MULTI-LEVEL SEQUENCE MINING BASED ON GSP

Michal ŠEBEK, Martin HLOSTA, Jan KUPČÍK, Jaroslav ZENDULKA, Tomáš HRUŠKA
Department of Information Systems, Faculty of Information Technology, Brno University of Technology,
Božetěchova 1/2, 612 66 Brno, Czech Republic, tel. +420 54114 1100,
e-mail: isebek@fit.vutbr.cz, ihlosta@fit.vutbr.cz, ikupcik@fit.vutbr.cz, zendulka@fit.vutbr.cz, hruska@fit.vutbr.cz

ABSTRACT

Mining sequential patterns is an important problem in the field of data mining and many algorithms and optimization techniques have been published to deal with that problem. An GSP algorithm, which is one of them, can be used for mining sequential patterns with some additional constraints, like gaps between items. Taxonomies can exist upon the items in sequences. It can be applyed to mine sequential patterns with items on several hierarchical levels of the taxonomy. If a more general item appears in apattern, the pattern has higher or at least the same support as the one containing the corresponding specific item. This allows us to mine more patterns with the same minimal support parameter and to reveal new potentially useful patterns.. This paper presents a method for mining multi-level sequential patterns. The method is based on the GSP algorithm and generalization of more specific sequences based on the information theory.

Keywords: *sequence pattern mining, GSP, taxonomy*

1. INTRODUCTION

A great amount of data is being collected and stored in databases for various purposes. After years the size of these databases became enormous. In such amount of data hidden and interesting patterns may occur. Data mining, also known as knowledge discovery in databases, is a field of study to find these patterns in data. One of the typical data mining task is discovery of frequent patterns and *association analysis*. The latter one is seeking for rules in the form $antecedent \rightarrow consequent$, which occur in the data often enough and are so called strong. The strength means that the conditional probability $P(consequent|antecedent)$ is above a given threshold. The association analysis is abundantly used in market basket analysis to discover habits of customers. Its results can be employed for product recommendation. An example of such a rule can be $TV \rightarrow DVD\,player$ telling that when customers buys TV it's likely that a DVD player will appear in their market basket too.

If time information occurs in the data, not only association analysis but also a *sequential patterns mining* task can be sspecified. In such a case, the task is to reveal frequently occurring sequences in a sequence database. According to previously mentioned market basket analysis, an example of such a sequence pattern is $\langle TV\, DVD\_player\rangle$. This means that customers very often buy a TV set and later a DVD player.

In addition, a one or multiple taxonomies of items can be stored in the database. This allows mining patterns with items on different levels of hierarchy specified by the taxonomies. Then, the resulting pattern set can contain items from more general levels that might not be directly stored in the database. This can provide the analyst better insight of the data and reveal patterns that he wouldn't get with a pure sequence mining algorithm. It also allows us to set the minimum support parameter to a higher value and to get results containing more database sequences, because sequences with more general items have at least the same support as their more specific variants. Some algorithms can only find patterns, where all items in one pattern are on the same level of the hierarchy. They are referred to as *intra-level patterns*. If items of the pattern can be on different levels of hierarchy, they are called *inter-level* or *level-crossing patterns*. Our method is capable of revealing both intra-level and inter-level sequence patterns.

The remainder of the paper is organized as follows. In section 2 there is formally defined the problem of mining sequential patterns with taxonomies and terms related to that problem and to the information theory. Algorithms related to our work are described in section 3. The proposed method for mining sequential patterns with taxonomies is described in section 4. In section 5 we present performance evaluation of our method.

2. PROBLEM DEFINITION

In this section we present definitions of notions for mining sequential patterns with taxonomies from databases and the problem is formally defined. The section begins with terms related to sequential pattern mining, then focuses on taxonomy and ends with terms of the information theory necessary for our algorithm.

2.1. Sequential pattern mining

Definition 2.1. (Itemset) *Let* $I = \{I_1, I_2, I_3 \dots I_k\}$ *be a set of all items, that are stored in the database. Then* itemset $i = (x_1 x_2 \dots x_m)$ *is a nonempty subset of* I *containing* m *distinct items. Thus itemset is an unordered list of items, but with no loss of generality we can assume that items in the itemset are ordered lexicographically. Given* $I = \{a, b, c, d, e\}$ *an example of an itemset is* (abd).

Definition 2.2. (Sequence) *A* sequence $s = \langle e_1 e_2 e_3 \dots e_n\rangle$ *is an ordered list of* n *itemsets, sometimes called as elements or events. Based on this notation, example of such a sequence is* $s_1 = \langle av(ab)(ce)d(edgi)\rangle$. *When the itemset contains only one item, the braces can be omitted as it is shown in this example. The* length *of a sequence is considered as a number of instances of its items. The sequence of length* l *is called* l-sequence. *The length of previously* *mentioned sequence is 11, thus it is called* 11-sequence. *A sequence* $\alpha = \langle a_1 a_2 \dots a_n \rangle$ *is a* subsequence *of sequence* $\beta = \langle b_1 b_2 \dots b_m \rangle$ *if there exist integers* $1 \leq j_1 < j_2 < \cdots < j_n \leq m$ *such that* $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \dots, a_n \subseteq b_{j_n}$. *We denote it* $\alpha \sqsubseteq \beta$ *and* β *is a supersequence of* α *[1].*

Definition 2.3. (Sequence DB) *A sequence database D is a set of tuples* $\langle SID, s \rangle$ *, where* SID *is the identification of a sequence and s is the sequence. The support of the sequence* s_1 *is defined as a number of tuples in the database in which sequences are supersequences of* s_1. *Formally, the support of sequence* s_1 *is*

$$support(s_1) = |\{\langle SID, s\rangle | (\langle SID, s\rangle \in D) \wedge (s_1 \sqsubseteq s)\}|. \quad (1)$$

Definition 2.4. *Sequence pattern is defined as a frequent sequence, support of which is greater than the* minimum support *parameter, which is provided by the user.*

Based on the defined terms, we can formally define the problem of mining sequential patterns as follows: Given a sequential database $D = \{i_1, i_2 \dots i_n\}$, where each i_i in this database is an itemset, and the minimal support parameter *min_sup*, we are looking for all the sequences with support $\geq min\_sup$.

2.2. Taxonomy

Definition 2.5. (Taxonomy) *A* taxonomy structure *is an ordered directed tree. The taxonomy structure has* $l+1$ *levels. The node on the level* $h = 0$ *is called* root node, *the nodes on a level h where* $0 \leq h \leq l$ *are* internal nodes *and the nodes on the level* $h = l+1$ *are* leaf nodes. *The edges between nodes form* is-a relation, *the* specialization *of terms related to nodes is from the root to the leaf nodes. The* generalization *is from the leaf node to the root. An example of a taxonomy structure is depicted in figure 1, which represents some food products. The cheese* is a *dairy product and all dairy products and pastry are food. In our problem the items can be considered as nodes in the taxonomy structure.*

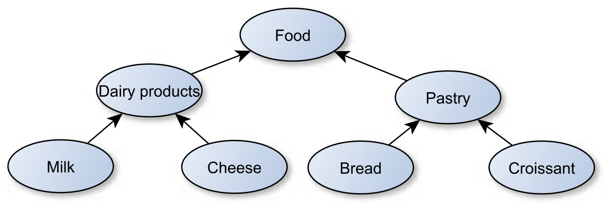

Fig. 1 Example of a food taxonomy with three levels and root containing all food.

Definition 2.6. (Parents) *Given an item i on a level h in the taxonomy structure, the* parent *of i, denoted as* $parent(i)$, *is its generalized item on level* $h-1$, *the* $ancestor(i)$ *is a set of all generalized items of i on a level l where* $0 \leq l < h$.

Given an element $e = \{i_1, i_2, \dots, i_n\}$, a set of *parent elements* of the element e is a set of the elements which are same as e but exactly one of the items is generalized. This is defined as

$$\begin{aligned} parent(e) \;=\; & \{\{j_1, j_2, \dots, j_n\} | \exists k : i_k \in e \\ & \wedge parent(i_k) = j_k \\ & \wedge \forall l \neq k : i_l = j_l\}. \end{aligned} \quad (2)$$

Notice that we took advantage of the property that items in elements are lexicographically ordered. Based on the definition of parent of an element, we can define the parent of a sequence. Given a sequence $s = \langle e_1 e_2 \dots e_n \rangle$, the parent of s is the set of sequences which are the same as the sequence s but one of their element is replaced by its parent. This can be defined as

$$\begin{aligned} parent(s) \;=\; & \{\langle f_1 f_2 \dots f_n \rangle | \exists k : e_k \in s \\ & \wedge parent(e_k) = f_k \\ & \wedge \forall l \neq k : e_l = f_l\}. \end{aligned} \quad (3)$$

Based on the definition of the parent of a sequence, the ancestor of the sequence s is defined as the set *ancestor*(s) as follows:

1. $parent(s) \in ancestor(s)$ (4)
2. $\forall x \in ancestor(s) : parent(x) \in ancestor(s)$.

Notice that the sequence s and all the sequences in the *ancestor*(s) has one common ancestor. This sequence consists of elements with root items of the items in sequence s. We denote this sequence as *root sequence*.

Example 2.1. *These notions can be well understood by looking at the figure 2 with two taxonomy structures for two root items A and B. Item A is an ancestor and a parent of items a and a'. The bottom of the figure depicts ancestors for the sequence* $\langle a(a'b)\rangle$. *The arrows in the figure represent which nodes are parents to the selected node. Sequences* $\langle a(a'B)\rangle, \langle A(a'b)\rangle, \langle a(Ab)\rangle$*are parents of the sequence* $\langle a(a'b)\rangle$, *because one of their elements is the parent of an element in* $\langle a(a'b)\rangle$, *for example* $(a'B)$ *is the parent of the* $(a'b)$. *All of the sequences in nodes above sequence*$\langle a(a'b)\rangle$ *are its ancestors and the sequence* $\langle A(AB)\rangle$ *is its root sequence.*

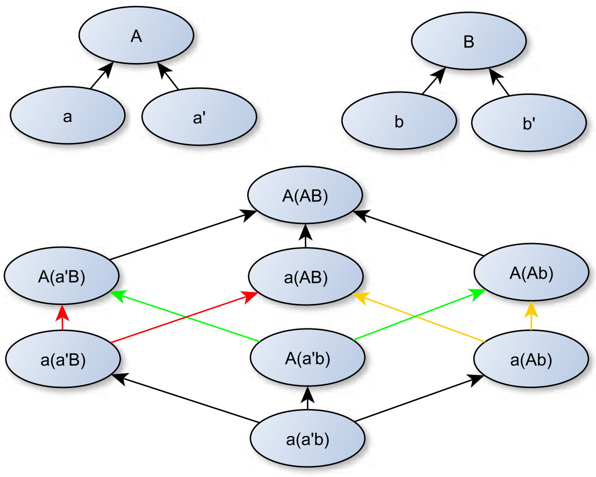

Fig. 2 Hierarchy in sequences.

Definition 2.7. *The* generalized support *gen_supp is based on the definition of support in definition 2.2. It's only necessary to redefine it, if an element e_1 is a subset of an element e_2. For this purpose we define the generalized subset relation $\subseteq_g$ as*

$$e_1 \subseteq_g e_2 \quad \Leftrightarrow \quad \forall i \in e_1 : i \in e_2 \vee \exists j \in e_2 : i \in ancestor(j). \tag{5}$$

A sequence $\alpha = \langle a_1 a_2 \ldots a_n \rangle$ is a *generalized subsequence* of 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 $\alpha \sqsubseteq_g \beta$. For completeness, the definition of *the generalized support* of a sequence s_1 is

$$gen\_supp(s_1) = |\{\langle SID, s \rangle | (\langle SID, s \rangle \in D) \wedge (s_1 \sqsubseteq_g s)\}|. \tag{6}$$

Example 2.2. *An example of how the generalized support of a sequence is computed can be shown on a sequence database from table 1 and taxonomy structures over items a, a', b, b' from figure 2. For sequence $s_1 = \langle a(Ab) \rangle$ generalized support $gen\_supp(s_1) = 2$, because it is generalized sequence of tuples with SID $\in \{1, 5\}$. If we take a parent of s_1, the sequence $s_2 = \langle a(AB) \rangle$, the generalized support will be even higher. In addition to s_1, s_2 is contained in the tuple with SID $= 4$, because $\langle a(AB) \rangle \sqsubseteq_g \langle b'aa(a'b')b \rangle$. The higher support is the consequence of $(AB) \subseteq_g (a'b')$. Element $(Ab) \nsubseteq_g (a'b')$, so that is why sequence s_1 has lower support than s_2.*

Table 1 Example of sequence database D with 5 sequences, $|D| = 5$.

SID	sequence
1	$\langle ab(a'b)b \rangle$
2	$\langle b(aa')b(ab')aa \rangle$
3	$\langle (ab)ba'a(bb') \rangle$
4	$\langle b'aa(a'b')b \rangle$
5	$\langle (ab)(ab)(ab) \rangle$

Having the definition of the taxonomy structure, we can define the task of mining sequential patterns with taxonomy in almost the same way as the problem without taxonomies. In addition, the presence of the taxonomies over the items in a sequence database causes that more general terms will occur in the result.

However not all of these patterns are interesting for an analyst. The question is: which of the generalized sequences are interesting to incorporate them into result set of patterns? This decision can be based on information theory which is described in more details in section 4.

3. RELATED WORK

The first algorithm for mining sequential patterns *AprioriAll* was published by Agrawal and Strikant in 1995 in [2]. It was the modification of the well-known *Apriori* algorithm for mining association rules such that it can mine sequential patterns. The general idea of both *Apriori* and *AprioriAll* is based on a candidates generate-and-test framework. The same authors then presented algorithm *Generalized Sequence Patterns (GSP)* in [3]. Our proposed algorithm is based on *GSP*, so this algorithm will be described in more detail.

3.1. GSP algorithm

As mentioned, the general idea of this algorithm is based on the Apriori property and candidates generation and testing approach. The apriori property states: *Every nonempty subsequence of a sequential pattern is a sequential pattern [1].*

The algorithm works in several phases. In each of them it makes a pass over the sequence database:

1. In the first phase, the support of items is counted and those having it higher than the *min_sup* are inserted in the resulting set L_1 containing *frequent 1-sequences*.

2. Each of the next phases iconsists of two sub-steps.

   (a) At first, in the *join step*, the *candidate set* C_k is generated from the set of frequent items from the previous phase L_{k-1}. The candidate sequences C_k contain one more item than sequences in L_{k-1}. Candidates of length two are generated from items in the manner, that either they occur in one transaction, thus forming an element, or one is after the other, thus forming a sequence. For example, for items a, b either $\langle (ab) \rangle$ or $\langle ab \rangle$can be generated . Candidates of higher length are generated from s_1 and s_2 such that if the first item of s_1 and the last item of s_2 are omitted, the resultant subsequences are the same. The sequences s_1 and s_2 are called *contiguous* and those *k-sequences* which has non-frequent contiguous subsequences are pruned.

   (b) After that, in *counting step*, the support of each candidate is counted resulting in frequent set L_k. The algorithm terminates in phase n when no candidate sequence satisfies the minimum support condition or no candidate sequence can be generated because of pruning step. The result set is composed of $\bigcup_{k=1}^{n-1} L_k$.

In comparison to *AprioriAll*, the GSP incorporates time constraints, sliding time windows and taxonomies in sequence patterns and thus allowing to mine for *generalized* patterns. The patterns with given time constraints and sliding time windows are achieved by a procedure which detects if the candidate is a subsequence of any sequence in database satisfying the constraints.

3.2. Other sequence pattern mining algorithms

Some modifications of algorithm GSP were published later, for example *PSP* in [4] introduces a different tree structure for maintaining candidates. In the category of algorithms based on candidate generating and testing, it's worth mentioning algorithm *SPADE [5]* which uses for mining vertical representation of sequence database; and also *SPAM* [6], which is similar to SPADE but uses internal bitmap structure for database representation.

The next family of algorithms is based on the *pattern-growth* principle. The key idea of these algorithms is that at first, it is created a representation of the sequence database to partition the space and then the search space is often traversed in depth-first manner to generate less candidate sequences. The most famous representative is the *PrefixSpan* published by Pei et al. in [7], which uses projected database with respect to some prefix for database representation.

The most recent algorithms are based on *early pruning*. The algorithms try to prune the searched space in the earliest phases based on the *position induction.* They store the last positions of items in database sequences and use this information to prune candidates which can't be appended to the current prefix. The rest of the algorithm is the same as in pattern-growth based. *LAPIN* [8] is the typical representative of these algorithms.

3.3. Hierarchies in sequence pattern mining

The authors of GSP presented the way how to incorporate the taxonomies into the process of sequential pattern mining [3]. The idea is based on replacing all the sequences in database with *"extended-sequences"*. In this extended form, in addition to the item, the information of all its ancestors is stored. For example sequence $\langle$(milk,bread)(croissant)$\rangle$ from the taxonomy in figure 1 is replaced by $\langle$(milk, pastry, dairy product, food) (croissant, pasrty, food)$\rangle$. Then the GSP algorithm is the same as was mentioned before on these *"extended-sequences"*. Although the authors proposed two optimizations, this approach requires much more space for storing the database sequences. It also allows to mine all the frequent sequences, even those which could be uninteresting.

In [9], the authors were using their framework for multidimensional sequence mining. They presented the *HYPE* algorithm to incorporate hierarchies into this framework and mine for multidimensional sequences over several levels of hierarchy.

In [10] T. Huang presented the concept of *fuzzy* multilevel sequential patterns. In this concept the item is allowed to belong among more general concepts, for example tomato can be considered either as fruit or vegetable. This relationship can be represented by a value between 0 and 1. They proposed the algorithm based on the divide-and-conquer strategy and an efficient algorithm to mine fuzzy cross-level patterns.

4. THE HGSP ALGORITHM

In this section we describe our algorithm *hGSP (hierarchical-GSP)* for mining level crossing sequence patterns. The algorithm is based on GSP algorithm [3]. The objective of the algorithm is to get the complete set of maximally concrete frequent sequences. The concreteness measure will be evaluated using information theory explained in following subsection. In following text we use *support* term in meaning of our defined *generalized support*.

4.1. Algorithm concept

The main idea of our algorithm is that if a sequence s has support $gen\_supp(s)$, there can exist a generalized sequence $s_g \in parent(s)$ such that $gen\_supp(s_g) > gen\_supp(s)$. This can be applied repeatedly. Notice that $\forall s_g \in parent(s) : gen\_supp(s) \leq gen\_supp(s_g)$. Unfortunately, during generalization some information is being lost. Because of this, the quality of mined generalized frequent items strictly depends on the selection of a the generalized sequence from the set of generalized sequences. The concepts of information theory is used for this purpose. Generally, we expect that more specific sequence s is more important result than it's generalized form s_g because the generalized s_g is more expectable in the result set. This corresponds with meaning of information content.

Definition 4.1. *The Shanon* information content *[11] of value x with probability $p(x)$ is defined as*

$$h(x) = \log_2 \frac{1}{p(x)}. \tag{7}$$

The probability $p(s)$ that sequence s occurs in source database D is

$$p(s) = \frac{gen\_supp(s)}{|D|}. \tag{8}$$

The *information content of sequence s* in database D is

$$h(s) = \log_2 \frac{1}{\frac{gen\_supp(s)}{|D|}} = -\log_2 \frac{gen\_supp(s)}{|D|}. \tag{9}$$

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 $h(s_g) < h(s)$. 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 ancestors.

Definition 4.2. Concreteness *The sequence s is more* concrete *than another sequence s_1 if $(h(s_1) < h(s)) \wedge (ancestor(s) \cup s) \cap (ancestor(s_1) \cup s_1) \neq \emptyset$.*

4.2. Algorithm hGSP

The hGSP algorithm uses the modified *join step* and *pruning step* of the GSP algorithm. The rest of algorithm remains the same.

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

The support of candidates is being counted almost the same as in the original GSP. The only difference is that we use $gen\_supp(s)$ defined in Definition 2.7 instead of common support. Thus only modified procedure for checking if a candidate is subsequence of a given database sequence is used.

The modification of *pruning step* is shown in Algorithm 1. The algorithm uses method for finding the most concrete generalization set of sequence which is described in Algorithm 2. We have formulated theorem, that is not necessary to evaluates all information contents with logarithm functions but it is possible to only compare ratios of supports of sequences and theirs generalized forms.

Let's have a sequence s and it's generalized form s_1. 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

$$\Delta h = \log_2 \left(\frac{\frac{gen\_supp(s_1)}{|D|}}{\frac{gen\_supp(s)}{|D|}} \right) = \log_2 \left(\frac{gen\_supp(s_1)}{gen\_supp(s)} \right). \quad (10)$$

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 $\frac{gen\_supp(s_1)}{gen\_supp(s)}$.

Algorithm 1: Method *find_generalization()*

Input: Candidate sequence s
Output: The set of the most concrete generalizations G_s.
Method:

```
    $G_s = \{\}$;
    $min\_supp\_ratio = +\infty$;
    foreach ($p_s \in parent(s)$) do:
      $ratio = gen\_supp(p_s)/gen\_supp(s)$;
      if ($gen\_supp(p_s) <> gen\_supp(s)$
              $\wedge ratio < min\_supp\_ratio$) then
        $G_s = \{p_s\}$;
        $min\_supp\_ratio = ratio$;
      elseif ($ratio = min\_supp\_ratio$) then
        $G_s = G_s \cup \{p_s\}$;
      endif;
    endforeach;
    return $G_s$;
```

Algorithm 2: hGSP Pruning Step

Input: The set of candidates - C_k, minimal support min_supp
Output: The set of frequent sequences L_k of length k
Method:

```
    $L_k = \{\}$;
    foreach $s_c \in C_k$ do:
      $C'_k = \{s_c\}$;
      $sequence\_added = false$;
      while ($sequence\_added = false \wedge |C'_k| > 0$) do:
        $G_s = \{\}$;
        foreach $s \in C'_k$ do:
          if $gen\_supp(s) \geq min\_supp$ then
            $L_k = L_k \cup \{s\}$;
            $sequence\_added = true$;
          else
            $G_s = G_s \cup find\_generalization(s)$;
          endif;
        endforeach;
        $C'_k = G_s$;
      endwhile;
    endforeach;
    return $L_k$;
```

5. EXPERIMENTS

We have used synthetic datasets created by the generator described in [12] with our hierarchical extension. The evaluation was performed on PC Intel Core Duo 2.66 GHz, 2GB RAM, OS Windows XP 32-bit. The implementation of hGSP algorithm was integrated into MS Analysis Services.

5.1. Performance evaluation

The performance test is focused on the execution time of the algorithm. We did not compare execution times with GSP because hGSP generates much more candidates. Also, GSP generates shorter sequences and it is finished after sequences of length 2 in most cases. Therefore, we show only execution times of hGSP. Parameters of datasets were:

- *average DB sequence length = 25,*
- *hierarchy count = 25,*
- *average hierarchy depth = 3,*
- *frequent sequences count = 0.2 % of* $|D|$
- *and average length of frequent sequences = 7.*

Results of experiment are shown in Figure 3. The algorithm was executed on datasets of a different size and the execution time was measured. The number of transaction items ranged from 25,000 in a database of 1,000 sequences up to 225,000 transaction items in a database of 9,000 sequences. The plot shows that processing time increases linearly with the dataset size because the algorithm checks if each candidate sequence is subsequent of any database sequence.

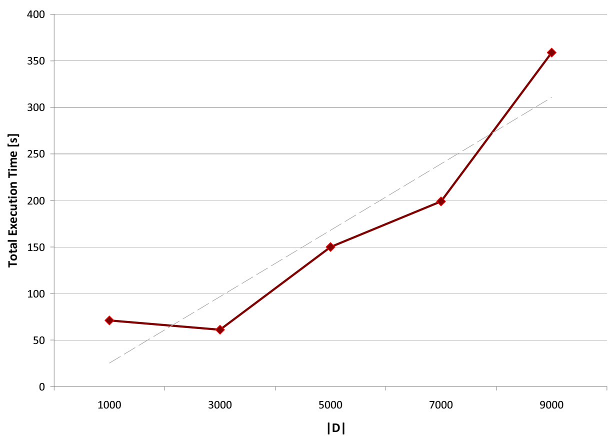

Fig. 3 Average processing time of one candidate sequence.

5.2. Generalization evaluation

The generalization evaluation experiment is focused on generalization property of hGSP algorithm. The methodology of the evaluation is following. It is known that the number of frequent sequences in results strictly depends on the minimal support value. Therefore, we have compared the number of candidates and mined frequent sequences for different values of the minimal support. The results were measured for relative supports from 20 % up to 80 %. Predefined parameters of frequent sequences in the dataset were: the average length 5 and support 50 %. Results of hGSP are shown in comparison to GSP results.

The results in Figure 4 show that the number of candidate/frequent sequences increases exponentially with descending value of minimal support for both hGSP and GSP. In general, hGSP creates more than 10 times more frequent sequences than the GSP because of the generalization property. This is the main substance for analyst – the hGSP does not prune important candidate sequences if there is a possibility to generalize them. In addition, Figure 5 shows that the hGSP creates sequences with more items than GSP.

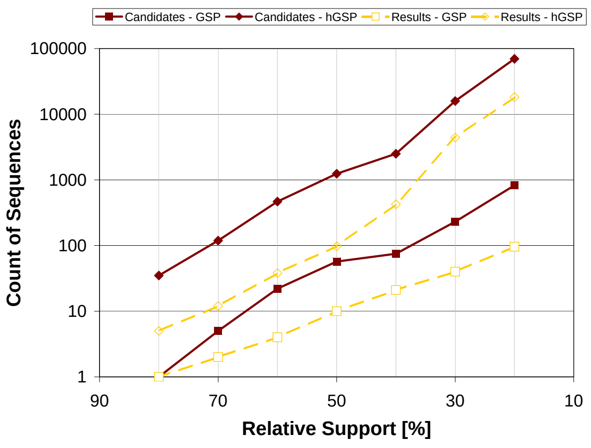

Fig. 4 Numbers of candidate and frequent sequences with dependancy on minimal support

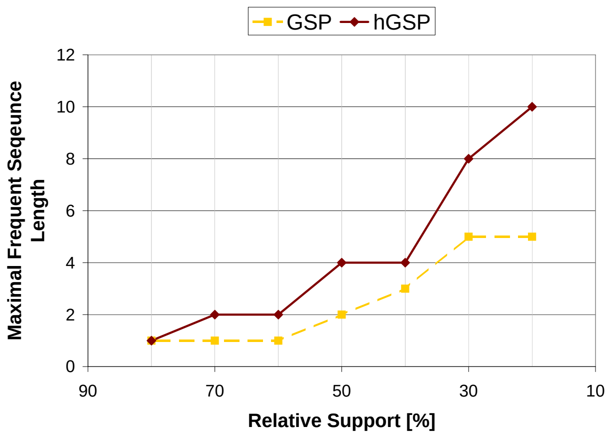

Fig. 5 Lengths of frequent sequences with dependancy on minimal support

5.3. Comparsion of hGSP algorithm and GSP using "extended-sequences"

Experiments show dependency of execution time and number of patterns on different number of frequent sequential patterns in database of length 7 $|L_7|$. The hGSP algorithm is compared with recommended taxonomy extension of GSP based on *"extended-sequences"* (see section 3.3) without using optimizations. Parameters of the datasets were *transactions count = 5 000, hierarchy count = 15, average hierarchy depth = 5*, other parameteres were same as in experiment in section 5.1.

In Table 2 is shown that hGSP is faster up to 10 % than the GSP using *"extended-sequences"*. The main disadvantage of GSP approach is shown in Table 3. GSP using *"extended-sequences"* generates a huge amount of redundant (candidate and frequent) sequences and the result has to be filtered, suggested author's optimization reduces the amount but the result set still contains many redunadnt patterns.

Table 2 Execution times of hGSP and GSP in seconds.

$\lvert L_7\rvert$	**hGSP**	**GSP**
25	1 536 s	1 546 s
50	3 968 s	4 229 s
100	1 327 s	1 515 s

Table 3 Total sum of sequential patterns of all lengths $|\bigcup_{k\geq 1} L_k|$.

$\lvert L_7\rvert$	**hGSP**	**GSP**
25	3 329	13 350
50	7 750	33 611
100	4 187	11 316

6. CONCLUSIONS

In this paper we presented a new way how to incorporate taxonomies into the process of mining generalized sequential patterns. Our approach is based on information theory and modifies the well known GSP algorithm. The algorithm is trying to generalize sequences only if it is requried. This is because of either low support of the sequence or inability to join two sequences with joinable ancestors. In the experiments, we showed that in comparison to GSP, the algorithm was able to find more patterns and patterns of higher length thanks to the generalization.

In the future work, we want to focus on optimization of the algorithm. Because the hGSP generates more candidates, it is naturally slower than GSP but outperforms the approach of extended databases for mining patterns with taxonomies. Because of unefficient candidates generating and testing we will also try to apply our ideas to more efficient algorithm such as PrefixSpan.

ACKNOWLEDGEMENT

This work has been supported by the research funding TAČR TA01010858, BUT FIT grant FIT-S-11-2, the Research Plan No. MSM 0021630528 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

REFERENCES

[1] HAN, J. – KAMBER, M.: *Data mining: concepts and techniques*. The Morgan Kaufmann series in data management systems. Elsevier, 2006.

[2] AGRAWAL, R. – SRIKANT, R.: Mining sequential patterns. pp. 3–14, 1995.

[3] SRIKANT, R. – AGRAWAL, R.: Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '96, pp. 3–17, London, UK, 1996. Springer-Verlag.

[4] MASSEGLIA, F. – CATHALA, F. – PONCELET, P.: The psp approach for mining sequential patterns. pp. 176–184, 1998.

[5] ZAKI, M. J.: Spade: An efficient algorithm for mining frequent sequences. In *Machine Learning*, pp. 31–60, 2001.

[6] 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, pp. 429–435, New York, NY, USA, 2002. ACM.

[7] PEI, J. – HAN, J. – MORTAZAVI-ASL, B. – WANG, J. – PINTO, H. – CHEN, Q. – DAYAL, U. – HSU, M.: Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Trans. on Knowl. and Data Eng.*, pp. 1424–1440, November 2004.

[8] YANG, Z. – WANG, Y. – KITSUREGAWA, M.: Lapin: effective sequential pattern mining algorithms by last position induction for dense databases. In *Proceedings of the 12th international conference on Database systems for advanced applications*, DASFAA'07, pp. 1020–1023, Berlin, Heidelberg, 2007. Springer-Verlag.

[9] PLANTEVIT, M.– LAURENT, A. – TEISSEIRE, M.: HYPE: mining hierarchical sequential patterns. In *DOLAP 2006, ACM 9th International Workshop on Data Warehousing and OLAP*, pp. 19–26, November 2006.

[10] HUANG, T. Ch.: Developing an efficient knowledge discovering model for mining fuzzy multi-level sequential patterns in sequence databases. *Fuzzy Sets Syst.*, 160:3359–3381, December 2009.

[11] MACKAY, D.: *Information Theory, Inference, and Learning Algorithms*. Cambrifge University Press, 2003.

[12] AGRAWAL, R. – SRIKANT, R.: Fast algorithms for mining association rules. In *Proc. of the VLDB Conference.*, pp. 487–499, Santiago, Chile, 1994. Expanded version available as ABM Research Report RJ8939.

Received December 12, 2007, accepted February 24, 2008

BIOGRAPHY

Michal Šebek is PhD student at Faculty of Information Technology, Brno University of Technology. He received his master's degree in Computer Science in 2009. His master theses was focused on development of open-source data mining system FIT-Miner. His PhD research focuses on knowledge discovery from data streams.

Martin Hlosta is a second year PhD student. He received his master's degree in Computer Science in 2010 from Faculty of Information Technology, Brno University of Technology. Both his bachelor and master theses were focused on methods of knowledge discovery in databases. His PhD research focuses on knowledge discovery from malware detection data.

Jan Kupčík received the Masters degree in Information Systems from Faculty of Information Technology, Brno University of Technology in 2007. He is currently pursuing the PhD degree at the same university. His research interests include database and OLAP technologies, data mining and software engineering.

Jaroslav Zendulka received his M.Sc. degree in computers and Ph.D. in technical cybernetics at the Brno University of Technology, Czech Republic. He is currently an Associate Professor at the Department of Information Systems at the Brno University of Technology. He has participated in several projects and has written tens of papers in international journals and conference proceedings. He is a PC member of several international conferences. His research interests include data and object modeling; database technology and information systems; data mining.

Tomáš Hruška received his Ing. (MSc.) and CSc. (PhD) titles from Brno University of Technology, Czech Republic. Since 1978 works at the Brno University of Technology, since 1998 as full professor. In 1978-1983, he dealt with research in the area of compiler implementation for simulation languages. He dealt with an implementation of an object-oriented database systems as a tool for modern information systems design and Lissom/Codasip project now. It is focused on the research of the processor description language for transformations of processor models