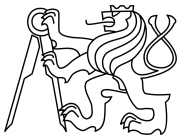
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF MASTER'S THESIS

Title:	Analysis of bank data
Student:	Bc. Evgeniya Nenenko
Supervisor:	doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Theoretical Computer Science
Validity:	Until the end of winter semester 2018/19

Instructions

The aim of the work is to analyze transactional data (bank data of financial account transactions) to identify different patterns of customer behavior. These can then be used, e.g., to optimize sales and marketing activities of the bank.

1) Get familiar with the structure of the transaction data and the problems of detecting patterns in time series.
2) Explore the transaction data in order to detect anomalies and identify interesting patterns of customer behavior.
3) Propose methods and algorithms to analyze the transactional data.
4) Design the proposed methods and algorithms in an appropriate analytical tool or implement them using appropriate programming language.
5) Verify the methods and algorithms on real data and evaluate the results.

References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague February 18, 2017

Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Computer Science

Master's thesis

Analysis of bank data

Bc. Evgeniya Nenenko

Supervisor: doc. RNDr. Ing. Marcel Jiřina, Ph.D.

9th May 2017

Acknowledgements

I'd like to thank the supervisor of my thesis work doc. RNDr. Ing. Marcel Jiřina, Ph.D. for advices, help and patience. Also, I'd like to thank Data Science department of KB bank, namely Tomáš Lancinger, Karel Šimánek and Tomáš Hubínek for the participation and cooperation. Last but not least, I'd like to thank my parents who were always supportive and caring throughout the years of my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 9th May 2017 ……

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Nenenko, Evgeniya. *Analysis of bank data.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Cílem této práce bylo analyzovat bankovní transakční data za účelem získání vzorců chování zákazníků. Osnovu této práce tvoří data mining periodických vzorců v databázích časových řad. Pro návrh vhodného analytického rámce byl proveden výzkum dostupných řešení s pozdější adaptací vhodných algoritmů tak, aby co nejlépe odpovídaly požadavkům zadání. Poté bylo provedeno vyhodnocení výsledků a nabídnuta doporučení týkající se nastavení tohoto rámce, budoucího vývoje a využití výsledků.

Klíčová slova Vytěžování dat, dobývání znalostí, databáze časových řad, periodické vzorce

Abstract

The aim of this work was to analyse bank transactional data in order to find patterns in the customers' behaviour. The main focus being periodic pattern mining in time series databases. To provide a suitable analytical framework, the research of the existing solution was made with a later adaptation of available algorithms to suit the requirements of the assignment. Finally, the

result evaluation took place with suggestions regarding framework settings and future usage.

Keywords Data mining, knowledge discovery, time series database, periodic patterns

Contents

List of Figures

List of Tables

Introduction

Modern data analysis technologies these days enable streamline accumulation of hundreds or thousands of records daily. Whether it is meteorological data that register changes in the temperature, atmosphere pressure or air humidity, stock market data that register changes in stock values, these types of information are vital to create a descriptive conception of changes that develop in time. These measurements, collected in order to observe changes and detect trends are called time series data.

Apart from perceiving time series data as a set of data points that was gathered in equal periods of time, another way to do so is in the form of records, which besides mere changes in one particular value, also contain attributes that describe these records. For instance, logbook of employees': such records might contain type of log, whether it is a log-in or log-out, timestamp, identification of facility and others. When such records are distributed uniformly during the day, this logbook is said to be time series database.

The bank sector is one of the most bountiful ground for innovative data analysis. Gathering vast amount of transaction data daily, it is truly the sphere of advanced data analysis technologies that enable multifaceted data mining. However, there exists an opinion that bank institutions tend to have historically rigid policies towards adapting new technologies and methods. This might be caused by several reasons: very strict regulations, conservative senior management, strong best practices traditions etc. Nevertheless, changes in this sector caused by natural market competitiveness, young technically advanced players, loss of exclusiveness of bank products offer brings the necessity of implementing an innovative approach to data analysis.

Same as to any other companies on the market, understanding data for bank institutions is a major question in modern business reality. The importance of creating the right data policy cannot be by any means overestimated, as the right business strategy is key to create a successful competitive ability. The capacity to propagate future events in time series data is important for the decision making process. This is why discovering knowledge from historical

time series data is a very interesting and promising area of research.

In the framework of this thesis paper, I'd like to focus on the problem of time series data mining with particular consideration for periodic pattern mining, and as a result suggest a number of innovations that will provide new perspectives of bank transactions' time series data and will add value to the company.

Motivation for this work

The motivation for this work was born and defined in KB bank's Data Science department. It was agreed that with considerable amount of historical data that was owned by KB at the moment of creating the technical specification for this thesis work, it was the highest time to make an attempt at discovering new successful and productive ways of utilizing it, to use it as a source of further innovations and creating a more advanced approach to bank transactions analysis. For the KB bank (further referred to as project owner) the main points of interests were questions such as:

- Do companies with similar characteristics interact in a similar manner with other companies?
- Are there patterns that can be detected and how can they be possibly represented and stored?
- Can the anomalies in the customers' behavior be discovered in a more efficient way rather than in a brute force observation and monitoring approach?
- Can we state, that by discovering common patterns for a group of companies, based on the type of activity (NACE code and turnover amount), we can project such patterns to individual clients?
- Can those detected and proven to be frequent and periodic patterns be used in order to predict customers' behavior?
- In what other ways can such patterns be used?

To be able to answer those questions, it was needed to create an analytical framework that would be able to recognize repetitive events discovered in historical data. More specifically, detect patterns in *aggregated time series*. Raw transaction data can be represented as records with standard attributes such as what amount of money was sent or received, account number of a transaction origin, recipient account number and additional details like timestamp etc., together with foreign keys as a reference to the other fact tables that contain further information about such transactions.

Analyzing historical data in the domain of individual transactions, however, may detract from the whole image of business reality, as they tend to be matchless, exclusive and either repeated constantly, and therefore redundant, or randomly and thus not periodic. However, while analyzing aggregated time series, e.g. transaction amount per category or market sector, that perspective enables one to ignore insignificant intermediate results and concentrate on the major trends and patterns. Aggregated time series are, in a way, data that were grouped by all of the record attributes, except for one, for example – total amount of money that was received, with beforehand defined time granularity. Such aggregated time series of bank transactions data are the subject of study in this thesis.

The output of the whole analytical framework may be then intended for better understanding the interconnection between different market sectors, event propagation and therefore improvement of the decision making process.

Definition of the problem

The main question that was given by the project owner as a definition of the problem and was tried to be answered in this work is: is it possible to detect a repetitive behavior for a particular company or a group of companies in order to identify regularities in one's business or a business sector? That is a primary question that invokes several other points of interest. Among them are for example: can these patterns be sector-specific? Does there exist particular seasonality in such behavior?

While the approach for detecting and mining periodic patterns in time series data remains universal, the result set of patterns may have various characteristics that respond to different business needs based on the input data and the post-evaluation:

- sequential patterns in time series data;
- patterns that indicate seasonality in market sectors and their interactions' cash flow;
- patterns in time series that appear to be sector-specific;
- meta patterns as a group of events that lead to particular deviations in customers' behavior

Goal of this work

In order to answer the given questions, the research upon the matter of time series data mining took place with later adaptation of the most suitable algorithms. The main goals of this work were to:

1. Get familiar with bank transactions data structure.
2. Consult and create appropriate discretizing method for the events in data.
3. Design an approach that will be used as a tool to generate, detect and mine periodic patterns in transactional data.
4. Embody this approach in an analytical tool.
5. Create a framework to find unusual changes in customers' behavior and notify business analytics about such events. (In the boundaries of this work notification is not meant to be sent or generated automatically).
6. Analyze and evaluate the outcome, consult the results' significance and value with the experts.
7. Propose a possible future use of the results.

Outline

The rest of this thesis work is organized as follows: chapter 1 is focused on the brief introduction to the theory of time series data and database, its use in general data mining. Chapter 2 represents research that was made in order to find appropriate methods of time series pattern mining. Chapter 3 is dedicated to further problem analysis and description of the given data set. Chapter 4 describes the suggested design of analytical framework and methods that were selected, including the adaptation of the chosen algorithms, its' alternations and extensions. Programing language, environment and libraries that were used are described in chapter 5. Finally chapter 6 is dedicated to the results' evaluation. In chapter 7 there is a discussion of the achieved outcome and its potential future use.

Chapter 1

Time series data

There are several ways to represent a bank transactional history database. One of them is the time series data or time series database. The purpose of this chapter is to provide a brief introduction into the problem of time series database and common data mining approaches to it. Also in this section, some preliminary terminologies are introduced to provide necessary background for better understanding of time series databases and pattern mining algorithms that will be discussed later.

1.1 Time series

These days, many aspects of everyday scientific and financial activities involve constant measurements and the storing of those resulting values in order to discover hidden knowledge in such data. These organized measurements collection are called time series. The process of discovering hidden knowledge is known as data mining.

In existing literature there can be found at least two main approaches to the time series data. The first one represents time series as a contiguous set of data points or a signal, whereas the second one suggests that time series consist of discrete multi-dimensional records or events.

A *time series* T is an ordered sequence of n real-valued variables, as shown in Formula 1.1

$$T = (t_1, ..., t_n), t_i \in R \tag{1.1}$$

Often, time series is a result of the observations of a particular underlying process in the course of which, values are collected from the measurements made at uniformly spaced time instants and according to a given sampling rate. A time series can thus be defined as a set of contiguous time instants [1]. Such data often represent only data points chosen by sampling or importance

level to scale out, simplify or smooth data. To illustrate the above-mentioned definition, see Figure 1.1 below:

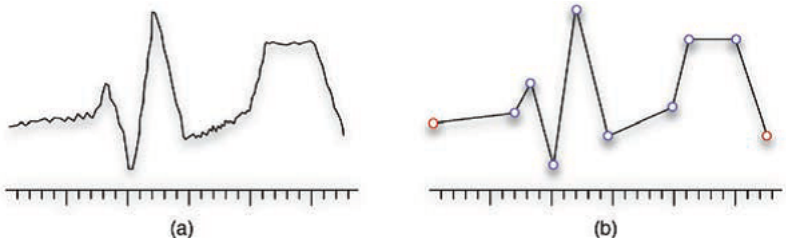


Figure 1.1: Example of application of segmentation system in time series data, adopted from [1]

The main issue with data mining this type of time series is the high dimensionality of it. To eliminate this constraint, time series databases mainly consist of merely simplified versions of time series.

This type of time series is further analyzed with the aim of extracting knowledge from the shape of the data, as illustrated in Figure 1.2. The initial tasks that are often required to preprocess data for further analysis are data representation and indexing. The core tasks for the time series data mining are segmentation and similarity measure between two time series. The data mining upon these time series may then include tasks like dimensionality reduction [2], segmentation [3], shape-based pattern discovery [4] or [5], clustering [6] and many others.

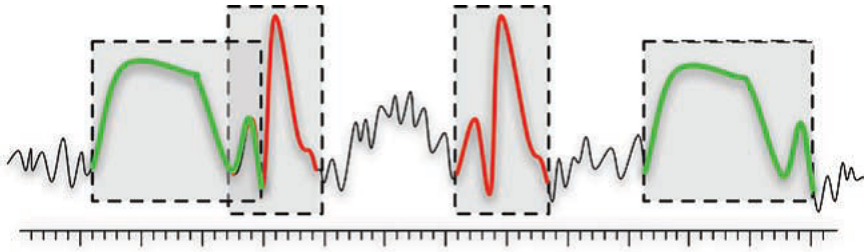


Figure 1.2: Typical example of motif discovery, adopted from [1]

Another way to understand time series data is to imagine them as a set of records uniformly distributed in time that kept all of their attributes, thus did not fall under dimensionality reductions and are treated as events, see Table 1.1. Such set of events is called time series database.

A *time series database* is a set of observations taken at specified times, usually at "equal intervals". Mathematically a time series database is defined by a set of values $Y_1, Y_2, \ldots, Y_n$ of a variable Y at times $t_1, t_2, \ldots, t_n$.

Thus, the relation among the variable and time values can be defined as $Y = F(t)$.

Table 1.1: Example of analogous database of a company's working hours for employees

	Day	Duty slot	Event
0	12/3/2016	08:00 - 10:00	Login
1		10:01 - 12:00	
2		12:01 – 14:00	
3		14:01 – 16:00	Logout
4		16:01 – 18:00	
5	14/3/2016	08:00 – 10:00	
6		10:01 – 12:00	Login
7		12:01 – 14:00	
8		14:01 – 16:00	
9		16:01 – 18:00	Logout

In the rest of this work we only consider this type of time series databases.

1.2 Pattern mining in time series database

Time series mining is a branch of data mining that consists of sequences of values or events that were obtained with respect to a certain time interval such as daily, weekly and monthly. In real-life applications, when there exists a certain possibility for an event to occur repeatedly after an equal period of time, the techniques of time series data mining are mostly used to discover interesting knowledge from the repeated events in the time series databases as for instance, stock market transactions and price changes, transaction analysis in supermarkets, computer logs, analysis of climate indicators such as temperature, ocean level, air pollution and others.

With the aim of effective analysis of time series, time series databases are often represented as a sequence of events, instead of tabular form. The procedure known as discretization [7] will transform a list of record into the ordered set of symbols, where each symbol represents a certain range of events. Consider the example above in Table 1.1. Each work day has 5 time slots, designated for an employee to log in and log out.

Imagine every slot of the time schedule is represented with one distinct symbol, where "a" stands for log-in, "c" for log-out, "b" for the time slot, describing the time spent at work, "d" for after-work hours and "x" for pre-work hours. Then the time series database from example in Table 1.1, can be represented as shown in equation 1.2:

$$T = \{abbcd\ xabbc\} \tag{1.2}$$

Pattern mining is one of the most important areas in data mining that

includes frequent pattern mining, sequential pattern mining, and periodic pattern mining.

A promising field of data mining is time series analysis, where a sequence of items or events can be found with respect to a fixed time interval. When an event or a sequence of events, also called pattern, repeats itself in a time series dataset with a specific time interval, it is known as *periodic pattern*.

For instance, in the example event sequence $T = \{bbaa\ abaa\ abac\ abdd\}$, the pattern "ab" is periodic where the period value $p = 4$ and starting position $= 4$.

Periodic pattern mining is an interesting research topic that allows possible prediction of future events or trends in different applications, for example business or scientific. *Periodic pattern mining* refers to the discovering of patterns, that appear to be frequent and periodic in the given time series database. Periodicity detection, that is a part of periodic pattern mining, is a process for checking regularities of patterns' occurrences within the time series database. In general, these three types of periodic patterns can be detected in a time series database:

- Symbol periodicity
- Sequence periodicity or partial periodic patterns
- Segment or full-cycle periodicity

If at least one symbol is repeated periodically in a time series database, such time series database is said to have symbol periodicity. In the same way, if a pattern consists of more than one symbol and is proven to be periodic in a time series database, this type of periodicity is called partial periodic patterns. Lastly, if the whole time series database can generally be represented as a repeated occurrence of a pattern or a segment, then this type of periodicity is called segment periodicity or full-cycle periodicity.

Pattern mining techniques that allow certain level of event skipping while scanning a database for the pattern occurrences is called flexible pattern mining. Flexible pattern is an ordered set of records or events, that might contain the do-not care symbol or "*", for example $\{a * b\}$, $\{a * *b\}$ or $\{a(*)^{\theta}b\}$. Usually, the maximum amount of skipped events, denoted by θ, is defined by the user. The ability to skip events is very advantageous for the pattern mining technique, as time series databases are not always perfectly structured or may contain unnecessary events.

1.3 Time series in bank and financial transactions

Among many examples of well-known real life datasets, which are widely used for time series database analysis, where periodic patterns can be discovered to

anticipate interesting information, there are financial and bank transactional data.

While stock market time series, that mostly represent stock prices evolving in time, tend to be much demanded for time series analysis and pattern mining, for example [8] or [9], to perform so called “technical analysis” and “fundamental analysis”, there are few if any studies, that suggest the ready-made approach for bank transactional data.

In the next section I’m introducing the result of existing work and algorithms research, which was made in order to choose suitable ones for the purpose of this work.

Chapter 2

State-of-the-art

Periodic patterns in time series data is this work's main point of interest. In order to design the pattern mining approach and create an appropriate analytical tool, I did the research of the existing methods and algorithms that are commonly used in a field of time series data mining. Pattern mining in time series databases is a very interesting data analysis problem that has been widely researched in the recent years.

Due to the nature of the task, it was vital, that we observe the events from a wider perspective and not only recorded changes of a one parameter in time. That said, it is extremely complicated to detect interesting patterns in transactional data if we only consider simplified time series, e.g. the balance curve of a particular company, the amount of money transferred between two particular companies, again, represented as a continuous variable. On the contrary, it was essential to have the ability to keep as much record dimensions as possible. That is why, algorithms that consider shape of the data were not suitable for the purpose of the given task.

As it was discussed in chapter 1.1, in the boundaries of this work I have approached time series database as a set of sequential events collected over equal time periods. Also, it was decided together with the project owner, that only historical data would be taken into consideration, so algorithms that mine patterns in the streaming data, such as for example method of pattern recognition with "sliding window" [10], were not suitable for this task. In the rest of this chapter I discuss algorithms that were considered and chosen for the implementation.

2.1 Existing algorithms

Most of the works that concentrate on the problem of periodic sequence pattern mining in time series databases have two main steps in their approaches, where first step is to find and trace repeated subsequences of time series data-

base and nominate those subsequences to be periodic patterns, and the second step represents the periodicity detection and check for such patterns to eliminate the false periodic patterns or not regular patterns. Here are presented some of the most suitable approaches for the purpose of this work.

First and the most discussed approach for periodic pattern mining in time series database is called "Efficient Periodicity Mining in Time Series Databases Using Suffix Trees", it has been used as a main reference in many works, which appeared after the time of its publishing. It is based on a suffix tree of time series string traversing [11]. The idea of this work served as a starting point for most of the works discussed in this section. In this algorithm the first step for the periodic pattern mining process is to create a suffix tree of time series string.

Suffix tree is a widely used data structure for string processing. Suffix tree for a string represents all of its suffixes. For each suffix of the string there exists a particular path from the root of the suffix tree to a corresponding leaf node, see illustration in Figure 2.1. A string of length n can possibly have exactly n suffixes, consequently the suffix tree for such string contains exactly n leaves. Each edge in such suffix tree is then labeled by the string, which it represents. Each leaf node stores a number, which represents the starting position of given suffix and is being yielded when traversing from the suffix tree root to that leaf. Each intermediate node stores a number which represents the length of the substring when traversing from the root to that intermediate node. Each intermediate edge reads a substring (from the root to that edge), which is repeated at least twice in the original string.

Having a suffix tree for the given time series string constructed, the algorithm of [11] then has it traversed bottom-up from each leaf to annotate intermediate nodes. Each intermediate node in a suffix tree represents patterns as individual symbols or a time series string subsequence that repeats itself inside a time series string more than once. Then, after occurrence of such patterns is recorded, see Figure 2.2, the periodicity of each pattern is calculated. The [11] work is known to be the first algorithm, that overcame the necessity of periodicity specification from the user. It means, that instead of testing the pattern for periodicity of user input, it assumes a range of possible period values and gradually checks each value.

This method tends to outperform other algorithms mentioned in this study's related works section, however, together with other studies based on the suffix tree approach, it has a limitation, that is, if we use a suffix tree to generate patterns and detect periodicity, we will fail to generate flexible and interesting patterns. Furthermore, using a suffix tree, it is not possible to skip a particular character in a generated pattern. The inability to skip characters, which represent particular events in time series database, turned out to be a limitation in the domain of bank transactional data, as they appear to be less regular, than the data set of oil market prices, for example.

Noise resilience is a major factor that increases the quality of the result set

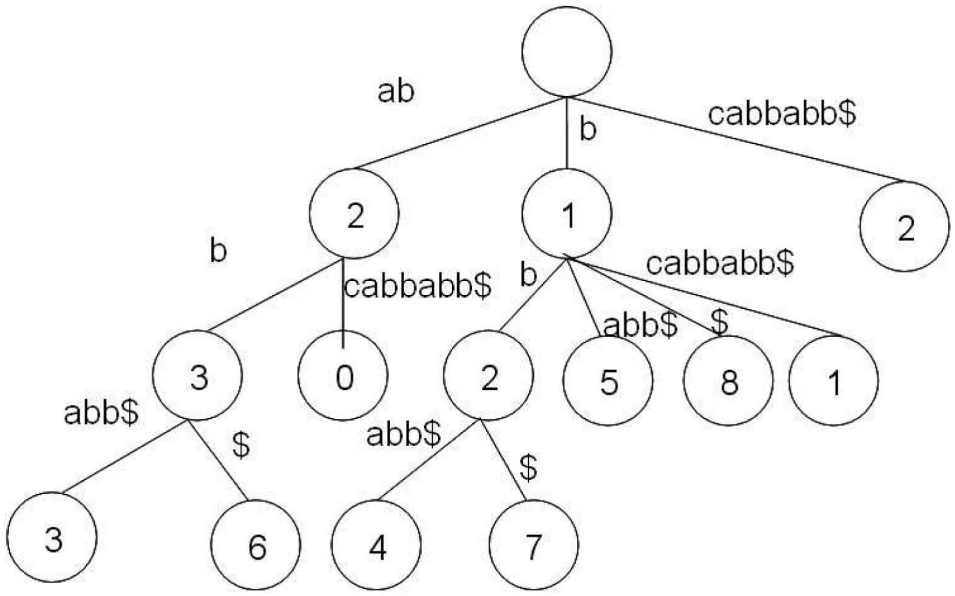


Figure 2.1: The suffix tree for string {abcabbabb$}, adopted from [11]

of patterns, eliminates unwanted output and also improves the performance of the mining process, as no redundant events are included in the pattern or undergo posterior analysis. However, it is needed to mention, that in terms of time series databases, noise in data seldom represents error data points that do not belong to the data. Even if an event acts as an outlier, it is still considered to be a meaningful record. Nevertheless, noise in the description of the algorithm's mechanism may actually represent an event that does not reflect value or interest in comparison to the others in time series database.

Another method, which was studied and taken into consideration, is the work of [12] that is called "A sequential pattern mining algorithm using rough set theory" and is an extended version of [13]. This work suggests the use of so-called itemsets. As the name of itemset suggests, it is a set of certain events in the time series database. This algorithm computes subsequences of a fixed size that are regarded as local patterns hidden inside sequences. A time series database is then represented not by symbols, which refer to a designated range of events, but rather by such itemsets. For instance:

- s_1 = aabcac
- s_2 = bca
- s_3 = cba

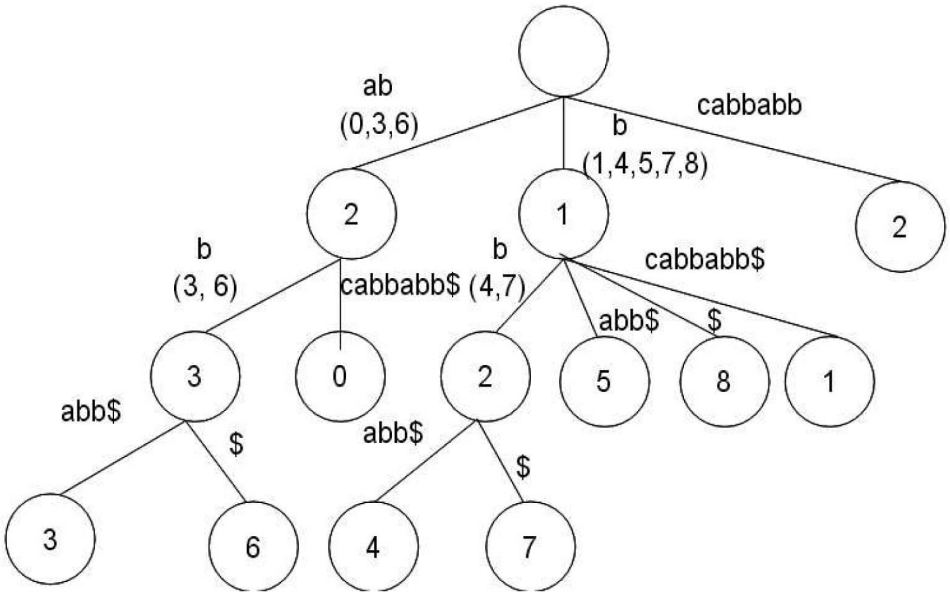


Figure 2.2: Annotated suffix tree of string {abcabbabb} after bottom-up traversal, adopted from [11]

The pattern mining algorithm then searches the time series database for the subsequence of an itemset present in different itemsets. The work of authors Kaneiwa and Kudo [12] includes exhaustive research that proves their algorithm to be effective. Nonetheless, the mentioned algorithm was not proven to be suitable for the purpose of this work, as itemsets turned out to be problematic data structure for bank transactional data.

Next work of Nishi et al. [7] titled "Effective periodic pattern mining in time series databases", also greatly refers to the work of Rasheed et al. [11]. The main point of interest of research in this paper was the requirement to overcome the limitation of generating flexible patterns by allowing event skipping in between interesting events. Using a suffix tree, it is not possible to skip a particular character in a generated pattern where the pattern is a combination of several characters and each one is the representation of each of the independent event in a time series database.

To get a clearer idea, consider a scenario where the time series database is represented as a string T as shown in 2.1:

$$T = \{abcd\ abed\} \tag{2.1}$$

Nishi et al. [7] argue that a suffix tree algorithm is only able to generate the eight types of patterns *abcdabed*, *bcdabed*, *cdabed*, *dabed*, *abed*, *bed*, *ed* and *d*.Assume the situation when the desired output represents generating a

pattern by skipping any intermediate symbol that stands for an unimportant event. It means, for example, that the desired result presumes “a”, “b” and “d” to be in the first, second and fourth position in the patterns respectively, and also the third positioned character is meant to be represented as don’t care event. By considering this description, the result pattern is $\{ab * d\}$.

However, when using a suffix tree, it is not possible generate patterns like the aforementioned $\{ab*d\}$ due to algorithm’s inability to skip any intermediate event in a generated pattern. Therefore, it is not possible to generate the pattern, which represents the combination of the important and unimportant events from the user’s point of view, using a suffix tree. As a consequence, from $T = \{abcd\ abed\}$, the algorithm proposed by Nishi et al. will result that $\{ab * d\}$ occurs in the position [0, 4] in the form of {abcd} and {abed} respectively.

The ability to generate flexible patterns in time series string is very important for the purpose of this work, if not the most significant. The records that represent bank transactions might not necessarily appear in the same order throughout time series. Also, it is likely, that in between important events of transactional data, there may appear additional less significant transactions, which must be possible to skip in order to successfully state the appearance of the pattern.

This ability of the algorithm to skip intermediate events in between the important ones is achieved by mining patterns in a special fashion and not using suffix trees, but by apriori based level-by-level sequential pattern mining approach to mine a specific pattern. First, single events are being mined and then the algorithm proceeds with generating gradually larger patterns by joining interesting and periodic smaller patterns in each pass. For instance, having time series string $T = \{accxacxdaxddbacx\}$, the gradual periodic pattern generating is illustrated in Figure 2.3 below:

Patterns of length 1

(a)	(b)	(c)	(d)	(x)
Occ_Vec	Occ_Vec	Occ_Vec	Occ_Vec	Occ_Vec
EID (a)	EID (b)	EID (c)	EID (d)	EID (x)
0	12	1	7	3
4		2	10	6
8		5	11	9
13		14		15

Patterns of length 2

(ac)				
		Occ_Vec	Diff_Vec	
EID (a)	EID (c)			
0	1	0	1	
0	2	0	2	…
4	5	4	1	
13	14	13	1	

(cx)			
EID (c)	EID (x)		
1	3	1	2
2	3	2	1
5	6	5	1
14	15	14	1

Patterns of Length 3

(acx)					
			Occ_Vec	Diff_Vec	
EID (a)	EID (c)	EID (x)			
0	1	3	0	[1,2]	
0	2	3	0	[2,1]	…
4	5	6	4	[1,1]	
13	14	15	13	[1,1]	

Figure 2.3: Periodic patterns of length 1-3 for time series string {accx acxd axdd bacx}, adopted from [7]

Nishi et al. [7] have adopted the periodicity detection and check algorithm from the work of Rasheed et al. [11] but also suggest, that in fact, any

periodicity check algorithm might be used after the patterns are constructed in each pass of pattern generating algorithm. Later, it was argued in [14], that it fails to find some significant periods. For instance, for the time series string $T = \{abcc\ abdc\ acdc\ abdc\}$, the occurrence vector or the indexes of time series string is {2, 3, 7, 9, 11, 15}. As the previous periodicity detection algorithm as shown in Figure 2.4 searches periods linearly, it fails to generate patterns with period = 6, which can be easily found from the difference of the 2^{nd} and the 4^{th} occurrence vector elements 3 and 9. The same period value is also identified for the 4^{th} and the 6^{th} occurrence vector elements 9 and 15, respectively. As a consequence, a good amount of periods remain undetected.

```
Input: a time series database of size n
Output: positions of periodic patterns

foreach occurrence vector, occur_vec of size k for pattern X do
    for j = 0 to k do
        p = occ_vec[j+1]-occ_vec[j]
        stPos = occ_vec[j]
        endPos = occ_vec[k]
        for i = j to k do
            if (stPos mod p==occur_vec[i]mod p) then
                increment count(p)
            end
        end
        conf(p) = count(p)/ PP(p, stPos, X)
        if (conf(p) >= threshold) then
            add p to the period list
        end
    end
end
```

Figure 2.4: periodicity detection algorithm, adopted from [11]

The work of [14] "An efficient approach to mine flexible periodic patterns in time series databases" presents a very interesting approach to the given problem. It utilizes the suffix trie, see Figure 2.5, as a data structure for finding all the repeating subsequences of time series string, and also allows skipping events to the maximum number that is defined by the user. However, the improvements that were made in this work, that distinguish performance level from several other existing approaches, including [7] and [11] were, as it appeared, hardly achievable in practice. Among them are: the assumption that period length must be known in advance and the very strict threshold of allowed skipped events. The problems with period length and regularity, together with capacity to use event skipping threshold are discussed in 4.2.1.

The most recent studies of periodic patterns "A new framework for mining weighted periodic patterns in time series database" [15] suggest, that existing algorithms generate vast amount of non-interesting patterns in dense databases that are not important enough to participate in the decision making process. This algorithm is once again based on the suffix trie traverse principle allowing event skipping and, as a novel feature, supporting different

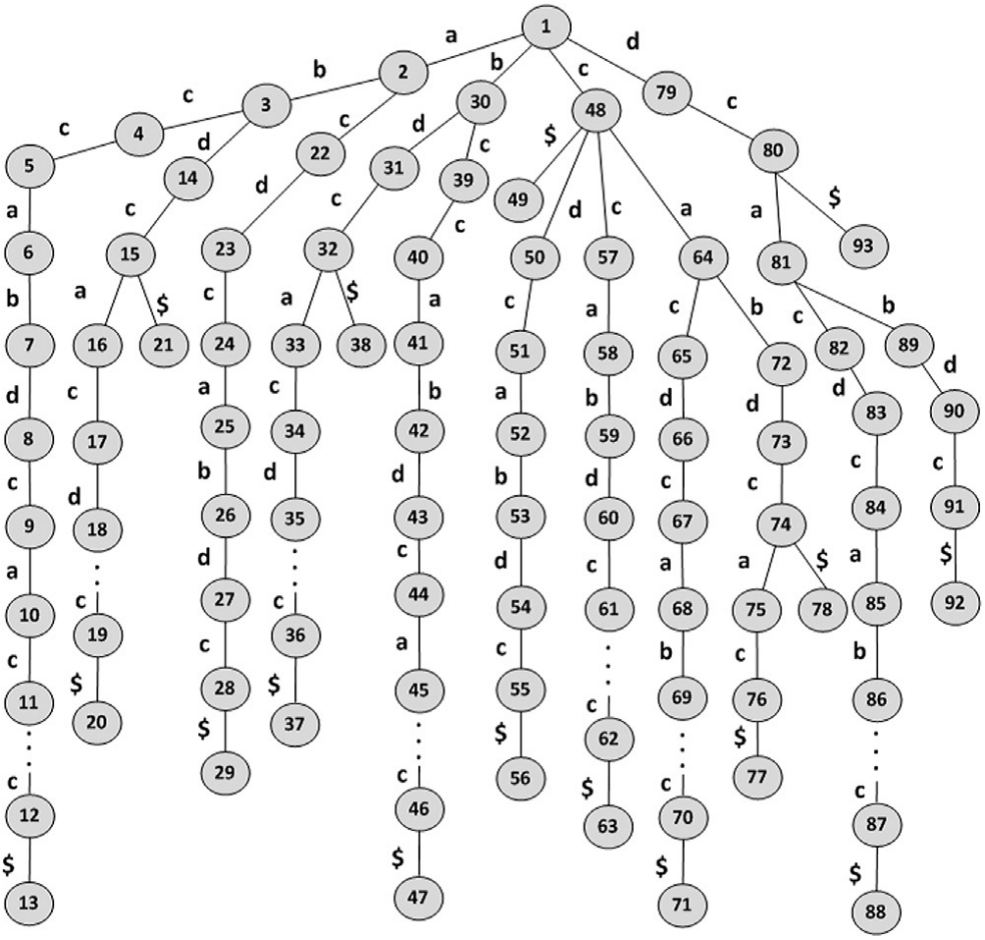


Figure 2.5: Suffix trie of {abcc abdc acdc abdc$}, adopted from [14]

weights to prioritize events or items in time series database. Even though the idea of weights in sequence pattern mining was researched before, e.g. work of [16], the authors of [15] succeeded in creating an algorithm that is modern and considers all of the achieved improvements of the previous studies.

In my opinion, the latter work might have interesting use for the project owner, but only after they become acquainted with the problem and output in full range. Another prerequisite for adapting these ideas from [15] is to be able to specify time series database with a fixed length of the period for all of the subjects (bank clients) which seemed to be problematic at the moment of writing this thesis paper. Before that, it is more of an advantage that chosen algorithms, make an exhaustive search of time series database and produce many patterns, taking into consideration every particular event with the same importance level.

Another interesting work is concentrated on the problem of outlier periodic patterns in time series databases. "A Framework for Periodic Pattern Detection in Time-Series Sequences" [17] argues, that most of the existing algorithms (at the time of publishing in 2014), that detect and mine periodic patterns in time series data, tend to produce vast amount of redundant patterns, that appear not to be interesting, nor are valuable in decision making processes. To overcome this tendency, they focused on the patterns in time series, which are still periodic, but less frequent and thus, more valuable for the user. Such unusual patterns in general time series may represent, for example, a decline in the economy or unusual natural phenomena.

The mining process of these outlier patterns begins once again with suffix tree of a time series database that was discretized to a string. After detecting all of the repeated subsequences in a string, one extra step in the analysis appears – every pattern in the set is being tested for its surprise level. *Surprise level* of the pattern shows, how unusual this pattern is in comparison to other patterns with the same length. After periodic outlier patterns were nominated, the algorithm proceeds with periodicity check for each pattern. Unlike previously mentioned periodicity detection algorithms, the one that is used in [17] does not require strict periodicity.

2.2 Chosen algorithms

The algorithms that I used as an inspiration for my work are based on the idea of sequential pattern mining in time series databases that were described above. However, for the purpose of this work, none of these approaches appeared to be suitable for a straightforward implementation, due to particular constraints such as the period length of bank transactional data that is described in 4.1 and the capacity to determine the maximum event skipping threshold. The alternations and extensions that were needed to apply first, are described in section 4.2.1.

In order to create an extensive pattern database, it was decided, to preprocess the given dataset in order to assemble it in accordance to the definition of the time series database. This process is described in detail in section 4.1. After the given dataset is preprocessed, its records are discretized to the set of so called symbols and time series database is then treated as time series string. Finally, my chosen algorithms are: [7] for detection and mining frequent periodic patterns as it allows the most exhaustive time series string search for presence of all types of patterns: symbol, sequence and segment patterns. This is achieved through the flexibility of a maximum skipping event threshold that may be set in advance as big as the period length itself, or even omitted. To escalate the effectivity of the periodicity check of found candidate patterns, I decided to use the patterns' periodicity detection algorithm from [14] in time series data. This will guarantee, that no interesting patterns are

left abandoned and therefore will not proceed in the next level of the pattern generating algorithm.

For detecting periodic outlier patterns, I added an optional extra step in between the pattern detection and the pattern periodicity check in the form of pattern surprise level check that is inspired by the work of [17]. That said, when patterns of every length are nominated to be periodic in every pass of the algorithm, only patterns that achieve the necessary user given surprise threshold level proceed to the periodicity check. This approach is then a form of pattern mining algorithm extension.

Chapter 3

Analysis

In this chapter I'd like to introduce the analysis that was made upon the given data set of the project owner's transactional bank data in order to define the necessary data preprocessing steps and identify the required data form.

One of the requirements for the implementation was the ability to approach data from different sectors in the same manner. This means that companies with different amounts of partners, transactions and turnover category must be still comparable in terms of their interaction with other market subjects, as the objective of the analysis wasn't creating clusters of similar companies, but rather investigate the similar interconnections.

Another important feature, that was meant to be available as part of the implementation, was the capacity to change the perspective of the time period. Whether it is a year, a month or even a week or a day, it should be available to mine patterns of different time granularity from the same data set and then, compare the different output sets.

The main challenge in the implementation was the data fuzziness and the need to adjust methods that are oriented to the exact output and do not allow semantic interpretation. This means, that sometimes, the expected results of periodic pattern mining might not only be handled differently, in terms of business meaning, but also, such patterns might allow a certain level of tolerance towards a set of events in the resulting pattern.

First, there was a problem of data representation and preparation. As an alternative to creating a continuous line graph that would represent the constant development of the balance of one company, it was decided to keep the discrete nature of data and rather than monitoring the balance curve of one company, instead keep track of its interaction with other business subjects. Not only would it allow to create more efficient data transformations, but also, the aggregation and dimensionality reduction would be less of a problem.

For this purpose, I have suggested the aggregation and discretizing technique, meant exactly for the transactional bank data, to transform data set of bank transactions into a sequence of events. The technique and implementa-

tion are described in chapter 4.2.2.1.

Second, I managed the problem of period length. The result sequence of event did not and could not have the same length of period due to:

- Different clients interact with different categories and amount of other subjects
- The bank records for each client naturally start at different a moment in time
- There could be any amount of skipped events (the absent payments towards a particular group of adverse parties)

More details about overcoming this limitation is presented in section 4.2.2.3.

Anomalies in time series in a domain of bank transactional data can be observed in two ways: where the first is the unusual events in a set of pattern events that appear less frequently. Imagine a set of usual transactions that lead to one particular event that happens every 3 years. These patterns don't have to be extensive, these anomalies can also be represented as single events in transactional history. The second way to imagine the observation of anomalies in bank data are certain trends that happen to be sector-specific. To detect such anomalies, it is needed to aggregate data without considering individual subjects, but rather whole sectors in general. These outliers can be obvious from the data visualization, however, due to the range of the data, it is needed for them to be searched automatically.

3.1 Data structure, example, statistics

For the purpose of creating suitable data procession steps and testing, the data set was extracted from the historical data of the project owner's database cluster. The structure of the data extracted from the database doesn't have to be strict. However, it is recommended, that the result data set has the same structure and properties, after following the steps of preprocessing and aggregating.

The data set that was given to be used as an inspiration and test data for the purpose of this work was represented as bank account transactions history. That included these parameters:

- Transaction id
- Transaction timestamp
- Account id of transaction origin
- Type of transaction

- Account id of transaction adverse party, including bank code
- Transaction amount

Original set of data did not contain many attributes and they were mainly added later through joining other tables with information about bank clients. That said, information about recipient of transaction, in case it is not client of bank, are limited. However, some attributes were still available:

- Party Id, derived from account id
- Party Id NACE code: identification of company economical activities
- Information about NACE code of adverse party

As it was mentioned before, it was agreed, that the analysis would only be held on the existing data, so, further on, data set will be referred to as historical data. Statistical description of a given historical data is described in the Table 3.1

The sample data set that was extracted from historical data, using Apache Impala, consists of transactions covering years 2011 – 2016. The transactions in the sample set were taken for the companies with primary NACE codes 1, 41, 42, 43, 55, 77 and turnover categories 1, 2 or 3 for each NACE code respectively. In each category there are approximately 100 companies.

Table 3.1: The given data set statistics

Description	**Value**
Total amount of transactions	41645807
Total amount of transactions with known NACE of adverse party	32026582
Period of time covered	1/2011 – 12/2016
Total PartyIds	1623

Project owner's historical data are stored in the format shown below. Table 3.2 contains several examples of transactions. Table 3.3 contains the description of the derived attributes.

Table 3.2: Example of the transactional data records

id	**tran_ date**	**tran_ time**	**account**	**tran_ sign**	**tran_ amount**	**adv_ bank**	**adv_ account**
20***14	2016-09-27	03:17:00	00***08	1	600.00	2010	00***77
20***20	2016-09-27	13:40:00	00***47	1	1500.00	0300	00***07
20***43	2016-09-27	16:56:00	00***97	1	30000.00	0100	00***87
20***31	2016-09-27	15:49:00	00***37	1	1000.00	0100	00***00
20***09	2016-09-27	10:47:00	00***67	-1	11000.00	0100	00***57

Table 3.3: The description of the derived attributes of the transactional data records

Attribute	Data type	Descriprion
id	decimal(16, 1)	Unique identifier of the tranasction, primary key
tran_date	date	The date of transaction
tran_time	string	The time of the transaction
account	string	The account number of the Party Id
tran_sign	int	The sign of the transaction where "1" stands for the incoming payment and "-1" for the outgoing one
tran_amount	decimal(17, 2)	Transaction amount in local currency
adv_bank	string	Code of the adversary party bank
adv_account	string	Account number of the adversary party

The *Party Id* that is mentioned in the *account* description refers to the bank's customer. There is a relation 1 to N between Account number and Party Id in the project owner's database, suggesting that one customer – person or company – may have several bank accounts at the same time, see Figure 3.1. The only bank clients that were chosen for this analysis are companies, which have their corporate accounts at the project owner's bank. For the sample set, transactions from several bank accounts of clients were merged together under one Party Id identification.

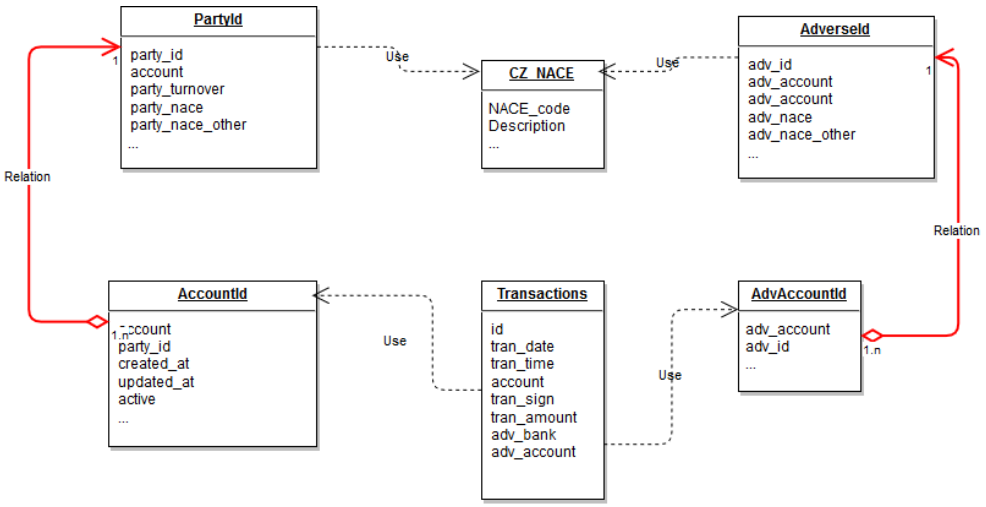


Figure 3.1: The simlified database diagram of project owner's historical data

Another relation is between Party Id and CZ_NACE and is of type 1 to (0, 1). In most cases, the main field of activities of a bank customer is known

and is described by so called NACE code.

"NACE code is the classification of economic activities in the European Union (EU). NACE is a classification providing the framework for collecting and presenting a large range of statistical data according to economic activity in the fields of economic statistics (e.g. production, employment and national accounts) and in other statistical domains developed within the European statistical system (ESS)" [18].

It is worth mentioning, that common practice for companies is to have several NACE codes listed as a description of its activities, while having one as primary. That primary NACE code is a major descriptive attribute and will be used in the further analysis. When aggregating companies in clusters, only the first two digits of the primary NACE were used.

Similarly to NACE codes, another characteristic is the company's turnover category. This attribute may have values of:

- 1 : 1 - 29 999 999 CZK
- 2 : 30 000 000 - 99 999 999 CZK
- 3 : from 100 000 000 CZK

The transaction data set was then joined with additional parameters such as the customer's primary NACE code and turnover category and the adverse party NACE code, if available. The prerequisites for the sample data set were:

- Party Id must be a company with a corporate account or accounts at the project owner's bank
- Party Id's account history must have at least 5 years of records
- Additional data like primary NACE code and turnover category must be available

Table 3.4: The description of the transactional data joined with the additional attributes

Attribute	**Data type**	**Descriprion**
id	decimal(16, 1)	Unique identifier of the tranasction, primary key
tran_date	date	The date of transaction
tran_time	string	The time of the transaction
party_id	bigint	The bank client company that originated the transaction
party_nace	string	Description of company's main economic activity
party_turnover	int	The range of company's yearly turnover
tran_sign	int	The sign of the transaction where "1" stands for the incoming payment and "-1" for the outgoing one
tran_amount	decimal(17, 2)	Transaction amount in local currency
adv_bank	string	Code of the adversary party bank
adv_id	int	Identification of the adversary party company
adv_nace	string	Description of adverse party conpany's main economic activity

The result data had the form that is shown in Table 3.4

Chapter 4

Design

This chapter describes the process of designing an approach for frequent periodic pattern mining, embodying it into an analytical tool and discussing the results and experiments that were made upon the given sample set of the project owner's data.

It contains prerequisites for input data, the preliminary data preprocessing, and the description of the algorithms that were implemented. Then there are described the constraints that were encountered during this implementation and experiments with algorithm parameters, the alternations and extensions that took place in the course of implementation.

The rest of the chapter is organized as follows: the first part is dedicated to the description of input data form and the statistics of the sample set provided by the project owner. The second part contains the description of the algorithm for periodic pattern mining that was adopted, its implementation and extension. The third part is a study of the experiments with framework attributes. The fourth part describes the anomaly detection approach, its description, implementation and results.

4.1 Data requirements

This chapter consists of the description of the preliminary work that was made upon the given data set. Partly, this topic was already discussed in the previous chapters, namely section 2 and 2.1, what type of time series data representation will be chosen and reasons for this choice. Here, I describe the raw data and its characteristics, such as structure and attribute.

It is worth mentioning, that the solution in this work is being designed based on the project owner's requirements and that otherwise, the structure of input data is not mandatory and may vary in other implementations. The important points are the final data set attributes and data types.

The data set is described in the next sections.

4.1.1 Data preprocessing and aggregation

As it was already discussed before, for the purpose of this work, the representation of time series was chosen in the form of a database as an ordered set of events, distributed in time over equal periods of time. However, the raw data set's unique transactions do not satisfy one of the main conditions of time series definition, as per one day there might be several transactions just as there might be none. In order to balance such irregularity, it was needed to first adapt the data.

While there are several ways to do so, including choosing random data points, smoothing the data points' curve, using regression, most of these approaches are leading to a particular loss of data. On the other hand, setting data points' granularity to just one second as a minimum regular period is not a meaningful representation of transaction history for this particular analysis, as the amount of transactions per day differ from company to company. See Figures 4.1, 4.2 below. Histogram of amount of transactions per day from history data that covers 1 year and represents a total of 109000 transactions of 210 companies.

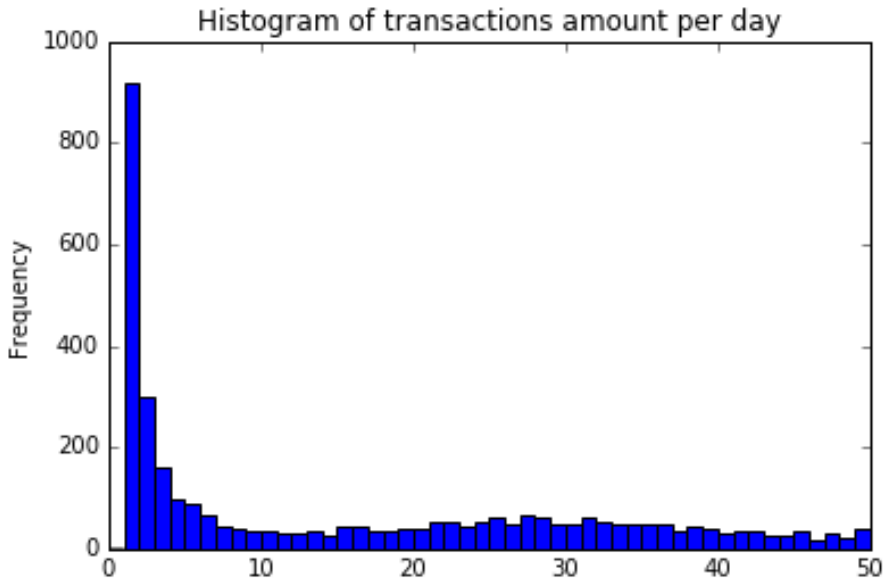


Figure 4.1: Histogram of example company's transactions distribution, amount 0 - 50

Aggregation however, will preserve the important attributes such as identification of the adverse party and the total transaction amount. Naturally,

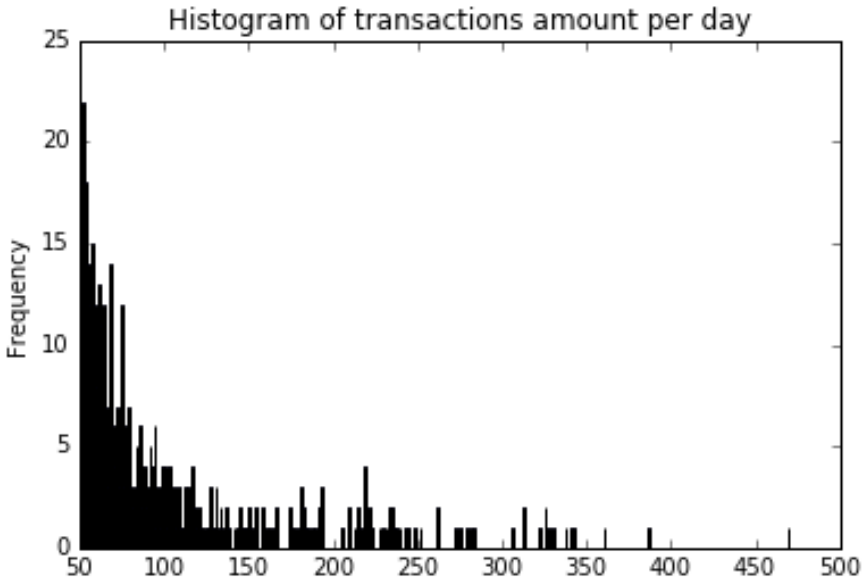


Figure 4.2: Histogram of example company's transactions distribution, amount 50 - 500

in order to be used effectively, the data set was aggregated into meaningful records. Such aggregated time series data are then represented in the form of data, grouped by a combined key, where all the records' attributes are part of that key, except for the data's studied numerical value.

Another reason to create an aggregated set of records, instead of choosing data points of, for example, a balance curve of one particular company, was to retain the whole perspective of the cash flow. Keeping the records with more than two attributes, such as timestamp and balance value, allows to see into data with a wider perspective. It allows to not only monitor one state throughout the time, but also follow and notice differences in a particular company's or the whole market's behavior from the position of cooperation.

So it was decided with the project owner, that records, which describe transactions or a set of transactions will be kept in the form of events, rather than data points. In order to approach transaction data as a sequence of events, it has to indeed be aggregated and reduced to a meaningful number of records. On the other hand, the resulting events must still be descriptive and, in a way, unique. After consulting with the project owner, it was decided, that monthly granularity is a sufficient trade-off between the representativeness of data and a meaningful amount of records. The important attributes were preserved:

- Customer's identification, including turnover category
- The general identification of adverse party as a description of financial operations
- Time granularity
- The sum of transaction amount

To identify the bank customer both composed_id and turnover category were used. The final data set then has the following structure as shown in Table 4.1:

Table 4.1: The description of the transactional data aggregated for the analysis

Attribute	**Data type**	**Descriprion**
composed_id	bigint	The composed id of bank client, used for the purpose of the data anonymization
nace_subject	int	The primary NACE code of the client – only first two digits
nace_partner	int	The primary NACE of the adverse party - only first two digits
tran_sign	int	Describes the nature of the transaction
turnover_category	int	As described in the analysis section
tran_month	int	Transaction month
tran_year	int	Transaction year
tran_amount	float	Sum of transaction amounts for this aggregated record, in CZK
tran_count	bigint	Amount of transactions, that belong to this aggregated record

4.2 Periodic pattern mining in time series databases

In this section I describe the algorithm adaptation steps, together with the constraints that were encountered in the process of implementation. Also, here are showcased the original approach ideas and the algorithm alternations, that were created with the aim to better suit the objectives of this work.

As it was mentioned before, the approach for periodic pattern mining in this work was mainly inspired by the "Effective periodic pattern mining in time series database" algorithm. Although, I did not use the whole implementation and suggested numerous adaptations and extensions, here I provide the description of Nishi et al. [7] work.

Input: a time series database of size n, Users interested pattern $\Phi = E_1(*)^{Y_1}E_2...E_{i-1}(*)^{Y_{i-1}}E_i$, Confidence Threshold σ, Maximum Event skipping Threshold θ
Output: positions of periodic patterns,The occurrence vectors and The number of times the period is generated in that period

```
1  begin
2    Perform Discretization Operation and Construct event-string sequence, S
3    foreach Alphabet [a...z][A...Z], α ∈ S do
4      Construct Occurence_Vector_α by processing S
5    end
6    Set Occ_Vec := ∀_α{Occurence_Vector_α} where α ∈ S
7    Set Φ' := ELIMINATE_STARS(Φ)
8    Set phase := LENGTH(Φ')
9    Set Pass, i := 1
10   while Occ_Vec != NIL and i ≤ phase do
11     Generate Patterns of Length, i by Joining any two i − 1 length patterns P1 and P2, i ≥ 2 and
       SUFFIX_{i−2}(P1) = PREFIX_{i−2}(P2)
12     Store the Patterns in P
13     foreach generated patterns e1, e2...ei ∈ P , i ≥ 2 do
14       Calculate Occurrence vector X1, X2, X3,...,Xi by Joining the occurrence vectors X1, X2, X3,...,Xi−1
         where EId(X1) <EId(X2) <EId(X3) <EId(Xi−1) <EId(Xi)
15       foreach occurrence vector do
16         Calculate Difference vector Z1, Z2, Z3,..., Zi−1
17         for (j=1 ; j <i-1 ; j++) do
18           if (Yj ≤ Zj) then
19             Assert "Search Pattern is absent"
20             Exit()
21           end
22         end
23       end
24     end
25     foreach occurrence vector of size i for pattern p do
26       for (j=0;j<n/2;j++) do
27         q=occur_vec[j+1]-occur _vec[j];StPos=occur_vec[j];endPos =occur_vec[i]
28         for (l=j ;l<i ; l++;) do
29           if stPos mod q==occur vec[l]mod q then
30             increment count(q);
31           end
32         end
33         conf(q)=count(q)/Perfect Periodicity(q,stPos, p)
34         if (conf(q) >= threshold) then
35           add q to the period list;
36         end
37       end
38     end
39     foreach Pattern, p ∈ P do
40       Calculate conf = CONFIDENCE(p)
41       if (conf ≥ σ) then
42         Set P' = P' ∪ p
43         Set Occ_Vec := Occ_Vec ∪ Occurence_Vector_p
44       end
45     end
46     Set P = P'
47     Set i := i + 1
48   end
49   foreach Pattern, p ∈ P do
50     foreach difference vector do
51       set p' := Generate e1(*)^{A1−1}e2(*)^{A2−1}e3....ei−1(*)^{Ai−1−1}ei Patterns
52       Calculate periodicity of p' using the periodicity of p
53     end
54   end
55 end
```

Figure 4.3: Pseudocode of periodic pattern mining algorithm, adopted from [7]

4.2.1 Original algorithm

The idea of this method is to effectively find all the repeating sub-sequences of events from a time series database that appears to be frequent, periodic and interesting. By interesting, authors recognize patterns or sub-sequences, consisting of events that are of interest to the user and that don't take into account all of the "don't care" or skipped intermediate events. The pseudocode of the algorithm is illustrated in Figure 4.3 above.

The initial step in this method is to apply discretization technique to the time series database. The discretization can be thought of as a mapping among the range of values of an entity and an ASCII character which represents a specific event and can be defined as a function of v, $f(v)$ see formula in Figure 4.4:

$$f(v) = \begin{cases} S(r_0), & \text{if } v \in r_0 \\ S(r_1), & \text{if } v \in r_1 \\ \dots & \\ S(r_{n-1}), & \text{if } v \in r_{n-1} \end{cases}$$

Figure 4.4: Discretization function, adopted from [7]

where, $S()$ is a function, that returns a specific symbol based on the given value r_1 and r_0, r_1, ..., r_n are ranges defined for the value of an entity [7].

In other words, there exists particular symbols for designated range of events. The discretization function takes every event in the time series database and returns the respective symbol pertaining to that event. The final set of symbols is then represented in the form of string, having all symbols set in the same order they were sorted in the original data set. This string is later used in the mining process.

Having original data set discretized, the next step is to generate interesting patterns, starting from patterns of length 1, and incrementing length with each pass, until the user's defined length is achieved or no more patterns can be generated, by joining interesting and periodic patterns. To start with, the patterns consisting of single events are first mined. For every single event, the occurrence vector of its appearance inside the time series string is recorded in the form of a set of indexes inside the string. An occurrence vector is also constructed for patterns in each pass.

After an occurrence vector is constructed, the algorithm generates all possible exclusive interesting patterns by allowing skipping intermediate events. The allowed number of skipped events, also referred to as not interesting events, and described as don't-care symbol or "*" is defined as maximum event skipping threshold θ. That means that for every found pattern, for instance {abc} and having maximum event skipping threshold $\theta = 1$, the time series string is searched for different versions of {abc} like {a*bc} or {ab*c}.

Each pattern in the set is then checked for periodicity. Periodicity describes how frequent a given pattern is within the board of a certain period value. Perfect periodicity is the number of occurrences that was theoretically possible, given the first occurrence of the pattern in time series string and pattern's period. Confidence is then calculated as a proportion of the pattern's actual periodicity and perfect periodicity for given period, see Formula 4.1. Patterns that satisfy the condition of confidence threshold are kept and proceed to the next algorithm iteration.

$$conf(p,\ StPos,\ X) = \frac{ActualPeriodicity(p,StPos,X)}{PerfectPeriodicty(p,StPos,X)} \tag{4.1}$$

with $conf$ = confidence level of periodicity of pattern X, with periodicity p and starting position $StPos$.

Increasing the length of generated patterns is possible by joining two existing patterns in the set. In order to suggest a new pattern in the pass i of algorithm with pattern length i, the algorithm merges together two patterns of length $i-1$ if the suffix of length $i-2$ of the first pattern is the same as the prefix of length $i-2$ of the second pattern. Events in both patterns must be ordered sequentially.

4.2.2 Algorithm adaptation

While approaching the implementation of the [7] algorithm, it was clear, that some functionality aspects might not be used without adapting the algorithm first, given the special domain of data structure used in this analysis. In sections 4.2.2.1 – 4.2.2.6 I'd like to describe the alterations that were made upon the original ideas and definition, and also describe my suggested extensions.

From this moment, it is important that we distinguish the two meanings of the term "period", as it may become confusing in some literature, when it is used together with different connotation. The first meaning is connected with time series database, as it contains data taken over regular intervals of times. For instance, consider the string of events "abcd abdd abcd aabb" from a time series database, which represents values in a measuring device that are taken every 6 hours, every day. It can be said, that the period of time series string is one day and the period value is 4, hence every 4 measures signifies one day of measures. Then, it can be said, that two events, for instance "a" and "b" belong to the same period, while two events "a" don't.

The second meaning of the term "period" is connected with the periodicity of patterns. For example, pattern "a" from the previous example appears in every period in positions [0, 4, 8, 12, 13]. Therefore pattern "a" is periodic and its period value is 4, with occasional deviations that are tolerated. Pattern "c" however appears in every second period of string in positions [2, 10]. So the period value of pattern "c" is 8 – or two periods of time series string.

One of the main constraints in implementing the work of Nishi et al. [7], which required a special approach, was the irregularity of the period length of time series database. Even after the data set was preprocessed and aggregated into time series database, particular gaps still exist. The period of such time series database is a year, or, for companies with more dense transactions history and more various interactions, a month. During a period of time series, there could be found various aggregated transactions with different partners. That said, the set of partners, amount of transactions and its density differ from period to period.

My first idea of how to deal with such irregularities was to fill the empty slots of a missing group of transactions with zero value symbols. However, not only did that increase the length of original time series database tremendously, sometimes as much as 18 times, and therefore affected the algorithm's effectivity, but also provided numerous meaningless outputs that reduced the overall quality of the result pattern set.

Here is an example of the original dataset with numerous gaps and the artificially stretched one in the Tables 4.2, 4.3.

Table 4.2: An example of the aggregated time series of transactional data

	composed_id	**nace_subject**	**nace_partner**	**tran_sign**	**turnover_category**	**tran_month**	**tran_year**	**tran_amount**	**tran_count**
0	77184	77	46	1	1	4	2015	7000	1
1	77184	77	01	-1	1	9	2015	29187	1
2	77184	77	01	1	1	9	2015	1407	1
3	77184	77	01	-1	1	7	2016	5500	1
4	77184	77	01	-1	1	8	2016	2017	1
5	77184	77	01	-1	1	12	2016	11859	1

The irregularity of the period length also affected the possibility to use θ threshold and its utility measure. It is visible from the example above, that as soon as it cannot be guaranteed that the period length will be set to have equal length, it becomes problematic to set the maximum amount of skipped events. Whether it can be a month, a year or just a subset of records of a particular adverse party, the θ threshold is dynamic and cannot be set or calculated for the whole time series database in advance.

Another constraint was the impossibility to use a single symbol that would represent the whole range of events. Even though 95 ASCII symbols might be sufficient for the data of one Party ID, it is not enough to use across the whole time series database, even after the data set was cleaned and aggregated.

The next sections describe the modifications that were required in order to implement the algorithm mentioned above.

Table 4.3: An example of the aggregated time series of transactional data, that was artificially stretched

	compo-sed_ id	**nace_ sub-ject**	**nace_ part-ner**	**tran_ sign**	**turno-ver_ cat-egory**	**tran_ month**	**tran_ year**	**tran_ amo-unt**	**tran_ count**
0	77184	77	1	-1	1	1	2015	0	0
1	77184	77	1	1	1	1	2015	0	0
2	77184	77	46	-1	1	1	2015	0	0
3	77184	77	46	1	1	1	2015	0	0
4	77184	77	1	-1	1	2	2015	0	0
5	77184	77	1	1	1	2	2015	0	0
6	77184	77	46	-1	1	2	2015	0	0
7	77184	77	46	1	1	2	2015	0	0
8	77184	77	1	-1	1	3	2015	0	0
9	77184	77	1	1	1	3	2015	0	0
10	77184	77	46	-1	1	3	2015	0	0
11	77184	77	46	1	1	3	2015	0	0
12	77184	77	1	-1	1	4	2015	0	0
13	77184	77	1	1	1	4	2015	0	0
14	77184	77	46	-1	1	4	2015	0	0
15	77184	77	46	1	1	4	2015	7000	1
...									
32	77184	77	1	-1	1	9	2015	29187	1
33	77184	77	1	1	1	9	2015	1407	1
...									
72	77184	77	1	-1	1	7	2016	5500	1
...									
76	77184	77	1	-1	1	8	2016	2017	1
...									
92	77184	77	1	-1	1	12	2016	11859	1

4.2.2.1 Discretize the events

In order to create a set of events for further analysis, data records with multiple attributes, event by event, were supposed to be represented as a simple ASCII symbol. However, it was obvious, that this amount of symbols was not enough to describe each event in the bank transaction time series database history. I suggested a technique of discretizing every event that took into account three main attributes:

- The NACE code of the adverse party

- The nature of the payment – income or expense
- The amount of payment

See Table 4.4 for details.

Table 4.4: An example of composed symbol in the proposed discretizing technique

NACE code	**Transaction sign**	**Binned transaction amount**	**Description**
61	0	001	Payment towards adverse party with NACE code 61, with transaction amount (0, $1 * bin\_size$]
52	1	07	Payment from adverse party with NACE code 52, with transaction amount ($6 * bin\_size$, $7 * bin\_size$]

To use the particular identification of the adverse party was not possible as these data were deprecated during the data aggregation and even if such data were still available, the uniqueness of each event would not allow the pattern to be discovered in the other companies' data.

NACE code was then represented as the symbol's two first digits. The nature of the transaction was simply replaced with an integer next to the NACE code, where 0 stands for the outgoing payment and 1 for the incoming one.

The transaction amount in Czech crowns should also be binned, of course, as putting the actual transaction amount into an event 'symbol' would once again create a set of unique events rather than grouping and describing the similar ones. My initial approach was to create a binning system, based on the statistical data of payment amounts for each subject. That said, I looked into the data, particularly, the median of the transaction amount for the given Party ID and then declared, that 50% of events should fit into 3-10 bins, based on the maximum amount value in the data. It might have created much more bins on the right side, but most of them would be empty, so the uniqueness of the event if not affected. This way, the pattern itself would be easily decoded from the event it contains.

For example, an event in the result set with name "id_1234_bin_1000" and symbol "350003" would be translated to: the outgoing payment to partner company with NACE code 35 and of amount in range 3000 to 4000 CZK. The highest amount of transaction in this set could be in a range of 1000 crowns x 100 and 1000 crowns x 999 as there are three digits in the symbol, reserved

for the bin identification. The precise bin range of 1 thousand crowns will of course be evident in the particular event.

Another approach, that I considered, was to create the same amount of bins on both left and right side from the median amount. That would create a similar output, but the data representation would be lost, as an additional track of transaction data should be kept onside. The common practice is, that the distribution of the transaction amount is more regular on the left side from median, than it is on the right side. Therefore, this approach would also put in the same bin, transactions with approximately 1 standard deviation from mean together with the absolute maximum, see Figure 4.5.

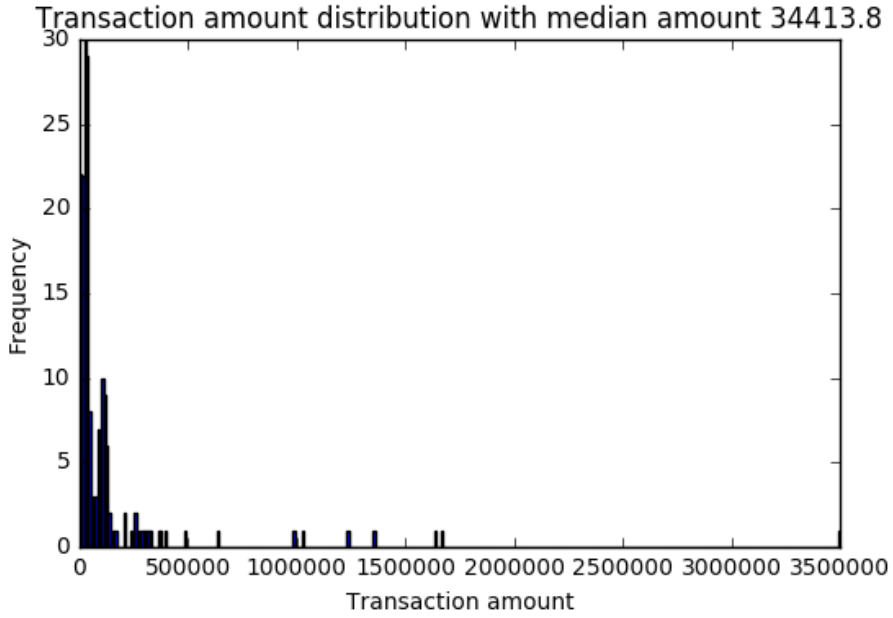


Figure 4.5: Example of transaction amount distribution

My next personal approach was to use the logarithmical value of the transaction amount. That approach would serve as binning itself, giving the idea of transaction amount. However, for companies with a wide range of amounts of transactions, it might create an unnecessary precision of binning. This way, logarithmical values might be binned as well. Therefore, an additional option when using this approach was to put the binned logarithmical value into the transaction symbol.

The last approach was to first analyze the nature of both outgoing and incoming payments of adverse party. That said, we cannot easily compare two patterns containing the same event, for example “350000”, if in the first case the bin range of 1000 crowns was used and in the latter case 5000. So

instead of doing additional work post factum, it was suggested to create a particular range of bin for every adverse party. The record of NACE codes, nature of payment and bin range should be kept aside, but the comparison of every pattern throughout the whole result set would be easier.

4.2.2.2 Create the occurrence vector

The first step in pattern generating is to get the patterns of length one, in other words all the unique events. The time series string is then searched for all the occurrences of single events. Once that's done, patterns with single occurrence are filtered out, as clearly, they cannot be periodic.

Most of the works listed in section 2.1 suggest, that from the moment of discretization, the time series should be treated as a string of symbols. However, it must be mentioned, that such a string assumes the same length of all the periods it represents, as a structure of data points taken at the equal time deltas. For example, string "aabd aabc abbc abcd" stands for the time series of a total of 16 events, with 4 periods of length 4.

However, the usage of this implementation has its limitations, as it cannot be guaranteed, that the given data will have the same amount of events in each period. As it was already illustrated in 4.2.1, some periods or its sectors, respectively years and months in time series database, might be missed or incomplete in general. It was decided, to keep the table organization of data, in order to maintain the period denotation alongside the event and its index in the time series.

In this manner, the occurrence vector does not only consists of a pattern identifier and an array of its positions in time series, but also, an array of period identifiers that refer to those positions. Consider an abstract example in the Table 4.5, which shows an occurrence vector of two events that were discretized in the manner that was described above.

Table 4.5: Occurrence vectors of two distinct events

Pattern	**Occurrence_vector**	**Period_vector**
"610001"	12	2011
	32	2012
	35	2012
	46	2013
"351000"	11	2011
	14	2011
	36	2012

This additional attribute will be needed while generating patterns of length two and up.

4.2.2.3 Increase the length of pattern

A pattern of length i is constructed from two patterns of length $i-1$ if the suffix of the first one matches the prefix of the second one. First, we compare the suffix and the prefix of all the existing patterns pairwise and if they match, we suggest it can be a new pattern.

Generating patterns of length two is different from generating patterns of length three and up. When generating patterns of length two, the step of comparison suffix and prefix is omitted as clearly, patterns of length one, from which those patterns are generated cannot contain neither suffix, nor prefix. So a pair of single events is nominated to be a new bigger pattern if the first event happens to be situated before the second one. That is a compulsory condition of ordering events in a pattern in sequential order.

Another condition is that all events in a pattern must belong to the same period. It was solved very elegantly in Nishi and al.'s work [7], simply by calculating the difference vector between two events. Within the boundaries of the implementation of this work, to make such a period check, we access the period label array of both patterns, accordingly to the position, that is being checked at the moment. Consider the example from table 5 that showcases two example patterns '610001' and '351000'. We say, that both ['610001', '351000'] and ['351000', '610001'] could be candidates for the new, bigger pattern. In the case of ['610001', '351000'], the possible pairs of event identifiers are: [12, 11], [12, 14], [12, 36], [32, 11], [32, 14], [32, 36], [35, 11], [35, 14], [35, 36], [46, 11], [46, 14], [46, 36].

First, the condition of right order of events is being checked. This check declined pairs [12, 11], [32, 11], [32, 14], [35, 11], [35, 14], [46, 11], [46, 14] and [46, 36] as they do not follow the sequential order. Then, the right ordered pairs are checked so as to guarantee the events belong within the same period. Even though pair [12, 36] satisfies the right order of events, it cannot be accepted as this pair of events belong to different periods of the time series string. The result occurrence vector for pattern ['610001', '351000'] is [12, 14], [32, 36], [35, 36]. Similarly, we test pair ['351000', '610001'], see result in Table 4.6 below:

Table 4.6: Occurrence vectors of two patterns of length 2

Pattern	**Occurrence_vector**	**Period_vector**
["610001", "351000"]	[12, 14]	2011
	[32, 36]	2012
	[35, 36]	2012
["351000", "610001"]	[11, 12]	2011

The pair of patterns is nominated to be a new longer pattern if two conditions are met: the event identifier of the first one precedes that of the second one and they both belong to the same period. In order to do so, we create the

Cartesian product of all the patterns from the set and compare their event identifiers and responding period labels pairwise.

Then we compare the event identifiers of all the events in the pattern – must be in ascending order and then we confirm that both patterns belong to the same period. The approach in the paper suggests that we check the difference between event identifiers, whereas in our case, such

4.2.2.4 Periodicity detection

After a certain step in the pattern generating algorithm is finished, the result set of patterns is being checked for periodicity and then the confidence level of each particular pattern. It was already mentioned in section 2.1, that the periodicity check algorithm in the work of Nishi et al. [7] is not effective and tends to abandon interesting patterns by not being able to find its actual periodicity. To overcome this ineffectiveness, I have suggested to use a more advanced periodicity check algorithm from the work of [14]. The pseudocode for this algorithm is presented in Figure 4.6 below:

This algorithm is proven to be more effective, as it suggests a period length value based on the actual occurrences of the pattern in time series string. However, not having the same period length in time series string once again enforces the alternated solution. While both the proposed algorithms check the periodicity of the pattern by calculating the difference of its occurrences, I proposed to transfer this algorithm to the domain of period labels.

Consider we have an example abstract pattern "abc" that appears at the positions [3, 11, 14, 28, 36, 44, 67] of the time series string. While calculating its periodicity from the occurrence identifiers might appear misleading, if we first transfer the occurrence vector to the domain of period label vector, then it will be represented as [2000, 2003, 2003, 2005, 2008]. That allows to proceed in the algorithm execution, however another addition is still needed. To calculate perfect periodicity, indicated as PP in the pseudocode above, see Formula 4.2, we need to also adapt the total length of the time series string value, indicated as |T| in the pseudocode.

$$PP := \lfloor \frac{|T| - st + 1}{period} \rfloor \tag{4.2}$$

Therefore, it is clear that the length of such time series string might not be used, nor can the unique values of period labels be used, as some of the periods might not be covered by transactional data. To calculate the time series string length we must refer to the first and last occurrence of the events in the time series string itself. Then, we store the period labels of these events and fill the total period list with the labels in range (minPeriodLabel, maxPeriodLabel + 1) in the PeriodList. Then, to calculate the upper part of the perfect periodicity equation, we replace it with Formula 4.3:

Input: An *occ_vec* of pattern X with size k, confidence threshold σ
Output: Periods with corresponding *occ_vec*: OP

```
1  begin
2      for i := 1 to (k-1) do
3          Set st := occ_vec[i-1]
4          for j := i to (k-1) do
5              Set δ := occ_vec[j]
6              Set period := δ - st
7              Initialize V := ∅
8              Add st and δ in V
9              for m := (j+1) to (k-1) do
10                 if (occ_vec[m]-st) % period == 0 then
11                     V := V ⋃ occ_vec[m]
12                 end
13             end
14             PP:=⌊(|T|-st+1)/period⌋
15             conf := size(V) / PP
16             if conf >= σ then
17                 OP := OP ⋃ V, with period
18             end
19         end
20     end
21 end
```

Figure 4.6: Periodicity detection algorithm, adopted from [14]

$$PP := \lfloor \frac{|PeriodList[\, firstPeriodLabel :]\,|}{period} \rfloor \tag{4.3}$$

Meaning that we only consider the length of the PeriodList that includes period labels, starting from the first period label that is recorded for the particular pattern.

That approach allows to use the periodicity check for all of the chosen time granularity of time series data. So that when having patterns mined at the yearly periodicity, the time period label of time series string will be designed in an appropriate manner – 2000, 2001, 2002 etc. When mining patterns that appear monthly, the label will be adjusted to the form of (2000, 2001, 2002) * 12, meaning 24000, 24012, 24024 etc. Same applies for the weekly or daily periodicity.

In the result implementation, it became obvious that the aforementioned algorithm works well for the sparse time series data, meaning it has a small

ratio of amount of period labels and events in time series data. However, it appeared to have different results in terms of effectiveness when it was used on different types of time series. Here is an example, consider a company with 136 events of 6 years history in the aggregated time series data. Then, if we consider yearly periodicity, there exist on average 22.6 events that belongs to one period label, while if we consider monthly granularity, there is only 1.8 events per period label. This difference has influenced the output.

While providing good results for time series with a monthly period with a wide period range and relatively small amount of occurrences, it seemed to be unnecessarily complex for the yearly time series, which have a small range of periods – there are rarely more than 10 years in the time series database for such type of analysis – and also, there are more occurrences for one pattern. To deal with this irregularity, I have suggested to separate dense and spare time series data, based on the period granularity.

To adapt periodicity detection algorithm for the dense time series data with yearly granulariry, I have suggested an extencion of the original approach from Nishi et al. [7] Wheh the occurrence vector is paired with period labels, we take all the distinct labels and calculate the difference between the pairs of adjoining labels. After this we take the most probble difference and call it a period length of a pattern. If there exist several differences with the same probability, the least is chosen.

4.2.2.5 Generated patterns mining

This fragment of the original method didn't require any additional adjustments, but was not implemented as the output was neither meaningful, nor illustrative. The term "mining" in the context of this method refers to the search and illustration of the generated pattern variations. With the aim of excluding ambiguities in understanding this term, it must be mentioned, that such form of mining is applied on already generated and found patterns within time series database.

This step is omitted in the result implementation as it is irrelevant due to the nature of the data. Basically, there can be as many do-not-care symbols as possible, in fact, the whole period sector might be represented with such symbols. The reason is, once again, that it is not effective to set the fixed period length and thus the theta constraint or maximum event skipping threshold cannot be used effectively.

For instance, there are mined patterns for pattern [”610001”, ”351000”] from 4.2.2.3. The algorithm has successfully found three occurrences of this pattern on positions 12, 32 and 35 of the time series string. Table 4.7 shows the patterns that were later mined, which are basically the variations of the initial pattern. Here, "*" symbol represents the don't-care symbol or skipped events. Once again, the amount of such don't-care symbols is bound by θ

constraint or maximum allowed amount of skipped events. For the purpose of reflectiveness, let's agree that θ threshold is 2.

Table 4.7: Example of mined patterns

Generated pattern	**Occurrence_ vector**	**Difference_ vector**	**Mined_pattern**
["610001", "351000"]	$[12, 14]$	2	["610001", $*$, "351000"]
	$[32, 36]$	4	["610001", $*$, $*$, "351000"]
	$[35, 36]$	1	["610001", "351000"]

Having guaranteed a fixed length of the period in time series database, this can be used as a very informative representation of pattern detection and pattern search flexibility. In the case of the given bank data, however, usage of θ constraint was deprecated. To "mine" patterns, after they were detected in the time series string, in the manner that is shown above, was also redundant as:

1. There might be as many as |period length| - |number of events in pattern| don't care symbols in the result "mined" pattern and therefore it obstructs the human perception of the result pattern set.

2. The amount of don't care symbols is not in any way representative, as it is not visible from the result data how many events the particular period contains. It seldom correlates with the period length itself and thus, doesn't give the real perspective of that fraction of period or how many period sectors were skipped.

3. From the managerial point of view, what matters the most is the appearance pattern in a given period. The exact position of a particular event in the pattern's chain of events is less important. Once again, certain deviation in the data is tolerated and handled by the final version of algorithm implementation.

4.2.2.6 Check for the result pattern similarity

Here I describe the original extension of the aforementioned algorithm, idea which originated from the collaboration with the project owner and is based on the specificities of the given task and the business meaning of the result interpretation. It was mentioned before, that a particular fuzziness in the data events allows for semantic interpretation, while having a strong order difference of the events might be occasionally tolerated. To describe this tolerance, I proposed the extension of the algorithm that is being executed as a post-procession part.

Once the patterns of desired length, together with its period, occurrence vector and confidence level are mined, we look into what they actually rep-

resent and if there are sets of similar patterns. For example, two patterns might consist of the same set of events, but in slightly different order. Another possible example: when having two patterns of length i, where i is the final desired length of the output pattern, it's visible that a particular event is inserted in one pattern, which increments the position of the rest of the events.

See the example below: having two example patterns p_1 and p_2, with the maximum achieved pattern length is 8. If we compare two patterns by checking every event in them pairwise, it may seem, that that those two patterns are barely similar. However, it is clear that, event "20114" is inserted in the beginning of pattern p_2, even though the rest of the pattern is left unchanged.

- p_1 = ["43107", "43107", "33108", "68107", "20112", "43108", "43107", "43107"]
- p_2 = ["20114", "43107", "43107", "33108", "68107", "20112", "43108", "43107"]

When encountering such pattern similarity, we can then either pronounce them to be the same pattern and merge their occurrences, or record the second pattern as a variation of the first one. This merged pair of patterns is then called "macro pattern". Macro pattern is the set of events that describes similar activities of the company behavior, with the assumption of a certain level of deviation.

To compare patterns pairwise, an algorithm inspired by Levenshtein distance was used. Levenshtein distance algorithm is a string metric that is commonly used to measure the difference between two sequences. This metric counts the changes that must be provided to create a first string sequence out of the second one or vice versa. These changes might be: deletion of a string symbol, insertion of a symbol or substitution of a symbol. We use these types of changes to check if a pair of patterns represents the variations of one another.

When having the same sequence length for both patterns, it must be mentioned, that the deletion or insertion of one symbol from the pattern is always accompanied by the opposite change: insertion or deletion respectively, to keep the same amount of events in the pattern. Also, the relocation of two symbols inside one sequence is counted as making one change. Keeping that in mind, the threshold for amount of changes becomes a user defined parameter.

To merge two or more patterns into one macro pattern, there was the question of which of the patterns must be chosen to represent the subset. To solve this issue, I've suggested a pattern ranking system. The principle of this system is that the result set of patterns is first sorted in a descending order based on the rank value. Then, the top pattern checks its similarity against the rest of the set, one pattern at a time. If changes or Levenshtein distance satisfy the given threshold, the pattern with lower rank is recorded as a variation of

the top one and is taken out of the rest of the set. Then, the top pattern is the next one in the rest of the pattern set. This algorithm iterates until no patterns are left in the set, or all of the patterns have empty variations sets. To rank the patterns in the pattern set, I assumed two characteristics of the patterns: the number of occurrences, counting the interchanged event identifiers, and the number of period labels, which respond to those occurrences, as one period may have several occurrences. The sum of these two parameters creates the rank and tends to be heterogeneous enough, so there is little chance, that two patterns have the same length.

After patterns are sorted by rank, the algorithm checks the Levenshtein distance between the top pattern and each lower rank pattern. If the Levenshtein distance is lower than the given threshold, the pattern which was considered as a possible variation of the top one, is listed in a variation array. There are two use cases of such check, with different purposes. The first one is to use the result of this similarity check in the business analysis purposes. In this case, after all the patterns below the top one are checked, the top pointer goes to the next pattern in the row and so on. This option keeps all of the original data and leaves the interpretation to the data science team.

The second one is to be used for the future pattern search in the new transactional time series. In this case, in the course of pattern check traversing, once the pattern with lower rank is found to be similar to the top one, it is removed from the original list and its occurrence vector is merged with the top pattern ones. This obviously affects patterns' periodicity and strengthen its confidence. After this, the top pattern's periodicity check is done again in order to update its confidence level.

4.3 Periodic outlier patterns and anomalies in time series database

Among many other possible statistical and economical descriptions, one way anomalies in transactional time series can be represented is in the form of either unusual transactions on companies' accounts or the drastic fluctuation in the market behavior, sector-wise. For the purpose of finding unusual events or set of events, I decided to use an algorithm that utilizes the same principles of periodic pattern mining in time series databases.

4.3.1 Algorithm and adaptation

Since the time of publishing of the last here quoted work of Rasheed, F. and Alhaji, R. in [11], these two authors came up with the extension of their suffix-tree based algorithms for time series database mining. Particularly, they published "A Framework for Periodic Outlier Pattern Detection in Time-Series Sequences" [17]. The aim of this work was to show, that while common

algorithms for periodic pattern mining are known for having a tendency of producing large amount of not interesting and redundant patterns, the valuable ones may not appear very often or periodically, but on the other hand are more valuable and descriprive.

This algorithm, apart from the main two steps – detection of repeating subsequences in time series together with its occurrence vectors – has a one additional intermediate step, which chooses patterns from the result set and calculates the surprise level of the patterns in each pass of the algorithm before the periodicity detection and check.

For the pattern detection this algorithm uses the suffix tree. The usage of the suffix tree data structure to detect patterns has already been discussed in this paper, so this step was replaced with the one I have described in the 4.2.1.

with the one I have described in the 4.2.1. After the set of patterns has been calculated in each pass of the algorithm, for all of the patterns in the set, the surprise level is calculated. Surprise level is then presented in Formula 4.4:

$$surprise(X) = 1 - \frac{f(X)}{\mu(f(X_i))}; \forall i \text{ such that } |X_i| = |X| \tag{4.4}$$

where $\mu(f(X_i))$ is the mean of the frequency of all patterns of same length as that of pattern X.

After this step, only patterns with surprise level higher than the user input value threshold, proceed to the periodicity check. To understand the surprise level threshold, here is an example: $thesurpriselevel(X) = 0.1$ means, that pattern X occurs in time series 90% as often, as an average pattern with the same length.

In their work [17] they suggest using the periodicity detection and check algorithm, which is very similar to the one, that is being used in [11] see Figure 4.7, however it has some limitations, one of them is that it only assumes occurrences that happened before the end position, or last occurrence in the time series data. This leads to the inconsistency in the results, as some pattern, which has last ocurrence in the time string database somewhere in the middle, might still get conidence = 1.0 and this is contradictive to the concept of periodic patterns that was defined before.

To overcome these limitations, I suggest using the same periodicity detection algorithm, that was described in 4.2.2.4.

4.4 Parameters and experiments with them

The complete version of analytical framework has a set of parameters that are defined by the user before each algorithms' initiation. During the construction of the analytical framework, I've observed different behavior of the framework

Algorithm 3 Occurrence Vector Processing Algorithm

procedure PROCESSOCCURRENCEVECTOR(pattern X, list $occur$, int $minSegLen$, real $conf_{min}$)
  $p_{pre} = -5, preCountPerCol = periodCol.Count$
  ▷ p_{pre} is previous period, $preCountPerCol$ is previous count of period collection
  for $m = 0$; $m < |occur| - 1$; m++ **do**
    if $m < |occur| - 1$ **then**
      $p = occur[m+1] - occur[m], i_{st} = occur[m], i_{end} = occur[|occur| - 1]$
      if $p_{pre} \neq p$ AND $(i_{end} + |X| - i_{st}) > (minSegLen * |s|)$ AND Not $AlreadyThere(X, i_{st}, i_{end}, p)$ **then**
        $periodCol.add(X, i_{st}, i_{end}, p)$ ▷ Add to test period list
      end if
      $p_{pre} = p$
    end if
    ▷ Verify current occurrence against test period list
    for $n = preCountPerCol; n < periodCol.count$; n++ **do**
      if $(periodCol[n].i_{st}$ mod $periodCol[n].p) ==$ $(occur[m]$ mod $periodCol[n].p)$ **then**
        Increment period frequency: $periodCol[n].f$
        $periodCol[n].i_{end} = occur[n]$
      end if
    end for
  end for
  ▷ Remove non-frequent and periods with shorter coverage
  for $y = 0, k = preCountPerCol; k < periodCol.count; k++$ **do**
    $f_{max} = \frac{periodCol[k].i_{end} + 1 - |X| - periodCol[k].i_{st}}{periodCol[k].p} + 1$
    $conf(X, i_{st}, i_{end}, p) = \frac{f}{f_{max}}$
    if $conf < conf_{min}$ OR $(i_{end} + |X| - i_{st}) > (minSegLen * |s|)$ **then**
      $periodCol.remove(X, i_{st}, i_{end}, p)$
    end if
  end for
end procedure

Figure 4.7: Occurrence vector processing algorithm, adopted from [17]

together with different outputs, while running the algorithm recurrent pipeline with different values of parameters. These parameters are:

1. Time period for time series database
2. Desired pattern length
3. Method of transaction amount binning
4. Periodicity confidence threshold

5. Difference threshold for macro patterns

In the sections 4.4.1 – 4.4.6 I discuss different values of the algorithm parameters and its impact on the results.

4.4.1 Time period label

In the course of the framework implementation and testing on the given data set from the project owner, I've used two types of time series database periodicity: yearly and monthly, however implementation is suitable to be extended for other possible periods for finer granularity as well. For all companies in the data set, there exists transactional history in range of years 2011 and 2016.

It was observed, that companies with less than 200 events are less likely to have any monthly periods in its transactional data, however, the problem with monthly period existing might be present even with more broad transactional history. On the other hand, companies that have more than 750 events, might take a vast time to compute patterns of length 6 and more. But of course, it also depends on the intermediate results such as the amount of patterns and probability of its matching for creating longer patterns.

In such cases, I suggest several optimization options:

1. Cut off the events, that contain NACE of adverse party that are no interest in the case of a particular company

2. Focus only on the NACE codes of the adverse parties, that appears in the patterns of companies with the same primary NACE and turnover category

3. Separate years of transactional activity

The choice of such options are, of course in the competence of business analytics, who are the final consumers of this analytical tool.

4.4.2 Target pattern length

The framework is design in such fashion, that if a desired pattern length cannot be achieved, it returns the set of patterns of the maximum achieved length. However, there are cases, like the one described in the previous sections, in which the desired pattern length cannot be achieved due to the massive amount of consumed memory. To deal with such cases, above are listed suggestions for the possible optimization.

4.4.3 Bin type

Different binning type options, basically different sizes of bins, provide different set of unique events and hence these events are more or less frequent

in the given time series database, for illustration see Figure 4.8. There is an inverse dependency between the number of unique events, which create the alphabet of time series database and the probability of creating longer patterns. Generally speaking, while having many unique events, most of them will have less occurrences than more general events would have and therefore, less periodic patterns can be mined. However, it doesn't necessarily mean less meaningful results, as an abundance of patterns in the earlier phases of the algorithm may lead to the quick escalation of the pattern number and hence, either a big RAM consumption, or meaningless result set of patterns.

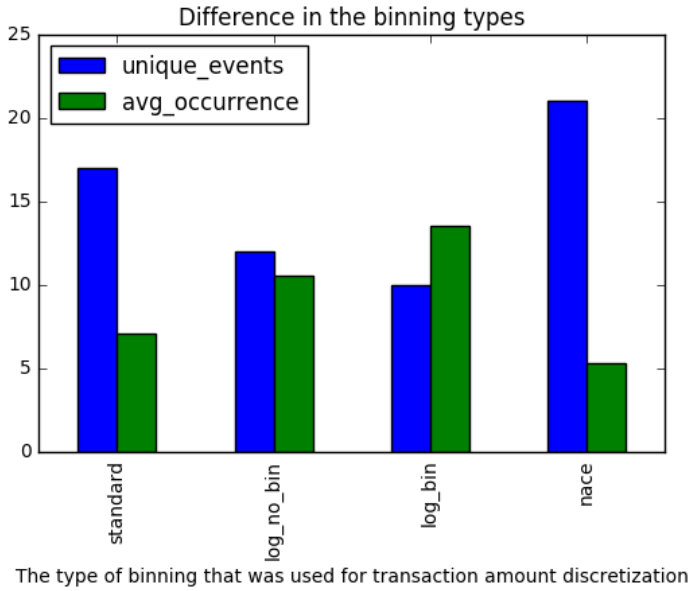


Figure 4.8: An example of the impact of different binning type for the transactional history with 515 events

4.4.4 Confidence threshold

There is an inverse dependency between confidence thresholds for pattern periodicity and amount of patterns in result set. As confidence threshold is used in every pass of the algorithm to cut off the less periodic patterns that will not proceed in generating longer patterns, this is a very important parameter. It also appears, that a confidence threshold less than 0.8 might lead to useless results in yearly period domain, while is tolerated for the monthly patterns.

4.4.5 Surprise level and confidence threshold for outlier patterns

The surprise level of the outlier patterns is another user defined parameter that affects the size of the result pattern set. While having a high surprise level threshold may end up as a little or empty result pattern set, having it set to low values may cause the result pattern set to overlap with the actual frequent periodic patterns that can be a source of misleading result interpretation. In my opinion, the appropriate range for this parameter is [0.2, 0.5]. For the threshold of outlier patterns, I suggest using the threshold value for the periodicity detection, reduced by the coefficient 0.3.

In order to find patterns, that consist of several evetns, but are not periodic, but still are intresting as they represent certain deviation in the customer's behaviour, I suggest using high surprise level threshold and confidence level = 0.

4.4.6 Macro patterns threshold

The threshold of pattern difference for the purpose of creating macro patterns was initially designed as another user input parameter, but after consulting the results of this post-processing of the result with the project owner, it was decided to fixate this parameter with value 2. This means, that there might be three possible differences in the pair of patterns:

- One event is absent in the pattern (another one is present, as two patterns are same length): one insertion is always compensated with one deletion and vice versa.
- Maximum of two events are replaced with others.
- Two events are interchanged inside the pattern.

More complex differences in between two patterns appear to be misleading, as it appears, that patterns are too different and in such cases it can no longer be perceived as semantic interpretation of business events.

Chapter 5

Implementation

As one of the main goals of this thesis work was to embody the suggested algorithms and their alternation into an analytical framework, here I include the description of the development environment and libraries that were used for the implementation.

5.1 Development environment

The development and testing of the analytical framework implementation was held on a single computer with the following characteristics:

- Operating system: Windows 8.1
- Processor: Intel Corei5-4210 CPU 1.70GHz 2.40 GHz
- RAM: 8 GB
- System Type: 64-bit Operating system, x64-based processor

The programming language of choice for the analytical framework implementation is Python 2.7. Among many reasons to choose this programing language, the main was a set of able libraries, designed for the needs of the data analysis:

- Pandas
- NumPy
- Matplotlib
- SciPy

These libraries are described in more details in the next section.

The development IDE of choice is IntelliJ IDEA Apache 2 licensed community edition. It is an open-source software developed by JetBrains s.r.o.

Another UI for the development that was used is Jupyter Notebook. It is an open-source web application that allows its users to create and share so-called Notebooks that represent documents with executable cells. These cells are used for interactive code execution, equations, visualizations and text. One of the advantages of using this application is the ability to control and look into the analyzed data at any moment, which improves the programming experience.

The diagram illustrations in the text of this work were created in Draw.io web application. This application is used to create different types of diagrams, including for example UML, engineering and business-processes. It is a free-to-license and all of the produced content belongs to the user, who created it.

5.2 Python libraries

Pandas is a software library, designed for data transformation and analysis. Pandas includes data structures like DataFrame and Series. It offers features like pivoting data sets, integrated handling missing data, data sets transformation, merging, slicing and grouping, together with data reading from external sources and writing into different formats. Pandas is a free software, released under the three-clause BSD license. The stable release that was used is 0.18.1. NumPy is a Python library that adds support for multi-dimensional arrays and matrices. It also contains high-level math functions that are used to operate on such arrays. NumPy is an open-source software with many active contributors. It is licensed under a new-DSD license. The stable release that was used is 1.11.2.

The data visualizations that are included in this work were created using the matplotlib library. It is a plotting library for Python programming language together with Python's extension NumPy. It allows many plot options like linear graphs, scatter graphs, histograms and more complex 3D visualizations. It is distributed under BSD-style license. The stable release that was used is 1.5.3.

SciPy is another open source Python library. It contains modules for scientific and technical computing. SciPy builds its functionality on NumPy arrays. The set of scientific libraries is constantly developing, its development is supported by an open community of developers. The SciPy is distributed under the BSD-new license. Stable release that was used is 0.18.1.

5.3 Algorithm limitation

It was mentioned in the work of [14] that the algorithm of Nishi et al. [7] tends to have certain performance limitations, as it uses an apriori based sequential mining approach to produce periodic patterns. That includes certain complexities in mining long sequences, such as, for P length pattern mining, in worst case, 2P number of patterns need to be handled and at least P number of times the database needs to be scanned. These limitations lead to an immense computational memory and time. Constrainst which I have also encountered, while working on the implementation.

In order to alleviate this disadvantage, I suggested several implementation alternations. One of which was to omit the on-the-fly check for the redundant occurrences of the pattern, which was created from two similar occurrences of patterns. Instead, the algorithm first collects all of the possible event identification of the pattern, then aggregates by pattern and then removes duplicate event identifications. The occurrence list can then be represented as a distinct set of itself, while referring to the event identifications by the first index.

Chapter 6

Results

This chapter showcases the achieved results, which represent the output of the analytical framework. The sample set of results was given to the analytics in KB's department of Data Science, to evaluate the result pattern sets of different companies, its accuracy and precision. As the evaluation of the result pattern could only be done manually, only a few companies were chosen for this task. The evaluation of the actual pattern sets' size or the events that are contained in those patterns against the expected values was limited due to several reasons such as:

- Different parameter settings provide a different output for each company's history. Therefore it is up to the experts' opinion of what the best suitable parameters set are for either different companies' characteristics, or different aim for the analysis.
- The expected set of patterns is a very subjective matter and thus cannot be used as an impartial measurement object for the result to be tested against.

The Data Science department has verified the provided results, but did not allow the publication of the pattern interpretation due to data protection. The demonstrative interpretation in the form of a case study would have been considered as validation of the data anonymization and protection of sensitive data. In the following sections I provide the results' description and the tendencies that are visible from those pattern sets.

6.1 Example results

The output of the analytical framework is then represented in the form of set of patterns that have structure of Table 6.1:

Here I enclose the results' statistics for group of companies, history length, time granularity and parameters used in the framework.

Table 6.1: The structure of the result patterns

Attribute	**Example**	**Description**
p	[”’46000002”, ”46100002”, ”61000001’”]	Generated pattern
occ	[3, 440]	Occurrence vector of the pattern
eid	((440, 441, 442), (3, 4, 6))	Event identificators of every event in the pattern
$period$	32	The periodicity of the pattern
$conf$	1.0	The confidence level of the pattern for the given periodicity
$occ_unfolded$	(2013, 1, 2015, 9)	The occurrences of the pattern translated back to the original period labels, in this case - months.

The companies’ classification that were taken for the test evaluation: 4 companies with NACE code 77 and of turnover category 3. Average length of aggregated time series database: 559

- Chosen time granularity: Month
  - Parameters used: Binning type: standard with bin size 50000 and confidence level threshold 0.8
    1. Average of maximum pattern length achieved: 3
    2. Average amount of patterns found: 4
    3. Average confidence of the result pattern set : 0.97
  - Parameters used: Binning type: NACE and confidence level threshold 0.8
    1. Average of maximum pattern length achieved: 3
    2. Average amount of patterns found: 17
    3. Average confidence of the result pattern set : 0.82
  - Parameters used: Binning type: logarithmic value without binning and confidence level threshold 0.8
    1. Average of maximum pattern length achieved: 2
    2. Average amount of patterns found: 3
    3. Average confidence of the result pattern set : 0.9
  - Parameters used: Binning type: logarithmic value with binning and confidence level threshold 0.8
    1. Average of maximum pattern length achieved: 2
    2. Average amount of patterns found: 3

3. Average confidence of the result pattern set : 0.9

- Chosen time granularity: Year
  - Parameters used: Binning type: standard with bin size 50000 and confidence level threshold 0.8
    1. Average of maximum pattern length achieved: 10
    2. Average amount of patterns found: 8
    3. Average confidence of the result pattern set : 1
  - Parameters used: Binning type: NACE and confidence level threshold 0.8
    1. Average of maximum pattern length achieved: 10
    2. Average amount of patterns found: 4
    3. Average confidence of the result pattern set : 1
  - Parameters used: Binning type: logarithmic value without binning and confidence level threshold 0.8
    1. Average of maximum pattern length achieved: 6
    2. Average amount of patterns found: 6
    3. Average confidence of the result pattern set : 1
  - Parameters used: Binning type: logarithmic value with binning and confidence level threshold 0.8
    1. Average of maximum pattern length achieved: 9
    2. Average amount of patterns found: 4
    3. Average confidence of the result pattern set : 1

The key parameter for the result output is the type of transaction amount binning in the process of time series data discretization. The smallest possible bin size leads to the statistically correct results, as it basically describes the regularities in the single events, which is, however, meaningless from the business point of view. In order to provide descriptive and significant results, one has to find a balance in the tradeoff between bin size and the amount of generated patterns.

Parameters were chosen with an aim to achieve the optimum results. Suggestions for the parameters are:

- Binning type based on the wider transactional database, that creates the right bin size based on the transactions of each segment throughout the whole database.

- Confidence level: 0.8

- Maximum pattern length: 12 for the yearly granularity, where the amount of events inside the aggregated time series is more than 250, and 5 for the monthly granularity, where the amount of events inside the aggregated time series is more than 350.

However, a further examination of the pattern settings is expected to take place in order to find the applicable expert setting for the different types of analysis.

6.2 Result evaluation

To estimate the framework's effectivity, instead of a comparison of the expected and real results, here we compare the expected framework's functionality against the actual pattern result description. The project owner's team had several particular assumptions regarding the hidden knowledge in the transactional data. Here is a list of hypothesis, which were defined and tested.

- Hypothesis: technical
  - Patterns
    - Frequent
      - Confirmed: Yes
      - Description: There exists several types of events, that form the group in the bank transactional history data and are frequent.
    - Periodic
      - Confirmed: Yes
      - Description: Among these groups of events or patterns there exists a subset that is not only frequent, but also periodic.
  - Periodicity
    - Detectability
      - Confirmed: Yes
      - Description: The periodicity of such patterns can be detected. The patterns with detected periodicity can then be tested for the confidence level of a pattern's occurrence in the time series data.
    - Priority
      - Confirmed: Yes
      - Description: The patterns with different confidence level of occurrences may be then priorities, based on the value of the confidence level.

  - Anomalies
    - Periodic
      - Confirmed: Yes
      - Description: There exist patterns with less occurrences in longer or irregular periods of occurrences, that should also be detected.
    - Not periodic
      - Confirmed: Yes
      - Description: These less frequent patterns may not be periodic at all, but represent a real unexpected behavior of the subject.
  - Macro patterns
    - Confirmed: Yes
    - Description: The result set of patterns might contain patterns that basically represent the same set of events. These patterns might be merged into the macro patterns.
  - Gaps overcoming
    - Confirmed: Yes
    - Description: The final implementation would be able to deal with gaps in the periods of aggregated time series of the transactional data.

Apart from the technical hypothesis, there existed certain business expectations, regarding what kinds of regular transactions it would be possible to find and observe in the result patterns.

- Hypothesis: technical
  - Regular fixed costs
    - Confirmed: Yes
  - Utilities
    - Confirmed: Yes
  - Maintenance
    - Confirmed: Yes
  - Salaries
    - Confirmed: No
  - Supply chain: Customers dynamics
    - Confirmed: Yes
  - Supply chain: Suppliers dynamics

        * Confirmed: Yes
    - Taxes
        * Confirmed: Yes

The overall result evaluation proves the suggested analytical framework to be sufficient and the work's goals to be fulfilled. It creates a reconstructive knowledge that corresponds to the business behavior.

Chapter 7

Discussion

Once the project owner's chosen part of the historical data are full-scanned and the patterns' extensive database is constructed, these will provide major opportunities for future use. In the following sections, I have summarized this database's advantages, which different departments of KB bank may benefit from.

7.1 Marketing and Sales

The contribution of this work to the project owner's Sales department may be described as a new tool and ability to create better planned sale offers for the customers. Whether it is for a particular customer or the whole group of companies, being able to propagate events in their financial interactions is a mighty instrument that enables the creation of customer-oriented offers. Considering a company's financial situation and perspective of the upcoming group of necessary payments that will take place with certain probability, it is easier to create a tailor-made offer that consists of the right bank products and appears at the right time.

The Figure 7.1 illustrates regular payments that are made by the companies and its purpose. Among others there are at least four main purposes: taxes and deductions, utility payments such as energy, etc., telco payments and different services.

Once these regularities are studied and understood, they might be removed from the analyzed data completely, as the most important interconnections that represent the great value to the project owner are the regularities in the customers-suppliers interconnections on the market. The dynamics of the interactions between companies is a major point of interest for the Marketing and Sales department. In the course of the analytical framework suggestion, these regular payments data were not removed from the sample data set as they demonstrate the general regularities of the economical subjects and are

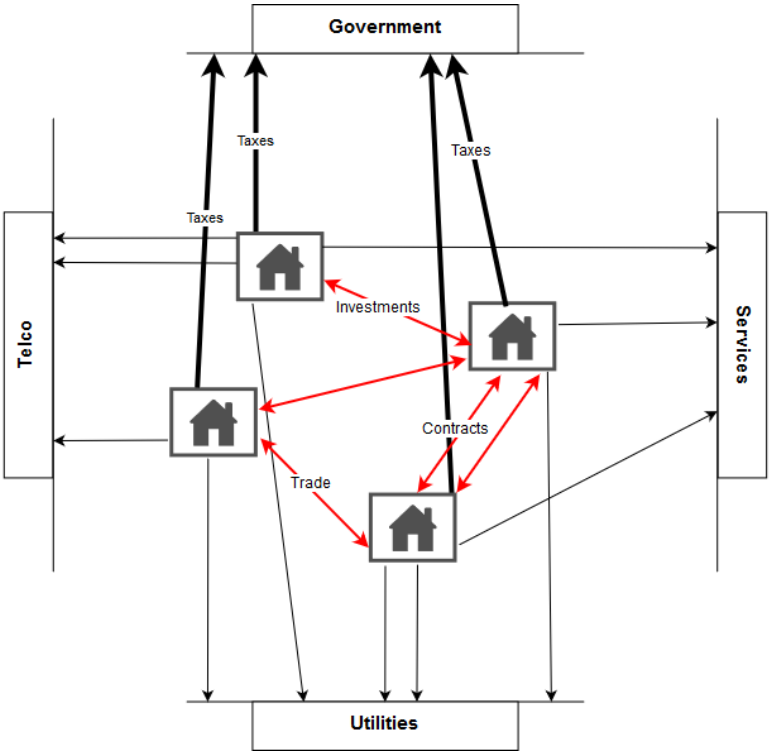


Figure 7.1: The distribution of different size companies on the market

a good source of training data.

7.2 Management and Business Intelligence

Another important output of the suggested analytical framework is the ability to analyze the customers' behavior in the automated processes. As Figure 7.2 below suggests, there exists a certain statistical dependency between the size of the company, its turnover amount and the number of such companies in the market. While the top corporate subjects are counted as units, and are well studied throughout the years of partnership with banks, small and medium businesses (SMB) are present in the wide range and could not be possibly studied individually.

The knowledge of the regularities of the SMB subjects' behavior is mostly concentrated among senior business analytics, which have many years of experience in this field. However, with a certain level of staff fluctuation, it becomes problematic to keep this knowledge capital within the company. Another knowledge management issue is the inability to successfully transmit the experience in suitable and efficient form. The analytical framework, how-

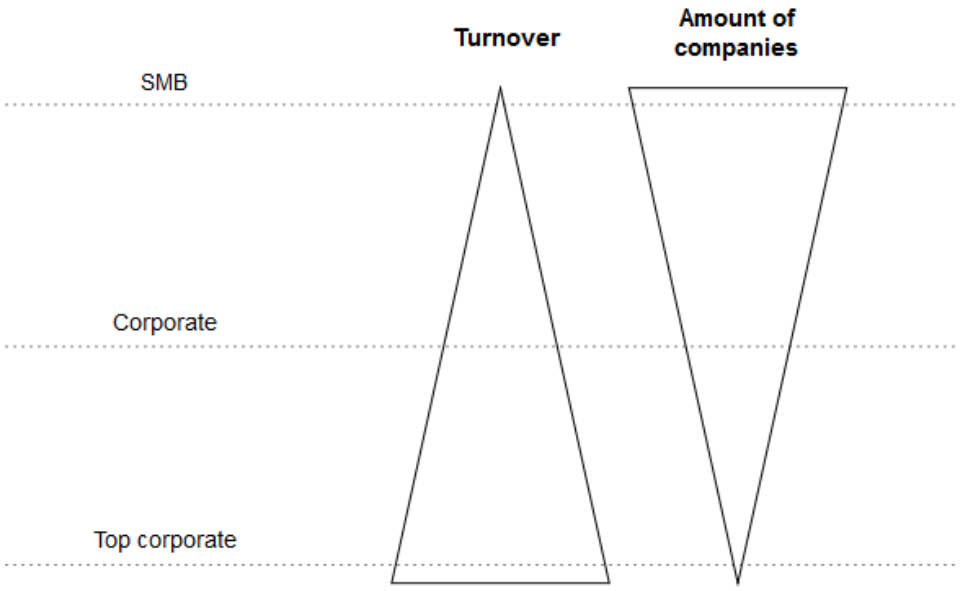


Figure 7.2: The distribution of different size companies on the market

ever, enables the discovery of restorable knowledge for a bigger amount of companies at once.

7.3 Future use

As it was mentioned in section 6.1, only a subset of the given data set was presented to the project owner in the form of result pattern sets. The framework will become a new module in the analytical portfolio of the project owner's department of Data Science.

So the first step for its future use implies the creation of the extensive database of patterns. The final goal of the analytical framework for periodic pattern mining was not to incorporate it in the everyday monitoring process, but rather to create a database of patterns, that will serve for the purposes of the different departments and will be maintained regularly. Next step assumes an extensive verification of pattern set results and tuning the patterns to suit different analysis goals.

The implementation of the given framework is intended for future development. This development assumes adding functionality modules like different time granularity options, apart from those shown in this work, being able to track financial regularities not only between company sectors, but also between particular chosen companies, etc.

Conclusion

In the course of working on this thesis I have focused on the problem of data mining of time series databases of bank transactional historical data. Periodic pattern mining in time series databases is a very promising field of data analysis, as it allows discovering the hidden knowledge in the historical or streaming data, such as, for example, bank transactional data. The goal of this work was to suggest a set of appropriate algorithms to be embodied into an analytical tool, which would be able to deal with the irregularities in the bank transactional data and detect periodic patterns regardless of this limitation. Here, I'd like to summarize the work that has been done.

Summary

To operate with transactional events effectively I have designed a discretizing technique for the transactional data. This technique takes into consideration the different nature or context of particular transactions so the result patterns are both descriptive and accurate in terms of business reality.

Then I have successfully managed to overcome the problem of data irregularities such as missing periods or gaps inside those periods. Even though I consider my solution to be efficient in terms of the output it provides, I suggest parallelized implementation for the more wide-ranging databases or in order to provide this analysis more often. Also, alternative algorithm concepts were suggested in the state-of-the-art descriptions.

After this, I have improved the periodicity detection algorithm to suit different time granularities in order to provide a set of significant patterns that also have business value and importance. This periodicity detection algorithm is then a part of the outlier detection module of the analytical framework. Then, the added extra step in the analytical framework checks the pattern similarity to create meta-patterns with higher confidence level.

Overall, I think that I have managed to design and implement useful and

effective methods of pattern detection and periodicity detection. An added value to this work is the delivery of a mechanism detecting the regularities in the behavior of business subjects, which have an extensive transactional history that may not be fully scanned and analyzed by the analytics without an automation in such analytical processes. An example of this historical data might be the case of transactional data of SMB subjects, which is not often possible to analyze in full range, given the amount of such subjects on the market.

Future work

The work that was done within the boundaries of this thesis leaves a lot of possibilities for improvement and further research. In my future work, I'd like to engage in the problem of pattern detection in transactional time series data in more detail. I'd like to propose working on a more precise analysis technique that would provide results with added descriptive business meaning. As this work only considered one main segment of the subject's economic activity, the result patterns might appear too complex. However, if it would be possible to have several defined segments of economic activity for each company, the resulting set of patterns might be then investigated to be composed of events that are typical for any of those segments.

Bibliography

[1] Esling, P.; Agon, C. Time-Series Data Mining. *ACM Comput. [online]*, volume 45, November 2012, [cit. 2017-03-09]. Available from: http://doi.acm.org/10.1145/2379776.2379788

[2] Krawczak, M.; Szkatula, G. An approach to dimensionality reduction in time series. *Information Sciences [online]*, volume 260, March 2014: pp. 15 – 36, [cit. 2017-03-07]. Available from: https://doi.org/10.1016/j.ins.2013.10.037

[3] Abonyi, J.; Feil, B.; et al. Modified Gath–Geva clustering for fuzzing segmentation of multivariate time-series. *Fuzzy Sets and Systems [online]*, volume 149, 2005: pp. 39 – 56, [cit. 2017-03-07]. Available from: http://www.abonyilab.com/time-series-mining/1-s2.0-S0165011404003069-main.pdf?attredirects=0

[4] Miao, S.; Vespier, U.; et al. Predefined pattern detection in large time series. *Information Science [online]*, volume 329, 2016: pp. 950 – 964, [cit. 2017-03-04]. Available from: https://doi.org/10.1016/j.ins.2015.04.018

[5] Chen, Y.; Chen, K.; et al. Effective and Efficient Shape-Based Pattern Detection over Streaming Time Series. *TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING [online]*, volume 24, 2012: pp. 265 – 278, [cit. 2017-03-07]. Available from: http://ieeexplore.ieee.org/document/5620913/

[6] J., L.; Keogh, E. Clustering of time-series subsequences is meaningless: Implications for previous and future research. volume 8, 2005: pp. 154 – 177, [cit. 2017-03-05]. Available from: http://link.springer.com/article/10.1007/s10115-004-0172-7#article-dates-history

[7] Nishi, M. A.; Ahmed, F. A.; et al. Effective periodic pattern mining in time series databases. *Expert Systems with Applications [online]*, volume 40, 2013: pp. 3015 – 3027, [cit. 2017-04-21]. Available from: http:/dx.doi.org/10.1016/j.eswa.2012.12.017

[8] Zanin, M. Forbidden patterns in financial time series. *Chaos: An Interdisciplinary Journal of Nonlinear Science [online]*, volume 18, March 2008, [cit. 2017-03-04]. Available from: http://dx.doi.org/10.1063/1.2841197

[9] Wan, Y.; Gong, X.; et al. Effect of segmentation on financial time series pattern matching. *Applied Soft Computing [online]*, volume 38, 2016: pp. 346 – 359, [cit. 2017-03-05]. Available from: http://dx.doi.org/10.1016/j.asoc.2015.10.012

[10] Moon, Y.; Whang, K.; et al. Duality-based subsequence matching in time-series databases. In *Proceedings of the 17th IEEE International Conference on Data Engineering [online]*, 2011, pp. 263 – 272, [cit. 2017-03-05]. Available from: http://ieeexplore.ieee.org/document/914837/?reload=true&arnumber=914837

[11] Rasheed, F.; Alshalalfa, M.; et al. Efficient Periodicity Mining in Time Series Databases Using Suffix Trees. *TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING [online]*, volume 23, January 2011, [cit. 2017-03-07]. Available from: http://ieeexplore.ieee.org/document/5467068/

[12] Kaneiwa, K.; Kudo, Y. A sequential pattern mining algorithm using rough set theory. *International Journal of Approximate Reasoning [online]*, volume 52, 2011: pp. 881 – 893, [cit. 2017-03-14]. Available from: http:/dx.doi.org/10.1016/j.ijar.2011.03.002

[13] Kaneiwa, K.; Kudo, Y. Local pattern mining from sequences using rough set theory. In *International Journal of Approximate Reasoning [online]*, Granular Computing (GrC), 2010 IEEE International Conference on, August 2010, [cit. 2017-03-06]. Available from: http://ieeexplore.ieee.org/document/5576058/

[14] Chanda, A. K.; Saha, S.; et al. An efficient approach to mine flexible periodic patterns in time series databases. volume 44, 2015: pp. 46 – 63, [cit. 2017-03-05]. Available from: http://dx.doi.org/10.1016/j.engappai.2015.04.014

[15] Chanda, A. K.; Ahmed, F. A.; et al. A new framework for mining weighted periodic patterns in time series databases. *Expert Systems with Applications [online]*, volume 79, 2017: pp. 207 – 224, [cit. 2017-04-28]. Available from: http:/dx.doi.org/10.1016/j.eswa.2017.02.28

[16] Yun, U. Efficient mining of weighted interesting patterns with a strong weight and/or support affinity. volume 177, 2007: pp. 3477 – 3499, [cit. 2017-03-05]. Available from: http://dx.doi.org/10.1016/j.ins.2007.03.018

[17] Rasheed, F.; Alhajj, R. A Framework for Periodic Outlier Pattern Detection in Time-Series Sequences. *TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING [online]*, volume 44, May 2014: pp. 569 – 582, [cit. 2017-03-04]. Available from: http://doi.acm.org/10.1109/TSMCC.2013.2261984

[18] Glossary: Statistical classification of economic activities in the European Community (NACE). http://ec.europa.eu/eurostat/statistics-explained/index.php/, [cit. 2017-03-05].

Appendix A

Contents of enclosed flash drive

- `readme.txt` ..... the file with flash drive contents description
- `src` ..... the directory of source codes
  - `pattern_mining` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis.ps` ..... the thesis text in PS format