Recognition of logical units in log files

Hassan Saneifar, Stéphane Bonniol, Pascal Poncelet, Mathieu Roche

Recognition of Logical Units in Log Files

Hassan Saneifar[1,2], Stéphane Bonniol[3],
Pascal Poncelet[1], Mathieu Roche[1,2]
[1] LIRMM - CNRS - University Montpellier 2,
161 rue Ada, 34095 Montpellier, France
{Firstname.Lastname}@lirmm.fr
[2] UMR TETIS, Cirad, Irstea, AgroParisTech
500, rue J.F. Breton
34093 Montpellier Cedex 5, France
{Firstname.Lastname}@cirad.fr
[3] Satin Technologies,
MIBI, 672 rue du Mas de Verchant, 34000 Montpellier, France
{Firstname.Lastname}@satin-tech.com

Abstract

With the development of new technologies more and more information is stored in log files. Analyzing such logs can be very useful for the decision maker. One of the probably best known example is the Web log file analysis where lots of efficient tools have been proposed to extract the top-k accessed pages, the best users or even the patterns describing the behaviors of users on a Web site. These tools take advantages of the well-formed structures of the data. Unfortunately, logs files from the industrial world have very heterogeneous complex structures (e.g., tables, lists, data blocks). For experts, analyzing logs to find messages helping to better understand causes of a failure, if a problem have already occurred in the past or even knowing the main consequences of a failure is a hard, tedious, time-consuming and error-prone task. There is thus a need for new tools helping the experts to easily recognize the appropriate part in logs.

Passage retrieval methods have proved to be very useful for extracting relevant parts in documents. In this paper we propose a new approach for automatically split logs files into relevant segments based on their logical units. We characterize the complex logical units found in logs according to their *syntactic characteristics*. We also introduce the notion of *generalized vs-grams* which is used to automatically extract the syntactic characteristics of special structures found in log files. Conducted experiments are performed on real datasets from the industrial world to demonstrate the efficiency of our proposal on the recognition of complex logical units.

Keywords: Log files, Text segmentation, Logical units, Generalized vs-grams

1 Introduction

With the development of new technologies more and more information is stored in log files. Analyzing such logs can be very useful for the decision maker [7, 15]. Probably one of the most well known example is the Web log analysis which aims at extracting information or knowledge from access log files. Actually these logs store information about connected users on a Web site and have a well-formed structure (e.g., IE, apache, Facebook, and so forth). Generally the volume of stored data is very high, e.g. Facebook has to manage more than 25 terabytes of data per day, that is equivalent of about 1,000 times the volume of mail delivered daily by the U.S. Postal Service[1].

Today very efficient tools exist to know the top-k users, the most downloaded pages and even the behaviors of users on a site. All these approaches take advantages on the well-formed structure but unfortunately, in industrial domains, lots of logs generated are very heterogeneous. For instance electricity company logs have to store information such as date, time, list of equipment affected, length of outage, and weather conditions. As they are usually fed by very different systems, these kind of logs can contain very complex data (e.g. tables, texts, numerical and symbolic data, and so on).

Today, specially in industrial domains, finding information in log files is crucial since the number of generated logs is dramatically increasing. For instance, for experts, it is important to determine which messages correspond to real problems in order to answer, for instance, to the following questions: *what can be the main causes of a failure?*, *did this problem already happened before?*, *what are the main consequences?*. Unfortunately analyzing these logs to get the relevant information, i.e. information that can represent answers to the questions of the domain, is a hard, tedious, time-consuming and error-prone task. Thus, experts need solutions for an automatic and accurate extraction of information from these logs.

Question answering systems (QAS) which is one type of information retrieval (IR) system that attempts to find exact answers to user's questions expressed in natural language has been revealed quite efficient (e.g., [1, 9, 18]). More precisely, passage retrieval (PR) is a major constituent of QAS which aims at retrieving relevant passages in documents which contain answers to questions. Usually, in this context, a passage is a fixed-length sequence of words which can begin and end anywhere in a document. Despite the advantages of passage-level information access, there is no general agreement about how one should

[1]source: http://www.datacenterknowledge.com/

define those passages in order to obtain an optimum performance [12]. Actually, the different PR systems mainly differ by the way they consider boundaries of a passage and how they evaluate its relevancy. In most of passage retrieval methods, we can distinguish two main phases: (1) *passage segmentation*, (2) *passage ranking*. Passage segmentation is the task of determining text segments in documents which are considered as candidate passages. Here the main issues considered are the following: *how to define a passage boundaries?* and *how to recognize them in a corpus?* On the other part passage ranking assesses the relevancy of passages according to a given query.

Here, we focus on the issue of *passage segmentation* for a particular kind of complex data: *log files generated by Electronic Design Automation (EDA)* tools. These logs, from the industrial world, represent a major source of information on designs, products or even causes of seen problems. They are used to answer specialized questions in the field of micro-electronics where, in this domain, to ensure the design quality, there are some quality check rules that must be checked. These rules are usually expressed in the form of natural language questions (e.g., "*Capture the total fixed cell STD*" or "*Captures the maximum Resistance value*"). Verification of these rules is principally performed, with an expert, by analyzing the generated log files. In the case of large designs where tools may generate megabytes or gigabytes of log files each day, the problem is to wade through all of this data *to locate the critical information* that we need to analyze the quality of rules.

In this paper we propose a new segmentation approach and characterize the complex logical units found in the log files according to *their syntactic characteristics*.

Recognition of these logical units is the main core of our segmentation approach. Within our approach, we also present the original notion of *generalized vs-grams* which is used to automatically extract the syntactic characteristics of special structures found in log files.

The paper is organized as follows. A detailed study on related work and their relevance in the context of log files is discussed in Section 2. Then, in Section 3, we present the main phases of our segmentation approach. Sections 4 and 5 are devoted to the development of different phases of our approach notably the characterization of logical units. We also present the original notion of "generalized vs-grams" which is used to characterize the logical units in Section 5. Results of experiments are presented in Section 6. A conclusion is addressed in Section 8.

2 Related Work

We detail here text segmentation methodologies which are used to split a document into candidate passages. According to [1] and [9], there are usually three

types of passages: *semantic* (thematic), *window* passages, and *discourse*. Semantic segmentation consists in identifying various topics conveyed by the text, to segment it into homogeneous units forming thematic blocks [19, 26]. Then, it is also possible to segment documents based on a sliding phrase or a line window which is called window-based segmentation. In the case of "discourse passages", the segmentation is carried out based on logical divisions or in other words, the discourse units in documents. Indeed, the documents often have logical (discourse) units like sentences, paragraphs, or sections. The logical components of documents can be regarded as passages (segments) [1, 9]. The choice of segmentation type depends largely on the objectives, application domain, and specially document characteristics.

Semantic segmentation. By Semantic segmentation a document is split into semantic pieces, according to the different topics in the document [9, 12, 20]. The semantic segmentation methods principally rely on the usage of word frequency, lexical co-occurrences, or lexico-semantic relations to identify subject changes in documents. For instance, TextTiling is a well-known semantic segmentation approach which uses word frequencies to recognize topic shifts [5]. In TextTiling, the main cues for identifying major subtopic shifts are patterns of lexical co-occurrence and distribution. Marti A. Hearst assumes in [5] that a set of lexical items is used during the course of a given subtopic discussion, and when that subtopic changes, a significant proportion of the vocabulary changes as well. TextTiling being a well-known semantic segmentation, we evaluate its application in the context of log files. This evaluation, presented in Section 7, gives a survey about the efficiency and relevance of semantic segmentation approaches in the context of textual data like log files.

The values of some measures like word repetition and redundant utterances are used to find the segmentation points [28]. For instance, [10] proposes to calculate a lexical cohesion profile which locates segment boundaries in a text. This approach relies on the similarity of words in a sentence. Word similarity, representing word cohesiveness, is calculated using a semantic network constructed from an English dictionary. There are also semantic segmentation methods calculating a semantic distance between terms. These methods often rely on external resources such as domain ontologies or semantic networks [11].

Studying main researches regarding semantic segmentation shows that topic change discovery mainly relies on the assumption of cohesion, which is a device for making connections between parts of the text [17]. As it is also mentioned in [17], we note that the current cohesion-based text segmentation approaches focus on either term iterations, term semantic relations, or both. These methods suppose that analysing term occurrences can guide to discover topic changes. Techniques based on semantic distance include relations between words that tend to co-occur in the same contexts, which have systematic and the non-systematic semantic relations [17].

In our domain, these methods suffer from some common points. We try to

explain below why these semantic segmentation methods are not suited to segment log files. First, the notion of subject change based on the lexical cohesion assumption is questionable in our context. Semantic segmentation methods use phrase or paragraph as the basic unit. Whereas log file, as explained above, are not natural language texts which means that notion of phrase or paragraph is not significant in this context. Furthermore, we show trough few examples that a change of co-occurrences or a change of term repetitions does not necessarily result in topic change in log files. This throws doubt on the relevance of assumptions based on term iteration or term occurrences which are used in above mentioned methods. As an instance, Fig. 1 demonstrate a fragment of a log file corresponding to a segment.

```
67 | ------ Design Statistics:
68 |
69 | Number of ports:           00
70 | Number of nets:            10
71 | Number of cells:           20
72 |
73 | Combinational area:        40
74 | Noncombinational area:     50
```

Figure 1: A fragment corresponding to a segment.

After a lexical co-occurrences and repetition analysis of this segment, we discover a co-occurrence change after line 72. Based on the above assumptions, we should consider lines 69 to 71 as a segment and lines 73 to 74 as another one which treat different topics thought the entire of this fragment should be considered as a single segment according to a domain expert. Indeed, the whole chunk from line 67 to 74 reports information about "design statistics". This issue is highlighted when we are dealing with tables or numerical data in log files. As another example, Fig. 2 illustrates a table in a log file.

```
76 | #patterns   #faults     #ATPG faults     test       process
77 | stored    detect/active    red/au/abort    coverage    CPU time
78 | -----     ---------      --------        ----       ----
79 | Begin determ.  ATPG: #uncollapsed_faults=2865, abort_limit=10
80 | 32        26155 2495         0/0/0        53.50%        0.08
81 | 50          2141 353         1/0/0       57.01%        0.11
82 | 77          219 129          4/2/9       57.37%        0.21
```

Figure 2: An example of a table in a log file.

In this table, a considerable vocabulary change is noticeable after line 79 as we have only numerical data. However, this change does not mean a topic

change. A table has to be always considered as a single segment in log files. Beside tables, we have many adjacent text chunks in log files which contain principally numerical data. Discovery topic changes in text chunks treating principally numerical data is not feasible by semantic segmentation methods which rely on vocabulary change or lexical cohesion.

Furthermore, in our situation, there are no resources (e.g., domain ontologies) to calculate the semantic correlations between words in the EDA context. Creation of such resources which enable to inference the semantic relations, is revealed time-consuming task, which also needs a lot of domain expert interventions. Thus, we are faced with some difficulties in our context in using the methods based on semantic network like ontologies. Finally, we seek methods quite independent of the vocabulary of log files which evolves over time (vocabulary changes) and also according to the tools.

Window Based Segmentation. Other segmentation methodology is based on the notion of sliding window [8]. As mentioned in [27], windows can be defined in terms of words [9] or in terms of sentences or paragraphs [29]. Window segmentation imposes a single model of text division. The fixed or variable size windows may be either non-overlapping or overlapping sliding windows [9, 13]. The window segmentation methods are also used along with passage scoring methods. In such a case, windows can start with a keyword of query and continue according to the presence of other keywords. In order to find the best segments, one slides the window or changes its size and calculates the relevance of new obtained segment based on the new position of the window.

Window models have the main advantage to be simpler to accomplish [12]. A drawback of these methods is the potential cutting of a relevant chunk of a document into several segments, whereas it should be seen as a single segment. This error, also noted in [1], can simply occur in the case of tables or data block in log files. Segmentation based on windows also ignores the structure of log files although log file structures contain relevant information to identify segments or extract information.

Discourse Passages. In the case of "discourse passages", the segmentation is carried out according to logical divisions or in other words, the logical units of documents. Indeed, texts are often composed of logical units like sentences, paragraphs, or sections. The logical (or *discourse*) units of documents can be regarded as candidate passages (segments) [1, 9, 22]. The segment definition is intuitive in this approach, because sentences should develop a single idea; paragraphs should address a single topic, and sections are the subject of a question [1, 9]. As mentioned in [6], in the last twenty years, the study of discourse has received continuous attention from the Natural Language Processing community. Discourse structure is fundamental to many text-based applications, such as Question Answering [24]. In discourse segmentation we can consider different granularities. Discourse segmentation for Information Retrieval or Question Answering is usually performed at the granularity level, distinguished in [14],

which consists on clause, sentence, paragraph, and section. The finer granularities are usually studied in Logical Parser studies to create logical structure of a document.

There are some solutions to identify classic logical units such as paragraphs in texts. We can also exploit the elements marking the logical units (i.e., logical divisions) such as "white lines" or "indentations" at the beginning of paragraphs. For instance, Deborah Schiffrin defines discourse markers in [23] as a set of linguistic expressions that brackets units of talk. There are also passage retrieval approaches which simply consider sentence logical unit as the base of segmentation. For example, a passage is defined as one or more adjacent sentences in the AskHERMES medical QA system [2]. A similar approach is also used in [4] which considers sentence as the base of segmentation.

As noted in [12], the discourse-based models look more effective since they are using the structure of the document itself. Similarly, Callan notes in [1] that discourse passages are expected to be the most effective, because discourse boundaries organize material by content. However, they also note several drawbacks and limits of discourse-based segmentation.

First, segmentation results could depend on the writing style of the document author [12]. Discourse passages require more consistency from writers than do semantic or window passages. If writers are sloppy or rushed then segmenting documents by textual discourse boundaries may be inappropriate [1]. Second, even though most documents are supplied with their structure, manual processing is required for those without it, thus making discourse passages impractical [9]. Third, logical unit markers (i.e., logical divisions) are not always available or ambiguous with other types of separators [27]. For example, *headers*, *list elements* or *table rows* might be separated in the same way as discourse related paragraphs (for example using an *empty line*). Problems with the classical discourse passage approaches often arise with special structures such as headers, lists, and tables which are easily mixed with other units such as proper paragraphs [27].

In our context, data have, for example, structures such as tables, data blocks, and specific strings *marking* the beginning of new information. Moreover, log files are generated by computing systems which means their structural organization is based on a defined grammar. Even though we do not have access to these grammars[2] or they vary from a log file type to another type, but it guarantees a consistency in writing log files. Therefore, the first and second mentioned drawbacks of discourse segmentation, regarding writing style and existence of document structure, do not exist in the context of log file. However,

[2]Although log files are semi-structured data and should conform to a grammar, acquisition or knowing of their generator grammar is challenging and in some situations irrelevant. According to some constraints like data confidentiality or software licensing, access to design tools is not always possible. Moreover, extracting language model or grammar of log files generated by a given tool requires sufficient amount of data (i.e., logs) generated in different configurations by that tool. Creation of such corpus is tedious, time/resource consuming and sometimes not feasible according to the available number of log files.

the third problem is emphasised in log files where conventional logical divisions are meaningless (e.g., paragraph indentions) or ambiguous (e.g., empty lines). In the same time, we deal with special structures such as headers, lists and tables in log files which can take different representation formats (e.g., different table types). These textual structures, more complex than conventional structures (classic logical units), are used to gather ideas and information. Thus, we consider these complex structures (which gather the correlated information) as logical units of log files.

We hence aim at proposing an approach to recognize special logical units in log files and overcome the above drawbacks notably ambiguity in logical divisions and issues in dealing with special structures such as headers and tables. Our approach enables to identify different complex kinds of logical units while they can take different forms or presentation formats. By means of our approach, we overcome the drawbacks of discourse-based segmentation in texts having special structures like tables, lists, or data blocks. In the following section, we describe how to discover the special logical divisions of log files and subsequently recognize their logical units.

3 Global Process of Logical Units Recognition

Since the logical structures of log files are different and more complex than those found in classical texts, we aim at identifying different kinds of discourse units in log files. To identify these logical units, we need to recognize different layouts and data presentation formats which are used to structure them. We hence look for syntactic cues that can be useful to reveal a separation of consecutive text chunks.

This means that we determine the *syntactic characteristics* of *logical units* in log files. They allow to distinguish the beginning of a logical unit (i.e., a logical division) from the other lines in the documents. For example, in classical texts, we can consider the "*indentation*" as one of the characteristics allowing to identify a paragraph. We call these syntactic characteristics *features*. We therefore characterize the logical units of log files by defining a number of "features". In order to perform the feature acquisition, we chose two different methods: (1) semi-automatic and (2) automatic. Fig. 3 shows the general outline of our approach of logical structure recognition.

In the semi-automatic way, we first define the features according to some heuristics based on an expert knowledge. Afterwards, we model the logical divisions based on the presence/absence of features.

In the case of the automatic method, we propose an original type of n-grams, called "*generalized vs-grams*". The features are determined automatically by extracting generalized vs-grams in a corpus without any need to expert knowledge. We will develop both methods of feature acquisition and the notion of generalized vs-grams respectively in Sections 4.1 and 4.2.

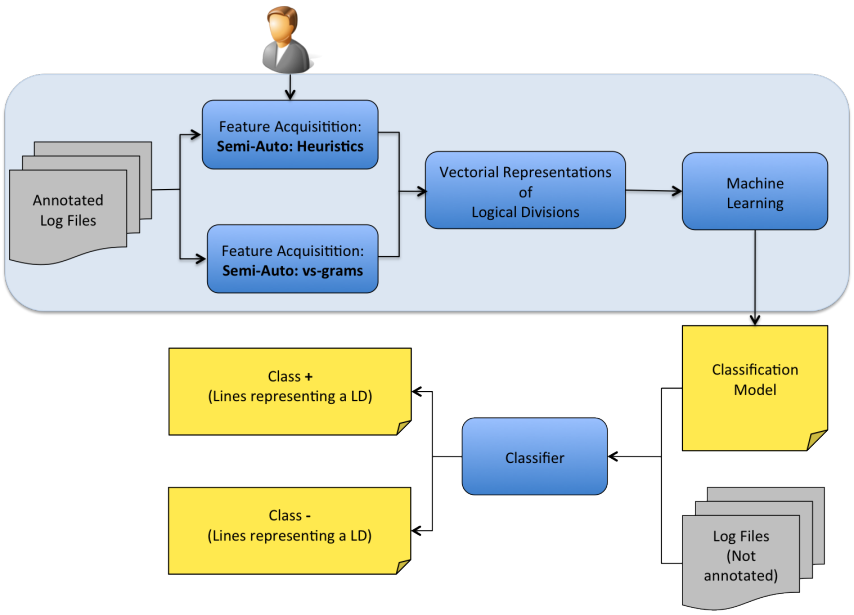

Figure 3: The global process of logical structure recognition.

Once all the features determined by one of these methods (automatic or semi-automatic), we represent the lines of the corpus by a binary vector in the vectorial space obtained by the set of features. For a given vector, the value of each element corresponds to the presence or absence of a feature. This representation allows us to build a training dataset based on the features. We use this dataset in a supervised classification system to obtain the rules (models) for the recognition of logical structures. The trained classification model is subsequently used in order to associate the lines of a new non-annotated corpus with a class: Class of lines representing a logical division (*positive*), and class of lines which are not associated with a logical division (*negative*). The learning phase is developed in Section 5.

4 Representation of Logical Divisions of Log Files

In this section, we present the steps to characterize and find the complex logical divisions: the feature acquisition and the classification model creation. We also introduce the original notion of generalized vs-grams, used in the automatic acquisition of features, which are the main contribution of our segmentation approach.

4.1 Heuristics Based Feature Acquisition

First, we assume that the identification of the beginning of a segment is sufficient to recognize a logical unit in a document. The end of a logical unit is determined by the start of the next one. Thus, we seek to identify *syntactic patterns* which exist in lines representing the beginning of a logical unit. These syntactic patterns allow to *differentiate* the beginning lines of logical unit (i.e., logical divisions) from other lines.

```
1
    set_ top i:/WORK/fifo
Setting top design to 'i:/WORK/fifo'
Status: Implementing inferred operators
```

Figure 4: A logical unit in a log file.

Fig. 4 shows an example of a logical unit in a log file. The highlighted line represents the beginning of the unit. In a simple way, we can characterize the beginning of this unit by a pattern as "`<abs-shift><string><:><string>`" [3] which means a line beginning with an absolute shift (indentation), followed by a string, then a ":", and finally followed by another string. We consider the *couple* of this pattern and its position around the beginning of the segment as a feature whose the presence helps to recognize this logical unit.

However, according to the domain characteristics and especially the heterogeneity that exists in such documents, we must characterize the beginning of segments more accurately. This means that it is necessary to characterize a logical unit by using a sufficient number of relevant features. To build this set of features, we identify syntactic patterns in a window of lines around the beginning of logical units. The window size being a parameter of our approach: "*nlp*" represents the number of lines before the beginning of the segment, and "*nls*" is the number of lines following the beginning of the segment. In the case of log files studied here, the values of these two parameters are fixed following an experimental protocol. The used values are presented in Section 6.

We detail below the constitution of features based on an example. Fig. 5 shows another fragment of a log file. The highlighted lines (lines 86 and 92) represent the beginning of two logical units (segments).

To simplify the example, we use a window size of five lines around the beginning of each segment (i.e., *nlp*=2 and *nls*=2). Regarding the first segment, we can identify the pattern "`<- - -><string><fin :>`" on the beginning line of the segment (line 86). We also have the pattern "`<string> <:> <string>`" on the both second lines before and after the beginning of the segment (i.e., lines 84 and 88). We also consider the empty line as a pattern. Thus, considering

[3] abs-shift: Absolute shift (indentation) at beginning of line.

```
84 | Total IO Pad Cell Area              : 1076208.64
85 |
86 | ------ Design Statistics:
87 |
88 | Number of Instances                 : 13628
89 | Number of Nets                      : 14293
90 | Maximum number of Pins in Net       : 531
91 |
92 | IO Port summary
93 | Number of Primary I/O Ports         : 388
94 | Number of Input Ports               : 259
```

Figure 5: Two logical units in a log file

the identified patterns and their locations, we obtain the following features for the first segment:

f_a(<---><string><fin :>, 0)
f_b(<emptyLine>, -1)
f_c(<emptyLine>, +1)
f_d(<string><:><string>, -2)
f_e(<string><:><string>, +2)

For each feature, the number after the pattern is the line number in the window. The zero corresponds to the beginning line of the segment. As an example, the f_a presents a feature consisting of the shown pattern, found on the beginning line of the segment (i.e., line 0 in the window around the beginning of the segment) which corresponds to the line 86 in the example. f_d also presents another feature consisting of the pattern found on second line before the beginning of the segment.

Regarding the second segment in Fig. 5, we can identify some of the features found around the first segment. For instance, on the second line before the beginning of the second segment (line 90), we found the pattern "<string><:><string>" which has been also found on the same position (second line before the beginning of the segment) around the first segment. Thus, we also note the same feature (i.e., f_d) for the second segment. Meanwhile, we also identify two new features (i.e., f_f and f_g) identified on lines 92 and 93. We note the below features around the beginning of the second segment.

f_f(<abs-shift><string>, 0)
f_b(<emptyLine>, -1)
f_g(<string><:><string>, +1)
f_d(<string><:><string>, -2)
f_e(<string><:><string>, +2)

Finally, by combining all the identified features (for both segments), we obtain a set of features. The following list represents this set of features obtained on the example of Fig. 5 = $\{f_a, f_b, f_c, f_d, f_e, f_f, f_g\}$

Then, by considering the feature set as a vector space, we represent both logical units in form of binary vectors. Therefore, the beginning of each logical unit of the example is represented by a binary vector as follows. Here, "1" means the presence of the feature and, "0" its absence.

$S_1 : \{1, 1, 1, 1, 1, 0, 0\}$ $S_1 : \{0, 1, 0, 1, 1, 1, 1\}$

To obtain the set of features, we established a corpus of different log files. The constitution of this corpus (presented in Section 6) is done in collaboration with a domain expert to ensure that all domain logical units exist in the corpus. The determined feature set contains 123 features in total. Once the feature set constituted, we characterize each line of the documents in the vector space defined by the feature set. Since the logical units in the log corpus are annotated by an expert, we obtain the vectors of positive instances (the lines corresponding to the beginning of logical units) and negative instances (the lines that do not match the beginning of logical units). Then, we use this positive and negative instances in a supervised learning system to obtain a model for the recognition of logical units (see Section 5).

4.2 Features Acquisition Using the Generalized VS-Grams

In the previous section, we have described a solution to build the feature set using syntactic patterns. Nevertheless, this method (see Section 4.1) requires expert knowledge to provide some heuristics and then define the patterns specific to them.

In the remainder of this work, we propose an automatic method that we use to create the set of features, without requiring human intervention. Thus, to create the set of features, we decided to use the n-grams to characterize the beginning of logical units in the log files. An n-gram is a set of n items in a sequence. In the field of NLP, an n-gram is a series of n items in a text where the items can be letters or words. N-grams are often used as features in the textual document classification tasks [25] to model the content and the sequence of words in a document.

But in our context, we are only interested in the structure of documents. That means we do *not* seek to identify the logical units according to their content (words or letters), *but* according to their textual structures (punctuation, symbols, layouts, etc.). This necessity led us to define and propose an *original kind of grams* that we call *generalized vs-grams*. Generalized vs-grams allow to model the textual structures (the layouts and the composition of letters and special characters) of a document while being insensitive to the contents of the latter. This enables to characterize the visual structure of a text document.

Generalized Vs-grams. We present here the concept of *generalized vs-grams* that we defined in the context of this work. We choose the term *vs-gram* as an abbreviation for "*Variable Size Grams*". To better understand the concept of

vs-grams, we first explain, via an example, the needs that led us to define vs-grams.

```
52  ****** Signal Coverages ******
53  CV_ INT[0:255]                    No No
54  ALG_ DATA_ INT [0:127]              No No
55  END_ KEY                          No No
...
62  **** MODULE INSTANCE ****
63  MODULE TB_ ECB_ VK_ DEC_ ITER.TOP_ LEVEL.CTRL
64  FILE /users/AES/src/controller_ iter.vhdl
```

Figure 6: logical unit in a log file.

Fig. 6 shows an extract from a log file. The highlighted lines (52 and 62) represent the beginning of two logical units. In this example, the beginning of the two logical units is characterized by a string preceded and followed by a series of the symbol "*". By extracting the fixed size n-grams (e.g., n = 3), we obtain the following tri-grams on the first segment[4]:

"`***`", "`␣si`", "`gna`", "`l␣c`", "`ove`", "`rag`", "`es␣`".

Similarly, we have the following tri-grams for the second segment.

"`***`", "`*␣m`", "`odu`", "`le␣`", "`ins`", "`tan`", "`ce␣`".

The extracted tri-grams show "the sequence of letters" in those two lines. However, the particular composition of special characters[5] and letters[6] which characterizes here the visual layout of these two logical units is hardly highlighted by the tri-grams. Indeed, the fixed size n-grams are not relevant when one focuses at layout and the composition of letters, symbols, and punctuation[7]. In order to describe the layout and the composition of letters and symbols in a line by means of a kind of feature, we define vs-grams where grams can have variable size.

We aim at defining a kind of feature which characterizes how letters, whitespaces, and symbols are located in a text line from the point of view of their positions. This makes it enable to describe the layout or visual structure of a line. Thus, we define vs-gram as a series of alphanumeric and non-alphanumeric characters whose boundaries are determined according to the type of seen characters. In order to determine the boundaries of a vs-gram, we define three conditions which make it enable to describe how the alphanumeric and non-

[4]The extracted grams are case-insensitive, as the letter case is not informative in this context.

[5]Non-alphanumerical character.

[6]Alphanumerical characters.

[7]In other words, how the alphanumerical and non-alphanumerical characters follow each other in a line.

alphanumeric characters follow each other in a line (i.e., visual structure of a line). Therefore, a vs-grams is a series of alphanumeric and non-alphanumeric characters, which is defined as follows:

- If the gram contains a series of alphanumeric characters, it ends with a non-alphanumeric character. The next gram begins with the non-alphanumeric character.
- If the gram starts with a series of non-alphanumeric characters, it ends with an alphanumeric character. The next gram begins with the alphanumeric character.
- if the seen character is a whitespace, it is systematically added to the current gram.

Taking the previous example (Fig. 6), we obtain the following grams for the first segment: "`******␣s`", "`signal␣coverage␣*`", "`******`"

Algorithm 1 presents how to extract the vs-grams. As shown in the algorithm, the extraction process consists in verification of three conditions: if the current character is a alphanumeric one, if it is a white space or finally it is non-alphanumeric character. Based on the type of the current char and that of previously seen char as well as the following one, we extract the vs-grams.

Contrary to the extracted 3-grams, the vs-grams show the composition model of the letters and special characters (here, "*") and their positions according to each other in this line. That is, these vs-grams express a composition of characters as a string of letters surrounded by symbols "*". This *pattern identified by vs-grams* marks the beginning of a logical unit in the log files.

The vs-grams are still sensitive to the content of texts. For example, here, the second extracted vs-gram presents a series of letters *comprising* the words "`signal`" and "`coverage`". However, the *essential knowledge* to take into account is the *presence of a string*. Similarly, the number of stars ("*") in the two other vs-grams, for example, is not informative. That is why we generalize vs-grams by replacing sequences of letters and special characters by some symbols representing their character type and a counter notion like "+"[8]. This means that we replace, for example, a string of alphanumeric characters with the symbol "`\w+`". The series of stars will be replaced by "`\*+`". Thus, in this example, we obtain the following generalized vs-grams: "`\*+␣s`", "`\w+␣\*+`", "`\*+`"

We finally obtain generalized vs-grams that express well the existence of a string of alphanumeric characters surrounded by a sequence of the "$*$" symbol.

[8] "+" means one repetition or more.

Algorithm 1: Extraction of vs-grams in a text file

Data: $Text(T) = \{ch_0, ch_1, ch_2, \ldots, ch_n \mid \text{n}=|T| \text{ and } ch_i \in \{ASCI\ chars\}$
Result: LIST(vsgrams) : list of extracted vs-grams

```
for i ← 0 to n do
    currentChar = char_i ∈ T;
    if currentChar is LetterOrDigit then
        if last char was not LetterOrDigit then
            Call AddGram();
        else
            gram ⟵ gram + currentChar;
    else if currentChar is Whitespace then
        gram ⟵ gram + currentChar;
    else if currentChar is SpecialChar then
        if last char was not LetterOrDigit then
            gram ⟵ gram + currentChar;
        else
            Call AddGram();
return LIST(vsgrams);
AddGram(){ gram ⟵ gram + currentChar;
LIST(vsgrams) ⟵ gram;
gram ⟵ currentChar }
```

The generalized vs-grams allow to generalize the content of the lines. This constitutes essential information while reducing the representation space compared to n-grams.

Once the notion of generalized vs-grams defined, we build the set of features by extracting the generalized vs-grams in a line window around the beginning of logical units. Like the previous solution, the generalized vs-grams are associated with line numbers (in the window) wherein they are extracted. A feature is hence a composition of a vs-gram and the position of the line (in the window) wherein it is extracted. The obtained feature set for the corpus of log files contains 1023 elements. The creation of positive and negative instances is conducted by using the feature set obtained by extracting generalized vs-grams in the tagged corpus of log files.

To exemplify the creation of vs-gram features, we use the same log file segments presented in Fig. 5, which were previously used to extract manual features. Taking the first segment of Fig. 5, we obtain the following features by extracting the generalized vs-grams:

f_1(\-+␣\w+, 0)	f_2(\w+ **:**, 0)
f_3(\w+ \s+ **:**, -2)	f_4(**:** \w+, -2)
f_5(\w+ \s+ **:**, +2)	f_6(**:** \w+, +2)

Feature f_1 contains a seen vs-gram (\-+␣\w+) on the beginning of the first segment. This vs-gram corresponds to "------ D" in the beginning of the segment (line 86). Then the rest of line is characterized by the next extracted vs-gram (\w+ **:**) which is used in feature f_2. Thus, the beginning line of the first segment, is characterized by the two features f_1 and f_2. Then, we have f_3 and f_4 which are composed of vs-grams extracted in the second line (line 84) before the beginning of the first segment. f_5 and f_6 correspond also to the second line after the beginning of the first segment.

5 Learning to Identify Logical Units

The objective of our work is to determine a model of rules from which we can identify the logical units in the log files. For this purpose, we first establish a corpus of log files. Then, we identify the set of features using the methods described in Sections 4.1 and 4.2. Each line of the corpus is seen as a positive or negative instance. The lines are represented as a boolean vector whose elements are features.

Afterwards, a supervised machine learning process based on a classification method is applied. The instances previously obtained are used as training set. The obtained classification model enables to classify the lines of corpus into two classes (beginning of segment / not beginning of segment).

Regarding the training data, we are faced with the balance of positive and negative instances problem. Indeed, in our corpus obtained from real data, the number of negative examples is seven times more than the number of positive

examples. The distribution of "+" and "-" examples can influence a classifier that uses the probability of each class to predict. Moreover, the lack of examples for special cases prevents the classifier to create the necessary rules. To address this unbalance problem several solutions can be applied: over sampling, under sampling, SMOTE. Interested reader may refer to [3] when an overview of the different approaches is proposed.

In the "Over sample", examples of the class with fewer examples are duplicated. If we want to retain the same size for the new dataset, it requires removing some of the examples of the other class. The choice of examples to be duplicated or removed is performed randomly. Nevertheless a real situation may create a problem of over-fitting.

The solution "Under Sampling" consisting of removing some examples of a class with more examples, has its own drawbacks. Indeed, we cannot guarantee the conservation of all instances of a particular case in that class. Moreover, this does not solve the lack of examples in the other class.

The "SMOTE" method consists in generating synthetic examples. This generation can build synthetic positive examples that cannot exist in real case according to the characteristics of the domain. Indeed, we observed that in some cases, by changing the value of a single feature in a positive example, we obtain an example which is inconsistent in the real world.

Depending on the characteristics of our data, we finally chose the "over sampling" solution. Actually, generating synthetic example is difficult in our context and with under sampling we can lost useful structures. The parameters of the algorithm like the size of the new dataset and the new class distribution rate are set after several tests. We tested several classification methods including "K Nearest Neighbors" (KNN) and "decision trees". The classification model obtained in the learning phase will be used to identify the lines representing the beginning of logical units in real industrial data. The classification results, explanation of parameters and obtained models are developed in Section 6. We also discuss the impact of both methods of feature acquisition in the next section.

6 Experiments

The experiments are conducted in two main directions to compare the semi-automatic approach (via heuristics of an expert) and the automatic approach (via extraction of generalized vs-grams):

- performance of the classification of the log file lines into positive and negative classes:
  - using features built based on defined patterns
  - using generalized vs-gram features

- comparison of our segmentation method with the well-know TextTiling segmentation method.

We use the same training corpus in all experiments. The learning corpus consists in 19 different log files from the industrial world. The log files contain real data and differ in content and especially in structure. The training corpus size is 1.1 MB and contains in total 19,638 lines.

We have calculated the performance of our approach in terms of precision and recall of the classification model. Precision for a class is the number of true positives divided by the number of instances *predicted* as positive. Recall in a class is the number of true positives divided by the number of instances *actually belonging* to the class. Finally, the harmonic average between precision and recall is calculated by the F-Score. More formally they are defined as follows:

$$Precision = \frac{Relevant\ segment\ retrieved}{retrieved\ segments}$$
$$Recall = \frac{Relevant\ segment\ retrieved}{relevant\ segments}$$
$$F\text{-}Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

We thus test the classification models obtained with each feature acquisition methods proposed in this chapter. As a reminder, in the classification, a line is a positive instance if and if it presents the beginning of a logical unit (i.e., logical division). A negative instance is a line which does not represent a logical division.

6.1 Tests Using Features Build by the Defined Patterns

At this stage of experiments, the feature set is created by the method described in Section 4.1. By using the training corpus to build the feature set, we identified 128 patterns. Consequently, we have 128 features to create instances. In the training data, there are initially (before dataset balancing) 1,142 positive examples against 18,496 negative examples. To perform the tests, we balanced the dataset using the "over sampling" method of Weka. We chose 0.8 as the equilibrium rate of the number of examples. We finally got a dataset in which there are 5,833 positive examples against 9,877 negative examples.

We present here the results obtained using the classification algorithms that yielded the best results: C4.5 decision tree [21] and KNN [16]. Although the results obtained by other algorithms like SVM are largely close to those obtained by KNN or C4.5, we do not provide the results of all tested algorithm as our objective if not to compare the performance of classification algorithm, but to evaluate the quality of build feature sets. C4.5 is particularly appropriate in our concern since at the end of the process we are provided with the classifier learnt by examining the learnt decision tree. In other words, we look to know if the extracted features (manually and automatically) are enough relevant to represent the visual characteristics of logical units. Their relevancy can be determined by the accuracy of the classification using these features.

To apply the classification algorithms, we use the built-in implementations in WEKA software[9]. To evaluate the classification performance, we use the cross validation (10 folds). Cross-validation is a method for estimating reliability of a model based on a sampling technique. We divide the initial training set into "n" samples and repeat the learning "n" times while choosing n-1 samples as training set and the last sample for evaluation. Each sample is used once for evaluation. Finally, the performance is given by the average of performances obtained in each iteration of the process.

	C 4.5				KNN		
Class	Precision	Recall	F-Score	Class	Precision	Recall	F-Score
Pos	0.98	0.99	**0.99**	Pos	0.98	0.99	**0.99**
Neg	0.99	0.99	**0.99**	Neg	0.99	0.99	**0.99**

Table 1: Classification performance according to each class - C4.5 (left) and KNN (right) - using the *balanced* dataset - Features obtained by "defined patterns"

Table 1 presents the results obtained for each class using the balanced dataset built using heuristic-based features. We observe that by using these heuristics-based features, we succeed in creating a set of rules (a classification model) with an overall F-score of 0.99.

6.2 Tests Using Generalized vs-grams Features

We experiment in this section the use of generalized vs-grams as features (automatic method). For this purpose, we use the same corpus described above. The tests are also conducted using the balanced datasets. Table 2 shows the results obtained by each classifier using the balanced dataset where the features are obtained via the extraction of generalized vs-grams.

	C 4.5				KNN		
Class	Precision	Recall	F-Score	Class	Precision	Recall	F-Score
Pos	0.92	0.74	**0.82**	Pos	0.94	0.75	**0.84**
Neg	0.96	0.98	**0.97**	Neg	0.97	0.98	**0.97**

Table 2: Classification performance according to each class - C4.5 (left) and KNN (right) - using the *balanced* dataset - Features obtained by "generalized vs-grams"

Using the features obtained by extracting the vs-grams, we reach a precision equal to 0.94 in the positive class and equal to 0.97 in the negative class. This means that the automatic method based on *generalized vs-grams* gives a classification performance close to the semi-automatic method which was based on

[9] *http* : *//www.cs.waikato.ac.nz/ml/weka/*

an expert heuristics. This shows that the generalized vs-grams represents well the characteristics of logical units of a document.

Finally, in order to improve the performance of classification, we have decided to create a set of features comprised mainly the automatic features (the generalized vs-grams), and a limited number of features from the heuristics of an expert (defined patterns). We use, in this test, ten features obtained by the heuristics. These features mostly characterize the *beginning* and the *end* of the line representing a logical division. Table 3 shows the performance of classification and therefore the recognition of logical units using a new dataset obtained by this new set of mixed features.

	C 4.5				KNN		
Class	Precision	Recall	F-Score	Class	Precision	Recall	F-Score
Pos	0.98	0.99	**0.99**	Pos	0.98	0.99	**0.99**
Neg	1	0.99	**1**	Neg	1	0.99	**1**

Table 3: Classification performance according to each class - C4.5 (left) and KNN (right) - using the dataset obtained by set of mixed features.

The results show that we obtain a performance similar to that obtained by using heuristics-based features. This shows that a minimum addition of expert knowledge (only ten semi-automatic features) significantly improves the results.

7 Discussing Semantic and Discourse Segmentation

Here we aim at studying how a semantic segmentation method would behave in the context of log files. For this purpose, we use the well-known TextTiling semantic segmentation method. This method, presented in Section 2, is a topic based segmentation method. It tries to recognises the topic changes in documents based on term co-occurrences and lexical cohesion measures. In order to perform the tests, we use the David James implementation of TextTiling, which is used in Lingua NLP package[10].

What is a relevant segment? Regarding the segments obtained by TextTiling, we observed that they hardly correspond to the segments initially tagged by a domain expert in the corpus of log files. In fact, we should consider that each segmentation method split a document into segments which may differ from segments obtained by another method. This point is emphasized when the segmentation methods use different strategies and assumptions. In the same

[10] http://search.cpan.org/~splice/Lingua-EN-Segmenter-0.1/lib/Lingua/EN/Segmenter/TextTiling.pm

time, we note that it is possible to have different kinds of relevant segmentation for a document. This means that we cannot say that there is only one relevant segmentation model for a document.

In our context, the log files are initially tagged by an expert based on their structure whereas TextTiling determines segments based on the identified topic changes. That is why it is not relevant to compare the segments obtained by TextTiling with reference segments which are based on the log file logical structure. Therefore, we asked the domain expert again to independently analyze the relevance of segments obtained by TextTiling. In this analysis, a segment is considered *irrelevant* if:

- it contains several text chunks that each one should be considered as a unique segment;
- it does not contain the entire of a text chunk that represents an unique segment.

In the first case, the obtained segment should be split into more segments as it contains different text chunks which are not significantly correlated. The fact that there are different kinds of information in a segment can bias the passage retrieval performance. A relevant segment should only contain correlated information.

In the second case, the issue is more important as the obtained segment do not contain the totality of the information. This means that the segmentation method has split a relevant segment into two or more irrelevant segments. In such situation, we lost some part of information which is not situated in the obtained segment.

Quality of TextTiling segments. For segments obtained by TextTiling, we evaluate them based on the two previously presented conditions. Once the relevant and irrelevant segments are determined, we define precision as the number of relevant segments divided by the number of all obtained segments. Since in the case of TextTiling, there are not a reference segmentation model, we do not provide a recall estimation. In other words, we just seek to evaluate the relevance of segments obtained by TextTiling.

In the case of our proposed approach, by using the automatic method (extraction of generalised vs-gram) to acquire the features, we do not need the expert knowledge nor to adapt our approach to different kind of log files. Whereas TextTiling needs to be parametrized according to the characteristics of documents. The main TextTiling parameters are Pseudo-sentence size and size (in sentences) of the block used in the block comparison method. After some tests on log file corpus, we set these two parameters to "8" which determine the size of Pseudo-sentence in terms of word and the size of text blocks in terms of sentences. In order to note also the performance of our approach, we provide the results obtained by our approach. In our approach, we used the vs-gram features and KNN classification.

	Obtained segments	Irrelevant	Relevant	Precision
Segmentation using vs-grams	1086	56	1030	**94**%
TextTiling	377	145	232	**61**%

Table 4: Segmentation by structure recognition using generalized vs-gram vs. TextTiling method

As shown in Table 4, we observe that only 61% of segments obtained by TextTiling are relevant. Regarding our method, the obtained results show that 94% of determined segments are relevant.

By analysing the obtained segments, we observed that TextTiling is not capable to recognize all topic changes, which results in large segments which should be split into more small segments. In the same time, TextTiling did not recognize the entire of tables as an unique segments. Most tables are split into several segments. This situation is explained via an example in Section 2 while we discussed the relevance of existing topic based methods. Moreover, there is a visual structure in log files which helps to recognize the correlated information whereas the topic based methods, like TextTiling, which use the lexical cohesion measures do not take this visual structure into account.

8 Conclusions and Future Work

We have presented an approach to segment log files (texts with complex logical structures). In our approach we created a set of features where each feature presents a syntactic characteristic of logical structures. To build the set of features, we have proposed two methods: semi-automatic approach (via heuristics of an expert), and automatic approach (via extraction of generalized vs-grams). The results show that *the generalized vs-grams*, introduced in this paper, can be used for modeling the complex logical structures of documents.

Using a training set based on the obtained features and a supervised classification method, we build a classification model which makes it possible to distinguish the logical structures in the log files with an F-Score of 0.99. These results have confirmed the experts to use our segmentation protocol in their industrial usages.

In our future work, we wish to experiment our approach with other learning algorithms and test our approach on other types of complex data. We also plan to learn automatically the classes of the most discriminatory features to identify a logical structure. Such knowledge can guide the user to add more relevant expert information in an automated process.

References

[1] J. P. Callan. Passage-level evidence in document retrieval. In *Proceedings of the 17th annual International ACM SIGIR Conference on Research and development in information retrieval*, SIGIR'94, pages 302–310, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[2] Y. Cao, F. Liu, P. Simpson, L. Antieau, A. Bennett, J. J. Cimino, J. Ely, and H. Yu. Askhermes: An online question answering system for complex clinical questions. *Journal of Biomedical Informatics*, 44(2):277 – 288, 2011.

[3] N. V. Chawla. *Data Mining and Knowledge Discovery Handbook*, chapter Data Mining for Imbalanced Datasets: An Overview, pages 853–867. Springer Verlag, 2013.

[4] M. Embarek. *Un système de question-réponse dans le domaine médical*. PhD thesis, Université Paris-Est, France, 2008.

[5] M. A. Hearst. Texttiling: segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23:33–64, March 1997.

[6] H. Hernault, H. Prendinger, D. A. duVerle, and M. Ishizuka. Hilda: A discourse parser using support vector machine classification. *Dialogue and Discourse*, 1(3), 2010.

[7] M. Jafari, F. SoleymaniSabzchi, and S. Jamali. Extracting users' navigational behavior from web log data: a survey. *Journal of Computer Sciences and Applications*, 1(3):39–45, 2013.

[8] M. Kaszkiel and J. Zobel. Passage retrieval revisited. In *Proceedings of the 20th annual International ACM SIGIR Conference on Research and development in information retrieval*, SIGIR'97, pages 178–185, New York, NY, USA, 1997. ACM.

[9] M. Kaszkiel and J. Zobel. Effective ranking with arbitrary passages. *Journal of the American Society for Information Science and Technology*, 52:344–364, February 2001.

[10] H. Kozima. Text segmentation based on similarity between words. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, ACL'93, pages 286–288, Stroudsburg, PA, USA, 1993. Association for Computational Linguistics.

[11] A. Labadié and V. Prince. Lexical and semantic methods in inner text topic segmentation: A comparison between c99 and transeg. In *Proceedings of NLDB'08*, pages 347–349, Berlin, Heidelberg, 2008. Springer-Verlag.

[12] F. Llopis, A. Ferrndez, and J. Vicedo. Text segmentation for efficient information retrieval. In A. Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volume 2276 of *Lecture Notes in Computer Science*, pages 13–29. Springer Berlin / Heidelberg, 2002.

[13] F. Llopis, J. L. Vicedo, and A. Ferrández. Passage selection to improve question answering. In *proceedings of the 2002 Conference on multilingual summarization and question answering - Volume 19*, MultiSumQA'02, pages 1–6, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[14] D. Marcu. *The theory and practice of discourse parsing and summarization.* MIT Press, 2000.

[15] D. Mehrzadi and D. G. Feitelson. On extracting session data from activity logs. In *Proceedings of the 5th Annual International Systems and Storage Conference*, SYSTOR '12, pages 3:1–3:7, New York, NY, USA, 2012. ACM.

[16] T. M. Mitchell. *Machine Learning.* McGraw-Hill International Edit, 1997.

[17] V. C. Nguyen, L. M. Nguyen, and A. Shimazu. Improving text segmentation with non-systematic semantic relation. In *Proceedings of the 12th International Conference on Computational linguistics and intelligent text processing - Volume Part I*, CICLing'11, pages 304–315, Berlin, Heidelberg, 2011. Springer-Verlag.

[18] B. Ofoghi, J. Yearwood, and R. Ghosh. A semantic approach to boost passage retrieval effectiveness for question answering. In *ACSC'06: Proceedings of the 29th Australasian Computer Science Conference*, pages 95–101, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[19] J. M. Ponte and W. B. Croft. Text segmentation by topic. In *Proceedings of the First European Conference on Research and Advanced Technology for Digital Libraries*, pages 113–125, London, UK, 1997. Springer-Verlag.

[20] J. M. Ponte and W. B. Croft. Text segmentation by topic. In *Proceedings of the First European Conference on Research and Advanced Technology for Digital Libraries*, pages 113–125, London, UK, 1997. Springer-Verlag.

[21] J. R. Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[22] G. Salton, J. Allan, and C. Buckley. Approaches to passage retrieval in full text information systems. In *Proceedings of the 16th annual International ACM SIGIR Conference on Research and development in Information Retrieval*, SIGIR'93, pages 49–58, New York, NY, USA, 1993. ACM.

[23] D. Schiffrin. *Discourse markers.* Cambridge University Press, Cambridge, 1987.

[24] M. Sun and J. Y. Chai. Discourse processing for context question answering based on linguistic knowledge. *Knowledge-Based Systems*, 20:511–526, August 2007.

[25] C.-M. Tan, Y.-F. Wang, and C.-D. Lee. The use of bigrams to enhance text categorization. *Information Processing and Management*, 38:529–546, July 2002.

[26] O. Tarek. La segmentation des documents techniques en amont de l'indexation : définition d'un modèle. *Revue d'Information Scientifique et Technique (RIST)*, vol. 13(no1):79–94, 2003.

[27] J. Tiedemann and J. Mur. Simple is best: experiments with different document segmentation strategies for passage retrieval. In *Coling 2008: Proceedings of the 2nd workshop on Information Retrieval for Question Answering*, IRQA'08, pages 17–25, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[28] M. A. Walker. Redundancy in collaborative dialogue. In *Proceedings of the 14th Conference on Computational linguistics - Volume 1*, COLING'92, pages 345–351, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[29] J. Zobel, A. Moffat, R. Wilkinson, and R. Sacks-Davis. Efficient retrieval of partial documents. In *Proceedings of the second Conference on Text retrieval Conference*, pages 361–377, Elmsford, NY, USA, 1995. Pergamon Press, Inc.