A New Algorithm for Fast Discovery of Maximal Sequential Patterns in a Document Collection

René A. García-Hernández José Fco. Martínez-Trinidad
Jesús Ariel Carrasco-Ochoa

National Institute of Astrophysics, Optics and Electronics (INAOE)
Puebla, México
{renearnulfo, fmartine, ariel}@inaoep.mx

Abstract. Sequential pattern mining is an important tool for solving many data mining tasks and it has broad applications. However, only few efforts have been made to extract this kind of patterns in a textual database. Due to its broad applications in text mining problems, finding these textual patterns is important because they can be extracted from text independently of the language. Also, they are human readable patterns or descriptors of the text, which do not lose the sequential order of the words in the document. But the problem of discovering sequential patterns in a database of documents presents special characteristics which make it intractable for most of the apriori-like candidate-generation-and-test approaches. Recent studies indicate that the pattern-growth methodology could speed up the sequential pattern mining. In this paper we propose a pattern-growth based algorithm (DIMASP) to discover all the maximal sequential patterns in a document database. Furthermore, DIMASP is incremental and independent of the support threshold. Finally, we compare the performance of DIMASP against GSP, DELISP, GenPrefixSpan and cSPADE algorithms.

1. Introduction

The *Knowledge Discovery in Databases* (KDD) is defined by Fayyad [1] as "the non-trivial process of identifying valid, novel, potentially useful and ultimately understandable patterns in data". The key step in the knowledge discovery process is the data mining step, which following Fayyad: "consisting of applying data analysis and discovery algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns over the data". This definition has been extended to *Text Mining* like: "consisting of applying text analysis and discovery algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns over the text". So, text mining is the process that deals with the extraction of patterns from textual data. This definition is used by Feldman [2] to define *Knowledge Discovery in Texts* (KDT). In both KDD and KDT tasks, special attention is required in the performance of the algorithms because they are applied on a large amount of information. In particular the KDT process needs to define simple structures that can be extracted from text documents automatically and in a reasonable time. These structures must be rich enough to allow interesting KD operations [2] having in mind that in some cases the document database is updated.

Sequential pattern mining has the goal of finding all the subsequences that are contained at least β times in a collection of sequences, where β is a user-specified support threshold. This discovered set of frequent sequences contains the *maximal frequent sequences* (MFSs), which are not a subsequence of any other frequent sequence. That is, the MFSs are a compact representation of the whole set of frequent sequences. So, the sequential pattern mining approaches play an important role in data mining tasks because these approaches allow us to identify valid, novel, potentially useful and ultimately understandable patterns in databases. In this case, we are interested in the extraction of this kind of patterns from textual databases. Due to its broad applications in text mining problems, finding textual patterns is important because they can be extracted from documents independently of the language without losing their sequential nature.

Most of the sequential pattern mining approaches have been developed for vertical databases, this is, databases with short sequences but with a large amount of sequences. A document database can be considered as horizontal because it could have long sequences. Therefore, sequential pattern mining approaches are not efficient for mining a document database. In order to guarantee human-legible and meaningful patterns we are interested in finding contiguous MFSs. Also, these special patterns could be of interest in the analysis of DNA sequences [10], data compression and web usage logs [9].

Furthermore, most of the sequential pattern mining approaches assume a short alphabet; that is, the set of different items in the database. So, the characteristics of textual patterns make the problem intractable for most of the apriori-like candidate-generation-and-test approaches. For example, if the longest MFS has a length of 100 items then GSP[3] will generate $\sum_{i=1}^{100}\binom{100}{i}\approx 10^{30}$ candidate sequences where each one must be tested over the DB in order to verify its frequency. This is the cost of candidate generation, no matter what implementation technique would be applied. For the candidate generation step, GSP generates candidate sequences of size $k+1$ by joining two frequent sequences of size k when the prefix $k-1$ of one sequence is equal to the suffix $k-1$ of other one. Then a candidate sequence is pruned if the sequence is non-frequent. Even though, GSP reduces the number of candidate sequences, it still being inefficient for mining long sequences.

Recent studies indicate that the pattern-growth methods could speed up the sequential pattern mining [4,5,6,10,11] when there are long sequences. According to empirical performance evaluations the pattern-growth methods like PrefixSpan[4], GenPrefixSpan[5] and DELISP[6] outperform GSP specially when the database contains long sequences. The basic idea is to avoid the cost of candidate generation step and to focus the search on sub-databases generating projected databases. An α-projected database is the set of subsequences in the database that are suffixes of the sequences with prefix α. In each step, the algorithm looks for frequent sequences with prefix α in the corresponding projected database. In this sense, pattern-growth methods try to find the sequential patterns more directly, growing frequent sequences, beginning with sequences of size one. Even though, these methods are faster than apriori-like methods, some of them were designed to find all the frequent sequences and not to get only the MFSs. Furthermore, none of them is incremental.

Other work related to searching of repeated substrings in a set of strings is the longest common substring (LCS) problem. From this point of view, the documents can be taken as strings of words. The objective of LCS is to find the longest substring that is repeated in all the set of strings. The LCS problem can be solved using suffix trees, but the LCS problem looks for only one substring (the longest) which appears in all the documents. However, we need to find all the maximal substrings that appear at least in β documents.

In this paper we propose a pattern-growth based algorithm (DIMASP) to **Di**scover all the **Ma**ximal **S**equential **P**atterns in a document database. First, DIMASP builds a novel data structure from the document database which is relatively easy to extract. Once DIMASP has built the data structure, it can discover all the MFSs according to the threshold specified by the user. If a new threshold is specified, DIMASP avoids rebuilding the data structure for mining with this new threshold. In addition, when the document database is increased, DIMASP updates the last discovered MFSs by processing only the new documents. DIMASP assumes that the data structure can fit in the main memory.

In section 2, the problem definition is given. Section 3 describes our algorithm. In Section 4, the experiments are presented. Finally in section 5 the conclusions and future work are given.

2. Problem Definition

A *sequence S,* denoted by $<s_1,s_2,...,s_k>$, is an ordered list of k elements called *items*. The number of elements in a sequence S is called the *length* of the sequence denoted by $|S|$. A *k-sequence* denotes a sequence of length k. Let $P=<p_1p_2...p_n>$ and $S=<s_1s_2...s_m>$ be sequences, P is a *subsequence* of S, denoted $P\subseteq S$, if there exists an integer i≥1, such that $p_1=s_i$, $p_2=s_{i+1}$, $p_3=s_{i+2}$, ..., $p_n=s_{i+(n-1)}$. We can consider a *document W* as a sequence of words, denoted as $<w_1,w_2,...,w_n>$.

The *frequency* of a sequence S, denoted by S_f or $<s_1,s_2,...,s_n>_f$, is the number of documents where S is a subsequence. A sequence S is *β-frequent* if $S_f \geq \beta$, a β-frequent sequence is also called a *sequential pattern*. A sequential pattern S is *maximal* if S is not a subsequence of any other sequential pattern.

In this paper, we are interested in the problem of discovering all the maximal sequential patterns in a document database.

3. DIMASP: A New Algorithm for Fast Discovery of all Maximal Sequential Patterns

The basic idea of DIMASP consists in finding all the sequential patterns in a data structure, built from the document database (DDB), which stores all the distinct pairs of contiguous words that appear in the documents, without losing their sequential order. Given a threshold β specified by the user, DIMASP reviews if a pair is β-frequent. In this case, DIMASP grows the sequence in order to determine all the possible maximal sequential patterns containing such pair as prefix. A possible maximal sequential pattern (PMSP) will be a maximal sequential pattern (MSP) if it is not a subsequence of any previous MSP. This implies that all MSPs which are subsequence of the new PMSP are deleted. The proposed algorithm is composed of three steps described as follows:

In the first step, DIMASP assigns an integer number, as an identifier, for each different word (item) in the DDB. Also, the frequency for each identifier is stored *i.e.* the number of documents where it appears. These identifiers are used in the algorithm instead of the words in the DDB like in the example of the table 1.

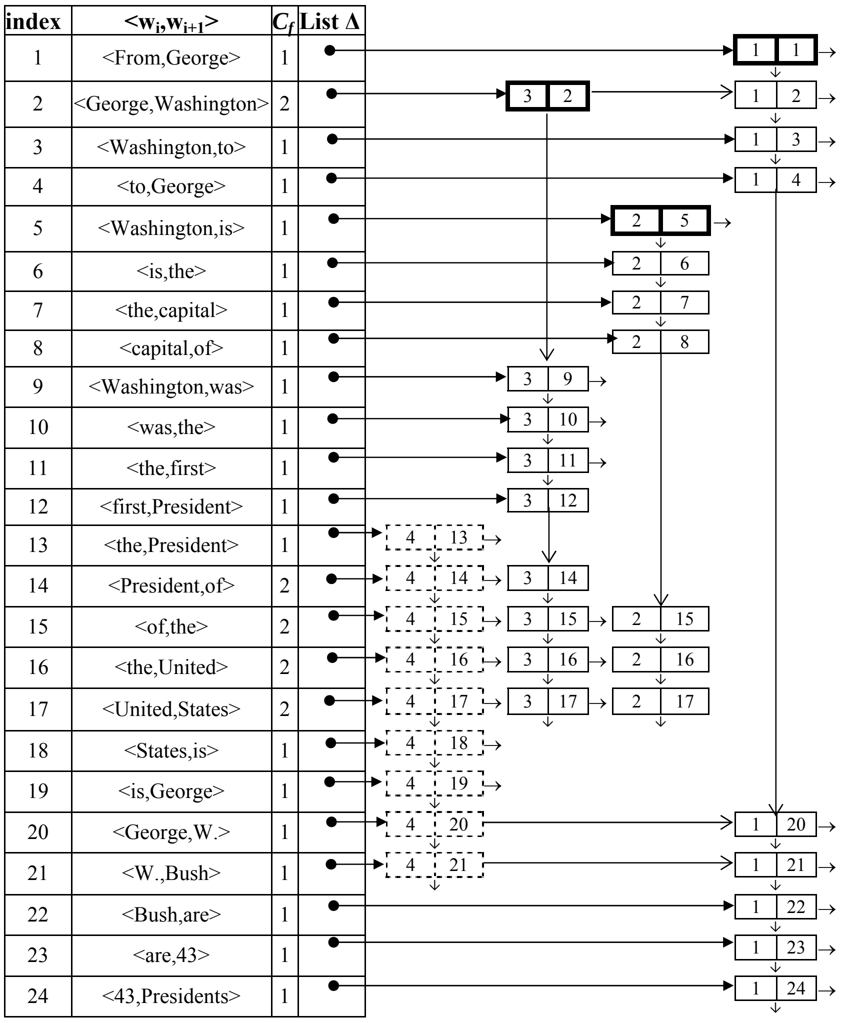
Table 1. An example of a document database and its identifier representation

D_J	**Document database**	**Integer identifiers**
1	From George Washington to George W. Bush are 43 Presidents	<1,2,3,4,2,5,6,7,8,9>
2	Washington is the capital of the United States	<3,10,11,12,13,11,14,15>
3	George Washington was the first President of the United States	<2,3,16,11,17,18,13,11,14,15>
4	*the President of the United States is George W. Bush*	*<11,18,13,11,14,15,10,2,5,6>*

```
Step 2: Algorithm to construct the data structure from the DDB
Input: A document database (DDB)    Output: The Array
For all the documents $D_J \in DDB$ do
    Array ← Add a document ( $D_J$ ) to the array
end-for
Step 2.1: Algorithm to add a document
Input: A document $D_J$  Output: The Array
    For all the pairs $\langle w_i, w_{i+1} \rangle \in D_J$ do
        $\delta_i$ ←Create a new Pair $\delta$
        $\delta_i$.Id ← J     //Assign the document identifier to the node $\delta$
        index ← Array[ $\langle w_i, w_{i+1} \rangle$ ] //Get the index of the cell where is <$w_i$,$w_{i+1}$>
        $\delta_i$.index ← index    //Assign the index to the node $\delta$
        $\alpha$ ← Get the first node of the list $\Delta$
        If $\delta_i$.Id $\neq \alpha$.Id then the document identifier is new to the list $\Delta$
            Increment $C_f$     //increment the frequency
        $\delta_i$.NextDoc ← $\alpha$     //link the node $\alpha$ at the beginning of list $\Delta$
        List $\Delta$← Add $\delta_i$ as the first node  //link it at the beginning of list $\Delta$
        $\delta_{i-1}$.NextNode ← $\delta_i$    //link the pair to do not lose the sequential order
    end-for
```

Fig. 1. Algorithms for steps 2 and 2.1 where is built the data structure for documents

In the second step, DIMASP builds a data structure from the DDB storing all the pairs of contiguous words <w_i,w_{i+1}> that appear in a document and some additional information to preserve the sequential order. The data structure is a special *array* which contains in each cell a pair of words C=<w_i,w_{i+1}>, the *frequency* of the pair (C_f), a Boolean *mark* and a list Δ of nodes δ where a *node* δ (see Fig. 2) stores a document identifier (δ.*Id*), an *index* (δ.*Index*) of the cell where the pair appears in the array, a link (δ.*NextDoc*) to maintain the list Δ and a link (δ.*NextNode*) to preserve the sequential order of the pairs with respect to the document. Therefore, the number of different documents presented in the list *Δ* is C_f. This step works as follows: for each pair of words $<w_i,w_{i+1}>$ in the document D_J, if $<w_i,w_{i+1}>$ is not in the *array* add it, and get its *index*. In the position *index* of the array, add a node δ at the beginning of the list *Δ*. The added node δ has J as δ.*Id*, *index* as δ.*index*, δ.*NextDoc* is linked to the first node of the list *Δ* and δ.*NextNode* is linked to the next node δ corresponding to $<w_{i+1},w_{i+2}>$ of the document D_J. If the document identifier (δ.*Id*) is new in the list *Δ*, then the frequency of the cell (C_f) is increased. In Fig. 2 the data structure built with the step 2 algorithm the document database of table 1 is shown.



index	$<w_i,w_{i+1}>$	C_f	List Δ
1	<From,George>	1	●
2	<George,Washington>	2	●
3	<Washington,to>	1	●
4	<to,George>	1	●
5	<Washington,is>	1	●
6	<is,the>	1	●
7	<the,capital>	1	●
8	<capital,of>	1	●
9	<Washington,was>	1	●
10	<was,the>	1	●
11	<the,first>	1	●
12	<first,President>	1	●
13	<the,President>	1	●
14	<President,of>	2	●
15	<of,the>	2	●
16	<the,United>	2	●
17	<United,States>	2	●
18	<States,is>	1	●
19	<is,George>	1	●
20	<George,W.>	1	●
21	<W.,Bush>	1	●
22	<Bush,are>	1	●
23	<are,43>	1	●
24	<43,Presidents>	1	●

Fig. 2. Data structure built for the document database of the table 1. Note, the dotted nodes δ corresponding to D_4 will be added when D_4 would be included, of course, it will be necessary to update the frequencies C_f of the array

```
Step 3: Algorithm to find all MSPs
  Input: Structure from step 2 and β threshold          Output: MSP set
  For all the documents $D_{J \cdots (\beta - 1)} \in DDB$ do
    MSP set ← Find all MSPs w.r.t. the document ($D_J$)
Step 3.1: Algorithm to find all MSPs with respect to the document $D_J$
  Input: A $D_J$ from the data structure and a β threshold  Output: The MSP set w.r.t. to $D_J$
  For all the nodes $\delta_{i=1 \cdots n} \in D_J$ i.e. $\langle w_i, w_{i+1} \rangle \in D_J$ do
    If Array [ $\delta_i$.index ].frequency ≥ β then      //if the pair has a frequency≥ β
      PMSP ← Array [ $\delta_i$.index ].$\langle w_i, w_{i+1} \rangle$ //the initial PMSP is the pair <$w_i$,$w_{i+1}$>
      $\Delta'$ ←Copy the rest of the list of Δ beginning from $\delta_i$.NextDoc
      $\Delta'_f$ ← Number of different documents in Δ'
      $\delta'_i \leftarrow \delta_i$
      While $\Delta'_f \geq \beta$ do the growth the PMSP
        $\Delta''$ ← Array [$\delta'_{i+1}$.index ].list Δ    //Denotes to Array [$\delta'_{i+1}$.index ].list Δ as $\Delta''$
        $\Delta'$ ← $\Delta' \cap \Delta''$ i.e. $\{\alpha \in \Delta' \mid (\alpha\text{.index} = \delta'_{i+1}) \wedge (\delta'_i\text{.NextNode} = \alpha)\}$
        $\Delta'_f$ ←Number of different documents in Δ'
        If $\Delta'_f \geq \beta$ then to grow the PMSP
          Array [ $\delta'_{i+1}$.index ].mark ← "used"
          PMSP ← PMSP + Array [ $\delta'_{i+1}$.index ].$\langle w_{i+1} \rangle$
          $\delta'_i \leftarrow \delta'_{i+1}$ i.e. $\delta'_i$.NextNode
      end-while
      If |PMSP| ≥ 3 then add the PMSP to the MSP set
        MSP set ← add a k-PMSP to the MSP set //step 3.1.1
    end-for
  For all the cells $C \in Array$ do the addition of the 2-MSPs
    If $C_f$≥ β and C.mark = "not used" then add it as 2-MSP
      2-MSP set ← add $C.\langle w_i, w_{i+1} \rangle$
```

Fig. 3. Algorithm to find all the MSPs using the data structure of step 2 and a threshold β

To prove that our algorithm finds all the MSPs we introduce the following proposition.

Proposition 1: DIMASP discovers all the maximal sequential patterns of a DDB.

Proof. To proof that DIMASP finds all the MSPs, suppose there is a *k*-MSP in the document database. Therefore, if there is a *k*-MSP then it is contained in at least β documents in the database. For $k \geq 2$ the *k*-MSP is $<w_1,w_2,w_3,...,w_k>$ which can be separated in its frequent pairs $<w_1,w_2> + <w_2,w_3> + \cdots + <w_{k-1},w_k>$. From step 2, we know that the pair $<w_i,w_{i+1}>$ and the list Δ in a cell of the array are stored, denoted by $\Delta(<w_i,w_{i+1}>)$, containing the registers of all documents that have this pair, without losing their sequential order. Therefore, as it was made in steps 3 and 3.1, we can index and get $\Delta(<w_1,w_2>)$, $\Delta(<w_2,w_3>)$, …, $\Delta(<w_{k-1},w_k>)$. Also, from the array, it is clear that the frequencies of such subsequences are ≥β. Now we have to proof that they form the k-MSP. Since the pairs do not lose their sequential order, we can establish that $\Delta(<w_1,w_2>) \cap \Delta(<w_2,w_3>) = \Delta(<w_1,w_2,w_3>)$ and $\|\Delta(<w_1,w_2,w_3>)\| \geq \beta$. Therefore, $\bigcap_{i=1}^{k} \Delta(<w_i,w_{i+1}>) = \Delta(<w_1,w_2,w_3,\ldots,w_k>)$ which is actually $\Delta(k\text{-MSP})$. And it can not grow because $\|\Delta(<w_1,w_2,w_3,\ldots,w_k>) \cap (<w_k,w_{k+1}>)\| < \beta$ since k-MSP is maximal. If k=1 then DIMASP includes this 1-MSP because in step 1 DIMASP includes all the frequent items which are not included in any other longer MSP.∎

In order to be efficient it is needed to reduce the number of comparisons when a PMSP is added to the MSP set. For such reason, a k-MSP is stored according to its length k, it means, there is a k-MSP set for each k. Also, for speed up the comparison of PMSPs, binary searches using the sum of the identifiers in a PMSP are performed. In this way, before adding a k-PMSP as a k-MSP, the k-PMSP must not be in the k-MSP set and must not be subsequence of any longer k-MSP. Two sequences might be equal only if they have the same sum. A sequence A could be a subsequence of another sequence B only if the sum of A is lesser than the sum of B. When a PMSP is added, all their subsequences are eliminated.

Step 3.1.1: Algorithm to add a PMSP to the MSP set

```
Input: A k-PMSP, MSP set                    Output: MSP set
If (k-PMSP ∈ k-MSP set) or
If (k-PMSP is subsequence of some longer k-MSP) then do not add anything
    return MSP set
Else
    k-MSP set ← add k-PMSP //add as a MSP
    {del S ∈ MSP set | S ⊆ k-PMSP }
```

Fig. 4. Algorithm to add a PMSP to the MSP set

Since the *array* has only the distinct pair of words $<w_i,w_{i+1}>$ the performance for comparing two sequences can be improved if instead of adding the identifiers w_i and w_{i+1} to PMSP only the *index* of the array where the pair appears is added. In this way, a sequence A is a subsequence of B only if the last and the odds items of A are contained in B. For example, if the array structure of Fig. 2 is used with the sequence A=<*the, President, of, United, States*> and B=<*the, President, of, United, States, is*> then the sequences A=<*13,14,15,16,17*> and B=<*13,14,15,16,17,18*>. Therefore it is enough, checking that the last and odds items of A=<*13,■,15,♦,17*> are contained in B=<*13,■,15,♦,17,●*>, to guarantee that A⊆B because only the items *14* and *16* can fit in ■ and ♦, respectively.

With the objective of do not repeat all the work to discover all the MSPs when one or a set of new documents are added to the database, DIMASP only preprocesses the part corresponding to these new documents. After the identifiers of these new documents were defined in step 1, DIMASP would only use the step 2.1 to add them to the array. Then, the step 3.1 is applied on the new documents and on the old MSP set, to discover the new MSP set. This strategy works only for the same β, however with a different β only the discovery step (step 3) must be applied, without rebuilding the data structure. For example, Fig. 2 shows with dotted line the new part of the data structure when D_4 of table 1 is added as a new document. Then, using β=2 for the algorithm of the step 3, the PMSPs <*President,of,the,United,States*> and <*George,W.,Bush*>

are discovered. The first PMSP eliminates the previous discovered maximal sequential pattern <*of,the,United,States* > because it is not maximal.

4. Experiments

The next experiments were accomplished using the well-known reuters-21578 document collection [7]. After a prune of 400 stop-words, this collection has 21578 documents with around 38,565 different words from 1.36 million words used in the whole collection. The average length of the documents is 63 words. In all the experiments the first 5000, 10000, 15000 and 20000 documents were used. Excepting for GSP, the original programs provided by the authors were used. In Fig. 5a the performance comparison of DIMASP, cSPADE[8], GenPrefixSpan, DELISP and GSP algorithms with β=15 is shown. Fig. 5b shows the same comparison of Fig. 5a but the worst algorithm (GSP) is eliminated, here it is possible to see that DELISP is not as good as it seems to be in Fig. 5a. In this case GenPrefixSpan had memory problems, so it was only possible to test with the first 5000 and 10000 documents. Fig. 5c compares DIMASP against the fastest algorithm cSPADE, the time of the steps 2 and 3 of DIMASP are also compared. Fig. 5d draws a linear scalability of DIMASP whit respect to β. An additional experiment with the lowest β=2 was performed, in this experiment DIMASP found a MSP of length 398, Fig. 5e shows the results. To evaluate the incremental scalability of DIMASP, 4000, 9000 14000 and 19000 documents were processed, and 1000 documents were added in each experiment. Fig. 5f shows the results and compares them against cSPADE which needs to recompute all the MSPs. Fig. 5g shows the distribution of the MSPs according to their length. Finally, Fig. 5h shows the number of MSPs when β = 1% of the documents in the collection was used.

5. Conclusions

In this paper, DIMASP a pattern-growth memory-based algorithm to discover all the maximal sequential patterns MSPs in a document database was proposed. To do that, DIMASP builds a data structure for the document database which speeds up the mining of MSPs. Our algorithm allows working with different support thresholds without rebuilding the data structure. Moreover, DIMASP is incremental with respect to document addition. According to the empirical evaluations, DIMASP outperforms GSP, DELISP GenPrefixSpan and cSPADE algorithms in discovering all MSPs in a document database and has a good scalability regarding to β. One of the reasons for which DIMASP is more efficient is because the algorithm begins to discover MSPs longer than 2 and the 1-MSPs and 2-MSPs, which are the majority of the MSPs, are discovered in one-pass. For example, Fig. 5g shows that $\sum_{i=1}^{2} \left| i - \mathrm{MSP} \right| > \sum_{i=3}^{14} \left| i - \mathrm{MSP} \right|$. For our experiments with the whole reuters-21578 collection DIMASP used around 30 Mbytes of main memory for the data structure built in step 2 which is able to be handled by most of the computers. This shows that, even though DIMASP needs the whole data structure to fit in main memory, it might process bigger document collections.

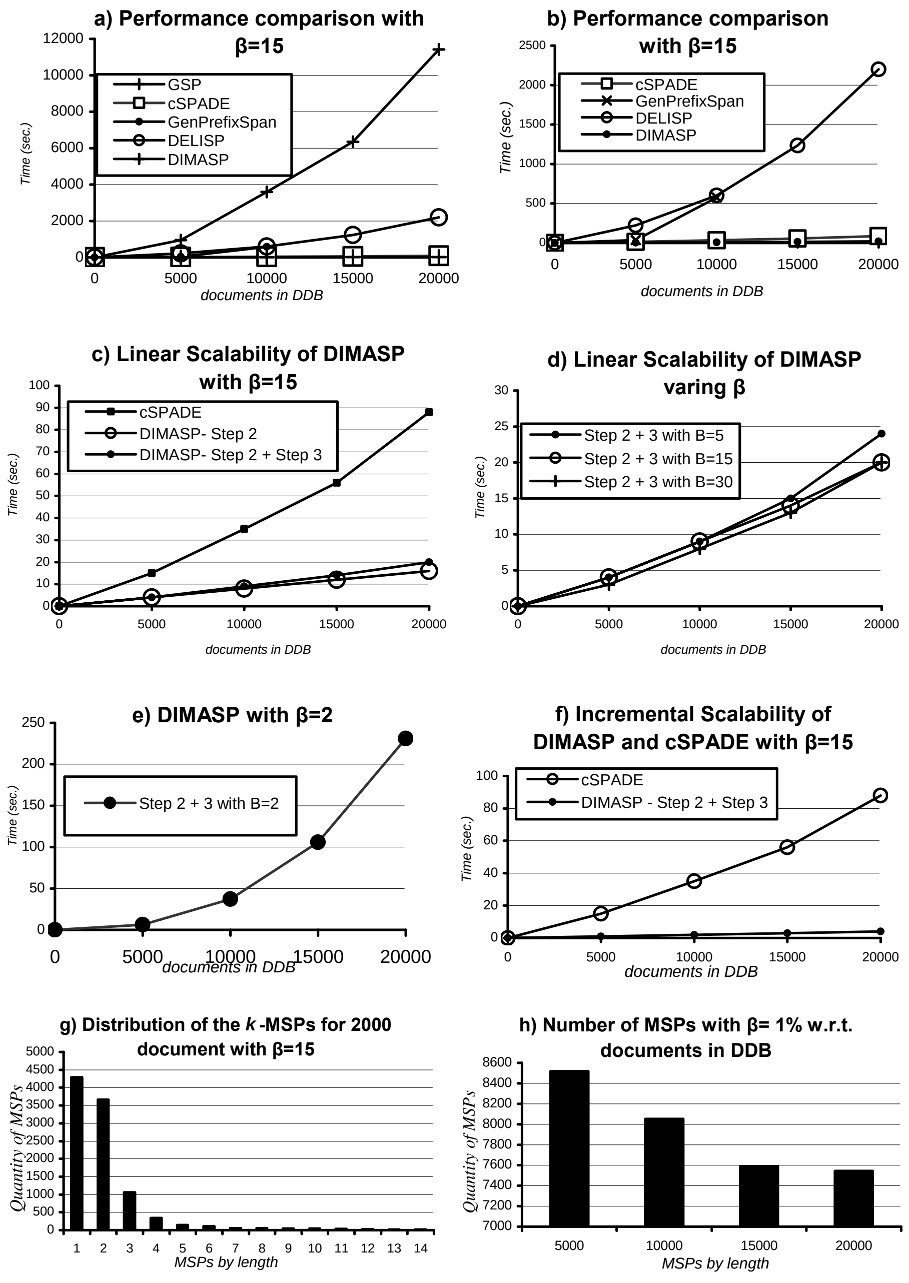


Fig. 5. Results of the performance experiments using the collection Reuters-21578

As future work we will extend the idea of DIMASP to be able of manage a gap constraint which allows a controlled separation between the items that form a sequential pattern. Also we are going to apply DIMASP on other kind of data like web logs or DNA sequences.

References

[1] Fayyad, U., Piatetsky-Shapiro G. "Advances in Knowledge Discovery and Data mining". AAAI Press, 1996.

[2] Feldman, R and Dagan, I. "Knowledge Discovery in Textual Databases (KDT)", *In Proceedings of the 1st International Conference on Knowledge Discovery (KDD-95)* 1995.

[3] Srikant, R., and Agrawal, R. Mining sequential patterns: Generalizations and performance improvements. *In 5th Intl. Conf. Extending Database Discovery and Data Mining*, 1996.

[4] Pei, J, Han, et all: "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth" in *Proc International Conference on Data Engineering* (ICDE 01), 2001.

[5] Antunes, C., Oliveira A. Generalization of Pattern-growth Methods for Sequential Pattern Mining with Gap Constraints. *Third IAPR Workshop on Machine Learning and Data Mining MLDM´2003*, 2003.

[6] Ming-Yen Lin, Suh-Yin Lee, and Sheng-Shun Wang, "DELISP: Efficient Discovery of Generalized Sequential Patterns by Delimited Pattern-Growth Technology," Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD02), Taipei, Taiwan, pp. 189-209, 2002.

[7] http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html

[8] Mohammed J. Zaki, Sequence Mining in Categorical Domains: Incorporating Constraints, in *9th International Conference on Information and Knowledge Management*, pp 422-429, Washington, DC, November 2000.

[9] Amir H. Youssefi, David J. Duke, Mohammed J. Zaki, "Visual Web Mining". *13th International World Wide Web Conference* , New York, NY, 2004.

[10] Jiawei Han and Micheline Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, August 2000. c.9 &10.

[11] Jian Pei, Jiawei Han, et. al. "Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, No. 10, October 2004.