract a list of pairs of write and read pattern actions (i.e., read-write pairs). The *read action* can be determined as an action that captures the data (e.g., copy or copy cell actions). The *write action* can be determined as an action that populates data (e.g., edit field or edit cell actions). We have considered pairs of reading and writing actions as a subject for the data transformations analysis, assuming that each non-writing action can be automated since the values of such action attributes are deterministic. The writing action cannot be automated if the data populated by that action does not depend on the data that captures previously executed actions (i.e., there are no data transformations for writing action). The read-write pair of actions consists of the reading action that specifies a source of the data (where data came from) and the writing action that specifies the target of the data (where data came to).

Figure 5 demonstrates the process of obtaining read-write pairs from the pattern in a symbolic representation. For each writing action, we found corresponding nearest read action and form a read-write pair.

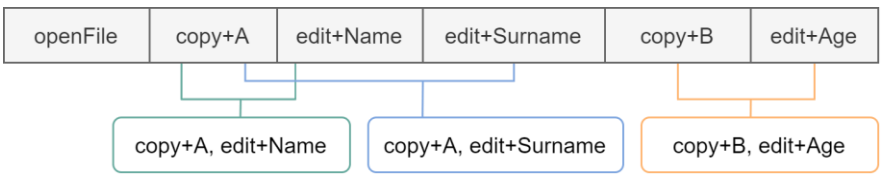


Figure 5. Example of extraction read-write pairs.

Once we have formed read-write pairs, we can analyze the actions within pairs on the presence of any syntactical transformations between write actions and corresponding read action. More precisely, we execute the Foofah algorithm. However, in optimization purposes, we need to group each writing action data attributes values into equivalence classes, where each class represents a different structural pattern of the input data [19]. After that, we can execute the Foofah algorithm for one randomly selected read-write pair per one structural pattern. The Foofah input list contains the data that corresponds to the value of specified data attributes of the read action. The Foofah output list contains the data that corresponds to the values of the data attributes of the writing action. The output of the data transformation identification submodule is a set of patterns and read-write pattern pairs that contain a non-empty list of data transformations.

It may be the case that for some write actions, that no data transformations have been identified. Such write actions are then given to the next submodule, which aims to identify whether they can be automated by discovering semantical transformations (i.e., functional dependencies).

A functional dependency is a relationship between attributes that uniquely determines the value of the attributes by the values of other attributes [26]. For discovering such relationships, we have chosen to use an efficient algorithm named Tane, since experimental results demonstrate that the algorithm is efficient on the large datasets [26]. Tane can discover functional dependencies by using a small-to-large searching strategy with excluding non-trivial dependencies (self-dependency) and pruning the search space. An example of the functional dependency is a dependency between the country name and the country ZIP code.

By executing the Tane algorithm using a database that consists of data attributes values of each action of the pattern, we will acquire a list of functional dependencies. More precisely, the list of data attributes that uniquely determine the values of attributes of the previously executed actions. If any writing action from the pattern does not have data transformations, we can identify if there are data attribute values of the pattern actions that uniquely determine the value of data attributes of such writing action. If a writing action has functional dependencies on the previously executed actions, the writing action can be automatable. Otherwise, the action is non-automatable.

Table 4 demonstrates an example of the UI log that contains writing actions whose value of the data attribute *"Content"* cannot be determined by data transformations since there is no way to transform the name of the county to the dealing code.

Table 4. UI log for functional dependencies example.

| Case Id | App | Action Type | Content | Field Id | Field value |
|---|---|---|---|---|---|
| 1 | Excel | Copy cell | Estonia | A1 | "Estonia" |
| 1 | Browser | Edit | Estonia | Country | "" |
| 1 | Browser | Edit | +372 | Dialing code | "" |
| 2 | Excel | Copy cell | USA | A2 | "USA" |
| 2 | Browser | Edit | USA | Country | "" |
| 2 | Browser | Edit | +1 | Dialing code | "" |
| 3 | Excel | Copy cell | Japan | A3 | "Japan" |
| 3 | Browser | Edit | Japan | Country | "" |
| 3 | Browser | Edit | +81 | Dialing code | "" |

However, the Tane algorithm will find a functional dependency between the *"Content"* attribute of the action *"Copy"* and *"Edit"* (that corresponds to editing country name field) and the *"Content"* attribute of the action *"Edit"* (that corresponds to editing dialing code field). Such functional dependencies are demonstrated in Figure 6.

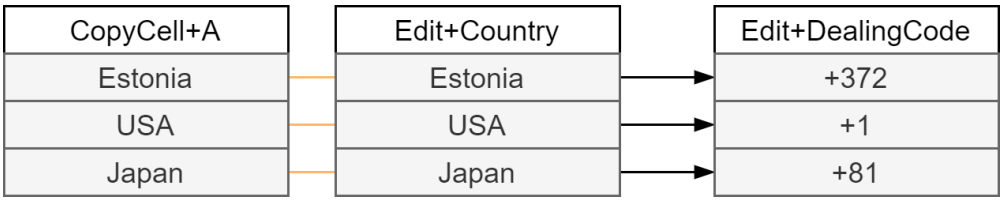


Figure 6. Functioal dndencies example.

The output of the functional dependencies submodule is a set of patterns and pattern writing actions that have no data transformations but have functional dependencies on previously executed actions. More precisely, the output of this submodule contains information about writing action attribute values that can be uniquely determined by the attribute values of actions that were executed earlier in the pattern.

The last submodule of the automatability assessment is responsible for routine automatability index calculation. RAI is a metric that determines the degree of pattern automatability. RAI can be calculated as follow:

$$RAI = \frac{|P|_A}{|P|} \quad (5)$$

where $|P|_A$ is the number of automatable actions in a pattern, and $|P|$ is the total number of pattern actions. Since the number of automatable patterns lies in the interval from 0 to $|P|$, the RAI value lies in the interval $[0, 1]$. If the RAI of a pattern equals 1, it means that the pattern is fully automatable. If the RAI of a pattern equals 0, it means that the pattern is non-automatable.

The output of the routine automatability assessment module is a set of routine specifications annotated with the value of RAI. The routine specification consists of the pattern mined at the routine identification step, the list of read-write pattern pairs and corresponding data transformations and the list of writing actions with a list of actions that uniquely determine the value of the writing action attribute values and the map of the writing attribute values and corresponding dependee actions' attribute values.

## 4.3 Approach implementation

### 4.3.1 Routine identification

In order to perform sequential pattern mining algorithms on the UI log, we need to parse the log file and transform each sequence of the file to the symbolic representation. For such a transformation, we need to determine a context for each action. For that, we group UI log actions by the action type and determine the number of unique values for each attribute from the payload. If the ratio between the number of unique values and the total number of actions in the group is higher then 0 and less than a specified threshold and the attribute from the payload is not a data attribute, the attribute will be defined as a context attribute.

Once we have converted the UI log sequences into symbolic representation, we can perform sequential pattern mining. For that, we have used the CloFAST algorithm [11], which implementation was provided by SPMF[1] – an open-source data mining Java library for discovering patterns from the datasets. The SPMF implementation of the CloFAST algorithm takes as input the value for the minimum pattern support and a file that represents a sequence database. The SPMF database should consist of a set of sequences – lists of the itemsets. Since itemset is an unordered set of items and the UI log sequences consist of actions, our solution specifies itemset as a set of a single action.

The implementation of the CloFAST algorithm provided by the SPMF library requires a specific sequence database format. Each line of a database should represent a single sequence, while its elements should have a format of positive integer numbers. The value "-1" flags the end of the itemsets. In our case, the value "-1" is used for delimiting actions. The value "-2" indicates the end of a sequence. For operation on string data, we are using SPMF library aliases: each item of a sequence has positive number alias. Since the action name and the context is enough for distinguishing the pattern item, we have defined SPMF items as a string literals that contain information about these two action attributes. Each line of the SPMF file represents a single UI log trace.

[1] https://www.philippe-fournier-viger.com/spmf/

Once we generated a file that contains log sequences in the format of the SPMF dataset, we can run the CloFAST algorithm in order to get a file with frequent closed patterns. Each mined pattern has support no lower than the specified minimum support value. The output of the SPMF CloFAST is a file that contains closed sequential patterns and the information about the pattern support in an SPMF format.

The result of running the SPMF CloFAST algorithm on the formatted UI log is a file that contains frequent closed sequential patterns. Each line of the output file represents a pattern. Besides the elements of the patterns, the output file contains information about pattern support.

In order to collect the information about the mined patterns that are presented in the output file, we should parse each line of the file and transform string representation of pattern actions to the set of patterns with specified pattern support.

Figure 7 demonstrates an example of the execution of the routine identification module. More precisely, it presents the UI log input, the middleware SPMF CloFAST output, and the output of the algorithm. For the current example, we have set a minimum support value to 30%. As we can see from Figure 7, the UI log contains 4 traces, and each trace contains 3 user interaction actions.

Overall, the UI log contains two routines that describe copying the value of the name or surname, editing the corresponding field and clicking the corresponding button:

1. copyCell+A; editField+Name; click+SubmitName.
2. copyCell+B; editField+Surname; click+SubmitSurname.

Since a minimum support value was set to 30% and a second routine has 25% support value, the SPMF CloFAST has mined the first pattern only that has support value equals to 75%.

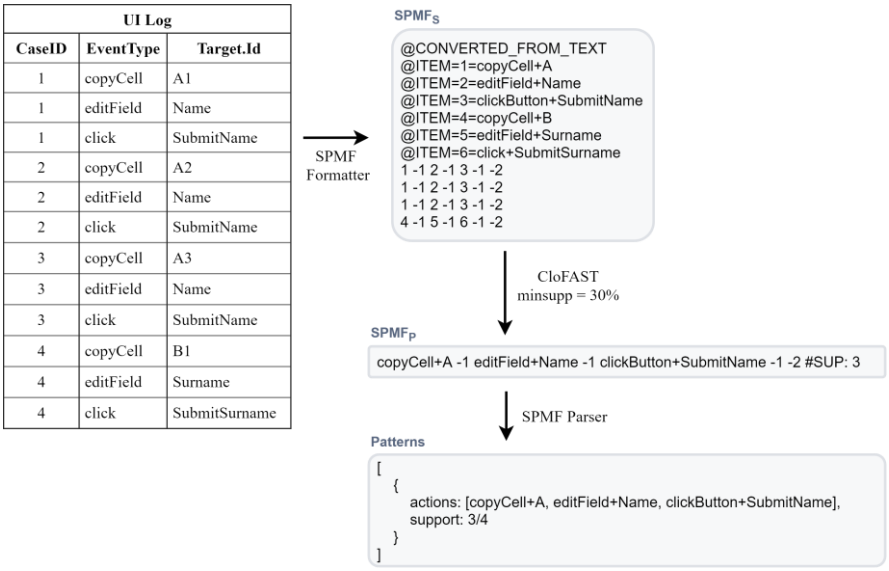


Figure 7. Routine identification workflow example.

Once we have mined closed frequent sequential patterns, we can calculate a membership-based cohesion score for each pattern. For that, we iterate over the list of UI log sequences in symbolic representation and extract minimum outliers based maximum length occurrence

window for each sequence in the UI log. After iteration over all sequences in the log, it is possible to calculate the pattern cohesion score. Given that we have calculated a cohesion score for each pattern, we can sort the list of patterns by the cohesion score value in descending order and cut-off the least valuable patterns. Since we have sorted the list of patterns by cohesion score in descending order, the first element of the pattern list is a top pattern, which means that this pattern is the most valuable. The first step of the cutting off process consists of iterating over the list of patterns skipping the first element of the list, and calculation of the drop-down value. If the drop-down value is higher than a specified cut-off threshold, it means that the difference of the cohesion scores is too high, and the pattern should be cut off from the resulting population of patterns.

### 4.3.2 Routine automatability assessment

For determining routine automatability, we execute two algorithms consistently. The first algorithm identifies data transformations for each read-write pair of a pattern, while the second algorithm identifies functional dependencies for writing actions that do not have data transformations. However, before executing the algorithm for data transformation identification, we need to extract the read-write pair from each pattern. For that, we execute Algorithm 1. In order to identify if the action is writing or reading one, we need to specify two lists. The first list contains action names that can be used for mapping the action to the group of reading actions. Similarly to the first list, the second one contains the action names for mapping the writing actions.

The input of the algorithm is a pattern $P$, while the output is a list of read-write pairs $\rho$ for the specified pattern. For extracting pairs of reading and writing actions, we iterate over the pattern actions (line 2), and if the action $a$ is reading one, we create a new pair of actions $\pi$ and set the iterable action $a$ as a reading action (lines 4- 6). If the iterable action $a$ is writing, we call the function $getLastPair$ that returns the last created pair $\pi$, and then we set the iterable action $a$ as a writing one (lines 7- 10). If the assembled pair is non-empty (i.e., the read and write actions are not empty), we add the pair $\pi$ to the list of pattern pairs (line 11).

**Algorithm 1:** Read-write pairs extraction

```
   input : Pattern P
   output : Read-write pairs ρ
1  ρ ← ∅;
2  foreach Action a ∈ P do
3      Pair π ← ∅;
4      if a is reading action then
5          Pair π ← ⟨R = a, W = ∅⟩;
6      end
7      if a is writing action then
8          Pair π ← getLastPair();
9          Pair π ← ⟨R = R, W = a⟩;
10     end
11     if π ≠ ∅ then ρ ← ρ ∪ {π};
12 end
13 return ρ;
```

Algorithm 2 describes the workflow for data transformation identification. The algorithm input is a set of patterns $P$ and a list of traces $C$ from the UI log. We iterate over a set of pattern and for each pattern $p_i$ we extract a list of read-write pairs by execution Algorithm 1 and collecting all pattern's read-write pairs to the list $\rho$ (line 4).

When we have extracted all read-write pairs of actions from the pattern, we need to collect all values from the traces for reading and writing actions from the pairs. For that, we iterate over the extracted pattern pairs $\rho$, and for each pair, we iterate over the traces $C$ (lines 5- 7). If the iterable trace contains the iterable pattern, we extract the values for the reading and writing actions in the pair from the trace to the variable $\tau$ and add the pair and corresponding values to the list $T$ (lines 8- 10). Once we end iterating over the traces, we can add the read-write pair $\langle R, W \rangle$ and the list of values T collected from each trace $c_j$ to the map $\Theta$ (line 13).

Finally, we can analyze if each pair of each pattern contains data transformations between reading and writing actions. First, for each read-write pair and its actions values, we need to group writing action values into equivalence classes, where each class represents a different structural pattern of the input data [19] (line 16). After that, we can execute Foofah algorithm for one randomly selected input-output example from the list $T_{tokenized}$ (line 17) and assigne the result of Foofah execution to the variable $\phi$. Variable $\phi$ represents found data transformation for the read-write pair $\langle R, W \rangle$.

**Algorithm 2:** Data transformations identification

```
    input : Patterns P, Traces C
    output : Set Φ = (p_i, ⟨⟨R, W⟩,ϕ⟩), where p_i ∈ P
1   Φ ← ∅;
2   foreach Pattern p_i ∈ P do
3       Θ ← ∅;
4       ρ ← extractReadWritePairs(p_i);
5       foreach Pair ⟨R, W⟩ ∈ ρ do
6           T ← ∅;
7           foreach Trace c_j ∈ C then
8               if p_i ⊂ c_j then
9                   τ = (⟨R, W⟩, ⟨R_value, W_value⟩) ← extractValues(⟨R, W⟩, c_j);
10                  T ← T ∪ {τ};
11              end
12          end
13          Θ ← Θ ∪ {⟨R, W⟩, T};
14      end
15      foreach ⟨R, W⟩, T ∈ Θ do
16          T_tokenized ← clusterByRegexPattern(T);
17          ϕ ← getFoofahTransformations(T_tokenized);
18          if ϕ ≠ ∅ then Φ ← Φ ∪ {p_i, {⟨R, W⟩,ϕ}};
19          end
20      end
21  end
22  return Φ;
```

If the transformation were found, we would add them to the resulting list Φ that map the pattern, pattern's read-write pairs, and data transformations for each of the pairs. If Foofah does not discover any transformations, the list $\phi$ will not contain any records, and we will not add the read-write pair to the resulting list (line 18).

The next step focuses on the functional dependencies identification within patterns actions. The identification is performed by applying Algorithm 3. The algorithm should be executed after the data transformations identification since it receives the list Ψ that contains writing actions from read-write pairs that do not contains any data transformations. Besides the list Ψ, the algorithm receives a set of patterns $P$, and a list of traces $C$ as an input. First, we iterate over the list of mined patterns and for each pattern $p_i$ we extract all functional dependencies that were found for the pattern actions attributes. More precisely, we create a file, each line of which represents values of a specified action attributes and execute the Tane algorithm that takes the file as an input (line 4). The input of the Tane algorithm is a database of the attributes in a format of comma-separated records without extra whitespaces.

Once we have found all functional dependencies within the pattern $p_i$ we can analyze functional dependencies for writing actions from read-write pairs of actions with empty data transformations (line 6). Iterating over the list Ψ that contains such writing actions, we need to extract all dependee actions for iterable writing action $W$ (lines 6- 7). Dependee actions are stored into the set of pattern actions dependencies ($P_{FD}$) and have a relationship with an iterable writing action $W$. The action $W$ can be considered as a depender action for the dependees.

Once we have assembled a depender and dependees pattern actions, we need to collect all values of attributes of the dependee actions and a depender action. For that, we iterate over each trace $c_j$ from a list of traces $C$ and extract values for attributes of the depender action and list of values for attributes of dependee values using $extractDepenerValue$ and $extractDependeesValues$ functions respectively (lines 10- 11). After values extraction, we can create a map $V$ that contains depender action and its values and corresponding dependee actions and their values (line 12).

The algorithm returns a set of pairs $FD$ that contain the pattern $p_i$ and the map $V$ that contains functional dependencies for the writing actions from the read-write pairs that contain empty data transformations (line 17).

**Algorithm 3:** Functional dependencies identification

```
    input : Patterns P, List Ψ, Traces C
    output : Set FD = (p_i, V), where p_i ∈ P
1   FD ← ∅;
2   foreach Pattern p_i ∈ P do
3       P_FD = ⟨Depender, Dependees⟩;
4       P_FD ← getFunctionalDependencies(p_i, C);
5       δ_FD ← ∅;
6       foreach W ∈ Ψ do
7           Dependees_W ← Dependees ∈ P_FD such that Depender = W;
8           V ← ∅;
9           foreach Trace c_j ∈ C do
10              v_depender ← extractDepenerValue(W, c_j);
11              v_dependees ← extractDependeesValues(Dependees_W, c_j);
12              V ← V ∪ {⟨W, v_depender⟩, ⟨Dependees_W, v_dependees⟩};
13          end
14      end
15      FD ← FD ∪ {⟨p_i, V⟩};
16  end
17  return FD;
```

For identification pattern automatability, we need to analyze if each pattern's action is automatable. As we have mentioned, we assume that each action is automatable except writing actions that were collected to the pairs of reading and corresponding writing actions. By execution Algorithm 4 that uses as input the outputs of Algorithm 2 and Algorithm 3, we can acquire routine specifications annotated with the value of the RAI metric. The input of the algorithm is a set of patterns $P$, a set $\Phi$ obtained from Algorithm 4 execution, a set $FD$ obtained from Algorithm 4 execution. By iteration over the patterns $P$ we extract all writing actions from the pattern $p_i$ to the variable $\Omega$ (line 3). Then we extract writing actions that occur in data transformations as an action from the read-write pairs or which occur in the functional dependencies as a depender action from the list $\Omega$ (lines 4- 9).

Finally, if we calculate RAI. If the length of the list Ω equals 0, the pattern does not contain any non-automatable actions. Consequently, the pattern is fully automatable and RAI equals to 1. If some writing actions do not have any dependencies, they cannot be automatable, and the RAI value decreases below 1.

**Algorithm 4:** RAI calculation

**input :** Patterns $P$, Set $\Phi = (p_i, \langle\langle R, W\rangle, \phi\rangle)$**,** Set $FD = (p_i, V)$
**output :** Routine specifications $RS = (p_i, \Phi, FD, RAI)$
1 $RS \leftarrow \emptyset$;
2 **foreach** Pattern $p_i \in P$ **do**
3 List $\Omega \leftarrow extractWritingActions(p_i)$;
4 **foreach** $\langle R, W\rangle, \phi \in \Phi$ **do**
5 $\Omega \leftarrow \Omega \backslash W$;
6 **end**
7 **foreach** $V = \langle W, v_{depender}\rangle, \langle Dependees_W, v_{dependees}\rangle \in FD$ **do**
8 $\Omega \leftarrow \Omega \backslash W$;
9 **end**
10 $RAI = 1 - Length(\Omega)/Length(p_i)$;
11 $RS \leftarrow RS \cup \{p_i, \langle\langle R, W\rangle, \phi\rangle \in \Phi, V \in FD, RAI\}$;
12 **end**
13 **return** $RS$;

The output of Algorithm 4 is a set of routine specifications. Each routine specification consists of the discovered pattern, a set of read-write pairs with corresponding data transformations, and a set of functional dependencies for the writing actions. Moreover, each routine specification is annotated by the RAI value.

# 5 Evaluation

This chapter reports on an empirical evaluation of the proposed technique aimed at answering the following research questions:

- RQ1: How accurate our technique rediscovers known patterns from an event log, both with and without noise?
- RQ2: How computationally efficient is the technique?
- RQ3: What is the effect of the support threshold on the quality of discovered patterns and efficiency?

The proposed approach for discovering automatable routines from user interaction logs has been implemented in Java as a command-line application[2].

Below we present the datasets description, experimental setup, and evaluation results.

## 5.1 Datasets

In order to empirically evaluate our approach, we use a collection of both synthetic and controlled-setting log datasets. Generally, we consider 9 synthetic[3] and 3 controlled-setting[4] datasets.

The synthetic UI logs were generated from Coloured Petri Nets for the baseline approach evaluation to correctly discover the automatable routines in the recorded UI log [15]. Synthetic datasets emulate the process of user interaction with web pages and files for data transferring. None of the synthetic logs contains noise. For compatibility with our approach implementation, we have converted the mentioned logs from XML format to CSV with the addition of auxiliary actions necessary for the log processing.

By contrast to synthetic datasets, controlled-setting datasets represent user interaction logs that correspond to real-life scenarios with noise. However, we have recorded controlled-setting datasets in the laboratory environment with artificial noise generation. Therefore, we should consider controlled-setting logs as a pseudo-real. For recording such controlled-setting UI logs, we have used the Action Logger tool [27], which offers relevant actions' granularity level. In total, we have recorded 3 logs that describe the same process of transferring university students' data from an Excel file to a web form. Every routine that occurs in the first log can be fully automated. The second log captures a routine that is not fully automatable. The routines described in the third log are fully automatable, but they contain noise (1% of the total number of actions in the log).

Both synthetic and controlled-setting logs contain the various number of routines that have different lengths and complexity. For evaluating synthetic datasets, we have used models that are a basis for logs generation as the ground truth. Controlled-setting logs contain predefined routines, which makes it possible to assess the quality of the discovered patterns.

[2] The software is available at https://github.com/stdevi/CohesionBasedRoutineDiscovery/tree/develop

[3] The CPNs and synthetic logs are available at https://doi.org/10.6084/m9.figshare.7850918.v1

[4] The controlled-setting logs are available at https://doi.org/10.6084/m9.figshare.12307631.v2

The properties of all the datasets used in the evaluation are reported in Table 5. More precisely, for each log, we have reported the number of unique actions (actions that are determined by action name and position in the routine), the number of routine unique variants, the number of traces, total number of actions, and average number of actions per trace.

Table 5. Properties of the datasets.

| ID | Name | # Unique actions | #Routines | #Traces | #Actions | #Action per trace (Avg.) |
|---|---|---|---|---|---|---|
| 1 | Log1 | 14 | 1 | 100 | 160 | 1.6 |
| 2 | Log2 | 19 | 3 | 1000 | 16804 | 16.8 |
| 3 | Log3 | 19 | 7 | 1000 | 15898 | 15.9 |
| 4 | Log4 | 17 | 4 | 100 | 1600 | 16.0 |
| 5 | Log5 | 16 | 36 | 1000 | 8775 | 8.78 |
| 6 | Log6 | 14 | 2 | 1000 | 9998 | 10.0 |
| 7 | Log7 | 19 | 14 | 1500 | 15838 | 10.56 |
| 8 | Log8 | 20 | 15 | 1500 | 18809 | 12.54 |
| 9 | Log9 | 32 | 38 | 2000 | 29818 | 14.91 |
| 10 | Log10 | 22 | 2 | 50 | 1080 | 21.6 |
| 11 | Log11 | 22 | 2 | 50 | 1080 | 21.6 |
| 12 | Log12 | 22 | 2 | 50 | 1090 | 21.8 |

## 5.2 Experimental setup

The implemented approach was executed on a PC with Intel Core i5-6200U@2.4GHz CPU with 8GB RAM, running Windows 10 and JVM 11.

In order to evaluate our approach and compare obtained results, we have chosen the most recent study [15] on discovering automatable routines from UI logs as a baseline. The study has introduced the approach that is divided into three steps.

The first step is flat-polygons (the candidate automatable routines) detection from the UI log. Each flat-polygon is obtained by analyzing each log trace on shared prefixes and suffixes. More precisely, if multiple traces share the same prefix, this prefix will be distinguished as an independent sequence or flat-polygon. The same rule applies to shared suffixes.

The second step of the baseline aims to determine if each action of the flat-polygons is deterministic (can be automatable) or not. It is possible to determine if the action is deterministic by analyzing if all action parameters' values using a constant or deterministic function. The action parameter using constant function if all parameter values are immutable. For discovering deterministic function, the approach implies two methods: looking for value-to value dependencies and apply the Foofah tool for finding data transformations. Despite the improved Foofah transformation discovery approach that was used by us, the baseline is used Foofah for each parameter of the chosen action, and each parameter of the

action encountered previously. Such Foofah usage affects the performance of the baseline approach in a negative way.

In the last step of the baseline, all non-deterministic actions are removed from the flat-polygons, and removed actions split each fat-polygon into multiple automatable routines. Once the automatable routine was discovered, the activation condition for triggering the routine should be detected using the JRipper tool. The goal of the JRipper tool is to find a set of rules (activation conditions) for the first action of each flat-polygon.

The comparison of the baseline and our approach has been made in terms of three metric groups. The measures from the first group describe qualitative measures, more precisely, the number of discovered patterns, the maximum length of the patterns, the average length of the patterns, total coverage, and the average routine automatability index. The second group contains characteristics of the discovered patterns, more precisely, the average precision, recall, and f-score. The last group describes computational efficiency metric – execution time.

*The number of discovered patterns* shows how many patterns we have found using both approaches. The number of discovered baseline patterns corresponds to the number of automatable sequences that were found at the last stage of the approach. Such sequences are represented by tuples of activation conditions and a sequence of actions.

*The maximum length of the patterns* is a measure that compares the ability of each approach for discovering long patterns. As was mentioned earlier, a baseline divides each automatable routine candidate according to shared prefix, suffix, and splits it by non-automatable actions. Consequently, each split decreases the maximum length of the pattern.

*The average length of the patterns* indicates a central tendency of the discovered patterns lengths.

*The total coverage* specifies how much behavior captured in the log can be described by the discovered patterns. It is measured in terms of a ratio of actions presented in the patterns. The total coverage is a measure in the [0, 1] interval, where 0 means that discovered patterns actions are not presented in the log, while one means that discovered patterns match all routine unique variants. The total coverage is a cumulative measure since it is made up of accumulated patterns coverages, and multiple patterns cannot cover the same action in the log. The total coverage (TC) can be calculated as:

$$TC = C(P_1) + C(P_1 \oplus P_2) + \cdots + C(P_1 \oplus \ldots \oplus P_n) = \sum_{i=1}^{N} C(\oplus_{j=1}^{j=i} P_j) \qquad (6)$$

where $C(P_i)$ is a coverage of the pattern $P_i$ and $P_i \oplus P_j$ is a cumulative coverage of the patterns $P_i$ and $P_j$.

*Execution time* shows the time needed to discover a set of automatable routines. The execution time was measured in seconds. Each experiment was conducted with enabled Foofah tool for discovering data transformations. Foth both approaches, we set Foofah timeout to 20 seconds, it means that if Foofah encounters a 20 seconds timeout, it reports an absents of

data transformations. Overall, Foofah brings an overhead in the baseline execution time, but it allows us to discover not constant data transformations.

*The average routine automatability index* indicates the ratio of the discovered pattern actions that can be automated. It should be pointed out that the baseline can discover fully-automatable routines only, which means that automatable routines that were discovered by the baseline always have RAI equals 1.

In order to present the remaining four metrics, it is essential to explain the confusion matrix concept in terms of the UI log and discovered patterns. In the field of the statistical classification confusion matrix is a layout for visualization of the performance of the algorithm. Each row of the matrix represents predicted or discovered actions, while each column represents actual actions or actions that are presented in the log. The four cells of the confusion matrix report on the true-positive, false-positive, false-negative, and true-negative samples.

*True-positive* actions are actions that are presented in the log routines that are the ground truth for the pattern and, at the same time, are presented in the discovered pattern.

*False-negative* actions are presented in the log routines that are the ground truth for the pattern, but the same list of actions is missing in the pattern.

*True-negative* actions are not presented in the pattern and in the log routines that are described by the pattern. The example of such actions is noise, which is also recorded. Since evaluated artificial logs are noise-free, the true negatives for such logs will always be 0.

*False-positive* actions are presented in the pattern but not in the log routine that is ground truth for this pattern. An example of such actions is a noise that is frequently observed in the log, so it may be discovered as a part of the pattern. Similar to true-negative actions, for the patterns discovered on synthetic datasets, the false-positive metric will always be 0.

One of the metrics that we have used for the evaluation is *precision* or positive predictive value (PPV). The precision is a fraction of action that is presented in the pattern and in the log routine among the discovered actions in the pattern. The precision can be calculated as:

$$PPV = \frac{TP}{TP + FP} \tag{7}$$

As we have mentioned, false-positive metric equals 0 in the case of synthetic datasets, which means that the precision can be calculated for patterns that were discovered on artificial logs as:

$$PV = \frac{TP}{TP + FP} = \frac{TP}{TP + 0} = 1 \tag{8}$$

*Recall,* or true positive rate (TPR) is a fraction of true-positive pattern actions amount all actions in the log routine that is ground truth for the pattern. TPR can be calculated as:

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{9}$$

where $P$ is a total number of actions in the log routine.

The last metric that was used for evaluation is F-score. F-score is a harmonic mean of precision and recall. The F-score is calculated as:

$$F_1 = 2 \cdot \frac{PPV \cdot TPR}{PPV + TPR} = \frac{2TP}{2TP + FP + FN} \quad (10)$$

Since patterns discovered on the artificial logs do not contain noise, the precision of such patterns always equals to 1. It means that the F-score for them can be calculated as:

$$F_1 = 2 \cdot \frac{PPV \cdot TPR}{PPV + TPR} = \frac{2TP}{2TP + FN} \quad (11)$$

In the evaluation purpose, we have calculated average values of precision, recall, and F-score for our approach and the baseline on both synthetic and controlled-setting datasets[5].

## 5.3 Results on synthetic datasets

In this section, we report the results on our evaluation of the synthetic datasets. As was mentioned earlier, we have evaluated the same nine synthetically generated UI logs that were used for the evaluation of the baseline approach. However, for the compatibility with our approach implementation, synthetic logs were converted and restructured.

One of the qualitative improvements to our approach concerns the way of using the Foofah tool. In the baseline approach, the Foofah tool takes as an input the array containing all the values of one parameter of an action executed before specified action, and the output is the set of the values assigned to the specified action parameters. In other words, the baseline approach executes Foofah for each parameter of the action where the constant function was not obtained and each parameter of each action executed before such action (until data transformation function is found). The baseline also emphasizes that Foofah brings an overhead in the execution time: when enabling Foofah, the baseline approach becomes up to 50x slower.

Another qualitative improvement to the baseline approach is the ability to identify routine automatability index. Since the baseline approach can discover fully-automatable routines only, our approach allows us to identify is the discovered pattern is fully automatable or partially (in percentage measure).

[5] Metrics calculations are available at https://docs.google.com/spreadsheets/d/18do3igE93Tif6

The evaluation results for synthetic datasets are presented in Table 6. The table reports the dataset ID for which the experiment was conducted, the first row for each log id row corresponds to the result of the baseline approach, while the second row corresponds to the result of our approach. The table includes information about the number of discovered patterns, the maximum and average length of the discovered patterns, total coverage, execution time, and average RAI for our approach only. Each experiment for our approach was conducted with the set input parameter – the minimum support at 5%.

Table 6. Results on synthetic datasets.

| Log ID | #Patterns | Length (Max) | Length (Avg) | Total coverage | RAI (Avg) | Execution time (s) |
|---|---|---|---|---|---|---|
| 1 | 1 | 14 | 14.0 | 1.00 | 1 | 3 |
| | 1 | 14 | 14.0 | 1.00 | 1.0 | 3 |
| 2 | 4 | 5 | 4.0 | 0.87 | 1 | 62 |
| | 2 | 16 | 16.5 | 0.95 | 0.94 | 16 |
| 3 | 6 | 6 | 3.66 | 0.76 | 1 | 224 |
| | 4 | 21 | 16.75 | 0.85 | 1.0 | 7 |
| 4 | 4 | 4 | 2.5 | 0.75 | 1 | 79 |
| | 4 | 16 | 16.0 | 1.0 | 0.83 | 13 |
| 5 | 2 | 2 | 1.5 | 0.25 | 1 | 49 |
| | 10 | 8 | 8.0 | 0.70 | 0.75 | 8 |
| 6 | 4 | 4 | 2.0 | 0.69 | 1 | 80 |
| | 2 | 11 | 10.0 | 1.00 | 0.74 | 7 |
| 7 | 5 | 2 | 1.2 | 0.23 | 1 | 50 |
| | 4 | 11 | 11.0 | 0.73 | 0.63 | 11 |
| 8 | 5 | 13 | 3.8 | 0.43 | 1 | 549 |
| | 9 | 16 | 11.0 | 0.98 | 0.65 | 26 |
| 9 | 11 | 4 | 2.3 | 0.53 | 1 | 1619 |
| | 6 | 18 | 15.5 | 0.95 | 0.81 | 23 |

From Figure 8, we can see that our approach tends to discover longer patterns, as the length of the longest discovered pattern by our approach is larger than the length of the longest pattern discovered by the baseline.

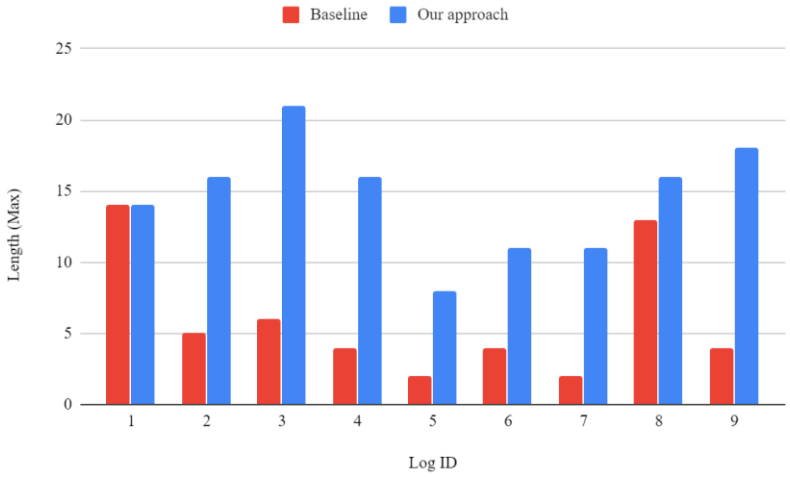


Figure 8. Comparison of the maximum length of patterns.

From Figure 9, we can see that besides maximum pattern length, the average pattern lengths also tend to be larger.

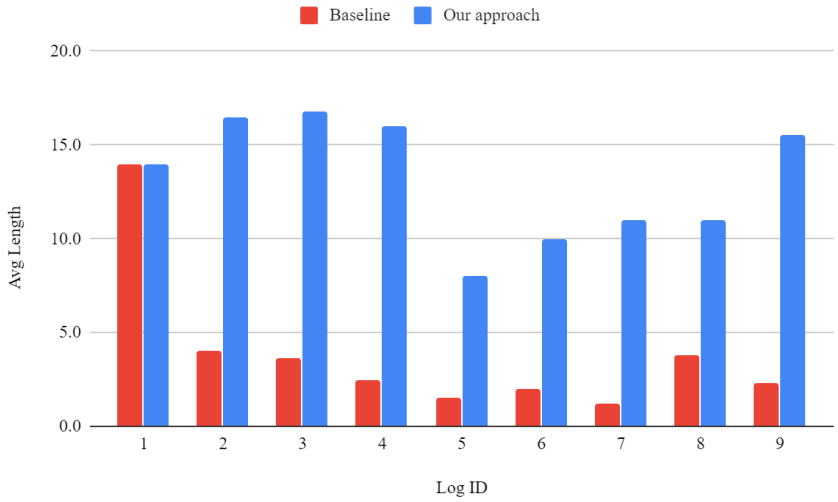


Figure 9. Comparison of the total coverages.

In Figure 10, the baseline and our approach were compared with respect to the total coverage metric. Our approach showed the same total coverage for the first dataset and higher total coverage for the rest datasets.

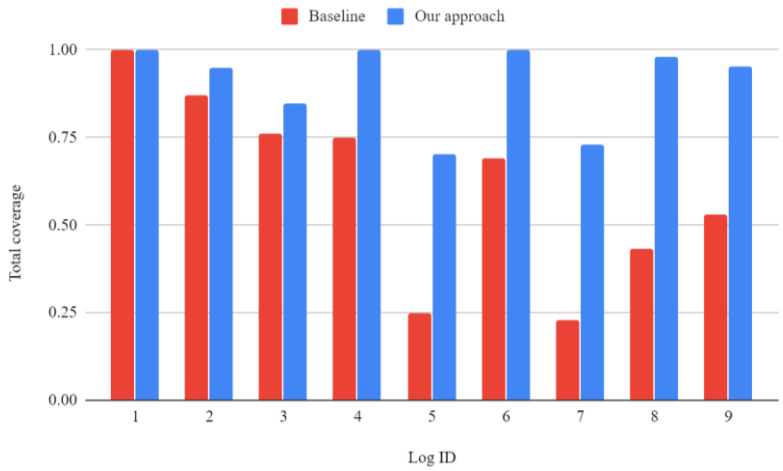


Figure 10. Comparison of the average patterns' length.

There are several reasons why the maximum pattern length, the average patterns' length, and total coverage is lower on the case of the baseline approach. At first, in the baseline approach, the first step is decomposing each log routine into sub-routines based on routine branching. It means that instead of analyzing routines as a whole entity, the baseline approach considers analyzing routine components. Since the length of sub-routines is lower than the routine itself, the length of discovered patterns by the baseline is lower.

Another reason for decreasing the pattern length in the case of the baseline is splitting discovered patterns by non-deterministic actions. Out approach considers discovered routines as an immutable entity that cannot be split by branching or non-automatable actions. Therefore, if the discovered pattern contains non-automatable action, the RAI of the pattern will decrease, but the length of the pattern will remain the same.

Figure 11 reports execution time (in seconds) for the baseline and our approach. The baseline approach requires much more time because of Foofah limitations that were mentioned before.

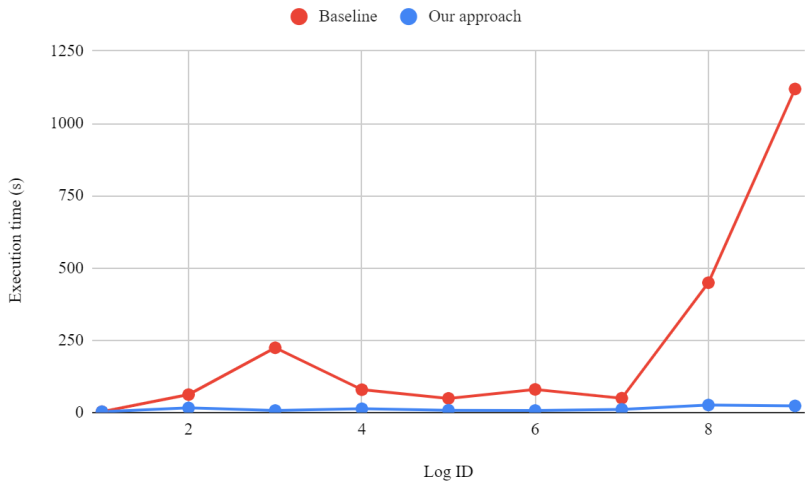


Figure 11. Comparison of the execution times.

Execution time depends on the complexity of the log, which leads to an increase in execution time in the case of ninth and eighth datasets. Nonetheless, our approach required no more than 25 seconds for rediscovering log routines.

Table 7 shows the following evaluation metrics: precision, recall, and F-score.

Table 7. Evaluation metrics for synthetic datasets.

| Log ID | Precision (Avg) | Recall (Avg) | F-Score (Avg) |
|---|---|---|---|
| 1 | 1.00 | 1.00 | 1.00 |
| | 1.00 | 1.00 | 1.00 |
| 2 | 1.00 | 0.29 | 0.45 |
| | 1.00 | 1.00 | 1.00 |
| 3 | 1.00 | 0.22 | 0.35 |
| | 1.00 | 0.81 | 0.89 |
| 4 | 1.00 | 0.19 | 0.30 |
| | 1.00 | 1.00 | 1.00 |
| 5 | 1.00 | 0.17 | 0.28 |
| | 1.00 | 0.87 | 0.93 |
| 6 | 1.00 | 0.20 | 0.32 |
| | 1.00 | 1.00 | 1.00 |
| 7 | 1.00 | 0.11 | 0.20 |
| | 1.00 | 0.93 | 0.96 |
| 8 | 1.00 | 0.27 | 0.38 |
| | 1.00 | 0.94 | 0.97 |
| 9 | 1.00 | 0.23 | 0.36 |
| | 1.00 | 1.00 | 1.00 |

As was mention earlier, int the case of synthetic datasets, precision equals to 1 for both approaches. As we can see, for each synthetic dataset, our approach outperforms the baseline. Since our approach can discover log routine entirely despite the baseline approach that discovers parts of the routine, the number of false-negative actions for our approach is much lower. The other reason why metrics for our approach are al least in half better is the gain the patterns' length. Since our approach can find longer patterns, it increases the number of true-positive action, which leads to improvements in the evaluation metrics.

Besides an evaluation of qualitative and quantitative metrics, we have evaluated the effect of changing the minimum support threshold. The minimum support is a parameter in [0, 1] interval. All patterns that are mined by sequential pattern mining techniques have support lower than the specified minimum support. It means that too lower minimum support can lean to discovering too many patterns, while minimum support that tends to 1 leads to discovering only a few patterns from a set of log routines.

The evaluation of the effect of the minimum support threshold on the results of our approach was done on Log2, Log4, Log7, and Log9. A choice of logs is justified by the fact that the complexity of each next log steadily increases compared to the previous. More precisely, Log2 contains three routine unique variants, while Log9 – 38 routine unique variants. A set of experiments was conducted with a minimum support threshold that corresponds to 5, 10, 25, 50, 75, 100, respectively.

Figure 12 reports the number of patterns that were discovered by our approach on synthetic datasets for a specified set of minimum support threshold. As we can see, decreasing the value of the threshold leads to discovering a larger number of patterns. With specified minimum support as 100, we have discovered only one pattern for each dataset. However, the more complex log is, the larger number of patterns we have discovered with decreasing the threshold.

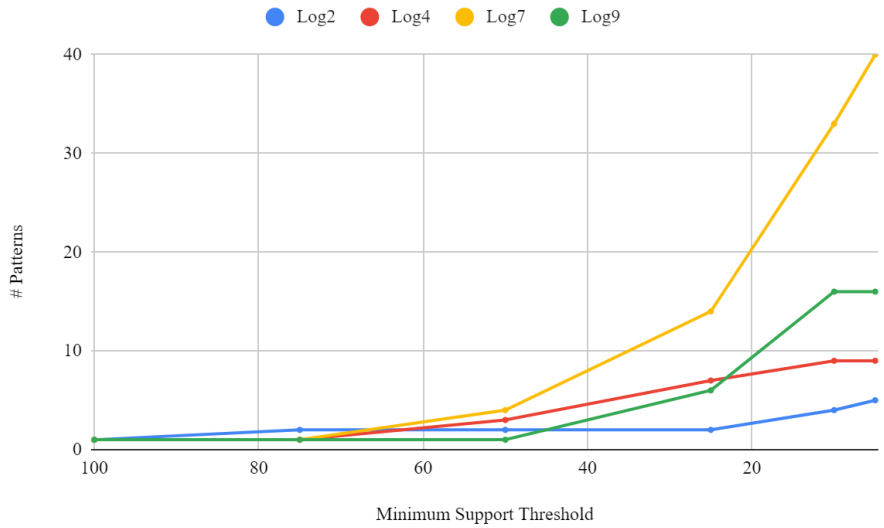


Figure 12. Number of patterns vs. minimum support threshold.

Figure 13 reports the total coverage of patterns discovered by our approach versus the specified threshold. For all considered datasets, the total coverage remains the same or increases with decreasing the minimum support. For all datasets, besides Log7, the total coverage continuously increases with a tendency to one. Since the Log7 dataset contains 14 complex routines, our approach was able to increase the total coverage of discovered patterns up to 72% by setting the minimum support threshold at 5%.

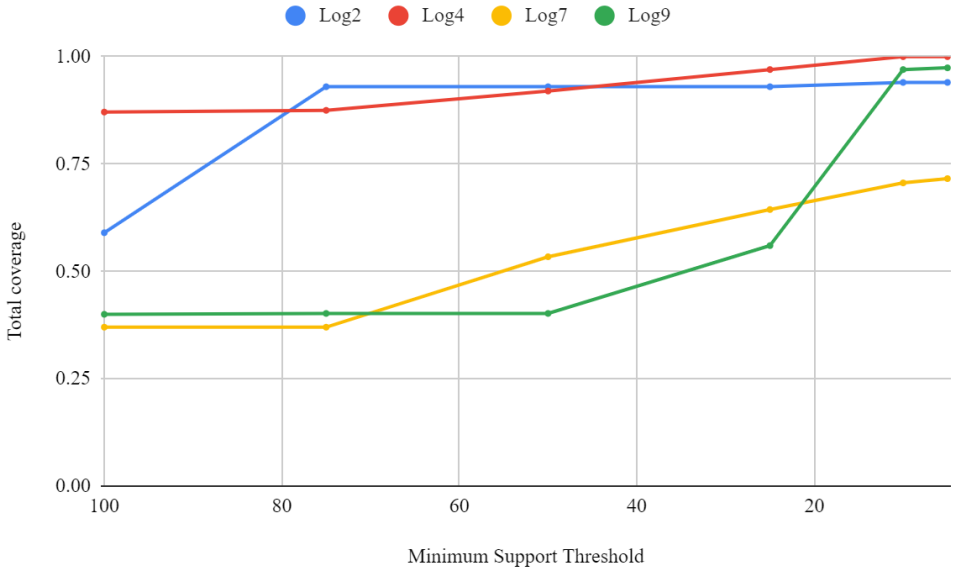


Figure 13. Number of patterns vs. minimum support threshold.

In Figure 14, we show the dependency of the F-score and minimum support threshold for four synthetic datasets. The results show that, in general, by decreasing a threshold, our approach discovers patterns with a higher average F-score.

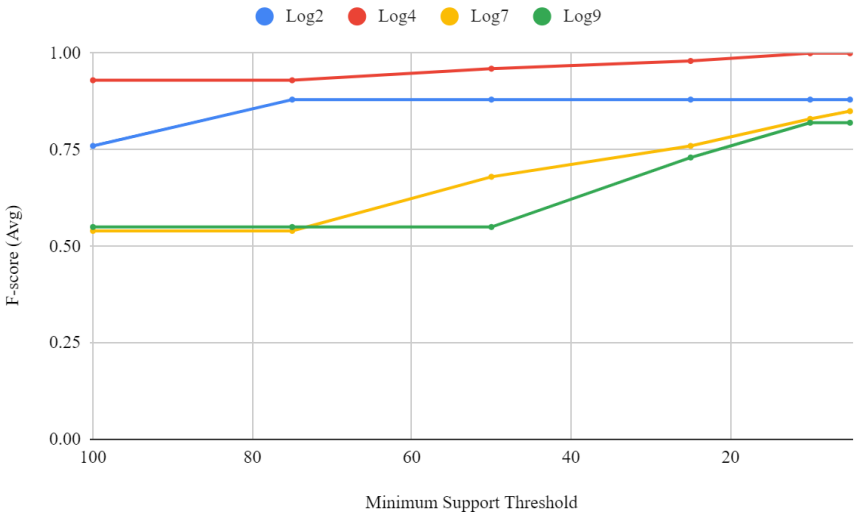


Figure 14. Average F-score vs. minimum support threshold.

The decrease of the minimum support threshold leads to a slight increase in the average F-score for the Log2 and Log4 datasets. In can be explained by the low complexity of the corresponding datasets. Since the average F-score has the value close to one even for the maximum threshold that equals 100%, it will not change a lot for smaller thresholds.

However, decreasing the threshold for Log7 and Log9 datasets increases the average F-score up to 0.82 for the minimum support set at 5%.

The increase in the total coverage and average F-score is explained by the limited number of patterns, which is set by the minimum support threshold. As we have shown on the example of synthetic datasets, the lower minimum support leads to increasing the number of discovered patterns, hence, to the increase in the total coverage and the average F-score, while high threshold decreases those metrics. However, too lower value of the minimum support threshold can lead to increasing the running time (in seconds) and memory consumption and an extremely large number of frequent sequences [11].

## 5.4 Results on controlled-setting datasets

Despite synthetic datasets, controlled-setting logs are much more similar to the real-life dataset. One of the most important differences is the fact that controlled-setting datasets contain noise. The noise represents events that are irrelevant to the routine that is presented in the log. If the baseline approach cannot deal with noise, our approach that mines frequent closed sequential patterns can handle noise filtering. Further, we will demonstrate it for the Log10 dataset.

Besides dealing with noise, our approach can discover functional dependencies in the routine. It increases the number of automatable actions in the discovered patterns, which, hence, increases patterns RAI.

From Table 8, we can see that for each controlled-setting datasets, the maximum pattern length, and the average patterns' length discovered by our approach is higher than by the baseline.

Table 8. Results on controlled-setting datasets.

| Log ID | #Patterns | Length (Max) | Length (Avg) | Total coverage | RAI (Avg) | Execution time (s) |
|---|---|---|---|---|---|---|
| 10 | 3 | 12 | 6.3 | 0.85 | 1 | 1419 |
| | 2 | 22 | 21.0 | 1.00 | 1 | 34 |
| 11 | 5 | 5 | 3.4 | 0.76 | 1 | 1420 |
| | 2 | 22 | 21.0 | 1.00 | 0.95 | 28 |
| 12 | 15 | 17 | 2.93 | 0.452 | 1 | 5979 |
| | 2 | 22 | 21.0 | 0.98 | 1 | 31 |

The first datasets contain actions that correspond editing country code field in the Web browser. That action depends on the previous one that corresponds to editing the country name also in the Web browser. The data about country code cannot be extracted from the previous actions directly. However, it depends on the county. It means that editing country code action can be automated by using functional dependencies. Since the baseline approach was not able to find a deterministic function for that action, the action was marked as "non-automatable," and the baseline split the discovered patterns on that action, which decreased the maximum and average length of the discovered patterns. For the same reason, the total

coverage of the baseline is slightly lower than for our approach. As we can see, our approach has managed to discover functional dependencies as the average RAI equals to one. Moreover, we have found two patterns out of two log routines, which covers 100% of the log.

The same difference in the evaluation measures can be seen in the case of the Log11 dataset since it contains one non-automatable action. It leads to decreasing in maximum and average patterns' length and total coverage.

For the Log12, the baseline shows the lowest evaluation measures. The noise present in the log can explain it. Since the limitation of the baseline, the approach is its inability to deal with noise. It discovers fifteen patterns for the Log12 dataset with total coverage equals to 0.45, while our approach that can deal with noise discover two patterns out of two log routines with total coverage equals 0.99. We have reported the average RAI equals to one for discovered patterns by our approach, which means that each action of the routine can be automatable and, in total, none of the patterns contain a noise.

Finally, from Table 8, we can see that the baseline execution time is at least 40 times higher than our approach execution time. In particular, for Log11, the execution time is 40 times higher, and for the Log12 datasets – 50 times higher. It can be explained by the high number of actions, which transformation function is not constant, but are determined by complex data transformations.

The results obtained on the Log12 datasets confirm the limitations of the baseline to deal with noise. In particular, the baseline execution time on the log with noise is higher in almost 200 times compared to our approach.

Table 9 shows the average precision, recall, and F-score metrics obtained for the controlled-setting datasets.

Table 9. Evaluation metricsfor controlled-setting dataset.

| Log ID | Precision (Avg) | Recall (Avg) | F-Score (Avg) |
|---|---|---|---|
| 10 | 1.00 | 0.30 | 0.43 |
| | 1.00 | 1.00 | 1.00 |
| 11 | 1.00 | 0.16 | 0.27 |
| | 1.00 | 1.00 | 1.00 |
| 12 | 1.00 | 0.14 | 0.21 |
| | 1.00 | 0.98 | 0.99 |

We can notice that the F-score for our approach is much higher compared to that of the baseline. In general, this and the difference recall implies that our approach is able to discover patterns with a larger number of true-positive actions – the actions that are presented in the log routine and less number of false-negative actions – the actions that are missed in the pattern but are presented in the routine.

Figure 15 reports on the number of the discovered patterns by our approach against the value of the minimum support threshold. Since the controlled-setting datasets contain only two routines, even the hight number of the threshold (75%) allows us to discover two out of two patterns.

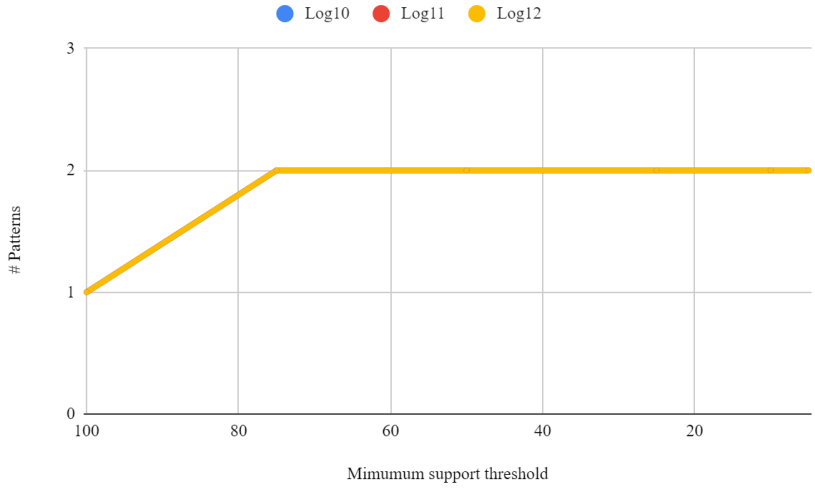


Figure 15. Number of pattern vs. minimum support threshold.

We have observed the same situation for the total coverage and the average F-score metrics. Even a small minimum support threshold that equals 75% leads to the higher value of the total coverage since both routines described in controlled-setting datasets share 11 out of 20 actions for the first routine and 11 out of 22 actions for the second routine.

# 6 Conclusion

In this thesis, we have presented an approach that consists of two modules. The first module is responsible for the identification of candidate routines for automation from the UI log. It exploits sequential pattern mining techniques with a user-specified minimum support threshold for discovering closed frequent sequential patterns within a log. Since sequential pattern mining operates on datasets of symbolic items, we overcome such a limitation by encoding UI log action from complex structural objects to the symbolic representation. We have shown the process of ranking of mined patterns based on membership-based cohesion score. Moreover, we have presented the cut-off technique that is based on a drop-down pattern value in order to select the most valuable pattern only.

The second module of the approach is responsible for routine automatability assessment. The goal of that module is to identify the degree of automatability for candidate routines for automation. We determine if the pattern is automatable by analyzing dependencies within pattern actions. More precisely, we consider syntactical and semantical transformations between action attributes. For discovering syntactical transformation, we exploit the optimized Foofah approach on pairs of nearest read and write pattern actions. Foofah allows us to discover data transformation between such actions. Once we have found writing actions that do not refer to any synthetical transformation, we investigate such actions on semantical transformation. Synthetical transformations are presented by functional dependencies – relationships between attributes that uniquely determines the value of the attributes by the values of other attributes. Finally, if any of the write action does not refer to synthetical or semantical transformation, the action is non-automatable. Furthermore, we have presented the RAI metric that corresponds to the degree of the pattern automatability.

An experimental evaluation was conducted both on synthetical and controlled-setting datasets. It provides a comparative analysis of the thesis approach and the baseline. The evaluation has shown that our approach outperforms the baseline, both qualitatively and quantitatively. More precisely, it allows us to discover longer routines with shorter execution time, determine not only fully-automatable routines but also partially-automated. This enables us to use the approach in conjunction with both attended and unattended RPA tools. Additionally, the evaluation demonstrates that our approach can deal with the noise contained in the UI log.

At the same time, the provided approach has several limitations. One of the limitations of the approach arises from the use of sequential pattern mining. While closed sequential pattern mining techniques have a high performance, it focuses on discovering tasks in which actions were executed sequentially. As a result, if a UI log describes a single routine, and each log trace contains routine actions in a different order, we may discover multiple variants of the same routine. Consequently, in order to address the limitation, the postprocessing of discovered routines is required. Another limitation of the sequential pattern mining is the ability to discover merely frequent patterns. If the UI log contains not frequent routines, they will not be discovered by the proposed approach. Another limitation comes up when the UI log contains too much noise. It leads to an increase in the number of outlier occurrences in the patterns. Hence, the cohesion score of such patterns will drop, which can lead to the

situation when the pattern with a high number of outliers will be cut off from the resulting set of discovered patterns.

# 7 References


[1] W. M. P. v. d. Aalst, M. Bichler and A. Heinzl, "Robotic Process Automation," *Bus Inf Syst Eng 60,* p. 269–272, 2018.

[2] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proceedings of the eleventh international conference on data engineering*, 1995.

[3] G. M. N, R. Rajeev and S. Kyuseok, "SPIRIT: Sequential pattern mining with regular expression constraints," in *VLDB*, 1999, pp. 7-10.

[4] A. Bosco, A. Augusto, M. Dumas, M. La Rosa and G. Fortino, "Discovering automatable routines from user interaction logs," in *International Conference on Business Process Management*, 2019.

[5] V. Leno, A. Polyvyanyy, M. Dumas, M. L. Rosa and F. M. Maggi, "Robotic Process Mining: Vision and Challenges," *Business & Information Systems Engineering,* pp. 1-14, 2020.

[6] H. Peter, S. Caroline and U. Nils, "Robotic process automation," *Electronic Markets,* pp. 1-8, 2019.

[7] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh and R. Thomas, "A survey of sequential pattern mining," *Data Science and Pattern Recognition,* vol. 1, pp. 54-77, 2017.

[8] S. Farzana Zerin and B.-S. Jeong, "A fast contiguous sequential pattern mining technique in DNA data sequences using position information," *IETE Technical Review,* vol. 28, pp. 511-519, 2011.

[9] C. Li and J. Wang, "Efficiently mining closed subsequences with gap constraints," in *proceedings of the 2008 SIAM International Conference on Data Mining*, 2008.

[10] G. Sirisha, M. Shashi and G. P. Raju, "Periodic pattern mining-algorithms and applications," *Global Journal of Computer Science and Technology,* 2014.

[11] F. Fumarola, P. F. Lanotte, M. Ceci and D. Malerba, "CloFAST: closed sequential pattern mining using sparse and vertical id-lists," *Knowledge and Information Systems,* vol. 48, no. 2, pp. 429-463, 2016.

[12] A. Aztiria, A. Izaguirre, R. Basagoiti, J. C. Augusto and D. J. Cook, "Automatic modeling of frequent user behaviours in intelligent environments," in *2010 Sixth International Conference on Intelligent Environments*, 2010.

[13] H. Leopold, H. van der Aa and H. A. Reijers, "Identifying candidate tasks for robotic process automation in textual process descriptions," in *Enterprise, business-process and information systems modeling*, Springer, 2018, pp. 67-81.

[14] W. Van Der Aalst, Process mining: discovery, conformance and enhancement of business processes, Springer, 2011.

[15] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, F. M. Maggi, A. Marrella, M. Mecella and A. Soo, "Automated discovery of process models from event logs: Review and benchmark," *IEEE Transactions on Knowledge and Data Engineering,* vol. 31, no. 4, pp. 686-705, 2018.

[16] M. De Leoni, M. Dumas and L. García-Bañuelos, "Discovering branching conditions from business process execution logs," in *International Conference on Fundamental Approaches to Software Engineering*, 2013, pp. 114-129.

[17] J. Srivastava, R. Cooley, M. Deshpande and P.-N. Tan, "Web usage mining: Discovery and applications of usage patterns from web data," *Acm Sigkdd Explorations Newsletter,* vol. 1, no. 2, pp. 12-23, 2000.

[18] A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li and J. L. Herlocker, "TaskTracer: a desktop environment to support multi-tasking knowledge workers," in *Proceedings of the 10th international conference on Intelligent user interfaces*, 2005, pp. 75-82.

[19] V. Leno, M. Dumas, M. La Rosa, F. M. Maggi and A. Polyvyanyy, "Automated Discovery of Data Transformations for Robotic Process Automation," *arXiv preprint arXiv:2001.01007,* 2020.

[20] Z. Jin, M. R. Anderson, M. Cafarella and H. Jagadish, "Foofah: Transforming data by example," in *Proceedings of the 2017 ACM International Conference on Management of Data,* 2017.

[21] A. Jimenez-Ramirez, H. A. Reijers, I. Barba and C. Del Valle, "A method to improve the early stages of the robotic process automation lifecycle," in *International Conference on Advanced Information Systems Engineering*, 2019.

[22] W. Van Der Aalst, "Data science in action," in *Process mining*, Springer, 2016, pp. 3-23.

[23] J. Han, J. Pei, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal and M. Hsu, "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth," in *proceedings of the 17th international conference on data engineering*, 2001.

[24] P. Fournier-Viger, A. Gomariz, M. Campos and R. Thomas, "Fast vertical mining of sequential patterns using co-occurrence information," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2014.

[25] H. Dev and Z. Liu, "Identifying frequent user tasks from application logs," in *Proceedings of the 22nd International Conference on Intelligent User Interfaces*, 2017.

[26] Y. Huhtala, J. Kärkkäinen, P. Porkka and H. Toivonen, "TANE: An efficient algorithm for discovering functional and approximate dependencies," *The computer journal,* vol. 42, no. 2, pp. 100-111, 1999.

[27] V. Leno, A. Polyvyanyy, M. La Rosa, M. Dumas and F. M. Maggi, “Action logger: Enabling process mining for robotic process automation,” in *Proceedings of the Dissertation Award, Doctoral Consortium, and Demonstration Track at 17th International Conference on Business Process Management,(BPM’19), Vienna, Austria*, 2019.

# Appendix

## I. Licence

**Non-exclusive licence to reproduce thesis and make thesis public**

I, **Stanislav Deviatykh**

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,
   **Discovering Automatable Routines from UI Logs via Sequential Pattern Mining**
   supervised by Marlon Dumas and Volodymyr Leno.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

*Stanislav Deviatykh*

***15/05/2020***