Finding Maximal Sequential Patterns in Text Document Collections and Single Documents

René Arnulfo García-Hernández
Autonomous University of the State of Mexico
Tianguistenco Professional Academic Unit
Paraje el Tejocote, San Pedro Tlaltizapan, Estado de México
E-mail: renearnulfo@hotmail.com, http://scfi.uaemex.mx/~ragarcia/

J. Fco. Martínez-Trinidad and J. Ariel Carrasco-Ochoa
National Institute of Astrophysics, Optics and Electronics
E-mail: fmartine@inaoep.mx, ariel@inaoep.mx

Keywords: text mining, maximal sequential patterns

Received: April 1, 2009

In this paper, two algorithms for discovering all the Maximal Sequential Patterns (MSP) in a document collection and in a single document are presented. The proposed algorithms follow the "pattern-growth strategy" where small frequent sequences are found first with the goal of growing them to obtain MSP. Our algorithms process the documents in an incremental way avoiding re-computing all the MSP when new documents are added. Experiments showing the performance of our algorithms and comparing against GSP, DELISP, GenPrefixSpan and cSPADE algorithms over public standard databases are also presented.

Povzetek: Predstavljena sta dva algoritma za iskanje najdaljših zaporedij v besedilu.

1 Introduction

Frequent pattern mining is a task into the datamining area that has been intensively studied in the last years [Jiawei Han et al. 2007]. Frequent patterns are itemsets, subsequences, or substructures that appear in a data set with frequency no less than a user-specified threshold. Frequent pattern mining plays an important role in mining associations, correlations, finding interesting relationships among data, data indexing, classification, clustering, and other data mining tasks as well. Besides, frequent patterns are useful for solving more complex problems of data analysis. Therefore, frequent pattern mining has become an important area in data mining research.

Frequent pattern mining was first proposed by [Agrawal et al. 1993] for market basket analysis finding associations between the different items that customers place in their "shopping baskets". Since this first proposal there have been many research publications proposing efficient mining algorithms, most of them, for mining frequent patterns in transactional databases.

Mining frequent patterns in document databases is a problem which has been less studied. Sequential pattern mining in document databases has the goal of finding all the subsequences that are contained at least β times in a collection of documents or in a single document, where β is a user-specified support threshold. This discovered set of frequent sequences contains the maximal frequent sequences which are not a subsequence of any other frequent sequence (from now on we will use the term Maximal Sequential Patterns, MSP), that is, the MSPs are a compact representation of the whole set of frequent sequences. Therefore, in the same way as occurs in transactional databases, the sequential pattern mining in document databases plays an important role, because it allows identifying valid, novel, potentially useful and ultimately understandable patterns. In this paper, we will focus in the extraction of this kind of patterns from textual or text document databases. Since maximal sequential patterns can be extracted from documents independently of the language without losing their sequential nature they can be used to solve more complex problems (all of them related to text mining) as question answering [Denicia-Carral et al. 2006; Juárez-González et al. 2007; Aceves-Pérez et al. 2007], authorship attribution [Coyotl-Morales et al. 2006], automatic text summarization [Ledeneva et al. 2008], document clustering [Hernandez-Reyes et al. 2006], and extraction of hyponyms [Ortega-Mendoza et al. 2007], among others.

In this article, we present two pattern-growth based algorithms, DIMASP-C and DIMASP-D, to **Di**scover all the **Ma**ximal **S**equential **P**atterns in a document collection and in a single document respectively. The rest of this article is organized in four sections: (2) related work, (3) problem definition, (4) algorithms for mining frequent patterns in documents (in this section

experimental results are also given) and (5) concluding remarks.

2 Related work

Most of the algorithms for sequential pattern mining 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, most of the algorithms for sequential pattern mining are not efficient for mining a document database. Furthermore, most of the sequential pattern mining approaches assume a short alphabet; that is, the set of different items in the database. Thus, the characteristics of textual patterns make the problem intractable for most of the *a priori*-like candidate-generation-and-test approaches. For example, if the longest MSP has a length of 100 items then GSP [Srikant et al., 1996] 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 another one. Then a candidate sequence is pruned if it is non-frequent. Even though, GSP reduces the number of candidate sequences, it still is inefficient for mining long sequences.

As related work, we can mention those pattern-growth algorithms that speed up the sequential pattern mining [Jiawei Han et al. 2000; Antunes et al. 2003; Jian Pei et al. 2004; Lin et al. 2005] when there are long sequences. According to the empirical performance evaluations of pattern-growth algorithms like PrefixSpan [Jian Pei et al. 2004], GenPrefixSpan [Antunes et al. 2003], cSPADE [Zaki 2000], and DELISP[Lin et al. 2005], they outperform GSP specially when the database contains long sequences, therefore in this paper we will use them in our experiments. The basic idea in these algorithms is to avoid the cost of the 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, instead of only finding the MSP. Furthermore, none of them is incremental.

In this paper, we present two pattern-growth based algorithms, DIMASP-C and DIMASP-D, to **Di**scover all the **Ma**ximal **S**equential **P**atterns in a document collection and in a single document respectively. First, DIMASP algorithms build a novel data structure from the document database which is relatively easy to extract. Once DIMASP algorithms have built the data structure, they can discover all the MSP according to a threshold specified by the user.

In contrast with PrefixSpan, GenPrefixSpan and DELISP; if the user specify a new threshold our algorithms avoid rebuilding the data structure for mining with the new threshold. In addition, when the document database is increased, DIMASP algorithms update the last discovered MSP by processing only the new added documents.

3 Problem definition

The problem of finding patterns in documents can be formulated following the same idea as in transactional databases, i.e., assuming that each document of the collection is a transaction in the database, in this way, a sequence of items in a document will be a pattern in the collection if it appears in a certain number of documents. We have denominated to this formulation as the problem of finding all the maximal sequential patterns in a document collection.

3.1 Finding all the MSP in a document collection

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 $i\geq 1$, such that $p_1=s_i$, $p_2=s_{i+1}$, $p_3=s_{i+2},...,p_n=s_{i+(n-1)}$. The *frequency* of a sequence S, denoted by S_f or $<s_1,s_2,...,s_n>_f$, is the number of documents in the collection where S is a subsequence. A sequence S is *β-frequent* in the collection if $S_f \geq \beta$, a β-frequent sequence is also called a *sequential pattern in the document collection*. A sequential pattern S is *maximal* if S is not a subsequence of any other sequential pattern in the collection.

Given a document collection, the problem consists in finding all the maximal sequential patterns in the document collection.

Another formulation of the problem is finding patterns in a single document. At first glance, this problem could be solved by dividing the document into sections or paragraphs, and by applying algorithms for finding all the MSP in a document collection. However, the result would depend on the way the document was divided.

In addition, a sequence of items will be a pattern in the document if it appears in many sections or paragraphs without taking account the number of times the sequence appears inside each section or paragraph. This situation makes the problem different, therefore we will consider that a sequence of items in a document will be a pattern if it is frequent or appears many times inside the whole document. We have denominated to this formulation as the problem of finding all the maximal sequential patterns in a single document.

3.2 Finding all the MSP in a single document

Following the notation used in the section 3.1. Let $X \subseteq S$ and $Y \subseteq S$ then X and Y are *mutually excluded sequences* if X and Y do not share items i.e., if ($x_n=s_i$ and $y_1=s_j$) or ($y_n=s_i$ and $x_1=s_j$) then $i<j$. A sequence S is *β-frequent* in a document T, if it is contained at least β times in T in a mutually excluded way. *A β-frequent* sequence is also called a *sequential pattern in a document*. A sequential pattern S is *maximal* if S is not a subsequence of any other sequential pattern in the document.

Given a document, the problem consists in finding all the maximal sequential patterns in the document.

4 Algorithms for mining sequential patterns in documents

In this section, two algorithms one for mining maximal sequential patterns in a document collection (DIMASP-C) and another for mining maximal sequential patterns in a single document (DIMASP-D), are introduced. Both of them build a data structure containing all the different pairs of contiguous words in the document or in the collection and the relations among them. Then the maximal sequential patterns are searched into this structure, following the pattern-growth strategy.

4.1 DIMASP-C

The basic idea of DIMASP-C consists in finding all the sequential patterns in a data structure, which is built from the document database (DDB). The data structure stores all the different pairs of contiguous words that appear in the documents, without losing their sequential order. Given a threshold β specified by the user, DIMASP-C reviews if a pair is β-frequent. In this case, DIMASP-C grows the sequence in order to determine all the possible maximal sequential patterns containing such pair as a 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 the stored MSP which are subsequence of the new PMSP must be deleted. The proposed algorithm is composed of three steps described as follows:

In the first step, for each different word (item) in the DDB, DIMASP-C assigns an integer number as identifier. Also, for each identifier, the frequency is stored, i.e., the number of documents where it appears. These identifiers are used in the algorithm instead of the words. Table 1 shows an example for a DDB containing 4 documents.

In the second step (Fig. 1), DIMASP-C 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 an *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 stores a document identifier ($\delta.Id$), an *index* ($\delta.Index$) of the cell where the pair appears in the array, a link ($\delta.NextDoc$) to maintain the list Δ and a link ($\delta.NextNode$) to preserve the sequential order of the pairs with respect to the document, where they appear. 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 $<w_i,w_{i+1}>$ does not appear in the *array,* it is added, and its *index* is gotten. 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 $\delta.Id$, *index* as $\delta.index$, $\delta.NextDoc$ is linked to the first node of the list Δ and $\delta.NextNode$ is linked to the next node δ corresponding to $<w_{i+1},w_{i+2}>$ of the document D_J. If the document identifier ($\delta.Id$) is new in the list Δ, then the frequency of the cell (C_f) is increased. In Fig. 2 the data structure built for the document database of table 1 is shown.

Table 1: Example of a document database and its identifier representation

D_J	**Document database**
1	From George Washington to George W. Bush are 43 Presidents
2	Washington is the capital of the United States
3	George Washington was the first President of the United States
4	*the President of the United States is George W. Bush*
	Document database (words by integer identifiers)
1	<1,2,3,4,2,5,6,7,8,9>
2	<3,10,11,12,13,11,14,15>
3	<2,3,16,11,17,18,13,11,14,15>
4	<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**
δ_i ←*Create a new* ***Pair δ***
δ_i.Id ← *J //Assign the document identifier to the node δ*
index←Array[$\langle w_i, w_{i+1} \rangle$] *//Get the index of the cell*
δ_i.index ← *index //Assign the index to the node δ*
α ← *Get the first node of the list Δ*
If δ_i.Id ≠ α.Id **then** *the document identifier is new to the list Δ*
Increment C_f *//increment the frequency*
δ_i.NextDoc ← α *//link the node α at the beginning of list Δ*
List *Δ*← *Add* δ_i *as the first node //link it at the beginning*
δ_{i-1}.NextNode ← δ_i *//do not lose the sequential order*
end-if
end-for

Figure 1: Steps 2 and 2.1 of DIMASP-C.

In the last step (Fig. 3), given a threshold β, DIMASP-C uses the constructed structure for mining all the maximal sequential patterns in the collection. For each pair of words stored in the structure, DIMASP-C verifies if this

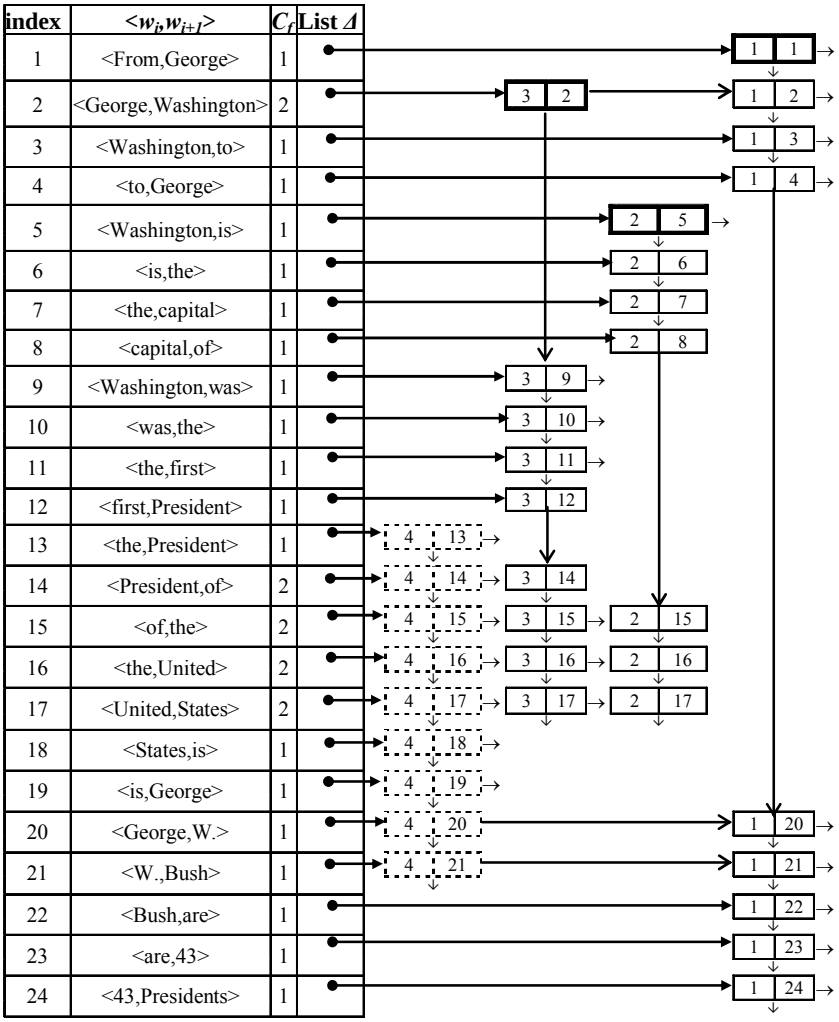


Figure 2: Data structure built by DIMASP-C for the database of the table 1.

pair is β-frequent, in such case DIMASP-C, based on the structure, grows the pattern while its frequency (the number of documents where the pattern can grow) remains greater or equal than β. When a pattern cannot grow, it is a possible maximal sequential pattern (PMSP), and it is used to update the final maximal sequential pattern set. Since DIMASP-C starts finding 3-MSP or longer, then at the end, all the β-frequent pairs that were not used for any PMSP and all the β-frequent words that were not used for any β-frequent pair are added as maximal sequential patterns.

In order to be efficient, it is needed to reduce the number of comparisons when a PMSP is added to the MSP set (Fig. 4). For such reason, a k-MSP is stored according to its length k, it means, there is a k-MSP set for each k. 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. When a PMSP is added, all its subsequences are eliminated.

For avoiding repeating all the work for discovering all the MSP when new documents are added to the database, DIMASP-C only preprocesses the part corresponding to these new documents. First the identifiers of these new documents are defined in step 1, then DIMASP-C would only use the step 2.1 (Fig. 1) to add them to the data structure. Finally, the step 3.1 (Fig. 3) is applied on the new documents using the old MSP set, to discover the new MSP set, for example, Fig. 2 shows with dotted lines the new part of the data structure when D_4 of table 1 is added as a new document. This strategy works only if the same β is used, however for a different β only the discovery step (step 3, Fig. 3) must be applied, without rebuilding the data structure.

The experiments were done using the well-known reuters-21578[1] document collection. After pruning 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 was 63 words. In all the experiments the first 5000, 10000, 15000 and 20000 documents were used. DIMASP-C was compared against GSP [Srikant et al.,

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

1996], an apriori-like candidate-generation-and-test algorithm, and cSPADE [Zaki 2000], GenPrefixSpan [Antunes et al. 2003] and DELISP [Lin et al. 2005], three pattern-growth algorithms. Excepting for GSP, the original programs provided by the authors were used. All the experiments with DIMASP-C were done in a computer with an Intel Pentium 4 running at 3 GHz with 1GB RAM.

Step 3: Algorithm to find all MSP
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 MSP w.r.t. the document** (D_J)
Step 3.1: Algorithm to find all MSP 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 [δ_i.index].*frequency* $\geq \beta$ **then**
PMSP←Array[δ_i.index]. $\langle w_i, w_{i+1}\rangle$ *//the initial pair*
Δ' ←Copy the rest of the *list of* Δ beginning from δ_i.NextDoc
Δ'_f ← Number of different documents in Δ'
$\delta'_i \leftarrow \delta_i$
While $\Delta'_f \geq \beta$ **do** *the growth the PMSP*
Δ'' ←Array[δ'_{i+1}.index].*list*
$\Delta' \leftarrow \Delta' \,\&\, \Delta''$ *i.e.* $\{\alpha \in \Delta' \mid (\alpha.\text{index} = \delta'_{i+1}) \wedge (\delta'_i.\text{NextNode} = \alpha)\}$
Δ'_f ←Number of different documents in Δ'
If $\Delta'_f \geq \beta$ **then** *to grow the PMSP*
Array [δ'_{i+1}.index].*mark* ← "used"
PMSP ← PMSP + Array [δ'_{i+1}.index]. $\langle w_{i+1}\rangle$
$\delta'_i \leftarrow \delta'_{i+1}$ *i.e.* δ'_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-MSP*
If $C_f \geq \beta$ **and** *C.mark* = "not used" **then** *add it as 2-MSP*
2-MSP set ← add $C.\langle w_i, w_{i+1}\rangle$

Figure 2: Step 3 of DIMASP-C.

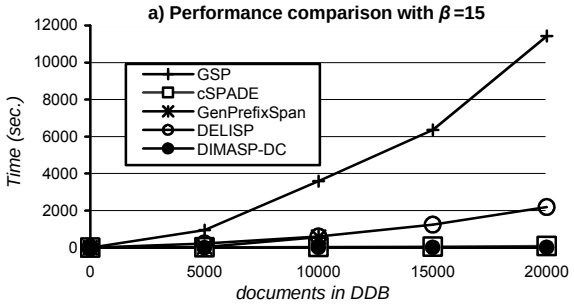
Step 3.1.1: Algorithm to add a PMSP to the MSP set
Input: A *k*-PMSP, MSP set **Output:** MSP set
If (*k*-PMSP ∈ *k*-MSP set) **or** (*k*-PMSP *is subsequence of some longer* k-MSP) **then** *// do not add anything*
return MSP set
else *//add as a MSP*
k-MSP set ← add *k*-PMSP
{***delete*** S ∈ MSP set | S ⊆ *k*-PMSP }
return MSP set

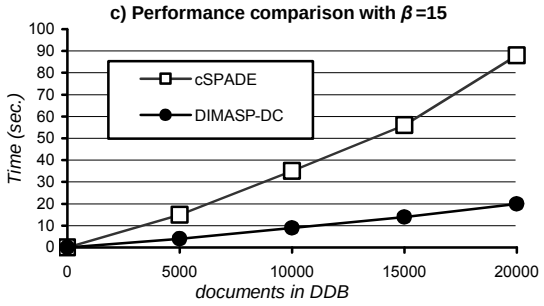
Figure 3: Algorithm to add a PMSP to the MSP set.

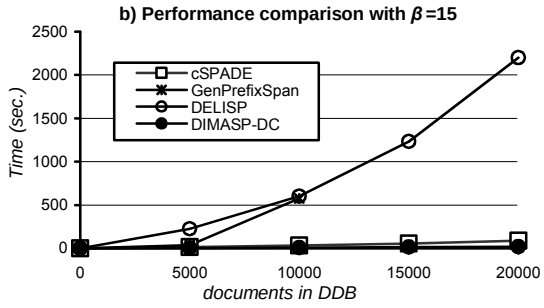
4.1.1 Experiments with DIMASP-C

In Fig. 5a the performance comparison of DIMASP-C (with all the steps), cSPADE, GenPrefixSpan, DELISP and GSP algorithms with β=15 is shown. Fig. 5b shows the same comparison of Fig. 5a but the worst algorithm (GSP) was eliminated, here it is possible to see that DELISP is not as good as it seems to be in Fig. 5a. In this experiment GenPrefixSpan had memory problems; therefore it was only tested with the first 5000 and 10000 documents. Fig. 5c compares only DIMASP-C against the fastest algorithm cSPADE with respect to β=15. Fig. 5d draws a linear scalability of DIMASP-C with respect to β=15. An additional experiment with the lowest β=2 was performed, in this experiment DIMASP-C found a MSP of length 398 because there is a duplicated document in the collection, Fig. 5e shows these results.

In order to evaluate the incremental scalability of DIMASP-C, 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 MSP. Fig. 5g shows the distribution of the MSP for β=15 according to their length. Finally, Fig. 5h shows the number of MSP when β=1% of the documents in the collection was used.







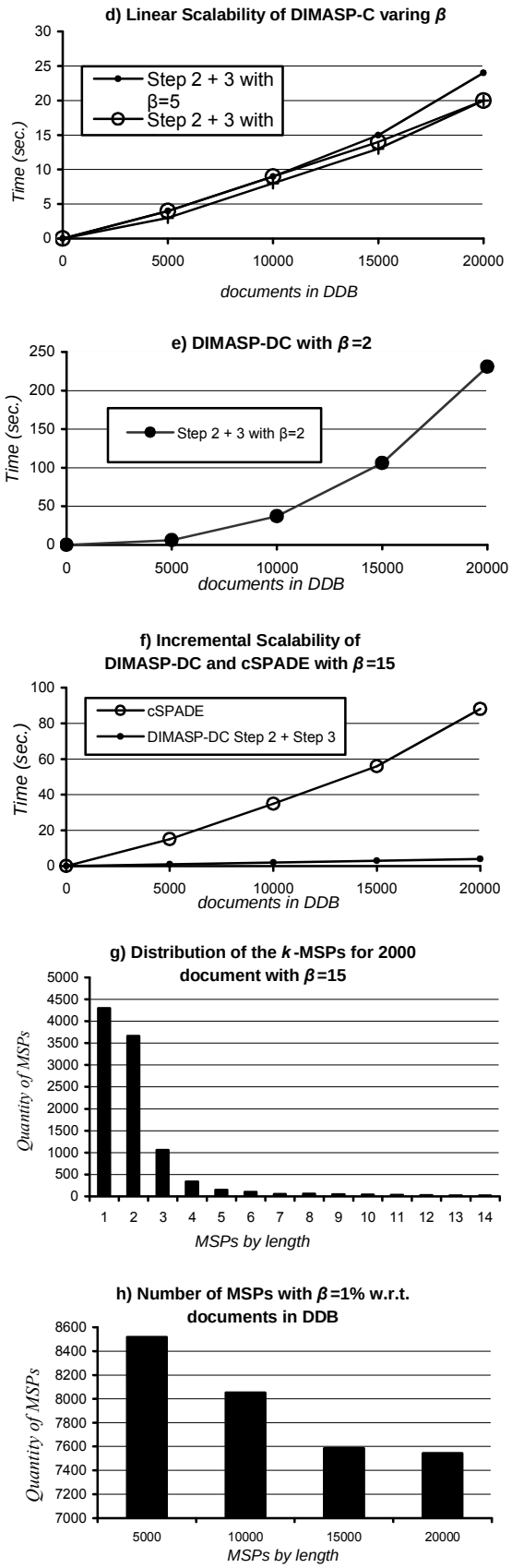


Figure 4: Performance results using Reuters-2157 collection.

4.2 DIMASP-D

Similar to DIMASP-C, the idea of DIMASP-D consists in finding all the sequential patterns in a data structure, which is built, in this case, from the single document to be analyzed. This structure stores all the different pairs of contiguous words that appear in the document, without losing their sequential order. Given a threshold β specified by the user, DIMASP-D reviews if a pair is β-frequent. In this case, DIMASP-D grows the pattern in order to determine all the possible maximal sequential patterns containing such pair as a prefix. The proposed algorithm has three steps described as follows:

In the first step, the algorithm assigns an *id* for each different word in the document.

The second step (fig. 6) consists in building the data structure. DIMASP-D will construct a data structure similar to the structure used for the document collection, but in this case containing only a single document. Since only one document is stored in the structure, the document index is not needed, instead of it, the position of the pair inside the document is stored, as it is shown in Fig. 7, this position is used to avoid overlapping among the instances of a maximal sequential pattern that appear into the document.

Input: A document T **Output:** The data structure
For *all the pairs* $[t_i,t_{i+1}] \in T$ **do**
// if $[t_i,t_{i+1}]$ it is not in *Array*, add it
PositionNode.Pos ←*index* ← *array* $[t_i,t_{i+1}]$;
Array[index].Positions ←**New** *PositionNode*
Array[index].Freq ← *array[index].Freq*+ 1
Array[LastIndex].Positions.NextIndex←*index;*
Array[LastIndex].Positions.NextPos←*PositionNode;*
LastIndex ← *index*;
End-for

Figure 5: Step 2 of DIMASP-D.

In the last step (Fig. 8), DIMASP-D finds all the maximal sequential patterns in similar way as DIMASP-C, but now the β-frequency is verified inside the document, counting how many times a pattern appears without overlapping.

4.2.1 Experiments with DIMASP-D

For the experiments, we chose from the collection Alex[2] the document "Autobiography" by Thomas Jefferson with around 243,115 chars corresponding to: 31,517 words (approx. 100 pages); and the document "LETTERS" by Thomas Jefferson with around 1,812,428 chars and 241,735 words (approx. 800 pages). In both documents the stop words were not removed and only the numbers and punctuation symbols were omitted. In order to show the behavior of the processing time against the number of words in the document, we computed the MSP using DIMASP-D with the minimum threshold value, β=2.

[2] Public domain documents from American and English literature as well as Western philosophy. http://www.infomotions.com/alex/

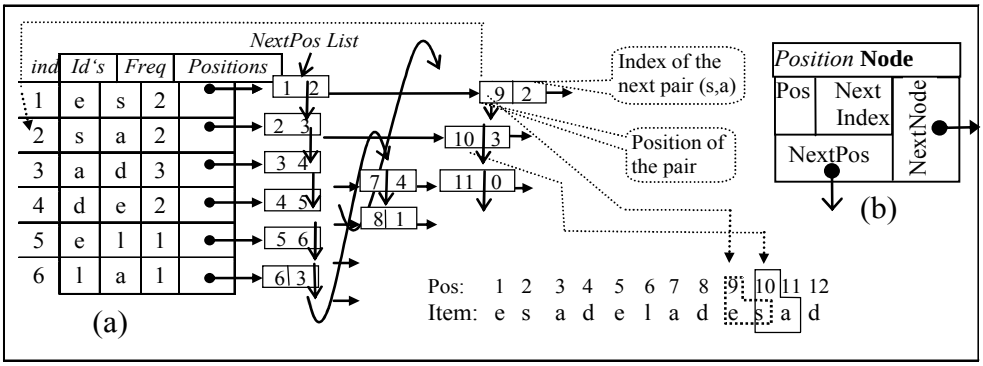
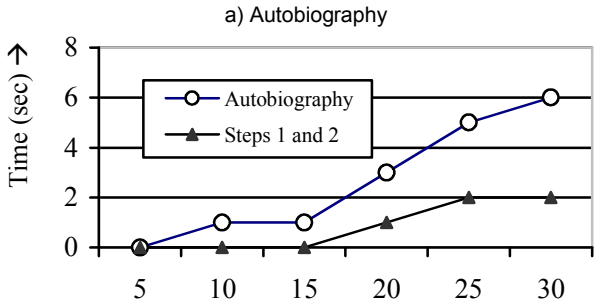


Figure 6: a) Data structure built by DIMASP-D for the text: "esadeladesad"
b) Node for positions list

Input: Data structure *from phase 2* **Output:** MSP list
Actual←1 //First element of NextPos List
while *Actual* ≠ 0 **do**
if *Array[Actual].Frequency* ≥ *β*, **then**
temporal ← Copy (*Array[Actual].Positions)*
PMS ← *Array[Actual].Id*$_1$ + *Array [Actual].Id*$_2$
aux ← *Array [Actual].NextIndex*
while aux ≠ 0 **do** //*expand the 2-sequence*
temporal ← **Match**((*temporal.Pos* + *1)* **AND** (*Array[aux].Positions.Pos)*
If | *temporal* | ≥ *β*, **then**
if *aux* = *Array*, **then** there is a cycle,
PMSP ← **Cycle(** *β*, *temporal*, *Array*, *Actual, aux* **)**
If the *PMSP* cannot grow **then** exit from the while
else *PMSP* ← *PMSP* + *Array[aux].Id*$_2$
aux ← *Array[Actual].NextIndex*
end-while
delete all the MSP ⊆ *PMSP*
if (*PMSP* ⊄ MSP) **then** MSP←Add(*PMSP*)
Actual ← *Array[Actual].NextIndex*
End-while

Figure 7: Step 3 of DIMASP-D.

Each graph in Fig. 9 corresponds to one document, processing different quantities of words. In the fig. 9a, we started with 5,000 words and used an increment of 5,000, in order to see how the processing time grows when the number of processed words is increased in the same document. In the fig. 9b, an increment of 40,000 words was used in order to see how the processing time grows for a big document. By the way, in both graphs the time for steps (1 and 2) is shown.



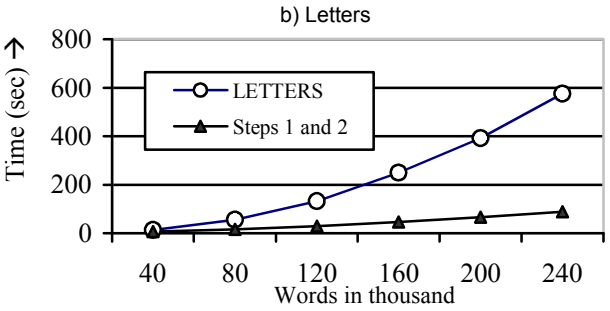


Figure 8: Processing time for: a) "Autobiography" and, b) "LETTERS".

For the same documents, the whole document was processed to find all the MSP, in order to appreciate (Fig. 10) how the performance of our algorithm is affected for different values of the threshold *β*. Fig. 10a shows the time in seconds for "Autobiography" and Fig. 10b for "LETTERS".

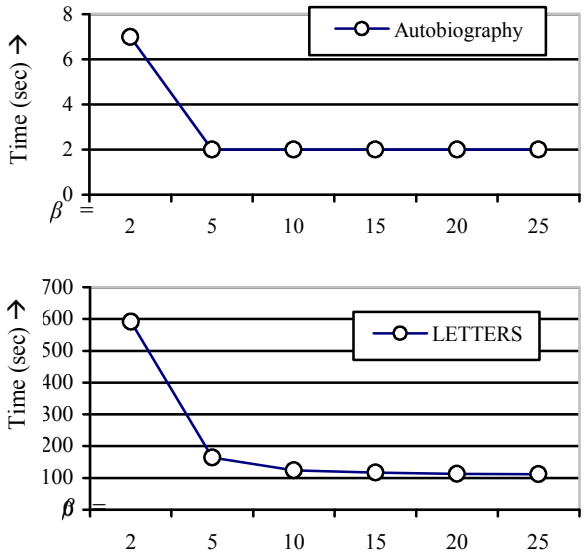


Figure 10: Time performance for different values of β for: a) "Autobiography" and b) "LETTERS".

Furthermore, we have included in Fig. 11 an analysis of the number of MSP obtained from the same documents for different values for β.

Additionally to these experiments, we processed the biggest document from the collection Alex, "An Inquiry into the nature …" by Adam Smith with 2,266,784 chars corresponding to 306,156 words (approx. 1000 pages) with β=2, all MSP were obtained in 1,223 seconds (approx. 20 min). All the experiments with DIMASP-D were done in a computer with an Intel Centrino Duo processor running at 1.6 GHz with 1GB RAM.

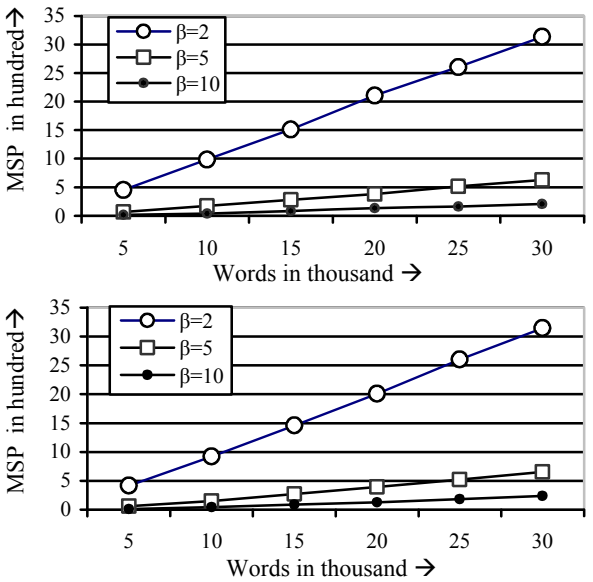


Figure 11: Amount of MSP generated for different values of β for a) "Autobiography" and b) "LETTERS".

5 Concluding remarks

In this work, we have introduced two new algorithms for mining maximal sequential patterns into a document collection (DIMASP-C) and into a single document (DIMASP-D).

According to our experiments, DIMASP-C is faster that all the previous algorithms. In addition, our algorithm allows processing the document collection in an incremental way; therefore if some documents are added to the collection, DIMASP-C only needs to process the new documents, which allows updating the maximal sequential patterns much faster than mining them over the whole modified collection by applying any of the previous algorithms.

DIMASP-D is a first approach for mining maximal sequential patterns into a single document, which allows processing large documents in a short time.

Since our proposed algorithms were designed for processing textual databases, they are faster than those proposed for transactional databases, therefore our algorithms are more suitable to apply maximal sequential patterns for solving more complex problems and applications in text mining, for example: question answering [Denicia-Carral et al. 2006; Juárez-González et al. 2007; Aceves-Pérez et al. 2007], authorship attribution [Coyotl-Morales et al. 2006], automatic text summarization [Ledeneva et al. 2008], document clustering [Hernandez-Reyes et al. 2006], and extraction of hyponyms [Ortega-Mendoza et al. 2007], among others.

As future work, we are going to adapt DIMASP-C and DIMASP-D for mining Maximal Sequential Patterns, allowing a bounded gap between words.

References

[1] Aceves-Pérez Rita Marina, Montes-y-Gómez Manuel, Villaseñor Pineda Luis, "Enhancing Cross-Language Question Answering by Combining Multiple Question Translations", *8th Intelligent Text Processing and Computational Linguistics (CICLing'2007), LNCS 4394*, Springer-Verlag, 2007, pp. 485–493.

[2] Agrawal R, Imielinski T, Swami A, "Mining association rules between sets of items in large databases", *Proceedings of the 1993ACM-SIGMOD international conference on management of data (SIGMOD'93),* Washington, DC, 1993, pp 207–216.

[3] 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 MLDM2003 LNCS 2734*, 2003, pp. 239–251.

[4] Coyotl-Morales Rosa Maria, Villaseñor-Pineda Luis, Montes y Gómez Manuel, Rosso Paolo, "Authorship Attribution Using Word Sequences", *11th Iberoamerican Congress on Pattern Recognition (CIARP'2006), LNCS 4225*, Springer Verlag, 2006, pp. 844–853.

[5] Denicia-Carral Claudia, Montes-y-Gómez Manuel, Villaseñor-Pineda Luis, García Hernández René, "A Text Mining Approach for DefinitionQuestion Answering", *5th International Conference on NLP (Fintal 2006), LNAI 4139*, Springer-Verlag, 2006, pp. 76–86.

[6] Hernandez-Reyes E, Garcia-Hernandez RA, Carrasco-Ochoa JA, J. Fco. Martínez-Trinidad, "Document clustering based on maximal frequent sequences", *5th International Conference on NLP (Fintal 2006), LNAI 4139*, Springer-Verlag, 2006, pp. 257-267.

[7] Jian Pei, Jiawei Han, et. al. "Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach", *IEEE Transactions on Knowledge and Data Engineering*, 16(10), 2004, pp. 1424–1440.

[8] Jiawei Han and Micheline Kamber, "*Data Mining: Concepts and Techniques*", Morgan Kaufmann Publishers, 2000.

[9] Jiawei Han, Hong Cheng, Dong Xin, Xifeng Yan, "Frequent pattern mining: current status and future directions", *Data Min Knowl Disc* 15, 2007, pp. 55–86.

[10] Juárez-González Antonio, Téllez-Valero Alberto, Delicia-Carral Claudia, Montes-y-Gómez Manuel and Villaseñor-Pineda Luis, "Using Machine

Learning and Text Mining in Question Answering", *7th Workshop of the Cross-Language Evaluation Forum, CLEF 2006, LNCS 4730*, Springer-Verlag, 2007, pp. 415–423.

[11] Ledeneva Yulia, Gelbukh Alexander, and García-Hernández René Arnulfo, "Terms Derived from Frequent Sequences for Extractive Text Summarization", *LNCS 4919*, Springer-Verlag, 2008, pp. 593–604.

[12] Lin, M. Y., and S. Y. Lee, "Efficient Mining of Sequential Patterns with Time Constraints by Delimited Pattern-Growth", *Knowledge and Information Systems*, 7(4), 2005, pp. 499-514.

[13] Ortega-Mendoza Rosa M., Villaseñor-Pineda Luis and Montes-y-Gómez Manuel, "Using Lexical Patterns for Extracting Hyponyms from the Web", *Mexican International Conference on Artificial Intelligence MICAI 2007, LNAI 4827*, Springer-Verlag, 2007, pp. 904-911.

[14] Srikant, R., and Agrawal, R., "Mining sequential patterns: Generalizations and performance improvements", *5th Intl. Conf. Extending Database Discovery and Data Mining, LNCS 1057*, Springer-Verlag, 1996, pp. 3–17.

[15] Zaki, Mohammed, "SPADE: An Efficient Algorithm for Mining Frequent Sequences", *Machine Learning*, Kluwer Academic Publishers, 42, 2000, pp. 31–60.