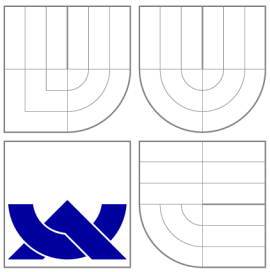
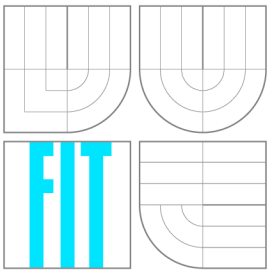


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MINING MULTI-LEVEL SEQUENTIAL PATTERNS

DOLOVÁNÍ VÍCEÚROVŇOVÝCH SEKVENČNÍCH VZORŮ

DISERTAČNÍ PRÁCE
PHD THESIS

AUTOR PRÁCE — Ing. MICHAL ŠEBEK
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. JAROSLAV ZENDULKA, CSc.
SUPERVISOR

BRNO 2016

Abstrakt

Dolování sekvenčních vzorů je důležitá oblast získávání znalostí z databází. Stále více průmyslových a obchodních aplikací uchovává data mající povahu sekvencí, kdy je dáno pořadí jednotlivých transakcí. Toho může být využito například při analýze po sobě jdoucích nákupů zákazníků.

Tato práce se zabývá využitím hierarchického uspořádání položek při dolování sekvenčních vzorů. V rámci práce jsou řešeny dvě základní oblasti – dolování víceúrovňových sekvenčních vzorů s křížením a bez křížení úrovní hierarchií. Dolovací úlohy pro obě oblasti jsou v práci formalizovány a následně navrženy algoritmy hGSP a MLSP pro jejich řešení. Experimentálně bylo ověřeno, že především algoritmus MLSP dosahuje výborných výkonnostních vlastností a stability. Význam nově získaných vzorů je ukázán na dolování reálných produkčních dat.

Abstract

Mining sequential patterns is a very important area of the data mining. Many industrial and business applications save sequential data where the ordering of transactions is defined. It can be used for example for analysis of consecutive shopping transactions.

This thesis deals with the using of concept hierarchies of items for mining sequential patterns. This thesis focuses on two basic approaches – mining level-crossing sequential patterns and mining multi-level sequential patterns. The approaches for the both data mining tasks are formalized and there are proposed data mining algorithms hGSP and MLSP to solve these tasks. Experiments verified that mainly the MLSP has good performance and stability. The usability of newly obtained patterns is shown on the real-world data mining task.

Klíčová slova

získávání znalostí z databází, dolování sekvenčních vzorů, konceptové hierarchie, uzavřené vzory, víceúrovňové sekvenční vzory

Keywords

data mining, mining sequential patterns, concept hierarchies, closed patterns, level-crossing sequential patterns, multi-level sequential patterns

Citace

Michal Šebek: Dolování víceúrovňových sekvenčních vzorů, disertační práce, Brno, FIT VUT v Brně, 2016

Contents

Chapter 1

Introduction

Nowadays, the total amount of stored data in different kinds of databases is growing. Many different applications save data about each transaction. For example, data about merchant transactions are saved for billing purposes. If the data are collected from a high number of customers, new dependencies about the customers' behavior can be formed. Another example is that data about insurance events can be stored. If the reporting period is long enough, the data can be used for the risk analysis of new contracts. Retrieving of such new knowledge from data is called *Data Mining* (or *Knowledge Discovery)* introduced in early 90^{th} of 20^{th} century.

An established definition is that *Data Mining is an extraction or "mining" of hidden knowledge from large amounts of data* [10]. Data Mining is a complex process where the application of the data mining algorithm is only one step of the process. The process is composed of following steps: data pre-processing (data cleaning, integration, transformation and reduction), data mining, pattern evaluation and knowledge presentation.

Various types of databases require different data mining tasks and provide different kinds of patterns. Examples of main data mining tasks and types of data to be mined are following:

- Classes Characterization and Discrimination – data are associated with classes and characterized and/or compared.
- Mining Frequent Patterns, Associations, and Correlations – frequent patterns are such patterns that occur frequently in data (in other words, data which occur in the dataset in a number which is higher than the given threshold value).
- Classification and Prediction – data are labeled into classes. The algorithms of classification try to find a model which describes each class and can be applied on new (unlabeled) data. The prediction has a continuous target attribute.
- Cluster Analysis – clustering algorithms try to find a model which can divide data into a specific number of groups. Clustering can be used for an initial labeling of data.

- Outlier Analysis – algorithms for outlier analysis reveal data records which have different values than the majority. Such problem is typically used for fraud detection.

This thesis deals with *mining sequential patterns*. Mining sequential patterns is a special case of mining frequent patterns with a defined order of transactions. It is used for many applications such as the analysis of customer patterns, web log data purchase, security applications, etc. The goal is to find sequential patterns that occur in the database frequently. Market basket analysis is a typical application example where the sequential patterns like $\langle PC\_minitower\ ink\_printer \rangle$ can be discovered. The pattern says that many people buy a minitower PC and then, later, they also buy an ink printer.

Items in the database can be assorted and categorized into one or more taxonomies. An example of taxonomies of items is shown on Figure 1.1. Taxonomies can be used to find patterns which items are on the different levels of hierarchy. We demonstrate it on the example of customer purchase analysis. Following the sequential pattern example above, the pattern $\langle PC\ printer \rangle$ can be found by replacing all items by items on a higher hierarchy level. Unfortunately, the amount of such patterns can grow enormously, but many of the patterns can be considered as useless. For instance, the pattern $\langle PC\ printer \rangle$ does not bring any new information if the number of its occurrence in the database is the same as a number of $\langle PC\_minitower\ ink\_printer \rangle$.

Example 1. For better illustration of the practical impact of the issue being solved, the thesis uses a simple real world example from a PC shop. There is an illustration of several categories representing products of the shop on Figure 1.1.

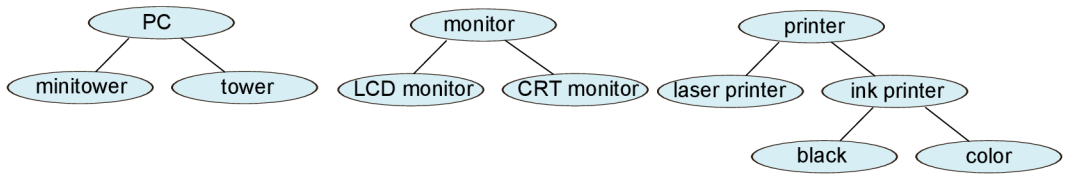


Figure 1.1: Example of the products structure in the shop.

When items are categorized in taxonomies, the sequential patterns can be divided into the following two categories:

- *multi-level* (known also as *intra-level*)
- and *level-crossing* (known also as *inter-level*) [9].

All items of *multi-level* sequential patterns are at the same level of hierarchy, whereas levels of items of *level-crossing* can be different. In Chapter 3 it is shown that very few algorithms deal with the problem of mining multi-level sequential patterns. The thesis deals with the both categories of sequential patterns and examines how to mine them effectively.

1.1 Goals of the Thesis

The Ph.D. thesis deals with the mining sequential patterns where taxonomies are defined over items in a sequence database. The hypothesis of the thesis is following:

"The existence of taxonomies makes it possible to find a new type of sequential patterns and a new method for mining it effectively can be developed."

The goal of the thesis is to verify the hypothesis. The goal is decomposed into the following three sub-goals.

1. To design and formally define the problem of mining sequential patterns with items in taxonomies.
2. To design and formally define a new method(s) or algorithm(s) which can solve the defined data mining problem effectively.
3. To experimentally evaluate properties of the developed method(s) or algorithm(s) and to compare it (them) with the existing methods.

1.2 Thesis Contribution

The main contributions of the thesis are as follows.

- The both multi-level and level-crossing categories of mining sequential patterns with items in taxonomies are discussed. There is proposed a new type of multi-level sequential patterns task which reduces redundant (useless) patterns using new constraints.
- New methods for mining of level-crossing and especially multi-level sequential patterns are introduced. There are proposed new optimization techniques which significantly increase the speed of the multi-level mining algorithm. The properties of algorithms are experimentally verified.

1.3 Structure of the Thesis

Pattern Mining is introduced in Chapter 2. The first part is focused on frequent pattern mining. Then the sequential pattern mining is defined and a problem is extended by the existence of taxonomies. The state of the art, especially existing algorithms for frequent or sequential pattern mining, is described in Chapter 3. The core of the thesis is described in Chapter 4. The first part deals with level-crossing sequential patterns and the second part is focused on the research of multi-level patterns. Here, a new data mining task is defined and algorithms to solve them are proposed. The experiments and their results are described in Chapter 5. First, the performance is compared on synthetic data, then, the practical results are shown. Finally, the results are summarized and possible following research is suggested in Chapter 6.

Chapter 2

Pattern Mining

Mining of frequent patterns is the most common of the data analysis and data mining tasks [10]. Basic concepts of mining frequent patterns and association rules are introduced first. Then, mining sequential patterns and mining with defined concept hierarchy are described. In addition, the chapter gives basic formal background to the pattern mining.

2.1 Mining Frequent Patterns

Mining frequent patterns was firstly studied by Agrawal et al. (1993) in the paper [2]. The main objective was to find such sets of items (shortly itemsets) which occur in transactions of input database more frequently than a given threshold. It is widely used to discovery of associations and correlations among input items. It produces simply understandable model of data and, therefore, the task is usually used for initial data analysis of an unknown dataset.

The task became very popular for industry and business, especially for decision making applications and marketing applications. The typical example of usage of the association rules mining is a market basked analysis. The goal is to find items which are usually purchased together. The example can be a typical computer shop which sells items such as computers, notebooks, monitors, printers, keyboards etc. The frequent pattern mining task can reveal that computers are usually purchased together with monitors and keyboards, but notebooks are purchased just alone. The task association rules analysis brings the results in the form of implication. It means that if a customer buys a computer, he probably buy also a keyboard.

2.1.1 Problem Definition

Here, the problem is described formally.

Definition 1. (Itemset) Let $I = \{i_1, i_2, i_3, \ldots, i_k\}$ be a nonempty finite set of items. Then an *itemset* T is a non-empty set of items I, such that $T \subseteq I$.

Definition 2. (Frequent Itemset) Let I be a set of items, $\mathcal{D}$ be set of transactions, such that each transaction T is $T \subseteq I$, and $A \subseteq I$ be an itemset. The transaction T

contains an itemset A iff $A \subseteq T$. A relative *support* of the itemset A is a percentage of transactions in $\mathcal{D}$ that contain A. Given the minimal support threshold value min_sup, the itemset A is called *frequent itemset* if its support is more than or equal to min_sup.

Definition 3. (Association Rule, Support of Association Rule, Confidence of Association Rule) Let I be a set of items, $\mathcal{D}$ be set of transactions and let A and B be itemsets such that $A, B \subseteq I$ and $A \cap B = \emptyset$. Then an *association rule* is the implication $A \Rightarrow B$. The *support of the association rule* $A \Rightarrow B$ is a percentage of transactions of $\mathcal{D}$ that contain $A \cup B$. The *confidence of the rule* $A \Rightarrow B$ is a percentage of transactions in $\mathcal{D}$ containing A which contain also B. This means

$$support(A \Rightarrow B) = P(A \cup B), \tag{2.1}$$

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support(A \cup B)}{support(A)}. \tag{2.2}$$

Definition 4. (Mining Frequent Patterns) Given a database $\mathcal{D}$ and a minimal support threshold min_sup, the task of finding of the complete set of frequent itemsets is called the *mining frequent patterns.*

2.1.2 Mining Maximal and Closed Frequent Itemsets

The huge number of result itemsets can be reduced using the maximal and closed restrictions of frequent itemsets [7, 15].

Definition 5. (Maximal Frequent Itemset) Let F be a set of frequent itemsets. A frequent itemset $x \in F$ is called *maximal frequent itemset* if it is not a proper subset of any other frequent itemset $x' \in F$.

Definition 6. (Frequent Closed Itemset) Let F be a set of frequent itemsets. A frequent itemset $x \in F$ is called *closed frequent itemset* if it is not a proper subset of any other frequent itemset $x' \in F$ such that $support(x) = support(x')$.

Definition 7. (Mining Maximal/Closed Frequent Patterns) Given a database $\mathcal{D}$ and a minimal support threshold min_sup, the task of finding of the complete set of maximal (closed) itemsets is called the *frequent maximal (closed) patterns mining.*

2.2 Mining Sequential Patterns

Sequential pattern mining was introduced by Agrawal and Srikant in 1995 [1]. The sequence is defined as an ordered list of transactions (itemsets) of one customer. The example of usage of the sequential pattern mining can be also demonstrated on market basket analysis. It can be expected that customers return for further purchases. Therefore, the sequential patterns over purchased items can be found. The example of such sequence can be, that a customer buys the computer with a

monitor in the one purchase and, later, the customer returns and buys a printer. If a sequence occurs in database more than a given threshold, it is called the sequential pattern.

2.2.1 Problem Definition

In this section the problem of mining sequential patterns is formalized. Firstly, the basic terms such as item, itemset, sequence and sequence database should be defined.

Definition 8. (Sequence) A *sequence* is an ordered list of itemsets. A sequence s is denoted by $\langle s_1 s_2 s_3 \dots s_n \rangle$, where s_j for $1 \leq j \leq n$ is an itemset. The itemset s_j is also called an *element* of the sequence. The *length* of a sequence is defined as the number of instances of items in the sequence. A sequence of length l is called an *l-sequence*. The sequence $\alpha = \langle a_1 a_2 \dots a_n \rangle$ is a *subsequence* of the sequence $\beta = \langle b_1 b_2 \dots b_m \rangle$ where $n \leq m$ if there exist integers $1 \leq j_1 < j_2 < \dots < j_n \leq m$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \dots, a_n \subseteq b_{j_n}$. We say that the sequence α is contained in the sequence β. We denote it $\alpha \sqsubseteq \beta$ and β is a *supersequence* of α .

Definition 9. (Sequence database) A sequence database $\mathcal{D}$ is a set of tuples $\langle SID, s \rangle$, where SID is a sequence identifier and s is a sequence.

Definition 10. (Sequence Support) Given sequence database $\mathcal{D}$, the support of a sequence s_1 in $\mathcal{D}$ is defined as the number of sequences in $\mathcal{D}$ containing a subsequence s_1. Formally stated, the support of a sequence s_1 in $\mathcal{D}$ is $support(s_1) = |\{\langle SID, s \rangle | (\langle SID, s \rangle \in D) \wedge (s_1 \sqsubseteq s)\}|$.

Definition 11. (Sequential Pattern, Mining Sequential Patterns) Given sequence database $\mathcal{D}$ and *minimal support threshold* min_supp, a *frequent sequence* is such a sequence s whose $support(s) \geq min\_sup$. A frequent sequence is called *a sequential pattern*. For a given sequence database D and a minimal support min_supp, the goal of *mining sequential patterns* is to find all frequent sequences in $\mathcal{D}$.

Example 2. (Item, Element, Sequence, Sequence Database, Sequential Pattern) For better understandability, the examples in the thesis are based on the *sequence database* in Table 2.1. The set of *items* for the example is the set $I = \{a_{11}, a_{12}, a_1, a_2, b_1, b_2, c_1, d_1, e_1, f_1, f_2, g_1, g_2, h_1, h_2\}$. The table represents a sequence database with sequences in the sequence column. Let's focus on the first row containing the sequence $s = \langle (c_1 d_1)(a_{12} b_1 c_1)(a_1 b_2 f_1)(a_{11} c_1 d_1 f_1) \rangle$. The sequence length is twelve, therefore the sequence is called *12-sequence*. The sequence consists of four *elements* (*itemsets*): $(c_1 d_1)$, $(a_{12} b_1 c_1)$, $(a_1 b_2 f_1)$ and $(a_{12} c_1 d_1 f_1)$. Note that if element contains only one item, than the parentheses around the itemset can be omitted, e.g. element e_1 on the second row. Items denoted by the same letter, which differ only in indexes, belong to one taxonomy. This notation will be described later in Example 3 on page 9.

For the following examples we assume the minimal support threshold min_supp=2, unless otherwise stated. The *support* of item b_2 is 3, denoted as $\langle b_2 \rangle : 3$, because the item is included in three sequences with SID 1, 2 and 3. Therefore, the 1-sequence

$\langle b_2 \rangle$ is frequent and is called *sequential pattern*. In contrast, the support of 1-sequence $\langle g_1 \rangle$ is 1 and it is not frequent. Further, the 2-sequences$\langle d_1 b_1 \rangle : 2$, $\langle d_1 f_1 \rangle : 2$ and $\langle (b_2 f_2) \rangle : 2$ are *sequential patterns* of length 2 with the support 2, e.g. the first sequence $\langle d_1 b_1 \rangle : 2$ is the subsequence of sequences 1 and 4 of the sequence database. In the sequence database there is no any sequential pattern longer than two.

Table 2.1: A sequence database $\mathcal{D}$ containing items on different taxonomy levels.

SID	Sequence
1	$\langle (c_1\ d_1)(a_{12}\ b_1\ c_1)(a_1\ b_2\ f_1)(a_{11}\ c_1\ d_1\ f_1) \rangle$
2	$\langle (a_{12}\ b_2\ f_2)\ e_1 \rangle$
3	$\langle (a_2\ b_2\ f_2) \rangle$
4	$\langle a_{11}\ (d_1\ g_1\ h_1)(b_1\ f_1)(a_2\ g_2\ h_2) \rangle$

2.2.2 Mining Maximal and Closed Sequential Patterns

In some special cases, some restrictions over the sequence length and support need to be defined. In general, there are two restrictions similar to those introduced in Def. 5 and Def. 6 – maximal sequential patterns and closed sequential patterns. In the case of maximal sequential patterns we are interested in sequences whose supersequences are not frequent (simply, algorithms find the longest sequences). This problem was deeply studied by Wang et al. [28]. In the case of closed sequential patterns proposed by Agrawal in [1], the change of support of the supersequences is important and it also bring us some information. In this case, we omit only subsequences whose support is the same as theirs supersequences.

Definition 12. (Closed Sequential Pattern) Given sequence database $\mathcal{D}$ and a frequent sequence s. If there is no proper supersequences of s with the same support, i.e. $\nexists s'$ such that $s \sqsubset s'$ and $support(s) = support(s')$, the sequence s is called *closed sequential pattern*.

Definition 13. (Maximal Sequential Pattern) Given sequence database $\mathcal{D}$, a frequent sequence s and *minimal support threshold* min_supp. If there is no proper frequent supersequence of s, i.e. $\nexists s'$ such that $s \sqsubset s'$ and $support(s') \geq min\_supp$, the sequence s is called *maximal sequential pattern*.

2.3 Concept Hierarchy and Taxonomies

Concept hierarchy allows describing relations between concepts (values of attributes) in database. The usage of concept hierarchy for data mining is summarized in [10] and [6]. In general, the concept hierarchies define relations between lower (more specific) and higher (more general) concepts. Formally, the concept hierarchy is a partially (or totalky) ordered set of concepts. A special case of concept hierarchy is a hierarchy of items referred as taxonomy.

Definition 14. (Concept Hierarchy) A Concept Hierarchy $\mathcal{CH}$ is a partially ordered set $(\mathbf{CH}, \preceq)$, or respectively a totally ordered set $(\mathbf{CH}, \prec)$, where $\mathbf{CH}$ is a finite set of concepts, and $\preceq$ and $\prec$ are partial and total order over $\mathbf{CH}$, respectively.

Definition 15. (Taxonomy) The taxonomy structure of an itemset V (abbr. *taxonomy*) and edges E is a rooted tree $\tau = (V, E)$ with a root $r \in V$. In the context of the tree, we refer to V as a set of nodes representing items. For each node v in the tree, let $UP(v)$ be a simple unique path from v to r. If $UP(v)$ has exactly k edges then the *level* of v is k for $k \geq 0$. The level of the root is 0. The *height* of a taxonomy is the greatest level in the tree. The *parent* of $v \neq r$, formally $parent(v)$, is the neighbor of v on $UP(v)$, and for each node $v \in V, v \neq r$ there exists a set of its *ancestors* defined as:

$$ancestors(v) = \{x | x \in UP(v), x \neq v\}. \tag{2.3}$$

The parent of r and the ancestors of r are not defined. If v is the parent of u then u is a *child* of v. A *leaf* is a node having no child [14].

In every taxonomy there exists a *is-a relation* which is defined as follows:

$$is - a : V \times V :\equiv \{(a, b) | b \in ancestors(a)\}. \tag{2.4}$$

Let $\iota = \{I_1, \ldots, I_m\}$ be a partition of a nonempty finite set of items I. Then a set of taxonomy structures of items I is a nonempty set of taxonomy structures $\mathcal{T} = \{\tau_1, \ldots, \tau_m\}$ corresponding to ι such that $\tau_i = (I_i, E_i)$ where $I_i \in \iota$ for $1 \leq i \leq m$. It means that each item $i \in I$ appears in exactly one taxonomy structure $\tau_i \in \mathcal{T}$. It should be noted that we do not require that items need to be only leaf nodes. Items $ancestors(i)$ will be referred to as *generalized* items of i.

Example 3. (Taxonomy of Items, Taxonomy Level, Parent, Ancestor, Generalized Item.) The tree structures on Figure 2.1 are called *Taxonomies of Items* which are used in the running example. The *root* symbols are alone letters from a to h which are called *root items*. Then, all descendants are denoted by down-indexes. By the definition, the *level* of item is the number of edges from item to root item, for example the *level* of a_{12} is 2. Note, that the count of down-index digits denotes the level of the item and the digit value denotes ordering of the item on the current taxonomy level.

Now, we focus on *relations* between items in the taxonomies. The a_1 is a *parent* of both a_{11} and a_{12}, denoted as $parent(a_{11}) = a_1$. Each item has almost one parent item. In contrast, the *ancestors* are a set for each item laying on the path to root, for example a_{11} has two ancestors a_1 and a, denoted as $ancestor(a_{11}) = \{a_1, a\}$. The *generalized items* of item a_{11} are both a_1 or a.

2.3.1 Mining Multi-Level and Level-Crossing

The necessity of mining association rules on different concept levels has been firstly mentioned by Agrawal et. al. in [3]. It is important to deal with multiple level pattern

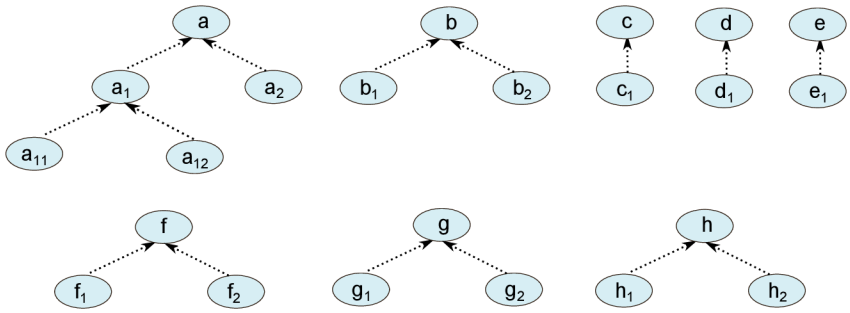


Figure 2.1: Visualization of taxonomies over items from the Example .

mining because association rules over leaf items may not satisfy minimal support but association rules over more general items in the taxonomy may satisfy it.

Therefore, the task of mining association rules is extended to the form of the generalized association rules [22] by the Def. 16.

Definition 16. (Generalized Association Rule) Let $\mathcal{D}$ be a set of transactions, $\mathcal{T}$ be a set of taxonomies and I the set of all items, where each transaction T is a set of items such that $T \subseteq I$. A transaction T *supports* an item $x \in I$ if $x \in T$ or x is an ancestor of some item in T. A transaction T *supports* a set $X \subseteq I$ if T supports every item in X. Then, a *generalized association rule* is an implication $A \Rightarrow B$, where $A, B \subseteq I, X \cap Y = \emptyset$ and no item in Y is an ancestor of any item in X. The support of the generalized association rule $A \Rightarrow B$ is a percentage of transactions in $\mathcal{D}$ which contain $A \cup B$ according to the *support* defined in this definition. The confidence of the generalized association rule $A \Rightarrow B$ is percentage of transactions in $\mathcal{D}$ supporting A that also support B.

Definition 17. (Mining Generalized Association Rules) Let $\mathcal{D}$ be a set of transactions and $\mathcal{T}$ be a set of taxonomies. The task of mining generalized association rules is to discover all rules that have support and confidence greater (or equal) than the user specified minimal support and minimal confidence values.

Chapter 3

State of the Art

The algorithms for mining sequential patterns have to deal with an ordering of transactions of customers. This section contains an overview of approaches to the sequential patterns mining. The algorithms based on candidate generation are described first and, then, the efficiency improvements based on pattern-growth approach are introduced.

3.1 GSP Algorithm

The algorithms AprioriAll and AprioriSome described in the previous section enable mining of non-constrained sequential patterns and maximal sequential patterns using post-processing procedure. Srikant introduced a new mining algorithm called *Generalized Sequential Patterns (GSP)* in [21]. The GSP allows different types of constraints of sequential patterns such as taxonomies, sliding windows and time constraints. Sliding windows and gap time constraints are not considered for the rest of the thesis. The algorithm works iteratively. It makes a pass over the sequence database in all iterations:

1. Initially, the support of items is counted in the first database pass. 1-sequences are created from items with higher support value than a minimal support *min_sup*. Such 1-sequences are inserted into a partial result set L_1 containing all *frequent 1-sequences*.

2. Then the following steps are processed iteratively until none k-sequential pattern is generated:

   (a) The *Candidate Generation* step generates C_k candidate sequences.

   (b) The *Counting Candidates* step filters the frequent sequences into the L_k sets.

3. The result set of sequential patterns is $\bigcup_k L_k$.

The *Candidate Generation* runs in Join and Prune steps.

1. In the *Join step*, a set of *candidate sequences* C_k is generated from sequential patterns in L_{k-1}. A pair of sequences $s_1, s_2 \in L_{k-1}$ can be joined if subsequences, generated by omitting of the first item of s_1 and the last item of s_2, are equal. Then, the candidate k-sequence is formed by adding the last item of the s_2 at the end of the sequence s_1 as:

   (a) the last new element containing one item x if x was in a separate element in s_2;

   (b) as a next item of the last element in s_1 otherwise.

   (c) When joining $x \in L_1$ with $y \in L_1$, both sequences $< (y)(x) >$ and $< (yx) >$ are generated as candidate sequences.

2. The *Prune step* removes candidates whose any $(k-1)$-subsequence is not frequent.

In the *Counting Candidates step*, the database is passed and the support of each candidate sequence is counted. Candidates with a support greater than *min_supp* are added into the set L_k of sequential patterns. The *contains test*, checking if a sequence s of the sequence databasecontains a candidate sequence s_c, is used for support evaluation.

3.2 PrefixSpan Algorithm

The *PrefixSpan* proposed by Pet et al. [17],[16] is a representative of the *pattern-growth algorithms*. The algorithm does not use the time-consuming generating of candidate sequences. The algorithm is based on the projected databases [11]. The PrefixSpan algorithm works as follows. In the first scan, the algorithm finds all 1-sequential patterns in the sequence database (the prefix is empty). Then, projected database construction and PrefixSpan procedure is applied for each 1-sequential patterns. Constructed projected databases are searched for a set of local frequent items again. Output sequential patterns are constructed by joining a prefix with all local frequent items. Finally, sequential patterns represents new prefixes and the PrefixSpan is run recursively.

3.3 Mining Multi-Level Sequential Patterns

The first method to deal with taxonomies over sequential patterns was described in [18]. The method is called *Uniform sequential approach* [18] . It allows using a common sequential patterns mining algorithm for mining multi-dimensional and multi-level sequential patterns. This intuitive approach is based on the extending of sequence database $\mathcal{D}$. Each sequence $s \in \mathcal{D}$ is replaced by a new sequence s' called *extended sequence* where each item of s is replaced by all its ancestors within an

element. Then, the presented AprioriSome and GSP algorithms will produce level-crossing sequential patterns. Han et al. used in [8] such *extended sequences* for mining multi-level sequential patterns by means of the PrefixSpan algorithm.

The topic of mining multi-level sequential patterns was deeply studied by Plantevit et al. [20] and [19]. They proposed several methods for mining different kinds of multidimensional and multi-level sequential patterns. The multidimensional database contains items from n distinct dimensions D_i. Then items for data mining tasks are n-tuples $i = (d_1, \ldots, d_n)$, where $d_i \in dom(D_i) \cup \{*\}$ and star symbol * denotes all items of domain D_i, called multidimensional items. They proposed an algorithm called HYPE (HierarchY Pattern Extension). They described an idea of generalization and specialization of sequential patterns [20]. The algorithm runs in two phases. In the first phase, the algorithm creates the most *specific* (items in leaf nodes of taxonomies) $n-$multidimensional items $i = (d_1, \ldots, d_n)$ denotes any item. The construction gradually replaces star symbols by specific items of domains by joining pairs of compatible hierarchical items – two items over $n-$dimensions are compatible, if items share $n-2$ items. In the second phase, the sequential patterns are iteratively mined using an Apriori theorem.

Chapter 4

Level-Crossing and Multi-level Sequential Pattern Mining

Basic concepts of mining sequential patterns were described in previous Chapters 2 and 3. There were introduced the field of mining level-crossing and multi-level sequential patterns. It was indicated that the usage of taxonomies of items can help to find new results and new knowledge.The analysis of the state-of-the-art exposed that such mining task is important and challenging, however, there does not exist any satisfying solution.

In this chapter, my research over level-crossing and multi-level sequential pattern mining is described. The main idea of my research is that the generalization of sequence items can be performed when the subsequence support does not reach the minimal support value. I have studied both sub-problems – level-crossing and, also, multi-level sequential patterns. The naive solution for mining the level-crossing sequential patterns uncovered the main issues of the task which are the huge search space and the large result set. Therefore, there are proposed the constrains for mining multi-level sequential patterns which simplifiy the complexity of the mining process. The chapter summarizes facts published in research papers [24], [25] and [26]. All those research papers were presented as results of TAČR research project TA01010858: "*Improving Security of the Internet by Using System for Analyzing of Malicious Code Spreading*" .

For the rest of the thesis two basic tasks will be distinguished:

- Level-crossing sequential patterns – items of sequential patterns can be generalized to any level of taxonomy.
- Multi-level sequential patterns – items of sequential patterns have to have the same level of taxonomy.

Example 4. The difference between the complexity of level-crossing and multi-level sequential patterns mining is shown on Figure 4.1. The multi-level mining approach creates only the most bottom and top sequences: $\langle a_1\ (b_2\ f_2)\rangle$ and $\langle a\ (b\ f)\rangle$ because levels of items of those sequential patterns must be the same. However, there are more 5 level-crossing sequential patterns between the pair of multi-level sequential

patterns of length three. With the length of the sequential patterns, the number of level-crossing patterns increases dramatically.

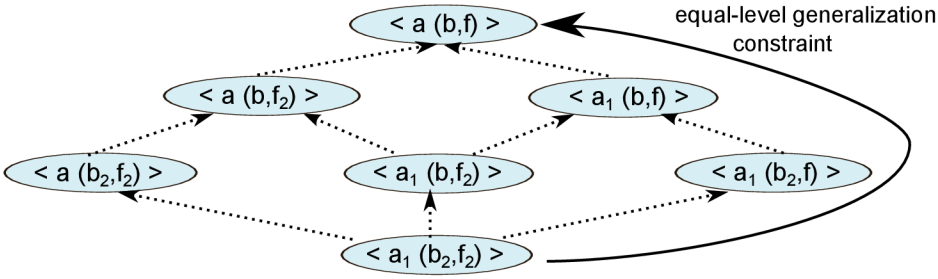


Figure 4.1: Difference between level-crossing and multi-level generalization of sequence $\langle a_1\ (b_2 f_2)\rangle$.

A new support measure called a *generalized support* is introduced. The *generalized support* *gen_supp* is based on Def. 10 but the sequence subset relation is replaced by the *generalized subset relation* $\subseteq_g$ for the generalized support. Then, the generalized support must test if the subsequence contains an item or any of its descendants.

Definition 18. (Generalized Support) Given elements $e_1, e_2 \subseteq I$, the *generalized subset relation* $\subseteq_g$ is defined as

$$e_1 \subseteq_g e_2 \quad \Leftrightarrow \quad \forall i \in e_1 : i \in e_2 \vee \exists j \in e_2 : i \in ancestors(j). \tag{4.1}$$

A sequence $\alpha = \langle a_1 a_2 \dots a_n\rangle$ is a *generalized subsequence* of a sequence $\beta = \langle b_1 b_2 \dots b_m\rangle$ if there exist integers $1 \leq j_1 < j_2 < \dots < j_n \leq m$ such that $a_1 \subseteq_g b_{j_1}, a_2 \subseteq_g b_{j_2}, \dots, a_n \subseteq_g b_{j_n}$. We denote $\alpha \sqsubseteq_g \beta$. Formally, the definition of *the generalized support* of a sequence s_1 is

$$gen\_supp(s_1) = |\{\langle SID, s\rangle | (\langle SID, s\rangle \in D) \wedge (s_1 \sqsubseteq_g s)\}|. \tag{4.2}$$

The rest of the thesis will use shortened term *support* instead of *generalized support*.

Definition 19. (Generalization Procedure) The *generalization procedure* (shortly *generalization*) is a procedure whose input is the node n of the taxonomy τ and the output is the subset N_a of ancestors of node n:

$$N_a \subseteq \tau \wedge n \in \tau \wedge N_a \subseteq ancestors(n). \tag{4.3}$$

4.1 Mining Level-Crossing Sequential Patterns

Sequential patterns are such subsequences which occur frequently in a sequence database. Level-crossing sequential patterns allow items to be on different levels of

taxonomies. On the other hand, the search space significantly grows for level-crossing sequential patterns.

This section is based on facts published in [24] and [25]. First, the problem of mining level-crossing sequential patterns is formalized. Especially, the relations as parents and ancestors are defined for level-crossing sequences. Then, the algorithm for mining level-crossing sequential patterns is proposed.

4.1.1 Problem Definition

Definition 20. (Element Parents) Given an element $e = \{i_1, i_2, \ldots, i_n\}$, an *element parents* of the element e is a set of the elements which are the same as e except one of the items which is generalized. Formally,

$$\begin{aligned} parents_{el}(e) &= \{e \setminus \{i_k\} \cup \{parent(i_k)\} | i_k \in e \\ &\quad \wedge parent(i_k) \notin e \wedge 1 \leq k \leq n\}. \end{aligned} \tag{4.4}$$

Note that items inside elements can be linearly ordered without lost of generality.

Now, the sequence parents and sequence ancestors can be defined using the element parent definition.

Definition 21. (Sequence Parents) Given a sequence $s = \langle e_1 e_2 \ldots e_n \rangle$, where e_k are elements. The *sequence parents* of s is the set of sequences that are the same as the sequence s except one of their elements which is replaced by one of its element parents. Formally,

$$\begin{aligned} parents_{seq}(s) &= \{\langle f_1 f_2 \ldots f_n \rangle | f_k \in parents_{el}(e_k) \\ &\quad \wedge 1 \leq k \leq n \\ &\quad \wedge \forall l \neq k, 1 \leq l \leq n : e_l = f_l\}. \end{aligned} \tag{4.5}$$

Definition 22. (Root Sequence) Given taxonomy τ, a *root sequence* is a sequence consisting of elements with items corresponding to root nodes only. The set of sequence parents of a root sequence is an empty set.

Definition 23. (Sequence Ancestors) Given the sequence s, the *sequence ancestors* of the sequence s is defined as follows:

$$\begin{aligned} ancestors_{seq}(s) &= M_i, \text{ for such } i \text{ that } M_{i+1} = M_i, \textit{where} \\ M_0 &= parents_{seq}(s) \\ M_{i+1} &= M_i \cup \{parents_{seq}(x) \mid x \in M_i\} \\ &\quad \text{for } i \geq 0. \end{aligned} \tag{4.6}$$

Example of sequence parents and sequence ancestors of sequence are in Example 5.

Example 5. For a given sequence $\langle a_{12}\ a_{11}\rangle : 1$, a set of parent sequences is the set of two sequences $\{\langle a_{12}a_1\rangle : 1, \langle a_1\ a_{11}\rangle : 1\}$. The set of ancestors of the sequence $\langle a_{12}\, a_{11}\rangle : 1$ is the set of sequences $\{\langle a_{12}\, a_1\rangle : 1, \langle a_1\, a_{11}\rangle : 1, \langle a_1\, a_1\rangle : 1, \langle a_1\, a\rangle : 2, \langle a\, a_1\rangle : 1, \langle a\ a\rangle : 2\}$. The sequence $\langle a\ a\rangle : 2$ is the root sequence and it has no parent and ancestor sequences.

4.1.2 The hGSP Algorithm

In this section, the algorithm *hGSP (hierarchical-GSP)* for mining level crossing sequential patterns is introduced. The algorithm is based on GSP [21] described in Section 3.1. In contrast to the method based on "extended-sequences"', the hGSP algorithm reduces the number of redundant patterns. If a sequence s is frequent and $s_1 \in ancestors_{seq}(s)$, then s_1 must be also frequent. Therefore, the sequence s_1 is *redundant* because it does not contain any new information. Due to the observation our algorithm does not generate all possible generalizations of frequent sequences. It performs generalization only when the sequence would be pruned. The hGSP is based on the idea of *concreteness* of each sequence. The concreteness measure is evaluated using information theory explained in the following subsection.

Concept of hGSP Algorithm

The main idea of the hGSP algorithm is that if a sequence s has support $gen\_supp(s)$, there can exist a generalized sequence $s_g \in parents_{seq}(s)$ such that $gen\_supp(s_g) > gen\_supp(s)$. This can be applied repeatedly. Note that $\forall s_g \in parents_{seq}(s) : gen\_supp(s) \leq gen\_supp(s_g)$.

Generally, more specific sequence s is more important result than its generalized form s_g because the generalized s_g is more expectable in the result set. It corresponds to the concept of Shannon *information content* [12].

For a sequence s, the dependence between information content $h(s)$ and generalized support $gen\_supp(s)$ causes that if the generalization from s to s_g is performed and $gen\_supp(s_g) > gen\_supp(s)$, then $h(s_g) < h(s)$. Some information is lost during generalization. Therefore, the generalization should be performed only if the candidate sequence is not frequent (i.e. $gen\_supp(s) < min\_supp$) or the GSP algorithm cannot perform *join* of two candidate sequences with joinable sequence ancestors.

Definition 24. (Concreteness) A sequence s_1 is more *concrete* than a sequence s_2 if $(h(s_2) < h(s_1)) \wedge (ancestors_{seq}(s_1) \cup s_1) \cap (ancestors_{seq}(s_2) \cup s_2) \neq \emptyset$.

It means that to be more concrete $s_1$must have higher information content then s_2 and both sequences must have at least one common ancestor or one sequence is ancestor of the other.

Algorithm details

The hGSP algorithm uses modified *join step* and *pruning step* of the GSP algorithm. The rest of the algorithm remains the same. The hGSP algorithm implementation assumes that items in elements are in lexicographic order.

The *join step* is modified for generating candidates of length $k \geq 3$. Let s_1 and s_2 be a pair of frequent sequences of length $k-1$. The join can be performed if subsequences of s_1 after omitting the first item and s_2 after omitting the last one have a common *sequence ancestor*. Then the joined sequence of length k is composed from the first item of s_1, the most concrete sequence ancestor of common part and the last item of s_2. The last item is added as in GSP.

Support of candidates is counted similarly to original GSP. The only difference is that we use $gen\_supp(s)$ defined in Def. 18 instead of support. Therefore, only the procedure for checking, if a candidate is a subsequence of sequences in a given sequence database, is modified.

The modification of the *pruning step* is shown in Algorithm 4.2. The algorithm uses a method for finding the approximation of the most concrete generalization set of sequences which is described in Algorithm 4.1. The hGSP algorithm is based on the *greedy optimization* technique [5]. The method $FindGeneralization(s)$ returns the set G_s of most concrete generalizations of the sequence s with higher information value. Then the $hGSP$ algorithm checks, if each sequence in G_s is frequent. If so, it is added into set of sequential patterns, otherwise the candidate sequence is generalized again. Therefore, the algorithm finds only sequences corresponding to the local optimum of concreteness measure. Finding of a global optimum would be extremely computationally complex. It is not necessary to evaluate information content using logarithm functions but it is sufficient to compare ratios of supports of sequences and theirs generalized forms.

Given sequence s and its generalized form s_1, the information contents of these sequences are $h(s) = -\log_2 \frac{gen\_supp(s)}{|D|}$ and $h(s_1) = -\log_2 \frac{gen\_supp(s_1)}{|D|}$. The information lost during generalization of s to s_1 is $\Delta h = h(s) - h(s_1)$. It follows that

$$\Delta h = \log_2 \left(\frac{\frac{gen\_supp(s_1)}{|D|}}{\frac{gen\_supp(s)}{|D|}} \right) = \log_2 \left(\frac{gen\_supp(s_1)}{gen\_supp(s)} \right). \tag{4.7}$$

The generalization of s with the smallest information loss is found because then the sequences will be the most concrete. Therefore, the algorithm minimizes ratio $\frac{gen\_supp(s_1)}{gen\_supp(s)}$.

Generalized sequences which contain ancestor item of another item in the same element are redundant and they are discarded.

4.2 Mining Hierarchically-Closed Multi-Level Sequential Patterns

This section presents the core result of my research work. It deals with the formal definition of the newly formulated task of *mining hierarchically-closed sequential patterns*, and then it describes a new algorithm for mining such sequential patterns.

The level-crossing kind of sequential patterns introduced in previous section is the natural taxonomical (hierarchical) extension of the sequential patterns. However, the mining process of such patterns is very difficult and computationally expensive.

Algorithm 4.1 Method FindGeneralization()

```
1: procedure FindGeneralization(s)
2:     G_s = {}
3:     min_supp_ratio = +∞
4:     for all p_s ∈ parents_seq(s) do
5:         ratio = gen_supp(p_s)/gen_supp(s)
6:         if (gen_supp(p_s) ≠ gen_supp(s) ∧ ratio < min_supp_ratio then
7:             G_s = {p_s}
8:             min_supp_ratio = ratio
9:         else if (ratio = min_supp_ratio then
10:            G_s = G_s ∪ {p_s}
11:        end if
12:    end for
13:    return G_s
14: end procedure
```

Therefore, the simplification of the problem was introduced in the research paper [26]. The improvement is based on the multi-level sequential patterns concept. The main idea is to find only patterns containing items of the same level. It reduces the number of searched paths during the mining process.

The difference is explained on the following example which uses the taxonomies of a shop from Example 1 on page 3. The possible result of mining level-crossing sequential patterns can contain e.g. sequences like $\langle$*PC_minitower ink_printer*$\rangle$, $\langle$*PC_minitower printer*$\rangle$ *or* $\langle$*PC printer*$\rangle$ because there is no constraint for the combination of the level of items. The multi-level sequential patterns, by contrast, must not contain the sequential pattern as $\langle$*PC_minitower printer*$\rangle$ because the levels of items *PC_minitower* and *printer* are different.

The hierarchically-closed sequential patterns follow the idea of the hGSP algorithm which reveals only the most concrete patterns using the information content measure. It was observed, that the result becomes more clear and revealing if the closed patterns are used [28], [1]. In our example, the analyst could be overloaded by redundant patterns if the result contains all $\langle$*(PC_minitower LCD_monitor) ink_printer*$\rangle$, $\langle$*PC_minitower ink_printer*$\rangle$, $\langle$*LCD_monitor ink_printer*$\rangle$, $\langle$*(PC_minitower LCD_monitor)*$\rangle$, etc. Moreover, no information is lost if the non-closed patterns are omitted and the longest sequential patterns with the equal support are found. On the other hand, the mining of the close patterns are more complicated, because result patterns must be retroactively pruned. In our work, the "close" problem is applied to the process of generalization. It leads to the similar type of redundant patterns like in mining closed sequential patterns – only the most specific patterns with the no support change are revealing. Ancestors of the frequent hierarchically-closed multi-level sequential patterns are always also frequent. However, the change of the support during the generalization is important. Therefore, we focus on the mining the hierarchically–closed instead of the hierarchically–maximal sequential patterns.

Algorithm 4.2 Pseudocode of hGSP Pruning Step

```
1: procedure hGSP(C_k, min_supp)
2:     L_k = {}
3:     for all s_c ∈ C_k do
4:         C'_k = {s_c}
5:         sequence_added = false
6:         while sequence_added = false ∧ |C'_k| > 0 do
7:             G_s = {}
8:             for all s ∈ C'_k do
9:                 if gen_supp(s) ≥ min_supp then
10:                    L_k = L_k ∪ {s}
11:                    sequence_added = true
12:                else
13:                    G_s = G_s ∪ FindGeneralization(s)
14:                end if
15:            end for
16:            C'_k = G_s
17:        end while
18:    end for
19:    return L_k
20: end procedure
```

4.2.1 Problem Definition

This section deals with the formal basics of the mining hierarchically-closed multi-level sequential patterns. It follows the definitions Def. 1 (Itemset), Def. 8 (Sequence), Def. 9 (Sequence Database), Def. 14 (Concept Hierarchy), Def. 15 (Taxonomy of Items) and Def. 18 (Generalized Support).

First, the multi-level (ML) extensions of element, sequence, parent and ancestors must be defined. The definitions of ML-element and the ML-sequence are derived from definitions of element and sequence. The Definition 25 extends the element and the sequence definitions using items from nodes of taxonomies where the level of all items must be the same. The rest of the definition remains unchanged.

Definition 25. (ML-element, ML-sequence) Let $l \in \boldsymbol{N}$ be a level of items in a taxonomy $T \in \tau$. Then an *ML-sequence* is an ordered list of itemsets $s_{ML} = \langle s_1 s_2 s_3 \dots s_n \rangle$ such that the levels of all items of the itemsets are equal to l. The itemset of the ML-sequence is called an *ML-element*. The *length, subsequence and supersequence* of an ML-sequence is defined analogously to the ones in Definition 8.

Example 6. Next examples will be based on running example Example 2 on page 7. Assume three sequential patterns $\langle a_1\ (bf_1) \rangle : 2$, $\langle a_1\ (bf) \rangle : 2$ and $\langle a\ (bf) \rangle : 2$. The first sequence $\langle a_1\ (bf_1) \rangle$ is not a ML-sequence because the level of items differs in the element (bf_1) – the level of b is 0 and the level of $f_1$is 1. The second sequence $\langle a_1\ (bf) \rangle$ is not a ML-sequence too because the level of items differs between elements. Finally, only the third sequence $\langle a\ (bf) \rangle$ is a ML-sequence because it satisfies Def.

25. Therefore, only the third sequence may be included in the result of the mining multi-level sequential patterns. Note that the sequence $\langle a\ (bf)\rangle$ is the root sequence because all its items are the root items.

Here, it is possible to define taxonomic relations between ML-sequences. The *ML-element parent* can be simply obtained by replacing all items of a ML-element by their parent items. Then for the *ML-sequence parent*, all its ML-elements are replaced by their ML-element parents. Note that the parent of a level-crossing sequence is a set of sequences but a ML-sequence has only one ML-sequence parent. The *ML-sequence ancestors* are the union of ML-sequence parents recursively up to the root of a ML-sequence. These statements are formalized in the following definitions.

Definition 26. (ML-element parent) Given an ML-element $e = \{i_1, i_2, \ldots, i_n\}$, an *ML-element parent* of the ML-element e is an element whose items are obtained by replacing all items of their parents. This is defined as

$$parent_{el}(e) = \{parent(i_k) | 1 \leq k \leq n \wedge i_k \in e\}. \tag{4.8}$$

Definition 27. (ML-sequence parent, ML-sequence ancestors) Given an ML-sequence $s = \langle e_1 e_2 \ldots e_n\rangle$, where e_k is a ML-element on a position k, the *ML-sequence parent* of s is an ML-sequence such that all ML-elements of s are replaced by their ML-element parents. Formally,

$$parent_{seq}(s) = \langle f_1 f_2 \ldots f_n\rangle,\ f_k = parent_{el}(e_k), 1 \leq k \leq n. \tag{4.9}$$

Definition 28. (ML-sequence ancestors) For a given set of taxonomies τ, a *root ML-sequence* is an ML-sequence consisting of ML-elements with items corresponding to root nodes of taxonomies. The ML-sequence parent of a root ML-sequence is not defined. Based on the definition of the ML-sequence parent, the *ML-sequence ancestors* of an ML-sequence s, $ancestors_{seq}(s)$ is defined recursively as follows:

$$\begin{aligned} ancestors_{seq}(s) &= M_i, \text{ for such } i \text{ that } M_{i+1} = M_i, where \\ M_0 &= \{parent_{seq}(s)\} \\ M_{i+1} &= M_i \cup \{parent_{seq}(x) \mid x \in M_i\} \text{ for } i \geq 0. \end{aligned} \tag{4.10}$$

Example 7. For a given ML-sequence $\langle a_{12} a_{11}\rangle : 1$, the ML-sequence parent is $\langle a_1 a_1\rangle : 1$. The set of ML-sequence ancestors of $\langle a_{12}\ a_{11}\rangle : 1$ is the set of two ML-sequences $\{\langle a_1\ a_1\rangle : 1, \langle a\ a\rangle : 2\}$. The ML-sequence $\langle a\ a\rangle : 2$ is the root sequence and it has no ML-sequence parent and ML-sequence ancestors. Note that if the input sequence is an ML-sequence, then the result parent sequence and ancestor sequences are ML-sequences too because of the principle of their construction.

The multi-level approach reduces the search space of the data mining task. Moreover, we try to reduce the number of redundant (unimportant) patterns. Recall the term *closed* in *closed sequential pattern mining*. The *closed* means that if a sequence s and a supersequence of s have the same support, then the result set will contain

only a supersequence of s. In this case, any omitted subsequence can be derived from the result set.

In the case of mining multi-level sequential patterns, the closeness property can be applied for taxonomic relations. If a ML-sequence s and the ML-sequence ancestor of s have the same support, then the result set will contain only the ML-sequence s. A new data mining task is called **mining hierarchically-closed multi-level sequential patterns**. It has the following two fundamental properties:

- Only ML-sequences are revealed. It ensures fulfillment of equal-level of all items in the sequential patterns. The generalization (level changes) are allowed during the mining process, however, all newly constructed sequences are ML-sequences.
- Sequences are filtered for the hierarchically-closed condition. If some ML-sequences are in the ancestor relation and have the same value of the generalized support, then only the most-bottom sequences in the meaning of taxonomies are revealed. The generalized support must be used because the generalized ML-sequences are supported by their more specific variants.

Let's summarize all three basic constraints for the task of mining hierarchically-closed multi-level sequential patterns:

- **Constraint 1 (C1)**: A sequential pattern s must have sufficient support.
- **Constraint 2 (C2)**: A sequential pattern s must be an ML-sequence.
- **Constraint 3 (C3)**: A sequential pattern s must be hierarchically-closed.

The mining problem is formalized in the Definition 29.

Definition 29. (Mining hierarchically-closed multi-level sequential patterns) The *set of hierarchically-closed ML-sequences* is such a set of ML-sequences which *does not* contain any ML-sequence s and its ML-sequence ancestor with *equal* generalized supports. Then, the problem of **mining hierarchically-closed multi-level sequential patterns** (hereinafter *ML-sequential patterns*) for a given input sequence database D and minimal generalized support threshold min_supp is to find a set L_{ML} of all ML-sequences in $\mathcal{D}$ such that:

$$\begin{aligned} L_{ML} \;=\; & \{s_{ML} \sqsubseteq s | \langle SID, s\rangle \in \mathcal{D} \wedge gen\_supp(s_{ML}) \geq min\_supp \\ & \wedge\, \nexists s_x \sqsubseteq_g s [gen\_supp(s_x) \geq min\_supp \\ & \wedge gen\_supp(s_x) = gen\_supp(s_{ML}) \\ & \wedge s_{ML} \in ancestor_{seq}(s_x)]\}. \end{aligned} \tag{4.11}$$

4.2.2 The MLSP Algorithm

Han et al. in their book [10] characterized the sequential pattern mining by following words: *"Sequential pattern mining is computationally challenging because such*

mining may generate and/or test combinatorial explosive number of intermediate subsequences." The task of *mining hierarchically-closed multi-level sequential patterns* is even more difficult because of the traversing taxonomies and the result pruning. As the research result, the algorithm MLSP (*Multi-Level Sequential Patterns* algorithm) was proposed in [26] for the effective data mining of multi-level sequential patterns.

The algorithm MLSP is based on the candidate generation principle (adapted from the GSP, see Section 3.1) combined with the on-demand generalization. The algorithm works in phases.

The first phase

The algorithm passes through the sequence database and the values of the generalized support are counted for all items. Unlike GSP, the MLSP continues the first phase by generalization procedure. Candidate 1-sequences are created from all items in the sequence database $\mathcal{D}$. Candidate sequences are processed as follows:

1. The set of candidate 1-sequences is expanded by their all ML-sequence ancestors.
2. The value of the generalized support is counted for all candidate 1-sequences.
3. All hierarchically-closed 1-sequences with the sufficient support are added into the set of sequential patterns.

Sequential patterns of length 1 are outputted by the algorithm and passed to the second phase.

The next phases

The next phases of the algorithm run repeatedly until any new sequential pattern is generated. There are two steps during each phase:

1. *candidate generation step,*
2. *counting candidates step.*

Candidate Generation Step

The *candidate generation step* is based on the *join* and *prune* principles. In the *join* procedure, all pairs of k-length ML-sequential patterns are taken. They are tested if they are joinable to the $(k+1)$-length candidate ML-sequences. Similarly to GSP, a k-length ML-sequential pattern s_1 can be joined with a k-length ML-sequential pattern s_2 if the subsequence created by removing the first item of s_1 and the subsequence created by removing the last item of s_2 are equal. Moreover in MLSP, the ML-sequences are also joinable if it is possible to perform such generalization of both subsequences of sequences $s_1$and s_2, in which a common ML-sequence ancestor can be found. The MLSP algorithm tries to find the common ML-sequence ancestor of the candidate ML-subsequences in a *bottom-up way.* If a common ML-subsequence

ancestor exists, then the generalized ML-sequences are joined into the new candidate ML-sequence, otherwise, no candidate is generated. Levels of ML-sequences $s_1$and s_2 can be different, but the levels of items of the generated ML-sequence are the same. Finally, the prune principle is applied. The pruning is based on the Apriori theorem of the possible frequent sequences. For the multi-level sequential patterns, the **Apriori theorem** must be modified as follows (further referred as to MLSP Apriori Rule): *All ML-subsequences and their ancestors of a frequent ML-sequence are frequent too.*

The procedure for candidate generation is shown in Algorithm 4.3. Finally, the whole procedure is explained in the running Example 8,

Algorithm 4.3 Method GENERATECANDIDATEMLSEQUENCES()

```
1: procedure GENERATECANDIDATEMLSEQUENCES(L_{k-1}, k)
2:     C_k = ∅
3:     for all s_1, s_2 ∈ L_{k-1} do
4:         if ML-subsequences MLSP join condition is fulfilled for s_1 and s_2 then
5:             Join sequences s_1 and s_2 to a new ML-sequence s'
6:             if the MLSP Apriori Rule is fulfilled for s' then
7:                 Add s' into C_k.
8:             end if
9:         end if
10:    end for
11:    return C_k
12: end procedure
```

Example 8. Assume the join of the following two 2-length multi-level sequential patterns: $\langle ba \rangle : 2$ and $\langle a_1 f_1 \rangle : 2$. These two ML-sequences are firstly tested if the join is possible. Because the ML-subsequences $\langle a \rangle$ and $\langle a_1 \rangle$ has a common ancestor $\langle a \rangle$ they are joinable. The second ML-sequence is generalized to $\langle af \rangle$ and then ML-sequences are joined into a new ML-sequence $\langle baf \rangle$. Finally, the ML-sequence is tested for MLSP Apriori Rule. The ML-subsequence $\langle bf \rangle$ and it's any ancestor is not frequent, therefore, the ML-sequence $\langle baf \rangle$ is also not frequent and the ML-sequence is not added to the set of candidate sequences. In another case, assume the join of ML-sequential patterns $\langle (ab) \rangle : 3$ and $\langle (bf) \rangle : 4$. The join condition is fulfilled and a new ML-sequence $\langle (abf) \rangle$ is created. The Apriori test verifies that all ML-subsequences $\langle (ab) \rangle$, $\langle (bf) \rangle$ and finally $\langle (af) \rangle$ are frequent . The ML-sequence $\langle (abf) \rangle$ is added to the set of candidate sequences.

Counting Candidates Step

When all candidate ML-sequences are generated, the frequent sequential patterns are filtered by the support value. The counting step consists of two substeps: test and generalization procedure and pruning of not hierarchically-closed sequential patterns.

The idea of the test and generalization substep is to read the sequence database and count the generalized support of all candidate ML-sequences $s_c \in C_k$. For each s_c, one of the following results is possible:

1. The generalized support value **satisfies** the minimal support threshold and the ML-sequence is marked as frequent one, denoted as s_c^f.

2. The generalized support value **does not satisfy** the minimal support threshold and then the generalization procedure is performed. The generalization of the ML-sequence tries to find a ML-sequence ancestor with the greatest sequence level **which satisfies** the minimal support threshold. The on-demand *bottom-up* generalization procedure GetFirstFrequentAncestor() is shown in Algorithm 4.4. Upper-level ML-sequence is tested recursively until the ancestor is found or the generalization procedure reach the root.

Algorithm 4.4 Method GetFirstFrequentAncestor()

```
1: procedure GetFirstFrequentAncestor(s, min_supp)
2:     repeat
3:         if gen_supp(s) ≥ min_supp then
4:             return s
5:         end if
6:         s ← parent_seq(s)
7:     until s is root sequence
8:     return null
9: end procedure
```

Example 9. The length 2 ML-sequence $\langle a_1 a_1 \rangle$ is generated in the running example from the 1-sequence $\langle a_1 \rangle : 3$ by the Candidate Generation step. All subsequences are frequent, therefore, the sequence may be frequent. However, after Counting Candidates, the generalized support of the sequence is 1 which does not satisfy the minimal support threshold value. Therefore, the generalization is performed by the MLSP algorithm and the ML-sequence parent $\langle a\, a \rangle$ is formed. The Counting Candidates step evaluates the generalized support to 2. The ML-sequence ancestor (ML-sequence parent) $\langle a\, a \rangle : 2$ of ML-sequence $\langle a_1 a_1 \rangle : 1$ is frequent and it is a ML-sequential pattern. Moreover, the hierarchically-close condition is satisfied and the sequence is hierarchically-closed multi-level sequential pattern by our definition.

The MLSP Algorithm Summarization

The algorithm MLSP has two inputs: a sequence database $\mathcal{D}$ with a taxonomy (or taxonomies) defined for its items and a minimal support threshold value.

The algorithm output is the set of hierarchically-closed multi-level sequential patterns. The algorithm MLSP runs in the phases. The sequence database $\mathcal{D}$ is passed once in each phase. The first phase generates 1-length hierarchically-closed multi-level sequential patterns. Next phases generate $(k+1)$-length hierarchically-closed

multi-level sequential patterns from the k-length sequential patterns. Because there can exist candidate ML-sequences that are not hierarchically-closed, it is necessary to verify that there is no child of the candidate ML-sequence with the same generalized support. The procedure for the effective check of this constraint is described in Section 4.2.5. The algorithm runs until any hierarchically-closed multi-level sequential patterns are generated. The algorithm generates the complete set of hierarchically-closed multi-level sequential patterns. The complete MLSP algorithm is formalized in Algorithm 4.5.

Algorithm 4.5 The pseudocode of the MLSP algorithm

1: **procedure** MLSP($\mathcal{D}, min\_supp$)
2: $k \leftarrow 1$ ▷ First phase.
3: $I \leftarrow$Insert all items and all their ancestors i in $\mathcal{D}$ and count their support $gen\_supp(i)$
4: $C_1 \leftarrow$Add all 1-ML-sequences for all items i from I
5: $L_1 \leftarrow \{\}$
6: **for all** $s_c \in C_1$ **do**
7: **if** $gen\_supp(s_c) \geq min\_supp$ and s_c is hierarchically-closed **then**
8: $L_1 \leftarrow L_1 \cup \{s_c\}$
9: **end if**
10: **end for**
11: **while** $L_k \neq \emptyset$ **do** ▷ Next iterative phases.
12: $k \leftarrow k + 1$
13: $C_k \leftarrow$GenerateCandidateMLSequences(L_{k-1}, k)
14: Count support $gen\_supp(s)$ in $\mathcal{D}$ for all candidate ML-sequences and their ML-sequence ancestors $s \in \cup_{s_c \in C_k} ancestor_{seq}(s_c) \cup \{s_c\}$
15: $L_{TMP} \leftarrow \{\}$
16: **for all** $s_c \in C_k$ **do**
17: $L_{TMP} \leftarrow L_{TMP} \cup$ GetFirstFrequentAncestor(s_c, min_supp)
18: **end for**
19: $L_k \leftarrow \{\}$
20: **for all** $s \in L_{TMP}$ **do**
21: **if** s is hierarchically-closed **then**
22: $L_k \leftarrow L_k \cup \{s\}$
23: **end if**
24: **end for**
25: **end while**
26: **return** $\bigcup_{i=1}^{k} L_i$
27: **end procedure**

4.2.3 Optimization 1: Is-generalized-subsequence Check in Linear Time-Complexity

The algorithm often performs "is-generalized-subsequence" test (e.g. for the generalized support counting). It uses the *generalized subset relation* $\sqsubseteq_g$. The test can be optimized to the linear time-complexity if a suitable complete ordering exists over items. The simple lexicographical ordering cannot be used for MLSP because the simple lexicographical ordering cannot be used because of the generalization which changes the order. Therefore, MLSP uses two step ordering.

1. It sorts taxonomies lexicographically by their roots. It provides for a grouping of items within elements by taxonomy.

2. Items within the taxonomy must be sorted unambiguously. The bottom-up order is suitable, because such ordering can be used for join step comparison and searching of the minimal necessary generalization. Suitable order type is a post-order walk [4].

3. It guarantees that it is possible to check for an ideal mapping to ancestors in linear time complexity.

The procedure IsGeneralizedSubsequence() tests if a sequence s_{sub} is the generalized subset of a sequence s_{super}: $s_{sub} \sqsubseteq_g s_{super}$. The maximal time complexity of the procedure is $m+n$ where m is a number of elements in sequence s_{sub} and n is the number of elements in sequence s_{super}. The same is for each tested element in subprocedure ContainsGeneralizedElement(). Finally, IsAncestor() runs with constant time-complexity using a hash table or with linear time complexity using tree traversal. Therefore, the whole procedure keeps linear time-complexity w.r.t. to lengths of sequences s_{sub} and s_{super}.

4.2.4 Optimization 2: Hash Table Pre-Check for Is-generalized-subsequence Check

The majority of "is-generalized-subsequence" tests return *false*. Such major *false* case can be optimized by the pre-check. The $s_{sub} \sqsubseteq_g s_{super}$ is *true* if all items of subsequence s_{sub} are contained in the set of items and their ancestors of supersequence s_{super}. If it is false, then the test results in false too. Then, a set of all items and their ancestors is constructed for each sequence. Finally, the procedure IsGeneralizedSubsequence() can be completed by the fast pre-check for false result. The procedure is denoted in Algorithm 4.6. The procedure assumes that there exists a simple function `GetAllItems()` which returns the set of all items in all elements of the sequence.

Such set is organized (stored) as a hash table in a main memory because its search time complexity is equal to 1 (details about generic hash table algorithms and their properties are in [4]). Maximal number of searches in the hash table is equal to the length of the sequence s_{sub} . Final time-complexity of the whole pre-check is

maximally linear too but it speeds-up the check for the most cases (see experiments in the next chapter).

Algorithm 4.6 Pre-check procedure pseudocode for method IsGeneralizedSubsequence()

```
1: for all i ∈ GetAllItems(s_sub) do
2:     if i ∉ GetAllItems(s_super) ∧ i ∉ ⋃_{x∈GetAllItems(s_super)} ancestor(x)} then
3:         return false
4:     end if
5: end for
```

4.2.5 Optimization 3: Is-redundant Fast Check

Sequential patterns created by the join and generalization algorithm steps may not be hierarchically-closed. Then, the post-processing (filtering) is necessary. A naive approach compares each pair of sequential patterns, if one ML-sequence is an ML-sequence ancestor of the other and prunes them, if so. Nevertheless, it is possible to utilize the *Counting Candidates Step* procedure to mark sequential patterns which are redundant.

- First, we associate a new helper indexed list of counters called a *redundant base* to all candidate ML-sequences before the counting step. During the counting step of a candidate ML-sequence s_c , the algorithm increments by one the redundant base counter on index s_c^f to all ancestors: $S = ancestors_{seq}(s_c^f)$ when the generalization sub-procedure finds the most specific frequent sequential pattern $s_c^f \in ancestor_{seq}(s_c) \cup \{s_c\}$. The redundant base of a ML-sequence $x \in S$ on index s_c^f is denoted as $\mathcal{RB}_x[s_c^f]$.

- Finally, the prune condition can be formulated as follows:

  - If there **exists any redundant base counter** with value equal to the value of the **generalized support** of the ML-sequence s_c^f, then the ML-sequence s_c^f **is redundant** and is pruned,
  - **else**, s_c^f **is hierarchically-closed multi-level sequential pattern**.

Chapter 5

Experimental Evaluation

The issue of mining sequential patterns is generally computationally expensive. If we imagine that the sequence length is the horizontal dimension, then the mining multi-level sequential patterns adds a new vertical dimension over the patterns. The complexity of the problem growths because the algorithm must deal with relations between different multi-level sequences.

This chapter deals with a comparison of different multi-level approaches and algorithms to solve them. The first section of experiments is focused on time comparison of mining different algorithms on synthetic datasets. The following algorithms are compared in experiments GSP, PrefixSpan, hGSP and MLSP. The advantage of synthetic datasets is the possibility to define specific probabilistic properties. The second section is focused on mining in real-world data. Mining in the real world dataset is an important evaluation because it shows if the algorithms can be used and if revealed results are useful. Commonly used real world testing dataset AdventureWorks [13] by Microsoft is absolutely inappropriate because it does not contain a long-period order history. Therefore, the five year order history of on-line e-shop VOPI [23] is used for the real world evaluation.

5.1 Evaluation on Synthetic Datasets

The synthetic dataset allows changing only a specific property of the dataset without changing others if necessary. The complete set of parameters of sequence databases with defined taxonomies are shown in Table 5.1. The general methodology of experiments are following. All parameters of generated datasets are fixed except one. Then, the effect of the changes of such dataset or algorithm parameter is evaluated. There was no generator for multi-level sequential patterns. This section describes a generator of multi-level or level-crossing sequence datasets developed by the author of this thesis published in [27]. .

Experiments were performed on a PC with CPU i5 3.3GHz, 8GB RAM, OS MS Windows 10. Because there is no algorithm for mining multi-level sequential patterns, results of our algorithms are compared with GSP and PrefixSpan. Authors of the GSP recommended using GSP over an extended database for mining sequential patterns with taxonomies. Algorithms GSP and PrefixSpan use post-processing to get

Table 5.1: Parameters of our hierarchical sequence generator.

Parameter	**ML-Seq.Patt.**
Dataset Size	$\|\mathcal{D}\|$
Avg. number of elements of sequences	$\|C\|$
Avg. size of elements	$\|T\|$
Avg. size of frequent elements	$\|I\|$
Number of items	N
Avg. length of (frequent) sequential patterns	$\|S\|$
Number of sequential patterns	N_S
Avg. support of sequential patterns	$Supp_S$
Number of taxonomies (roots)	$\|R\|$
Avg. taxonomy height	R_h
Probability of children (general items)	P^I_{ch}
Probability of children (items of freq. sequences)	P^S_{ch}

complete set of hierarchically-closed multi-level sequential patterns. All algorithms were implemented in C# on .NET platform using the MS SQL Server database.

5.1.1 Experiment 1: Dataset Size – Scalability

The first experiment is focused on scalability of the algorithms. The methodology of the experiment is to measure the dependency of execution time on a dataset size. For example, the values of fixed parameters denoted as C4T1.2S3I1.2N15%|Đ|R1k are explained in Table 5.2. The suffix 'k' of a number means that the value is $\times 1000$.

Table 5.2: Dataset Parameters for Experiment 1

Dataset	$\|C\|$	$\|T\|$	$\|S\|$	$\|I\|$	$\|N\|$	$\|R\|$	P^I_{ch}, P^S_{ch}	$Supp_S$
C4T1.2S3I1.2N150kR1k	4	1.2	3	1.2	15% $\|\mathcal{D}\|$	$1k$	0.9	0.045

The variable is the dataset size. The dataset size is set to different number of sequences $|\mathcal{D}| \in \{100\,000, 250\,000, 500\,000, 750\,000, 1\,000\,000\}$ where each sequence is of the average length 4 – it results in about from 400 000 up to 4 000 000 items in the synthetic datasets. The number of items $|N|$ cannot be set statically but it must be related to $|\mathcal{D}|$ because the small number of items increases their support in the dataset if the dataset naturally grows. Number of frequent sequences $N_S = 5$ with average support is $Supp_S = 0.045$ (4.5 %).

The execution time in seconds was measured for the evaluation. Lower execution time represents better scalability. Results are shown on Figure 5.1. The slowest is basic GSP algorithm. Moreover, for the $|D| = 1\,000\,000$ the run does not finish. Results of GSP can be better using the hash optimization for fast is-generalized-subsequence pre-check. Similarly the time complexity of our algorithm hGSP is computationally hard and is comparable to optimized GSP using the hash is-subsequence check.

Algorithms PrefixSpan and MLSP have better results. PrefixSpan is approximately 7× faster than MLSP without optimizations. However, the MLSP algorithm can be improved using the optimized *Is-generalized-subsequence Check*. The optimized MLSP (with hash is-generalized-subsequence check, denoted by "hash" suffix) is the fastest of all the algorithms. It is in average 4× faster than the second PrefixSpan.

5.1.2 Experiment 2: Changes of Minimal Support Threshold

The Experiment 1 showed that the *Is-generalized-subsequence check* brings important speed-up of the algorithms GSP and MLSP. Therefore, next Experiments uses only optimized variants for both GSP and MLSP. All further experiments were limited by maximal execution time up to 3 600 seconds (1 hour) which should be sufficient according to average execution times of Experiment 1.

In practice, the optimal minimal support threshold is not known on the beginning of the analysis. The optimal minimal support is usually determined experimentally when the data mining starts with the high minimal support threshold value and it is gradually decreased until sequential patterns are found. The decreasing of the minimal support increases the number of generated candidate sequences and sequential patterns while the dataset remains the same.

The setup of this experiment is following. The dataset parameters are C4T1.2S3I1.2N15kR1k, $|\mathcal{D}| = 100\,000$. All algorithms were run with several values of minimal support threshold $min\_supp \in \{0.025, 0.035, 0.045, 0.055, 0.065\}$. Results of this experiment are similar to Experiment 1.

Figure 5.2 shows that the execution times of PrefixSpan and MLSP algorithms are similar for high values of the minimal support. While the MLSP keeps the stable execution times, the performance of the PrefixSpan get worse with the decreasing value of the minimal support value parameter. The GSP and hGSP algorithms are much slower in all cases.

5.1.3 Experiment 3: Length of Sequential Patterns

Next parameter which can affect the performance of the algorithm is the average length of sequential patterns because the length of sequential patterns leads to a higher number of frequent subsequences and candidate sequences during the mining process. The Experiment 3 is focused on the gradually increasing length of the sequential patterns from $|S| \in \{3, 5, 7, 9\}$. Number of elements of sequences in the database $\mathcal{D}$ was determined to value $|C| = 10$. Note that the longer sequences result into higher number of items in the database. Therefore the Experiment 3 is divided into two parts.

First part analyses the dependence of the execution time on the length of sequences in $\mathcal{D}$ because the average sequence length must be at least the average length of sequential patterns. The experiment uses average number of elements of sequences $|C| \in \{4, 6, 8, 10\}$. The fixed dataset parameters are T1.2I1.2N15kR1k, $|\mathcal{D}| = 100\,000$ and $|N_S| = 3$. The results are shown in Table 5.3. The fastest algorithm on such

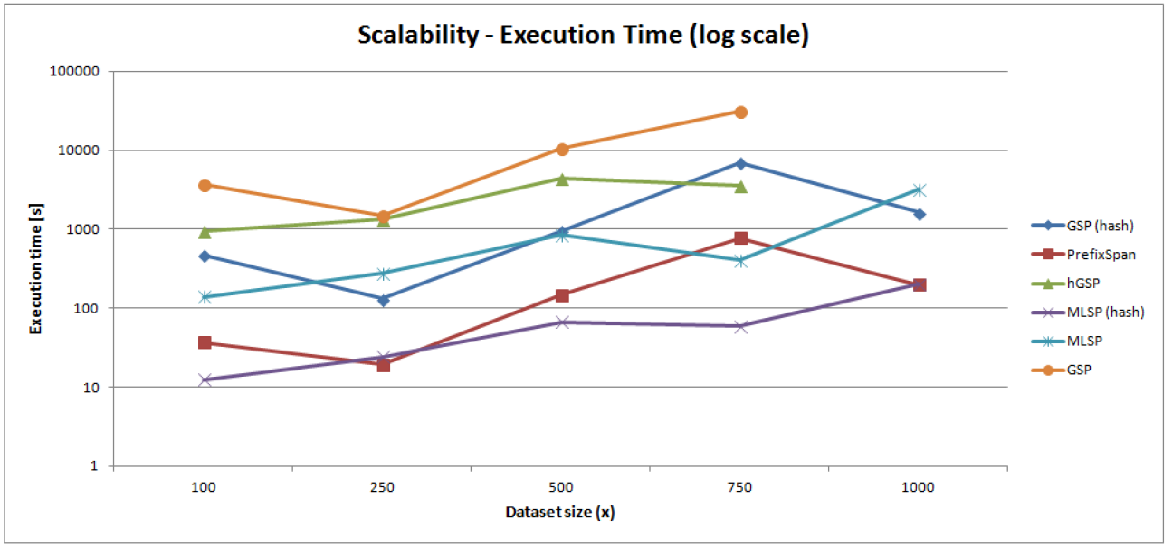


Figure 5.1: Comparison of execution time w.r.t. dataset size *1000 (in logarithmic scale).

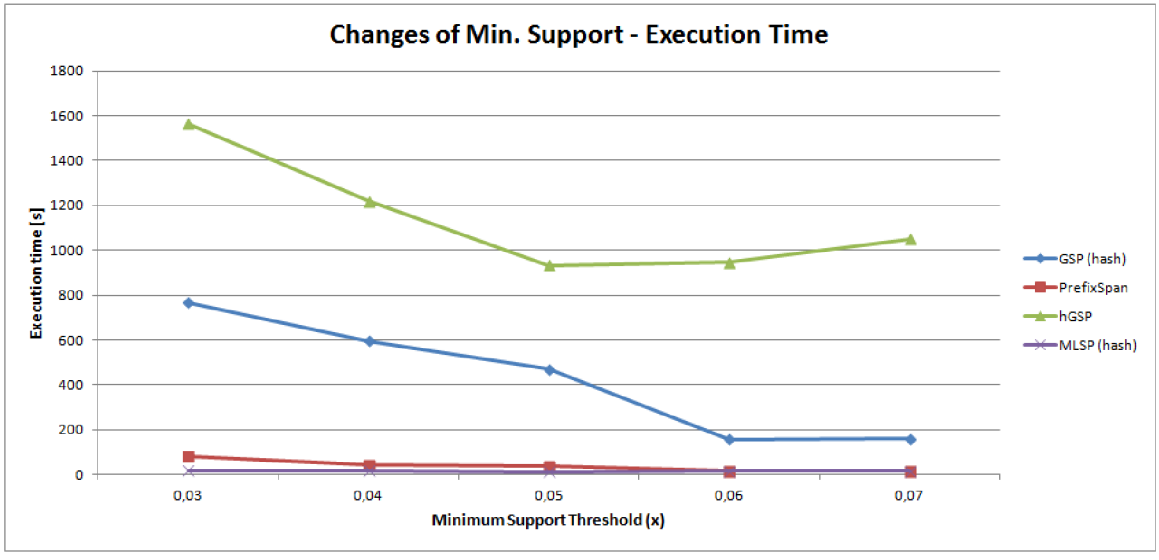


Figure 5.2: Comparison of execution time w.r.t. the minimal support threshold value..

Table 5.3: The dependency of the execution time on the average length $|C|$ of sequences in the database.

$\|C\|$	**GSP (hash)**	**Prefix Span**	**hGSP**	**MLSP (hash)**
4	229	11	137	**5**
6	633	79	2660	**52**
8	3253	**136**	N/A	333
10	967	**34**	N/A	367

Table 5.4: The dependency of the execution time on the average length of sequential patterns $|S|$.

$\|S\|$	**GSP (hash)**	**Prefix Span**	**hGSP**	**MLSP (hash)**
3	967	**34**	N/A	367
5	N/A	1229	N/A	**301**
7	N/A	N/A	N/A	**487**
9	N/A	N/A	N/A	**254**

databases is the PrefixSpan. The MLSP is slower mainly for longer sequences. The reason is that the combinations of candidate sequences which grows massively (there are 116 candidate sequences for $|C| = 4$ and 17250 candidate sequences for $|C| = 10$). Nevertheless the MLSP is up to 10× faster than the GSP algorithm. The hGSP does not finish for longer sequences because of the large search space.

The second part analyses the dependence of execution time on the length of sequential patterns. Experiment results are shown in Table 5.4. The experiment shows the strongest point of the algorithm MLSP. The longer sequential patterns lead to large number of candidate sequences and projected databases of the algorithms based on extended databases. In contrast, the performance of the MLSP is not affected by the length of sequential patterns but only by the number of final sequential patterns. Therefore, the MLSP is the only algorithm which is able to finish on all test cases. The other algorithms PrefixSpan and GSP do not finish for the cases $|S| \in \{7, 9\}$ the time limit of one hour.

5.1.4 Experiment 4: Number of Sequential Patterns

This experiment analyses the dependence of the execution time on the number of sequential patterns. The experiment setup is following: fixed dataset parameters are $C4$T1.2S3I1.2N15kR1k, $|\mathcal{D}| = 100\,000$, $|N_S| = 3$ and variable is $|N_S| \in \{3, 5, 7, 9, 30\}$. The results of experiment are shown in Table 5.5. The best results of the experiment were achieved by the PrefixSpan algorithm. The MLSP algorithm gives also satisfactory results. GSP and hGSP algorithms achieved by the order of magnitude worse results.

Finally, the experiment also tests the behavior of the algorithms when number

Table 5.5: The dependency of the execution time on the total count of sequential patterns $|N_S|$.

$\|N_S\|$	**GSP (hash)**	**Prefix Span**	**hGSP**	**MLSP (hash)**
3	201	9	94	**5**
5	8	**1**	82	3
7	17	**1**	219	5
9	217	**11**	893	16
30	57	**1**	889	16

Table 5.6: The dependency of the execution time on the number of taxonomy levels $|R_h|$

$\|R_h\|$	**GSP (hash)**	**Prefix Span**	**hGSP**	**MLSP (hash)**
2	4.1	**0.5**	35.6	2.7
6	4.7	**0.7**	34.4	2.2

of sequential patterns is higher $|N_S| = 30$. In that case the results were similar to previous cases. Therefore, we can say, that the number of sequential patterns does not negatively affect the execution time of the algorithms, especially examined MLSP and hGSP algorithms.

5.1.5 Experiment 5: Taxonomy Height

The last experiment on the synthetic dataset analyses the dependency of the execution time on the average taxonomy height (the total number of levels of all taxonomies). The fixed parameters of the experiment are $C4$T1.2S3I1.2N15kR1k, $|\mathcal{D}| = 100\,000$, $|N_S| = 3$. The variable parameter is $|R_h| \in \{2, 6\}$. The first case of the average height 2 shows algorithms behavior on low item categorizations. On the other hand, the second case deals with the taxonomies of average height 6. The results are shown in the Table 5.6. It is shown that the height of taxonomies do not affect the execution time and the complexity of the run of the algorithms. The results are the same for the both test cases.

5.1.6 Experiments Summary

Experiments on synthetic datasets compared algorithms GSP, PrefixSpan, hGSP and MLSP. Best results are given by the algorithms MLSP and PrefixSpan. The other algorithms are over a magnitude worse. The performance of the PrefixSpan is significantly slower for mining long sequential patterns. Only the algorithm MLSP finishes all the experiments and it proved very good results for mining hierarchically-closed multi-level sequential patterns.

.

5.2 Evaluation on Real-World Datasets

The previous experiments verify the behavior of algorithm MLSP however it does not deal with usability on a real world dataset. The real world data are much more suitable to test the usability. The dataset of orders history of the e-shop VOPI is used for the experiment. First, the dataset is described from general and statistical points of view. Second, the sequential patterns obtained by the MLSP algorithm are discussed. Note, that some kind of data were anonymized or marked as N/P (not presentable).

It was shown that the MLSP produces new sequential patterns on the real-world datasets. The execution time of the MLSP on the real world dataset was in minutes depending on parameters settings. Therefore ,we can say that the algorithm is fully usable for real world data mining problems.

Chapter 6

Conclusions

Mining sequential patterns, especially mining multi-level sequential patterns, is a challenging task. The main goal of the thesis was to confirm the hypothesis that taxonomies lead to find out new patterns and a new method to mine them effectively can be formulated.

In my research, I focused on two main approaches of dealing with items in taxonomies. The first approach is to find out patterns called level-crossing sequential patterns. A new algorithm called hGSP was proposed but the level-crossing approach came out as extremely time-consuming. The second approach adds some special constraints which simplify the task while keeping important patterns in the result. It leads to the definition of a new type of data mining task called mining hierarchically-closed multi-level sequential patterns. Mining hierarchically-closed multi-level sequential patterns produces results without redundant patterns useful for the analyst. These research results confirm the first part of the thesis hypothesis.

The thesis introduces a new algorithm called MLSP designed for mining hierarchically-closed multi-level sequential patterns. Both the hGSP and MLSP algorithms prefer the generalization of a sequence to dropping it. The performance of the algorithms was evaluated in several experiments. The experiments were focused on comparison of the performance in dependence on the dataset size (scalability), sequential patterns length and size and the taxonomies sizes. The best results are provided by MLSP and PrefixSpan algorithms. The other algorithms were more than over a magnitude slower. It was shown that the average length of sequential patterns has a significant effect on the execution time. The PrefixSpan did not finish for sequences containing 7 and more elements. Only the MLSP algorithm finished in all runs. The usability of the MLSP algorithm was shown on real dataset where the algorithm has found some new useful knowledge. This confirms the second part of the hypothesis related to a new data mining method.

As a result, both parts of the hypothesis were confirmed. Therefore, the thesis hypothesis was completely confirmed and the goal of the thesis was fulfilled.

It was shown that mining hierarchically-closed multi-level sequential patterns is suitable for tasks of the analysis of customer behavior. But there are some other domains where this type of mining task can be useful, for example security analysis of Domain Name System because domains are also organized in taxonomies.

The future work can be focused on the research of other constrains while mining level-crossing or multi-level sequential patterns. In the field of mining methods, research may continue exploring other optimizations.

Bibliography

[1] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*, Mar 1995, pp. 3–14.

[2] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *SIGMOD Rec.*, vol. 22, no. 2, pp. 207–216, Jun. 1993.

[3] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.

[4] A. V. Aho, J. E. Hopcroft, and J. Ullman, *Data Structures and Algorithms*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1983, 427 p.

[5] T. Cormen, *Introduction to Algorithms*. MIT Press, 2009, 1312 p.

[6] M. E. M. Di Beneditto and L. N. de Barros, "Using concept hierarchies in knowledge discovery," in *Advances in Artificial Intelligence–SBIA 2004*. Springer, 2004, pp. 255–265.

[7] K. Gouda and M. Zaki, "Efficiently mining maximal frequent itemsets," in *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, 2001, pp. 163–170.

[8] J.-W. Han, J. Pei, and X.-F. Yan, "From sequential pattern mining to structured pattern mining: A pattern-growth approach," *Journal of Computer Science and Technology*, vol. 19, no. 3, pp. 257–279, 2004.

[9] J. Han and A. Fu, "Mining multiple-level association rules in large databases," *IEEE Trans. on Knowledge and Data Engineering*, vol. 11, no. 5, pp. 798–805, 1999.

[10] J. Han and M. Kamber, *Data mining: concepts and techniques*, ser. The Morgan Kaufmann series in data management systems. Elsevier, 2006, 800 p.

[11] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu, "Freespan: Frequent pattern-projected sequential pattern mining," in *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '00. New York, NY, USA: ACM, 2000, pp. 355–359.

[12] D. MacKay, *Information Theory, Inference, and Learning Algorithms.* Cambrifge University Press, 2003, 640 p.

[13] Microsoft, "Microsoft sql server product samples database," [cit. 2016-01-15, on-line], 2016, published on: http://msftdbprodsamples.codeplex.com/.

[14] S.-I. Nakano, "Efficient generation of plane trees," *Inf. Process. Lett.*, vol. 84, no. 3, pp. 167–172, nov. 2002.

[15] J. Pei, J. Han, R. Mao *et al.*, "Closet: An efficient algorithm for mining frequent closed itemsets." in *ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, vol. 4, no. 2, 2000, pp. 21–30.

[16] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Mining sequential patterns by pattern-growth: the prefixspan approach," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 16, no. 11, pp. 1424–1440, Nov 2004.

[17] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth," in *Proc. of the 17th International Conference on Data Engineering.* IEEE Computer Society, 2001, pp. 0215–0215.

[18] H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen, and U. Dayal, "Multi-dimensional sequential pattern mining," in *Proceedings of the Tenth International Conference on Information and Knowledge Management*, ser. CIKM '01. New York, NY, USA: ACM, 2001, pp. 81–88.

[19] M. Plantevit, A. Laurent, D. Laurent, M. Teisseire, and Y. W. Choong, "Mining multidimensional and multilevel sequential patterns," *ACM Trans. Knowl. Discov. Data*, vol. 4, no. 1, pp. 4:1–4:37, Jan. 2010.

[20] M. Plantevit, A. Laurent, and M. Teisseire, "Hype: Mining hierarchical sequential patterns," in *Proceedings of the 9th ACM International Workshop on Data Warehousing and OLAP*, ser. DOLAP '06. New York, NY, USA: ACM, 2006, pp. 19–26.

[21] R. Srikant and R. Agrawal, *Mining sequential patterns: Generalizations and performance improvements.* IBM Research Division, 1996, 29 p., research Report.

[22] R. Srikant and R. Agrawal, "Mining generalized association rules," *Future Generation Computer Systems*, vol. 13, no. 2, pp. 161–180, 1997, data Mining.

[23] VOPI, "Vopi.cz," [cit. 2016-01-15, on-line], 2016, published on: http://www.vopi.cz/.

[24] M. Šebek, M. Hlosta, J. Kupčík, J. Zendulka, and T. Hruška, "Multi-level sequence mining based on gsp," in *Proceedings of the Eleventh International Conference on Informatics INFORMATICS'2011*, ser. 1. Faculty of Electrical Engineering and Informatics, University of Technology Košice, 2011, pp. 185–190.

[25] M. Šebek, M. Hlosta, J. Kupčík, J. Zendulka, and T. Hruška, "Multi-level sequence mining based on gsp," *Acta Electrotechnica et Informatica*, no. 2, pp. 31–38, 2012.

[26] M. Šebek, M. Hlosta, J. Zendulka, and T. Hruška, "Mlsp: Mining hierarchically-closed multi-level sequential patterns," in *Advanced Data Mining and Applications*, ser. Lecture Notes in Computer Science, H. Motoda, Z. Wu, L. Cao, O. Zaiane, M. Yao, and W. Wang, Eds. Springer Berlin Heidelberg, 2013, vol. 8346, pp. 157–168.

[27] M. Šebek and J. Zendulka, "Generator of synthetic datasets for hierarchical sequential pattern mining evaluation," in *Proceedings of the Twelfth International Conference on Informatics 2013*. The University of Technology Košice, 2013, pp. 289–292.

[28] J. Wang and J. Han, "Bide: efficient mining of frequent closed sequences," in *Data Engineering, 2004. Proceedings. 20th International Conference on*, March 2004, pp. 79–90.

Curriculum Vitae

Personal information

Name	Ing. Michal Šebek
Date and place of birth	16th July 1985, Nové Město na Moravě
Email	sebek.michal@centrum.cz
Nationality	Czech Republic

Education

2009 - present	Ph.D. Student *Brno University of Technology* *Faculty of Information Technology* *Programme: Computer Science and Engineering*
2007 - 2009	Master Study Programme (Ing.) *Brno University of Technology* *Faculty of Information Technology* *Programme: Information Systems*
2004 - 2007	Bachelor Study Programme (Bc.) *Brno University of Technology* *Faculty of Information Technology* *Programme: Information Technology*

Professional experience

2010 - present	PHP Programmer *Internet Stream s.r.o.* *Nové Město na Moravě, CZ*
2011-2013	Software Developer *AVG Technologies, s.r.o.* *Brno, CZ*

Research interest

Data Mining, Sequential Patterns

Languages

Czech	Native
English	Full Professional Proficiency
Russian	Basics