MLSP: Mining Hierarchically-Closed Multi-Level Sequential Patterns

Michal Šebek, Martin Hlosta, Jaroslav Zendulka, and Tomáš Hruška

Faculty of Information Technology, IT4Innovations Centre of Excellence, Brno University of Technology, Božetěchova 2, Brno, Czech Republic,
`{isebek, ihlosta, zendulka, hruska}@fit.vutbr.cz`

Abstract The problem of mining sequential patterns has been widely studied and many efficient algorithms to solve this problem were published. In some cases, there can be implicitly or explicitely defined taxonomies (hierarchies) over input items (e.g. product categories in a e-shop or sub-domains in the DNS system). However, how to deal with taxonomies in sequential pattern mining was discussed marginally. In this paper, we formulate the problem of mining hierarchically-closed multi-level sequential patterns and demonstrate its usefulness. We present the MLSP algorithm based on the on-demand generalization that outperforms other similar algorithms for mining multi-level sequential patterns.

Keywords: closed sequential pattern mining, taxonomy, generalization, GSP, MLSP

1 Introduction

Mining sequential patterns is interesting and long studied research problem in a data mining community and it is used for many applications such as analysis of customer purchase patterns, analysis of web log data or biology sequences. Given a sequence database, the goal is to find sequential patterns that occur frequently in this database, i.e. more often than a given user defined threshold. Taking market basket as an application example, the sequential pattern $\langle PC\_midtower\ ink\_printer \rangle$ can be discovered. This means that many people buy midtower PC and later, they also buy an ink printer. The problem was firstly presented by Agrawal and Srikant in [1] and since then, many of algorithms has been published trying to solve this problem.

Often, one or more taxonomies of items can be stored in the database. These taxonomies can be utilized to find patterns which items of sequences can be on different levels of hierarchy. It allows to find patterns, which wouldn't be retrieved without defined taxonomies. Our motivation to deal with the taxonomies is primary to analyse some important facts over internet domains taxonomy (top-level domain, second-level domain, etc.), but, due to privacy issues, we demonstrate our examples on customer purchase analysis. Following the previous sequence pattern example, in addition to $\langle PC\_midtower\ ink\_printer \rangle$ the pattern $\langle PC\ printer \rangle$ can be found by replacing the items by items on higher level of hierarchy. Unfortunately, the amount of such patterns can grow enormously, but many of the resulting patterns can be considered as unuseful. For example the pattern $\langle PC\ printer \rangle$ doesn't bring any new information if the number of its occurrence in the database is the same as for $\langle PC\_midtower\ ink\_printer \rangle$.We define hierarchically-closed sequential patterns which are the most useful for the analyst. In addition, the patterns can be divided to *multi-level* and *level-crossing* [2]. In *multi-level* (known also as *intra-level*) patterns, all items in one pattern are one the same level of hierarchy whereas in *level-crossing* (known also as *inter-level*), the level can be different. The amount of retrieved level-crossing patterns is often greater than the number of multi-level patterns, and so is the search space. This leads to longer execution time of algorithms that mine level-crossing patterns. In contrast to the amount of algorithms for mining sequential patterns, very few algorithms tackled this problem with taxonomies.

Our contribution in the presented paper can be summarized as: formal definition of the mining hierarchically-closed multi-level sequential patterns problem, the MLSP algorithm for solving the problem and the experimental comparison of the algorithm with existing approaches.

The paper is organized as follows. In the next section, the mathematical basics for mining sequential patterns are described. In Section 3, the related work in mining sequential patterns is discussed. The section is finalized by analysis of using taxonomies in this research area. In Section 4, terminology for multi-level sequential pattern mining is introduced. Our algorithm MLSP for mining sequential patterns is presented in Section 5. In Section 6, the algorithm is compared with techniques presented in other papers. Conclusions and future work are presented in Section 7.

2 Preliminaries

In this section the problem of mining sequential patterns is formalized.

Definition 1. (Itemset) Let $I = \{i_1, i_2, i_3, \ldots, i_k\}$ be a nonempty finite set of items. Then an *itemset* is a nonempty subset of I.

Definition 2. (Sequence) A *sequence* is an ordered list of itemsets. A sequence s is denoted by $\langle s_1 s_2 s_3 \ldots s_n \rangle$, where s_j for $1 \leq j \leq n$ is an 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 $\alpha = \langle a_1 a_2 \ldots a_n \rangle$ is a *subsequence* of the sequence $\beta = \langle b_1 b_2 \ldots b_m \rangle$ where $n \leq m$ if there exist integers $1 \leq j_1 < j_2 < \cdots < j_n \leq m$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \ldots, a_n \subseteq b_{j_n}$. We denote it $\alpha \sqsubseteq \beta$ and β is a *supersequence* of α .

Example 1. In the following examples, we use well established convention from sequence pattern mining for denoting elements of sequences [3]. An element that has m items is denoted as $(i_1 i_2 \ldots i_m)$ where $m \geq 1$. If an element contains only one item, the braces are omitted for brevity.

Given $I = \{a, b, c, d, e\}$, an example of an itemset is $\{a, b, d\}$, example of a sequence is $\langle a(ab)(ce)d(cde)\rangle$. This sequence has 5 elements and 9 items. Its length is 9 and it is called a *9-sequence*.

Definition 3. (Sequence database) A sequence database D is a set of tuples $\langle SID, s\rangle$, where SID is a sequence identifier and s is a sequence. The support of a sequence s_1 is defined as the number of sequences in D containing a subsequence s_1. Formally, the support of a sequence s_1 is $support(s_1) = |\{\langle SID, s\rangle|(\langle SID, s\rangle \in D) \wedge (s_1 \sqsubseteq s)\}|$.

Definition 4. (Sequence Pattern, Mining Sequential Patterns) Given sequence database $\mathcal{D}$ and *minimum support threshold* min_supp, a *frequent sequence* is such a sequence s whose $support(s) \geq min\_sup$. A frequent sequence is called *sequence 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 $\mathcal{D}$.

In the problem of mining multi-level sequential patterns, taxonomies exist over items in D and they are defined as follows.

Definition 5. (Taxonomy of Items) Taxonomy structure of an item set V is a rooted tree $T = (V, E)$ with 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 a tree, let $UP(v)$ be the 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 neighbour of v on $UP(v)$, and for each node $v \in V, v \neq r$ there exists a set of its ancestors defined as:

$$ancestors(v) = \{x | x \in UP(v), x \neq v\}. \quad (1)$$

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 [4].

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

$$is - a : V \times V :\equiv \{(a, b) | b \in ancestors(a)\}. \quad (2)$$

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 $\tau = \{T_1, \ldots, T_m\}$ corresponding to ι such that $T_i = (V_i, E_i)$ 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 $T_i \in \tau$. Notice that we do not require that items need to be only leaf nodes. In addition we refer to any ancestor of a node representing a item x a *generalized item* of the item x.

Example 2. We use s running example based on taxonomies on Figure 1. In terms defined above, the parent of *black* is *Ink printer*; ancestors is the set *{Ink printer, printer}* and the level of *black* is *2. In next examples, some node names will be shorted (e.g.* LCD monitor *to* LCD*).*

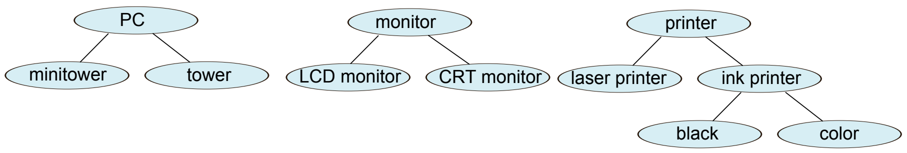


Figure 1: Three taxonomies for running example.

3 Related Work

Sequential pattern mining is a temporal extension of frequent pattern mining [5], [6]. Basic algorithms for mining sequential patterns are AprioriAll [1] and GSP [7] which are based on the generating and pruning candidates sequences. Other representatives of this category are SPADE [8] algorithm using a vertical format of database and SPAM [9] using a bitmap structure as an optimization. The second family of algorithms is based on the pattern growth. It is represented by the well-known fast algorithm PrefixSpan [10]. For reduction of search space, the closed patterns mining was defined by CloSpan [11].

The GSP algorithm is important for this work therefore it will be described in more detail. The algorithm works iteratively. In each iteration it makes a pass over the sequence database:

1. Initially, the support of items is counted in first pass over the sequence database and those having it higher than a minimal support min_sup are inserted in the resulting set L_1 containing *frequent 1-sequences*.
2. Then the following steps are executed iteratively in $1 < k \leq n$ iterations until no k-sequence is generated:
   (a) The *Join step*, a *candidate set* 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 sub-sequences generated from s_1 and s_2 such that if the first item of s_1 and the last item x of s_2 are omitted, are the same. Then the candidate k-sequence is formed by adding the last item of the s_2 at the and of the sequence s_1 as:
      i. the last new element containing one item x if x was in a separate element in s_2;
      ii. as a next item of the last element in s_1 otherwise.
      iii. When joining $x \in L_1$with $y \in L_1$ both sequences $< (y)(x) >$ and $< (yx) >$ are generated as candidate sequences.
   (b) In the *Prune step*, the database is passed and the support of each candidate sequence is counted. Candidates with support greater than min_supp are added into the set L_k of sequential patterns.
3. The result sequential patterns set is $\bigcup_{k=1}^{n} L_k$.

Multi-level frequent patterns (itemsets) and association rules was described by Han and Fu in [2]. They proposed a family of algorithms for mining multi-level association rules based on Apriori which process each level separately. It allows to use a different minimum support threshold for different levels of taxonomies. However, mining multi-level sequential patterns has not been deeply studied. The straightforward idea how to mine level-crossing sequential patterns using GSP was presented in [7]. Authors proposed to use an extended-sequence database where each item of a sequence of origin database $\mathcal{D}$ is extended by all ancestors within its element. The disadvantage is that many redundant sequences are generated by this method.

Plantevit et al. in [12] firstly describe the idea of mining the most specific sequences to avoid redundancy. The presented algorithm performs a generalization only in the first step over single items to create maf-subsequences (maximally atomic frequent 1-sequences). However, the algorithm does not perform the generalization during generation of their supersequences.

The information theory concept was used for solving the problem of mining level-crossing sequential patterns in [13]. Proposed hGSP algorithm does not reveal a complete set of level-crossing sequential patterns but only the most specific sequential patterns. The algorithm works on the bottom-up principle and performs generalization only if the candidate sequence should be pruned . It was demonstrated that mining level-crossing sequential patterns has extremely large search space.

In this paper, we focus on benefits of multi-level mining against to level-crossing sequential pattern mining from an algorithm's performance point of view.

4 Mining multi-level sequential patterns

The search space of mining level-crossing sequential patterns problem (studied in [13]) can be reduced by the *constraint* where levels of all items of frequent sequence are equal to a constant l_s (firstly used in [2]). The constraint could lead to substantial simplification of the mining process. The constraint brings a compromise between an execution time of the algorithm and specificity of sequential patterns for a user. The experimental verification is discussed in Section 6.

Example 3. The difference between the complexity of level-crossing and multi-level sequential patterns mining is shown on Figure 2. In level-crossing way, the sequence *<tower (LCD, laser)>* has 6 ancestors whereas in multi-level way it has only one ancestor sequence.

A constrained element (ML-element) and a constrained sequence (ML-sequence) are derived from definitions of element and sequence as follows.

Definition 6. (ML-element, ML-sequence) Let $l \in \boldsymbol{N}$ be a level of items in a taxonomy $T \in \tau$. Then a *ML-sequence* is an ordered list of itemsets $s_{ML} = \langle s_1 s_2 s_3 \dots s_n \rangle$, such that the level of all items of the itemsets is equal to l. The itemset of the ML-sequence is called an *ML-element*. The *length, subsequence and supersequence* of ML-sequence is defined analogously to ones in Definition 2.

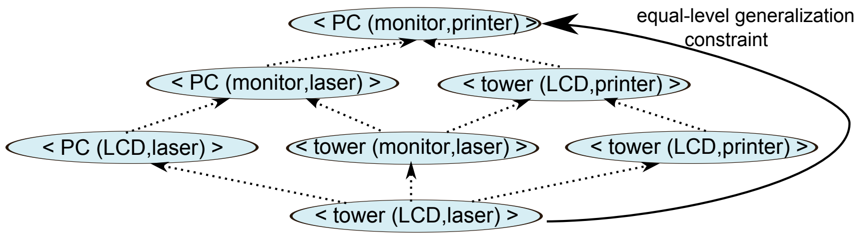


Figure 2: Level-crossing and constrained multi-level generalization combinations.

Example 4. For better understanding of defined notions, we return to our running example and the Figure 2 using taxonomies from the Figure 1. From all the sequences here, only the most specific $\langle tower(LCD, laser)\rangle$ and the most generic $\langle PC(monitor, printer)\rangle$ are ML-sequences. In contrast, $\langle PC(LCD, laser)\rangle$ is not a ML-sequence because the level of item PC is 0 and the level of the items in the element $(LCD, laser)$ is 1. The levels of items in the other sequences are different.

The existence of the taxonomy over items allows is to introduce ML variants of a parent and of ancestors.

Definition 7. (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 all items are replaced by their parents. This is defined as

$$parent_{el}(e) = \{parent(i_k)|1 \leq k \leq n \wedge i_k \in e\}. \tag{3}$$

Definition 8. (ML-sequence parent, ML-sequence ancestors) Given a ML-sequence $s = \langle e_1 e_2 \ldots e_n\rangle$, where e_k is ML-element on position k, the *ML-sequence parent* of s is a ML-sequence such that all ML-elements of s are replaced by their ML-element parents. Formally,

$$parent_{seq}(s) = \langle f_1 f_2 \ldots f_n\rangle,\ f_k \in parent_{el}(e_k), 1 \leq k \leq n. \tag{4}$$

For a given set of taxonomies τ, a *root ML-sequence* is a 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 a ML-sequence s, $ancestors_{seq}(s)$ is defined recursively as follows:

$$\begin{aligned} ancestors_{seq}(s) &= M_i, \text{ if } M_{i+1} = M_i, where \\ M_0 &= \{parent_{seq}(s)\} \\ M_{i+1} &= M_i \cup \{parent_{seq}(x) \mid x \in M_i\} \text{ for } i \geq 0. \end{aligned} \tag{5}$$

Example 5. The element$(monitor, printer)$ is ML-element parent of the *(LCD, laser)* element because all of its items are replaced by their parents. The ML-sequence $\langle PC(monitor, printer)\rangle$ is the ML-sequence parent of the sequence

$\langle tower(LCD, laser)\rangle$ and it is also the only member of its ML-ancestors set because $\langle tower(LCD, laser)\rangle$ does not have any ML-sequence parent.

The *generalized support gen_supp* is based on the definition of support in Definition 2. It's only necessary to redefine the subset relation for two elements to deal with taxonomies according to Def. 9.

Definition 9. (The generalized support) A *generalized subset relation* $\subseteq_g$ is defined as

$$e_1 \subseteq_g e_2 \Leftrightarrow \forall i \in e_1 : i \in e_2 \vee \\ \exists j \in e_2 : i \in ancestors(j). \quad (6)$$

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

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

Recall the term *closed* in *closed sequential pattern mining*. The *closed* means that if a sequence s and a supersequence of s has 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 contrast, the problem disscussed in the paper allows to modify sequence in two dimensions – a sequence length (same with *closed sequential pattern mining*) and a level of sequence (new for ML-sequences). We will discuss closed only in the taxonomy dimension.

5 The MLSP algorithm

In this section, the problem of mining hierarchically-closed multi-level sequential patterns is specified and the effective **MLSP** *(Multi-Level Sequential Patterns)* algorithm for mining these patterns is presented.

Definition 10. (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 ML-sequence ancestor of s with *equal* generalized support. Then, the problem of **mining hierarchically-closed multi-level sequential patterns** (hereinafter *ML-sequential patterns*) for given input sequence database D and minimal generalized support threshold *min_supp* is to find a set L_{ML} of all ML-sequences over $\mathcal{D}$ such that

$$\begin{aligned} L_{ML} = \{s_{ML} \sqsubseteq s | \langle SID, s\rangle \in \mathcal{D} \wedge gen\_supp(s_{ML}) \geq min\_supp \quad (7) \\ \wedge \nexists s_x[s_x \sqsubseteq_g s \wedge gen\_supp(s_x) \geq min\_supp \\ \wedge gen\_supp(s_x) = gen\_supp(s_{ML}) \\ \wedge s_{ML} \in ancestor_{seq}(s_x)]\}. \end{aligned}$$

The proposed algorithm MLSP is based on the candidate generation principle (adapted from the GSP, see Section 3) combined with the on-demand generalization. The algorithm runs in two steps (*join step* and *prune step*).

Prune step modification. The idea is that if the candidate ML-sequence should be pruned, the generalization of the ML-sequence is performed. The generalization of the ML-sequence means to find a ancestor with the greatest level of the sequence which satisfies minimal support threshold. The on-demand *bottom-up* generalization procedure GetFirstFrequentAncestor() is shown in Algorithm 1.

Algorithm 1 Method GetFirstFrequentAncestor()

```
1: procedure GetFirstFrequentAncestor($s, min\_supp$)
2:     repeat
3:         if $gen\_supp(s) \geq min\_supp$ then
4:             return $s$
5:         end if
6:         $s \leftarrow parent_{seq}(s)$
7:     until $s$ is root sequence
8:     return null
9: end procedure
```

Example 6. *Prune step generalization.* If none of the candidate ML-sequences ⟨*minitower CRT*⟩ and ⟨*minitower LCD*⟩ is frequent, there can exist the ancestor of both ML-sequences ⟨*PC monitor*⟩ which could be frequent and be important.

Nevertheless, this modification of prune step does not guarantee the algorithm completeness. Also, the generalization has to be performed for *join step*. Recall that the GSP allows to join a pair of sequences if the common subsequence is the same (see Section 3 for details). In difference, the MLSP algorithm firstly tries to find the common ancestor of the candidate ML-subsequences in *bottom-up way*. If the common ML-subsequence exists, then the generalized ML-sequences are joined to the new candidate ML-sequence, otherwise, no candidate is generated. Notice that the levels of input ML-sequences can be different but the levels of items of a generated ML-sequence are the same. The modified procedure for candidate generation is shown in Algorithm 2.

Example 7. *Join step generalization.* The pair of 2-ML-sequences ⟨*minitower CRT*⟩ and ⟨*LCD laser_printer*⟩ cannot be joined in GSP, because items *CRT* and *LCD*. However, they have a common parent *monitor*. Therefore, the items should be generalized to the common subsequence <*monitor*> and the sequences are joined to the 3-ML-sequence <*PC, monitor, printer*> by MLSP.

Finally, we summarize the complete MLSP algorithm in Algorithm 3. In the first phase, the algorithm count the generalized support for all items of the database D and their parents and 1-sequences are formed as candidate ML-sequences

Algorithm 2 Method GenerateCandidateMLSequences()

```
1: procedure GENERATECANDIDATEMLSEQUENCES($L_{k-1}, k$)
2:     $C_k = \emptyset$
3:     for all $s_1, s_2 \in L_{k-1}$ do
4:         if ML-subsequences wrt. GSP join rule of $s_1$ and $s_2$ are equal
           or can be generalized to common subsequence/s then
5:             Add ML-sequence joined from $s_1, s_2$ on the greatest level into $C_k$.
6:         end if
7:     end for
8:     return $C_k$
9: end procedure
```

C_1. Then, the ML-sequences for each candidate 1-sequence with sufficient support are generated as ML-sequential patterns or the generalization is performed.

Next phases run iteratively while some ML-sequential patterns are generated in the previous phase. In each phase, candidate ML-sequences are generated by the generalized join procedure (join step). In the prune step, the database D is passed and the generalized supports for the generated candidate ML-sequences are counted. It is effective to count also the generalized supports for the all ancestors of candidate ML-sequences, because it will be be used by following generalization procedure. The generalization tries to find the frequent ML-sequence with the greatest level for each candidate ML-sequence.

Sets of candidate ML-sequences can contain non-hierarchically-closed ML-sequences, the prune step verifies does not exist any child of the candidate ML-sequence with the same generalized support in the set of candidates, otherwise it is pruned.

Algorithm 3 The MLSP algorithm

```
1: procedure MLSP($D, min\_supp$)
2:     $k \leftarrow 1$                                                    ▷ First phase.
3:     Count $gen\_supp$ for all items in $D$                 ▷ Count also for ancestors.
4:     $C_1 \leftarrow$Add all 1-ML-sequences for candidates $c$ created from all items in $DB$
5:     Process sequences $c \in C_1$ by procedure GETFIRSTFREQUENTANCESTOR(c)
        and add resulting sequences into $L_1$
6:     while $L_k = \emptyset$ do                                ▷ Next iterative phases.
7:         $k \leftarrow k + 1$
8:         $C_k \leftarrow$GENERATECANDIDATEMLSEQUENCES($L_{k-1}, k$)
9:         Count $gen\_supp$ for all candidate sequences in $C_k$ and their ancestors
10:        Process items $c \in C_k$ by procedure GETFIRSTFREQUENTANCESTOR(c)
           and add resulting hierarchically-closed sequences into $L_k$
11:    end while
12:    return $\bigcup_{i=1}^{k} L_i$
13: end procedure
```

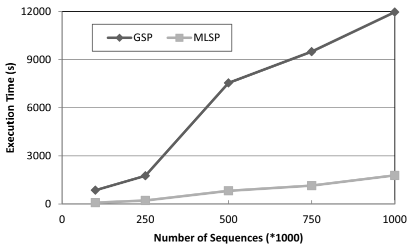
Algorithm heuristic. The algorithm often performs "is a subsequence" test (e.g. for the generalized support counting). This test can be optimized to the linear time-complexity if a suitable complete ordering exists over items. The simple ordering is not usable for MLSP because the subsequence uses the *generalized subset relation* $\sqsubseteq_g$. Therefore, MLSP uses two step ordering: 1) firstly sorts taxonomies lexicographically by their roots, 2) items in one taxonomy are sorted in the post-order walk. The first rule provides grouping of items in elements by a taxonomy, the second rule guarantees that it is possible to check for an ideal mapping to ancestors in the linear time complexity (the lexicographical ordering cannot be used because of the generalization which changes the order).

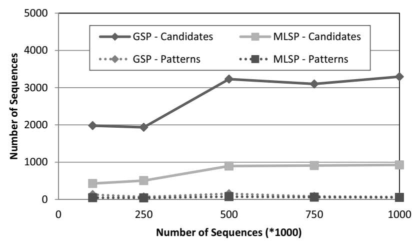
6 Experiments

We performed experiments on a PC in the configuration CPU i5 3.3GHz, 8GB RAM, OS MS Windows 7. Because there was not published any algorithm for the mining multi-level sequential patterns, we compare results of our algorithm to the GSP. Authors of GSP recommended using the GSP over an extended database [7], which is a modification of the origin dataset $\mathcal{D}$ such that an each item of the origin database $\mathcal{D}$ is replaced by a set of all its ancestors within its element, for mining sequential patterns with taxonomies. Both GSP and MLSP algorithms were implemented in C# on .NET platform. The experiments were executed over synthetic datasets generated by a random generator similar to [1] modified to generate sequences with items on different levels of taxonomies.

The first experiment compares scalability of GSP and MLSP. The parameters of the synthetic datasets are: $|\mathcal{D}|$ = *from 100 000 to 1 000 000, avg. sequence length = 4, sequential patterns count = 5, avg. sequential pattern length = 3, avg. support of sequential patterns=5%, taxonomies count = 0.1% of* $\mathcal{D}$*, avg. count of taxonomies levels=4.* The comparison of execution times is shown on Fig. 3a. The *MLSP* algorithm is about *5-10 times* faster than the *GSP* on all sizes of datasets. For both GSP and MLSP, the execution time grows approximately linearly with the size of the dataset (it corresponds to results presented by Agrawal et. al. in [1]). The main reason of higher speed is a smaller set of candidates (see the Fig. 3b). It results in much lower number of sequence comparisons when computing support. Moreover, the GSP algorithm generates a higher number of sequential patterns in contrast to MLSP, because sequential patterns generated by GSP are not hierarchically-closed. Notice that the numbers of candidate patterns and sequential patterns were summarized over all phases (when the average sequential pattern length is 3, then expected number of phases is 4).

The second experiment shows the dependency of the execution time and of the number of sequential patterns on the minimum support threshold. In this case, all experiments were executed over the same dataset and only the minimum support threshold parameter was changed. It was used a smallest dataset from the previous experiment where $|\mathcal{D}|$ = *100 000* sequences. The minimum support threshold was set to *1%, 3%, 5% and 7%.* The results are shown on Fig. 4a and Fig. 4b. Total execution time decreases with greater minimum support threshold.

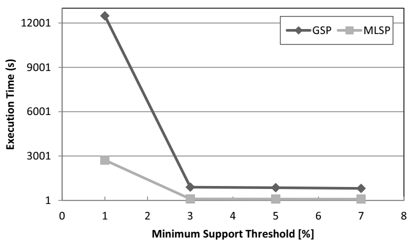


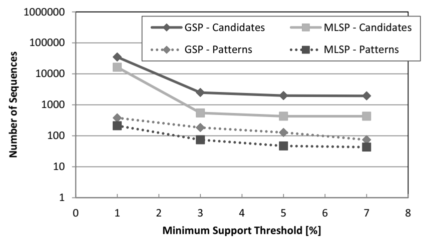


(a) Execution times of algorithms. (b) Number of cadidates and patterns.

Figure 3: Scalability experiment of GSP and MLSP algorithms.

Also, the number of sequential patterns decreases rapidly. In all cases, MLSP algorithm is much faster than GSP and produces more readable result set.





(a) Execution times of algorithms. (b) Number of cadidates and patterns.

Figure 4: Dependency on the minimum support threshold.

7 Conclusion

In this paper, we discussed the important problem of the multi-level sequential pattern mining. We introduced the term hierarchically-closed sequential pattern mining which reduces the number of output multi-level sequential patterns. We also proposed the MLSP algorithm for mining hierarchically-closed multi-level sequential patterns. We experimentally verified, that MLSP generates much smaller result set of sequential patterns without the loss of useful information for the user and the mining process is much more effective comparing to GSP when the extended database is used. In a future work, there is a challenge to develop an algorithm what does not generate candidates. We also plan to research the problem of mining multi-level sequential patterns for data streams.

Acknowledgement

This work has been supported by the research programme TAČR TA01010858, BUT FIT grant FIT-S-11-2, by the research plan MSM 0021630528, and the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070).

References

1. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Data Engineering, 1995. Proceedings of the Eleventh International Conference on. (mar. 1995) 3 –14
2. Han, J., Fu, A.: Mining multiple-level association rules in large databases. Knowledge and Data Engineering, IEEE Transactions on **11**(5) (1999) 798–805
3. Han, J., Kamber, M.: Data mining: concepts and techniques. The Morgan Kaufmann series in data management systems. Elsevier (2006)
4. Nakano, S.I.: Efficient generation of plane trees. Inf. Process. Lett. **84**(3) (nov. 2002) 167–172
5. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. VLDB '94 (1994) 487–499
6. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. SIGMOD Rec. **29**(2) (may 2000) 1–12
7. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In Apers, P., Bouzeghoub, M., Gardarin, G., eds.: Advances in Database Technology - EDBT '96. Volume 1057 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1996) 1–17
8. Zaki, M.: Spade: An efficient algorithm for mining frequent sequences. Machine Learning **42** (2001) 31–60
9. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. KDD '02, New York, NY, USA, ACM (2002) 429–435
10. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: Mining sequential patterns by pattern-growth: the prefixspan approach. Knowledge and Data Engineering, IEEE Transactions on **16**(11) (2004) 1424 – 1440
11. Yan, X., Han, J., Afshar, R.: Clospan: Mining closed sequential patterns in large datasets. In: In SDM. (2003) 166–177
12. Plantevit, M., Laurent, A., Laurent, D., Teisseire, M., Choong, Y.W.: Mining multidimensional and multilevel sequential patterns. ACM Trans. Knowl. Discov. Data **4**(1) (jan. 2010) 4:1–4:37
13. Šebek, M., Hlosta, M., Kupčík, J., Zendulka, J., Hruška, T.: Multi-level sequence mining based on gsp. Acta Electrotechnica et Informatica (2) (2012) 31–38