American Journal of Applied Sciences 2 (9): 1309-1314, 2005
ISSN 1546-9239

A Methodology to Segment the Text for Index Terms

Muhammad Shoaib and Abad Ali Shah
Department of Computer Science and Engineering
University of Engineering and Technology, Lahore, Pakistan

Abstract: The problem of information overload is a hot issue with the growth of the world wide web. The need for the tools those should be able to absorb this huge information and eliminate this problem is evident especially for IR systems. Text is not a simple sequence of words but carries a structure. It is essential to handle these uncontrollable complex structures of sentence, grammatical and lexical irrelevancy of different units. The main idea to handle these problems is to segment the text into elementary units, which will be simpler and lesser complex than their equivalent text. We have used cue phrases, punctuations. We are presenting an algorithm, which is not only efficient but also handling more than 500 cue phrases and most of punctuations. This proposed algorithm can yield elementary units, which can be used by Rhetorical Relations Finder to get relations among them, which can be used by the RST Parser for the construction of RST Tree which will be used to design an RST based indexer. In future, the algorithm can be enhanced for handling other discourse markers, which will enable us to handle the most complex cases where cue phrases and punctuations are not applicable.

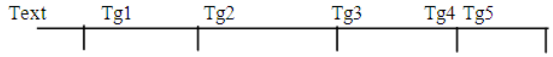
Key words: RST, text segmentation, cue phrases, punctuations

INTRODUCTION

It is commonly admitted that the text has structure, which is independent of world and domain specific knowledge[1]. Sentences in the text are closely interrelated and grouped in certain ways to form the whole text. Sentences relations are weaker than the relation that exists between words but the sentences are interpreted jointly and they meant to coexist. IN the same way there exist a relation between the paragraphs in any document[1-5].

To find the relationship between two adjacent short paragraphs is much easier than to find the relationship between two longer units of text. This fact has caused to the distinction between the global and local structures of discourse.

The theories[6,7] support this distinction. The theory developed by Gresz and Sinder deals specifically with local discourse structure[6]. This shows the need for identifying the boundaries of the segments of text. Segmentation is the process to identify the units of text whose sentences are strongly connected to each other. Text segmentation of any document problem focuses on how to identify the regions of text ends and another begins of that document. Simply we can say that a text is divided into N Segments, which display certain characteristics (i.e Text spans, a topic or an idea).



There are many application areas of the text segmentation like Information retrieval[8-10].

Text Segmenter is also helpful in finding the subtopic, which facilitates the user to jump from one topic to another topic or to his required information. The Text Segmenter also provides the structural information about the document, which enables to find the relations, which exists in any document. It can also be used for effective query and analysis. Our main interest in text segmentation is to use for Indexing purpose and ultimately to improve the performance of Information Retrieval Systems. There are many ways in which text can be broken down into segments[4]. We have used cue phrases and punctuations for text segmentation.

Cue phrases are words that connect two or more spans and add structure to the discourse of text, for example, some cue phrases are given: “first”, “and”, “now”, “accordingly”, “actually”, “also”, “although” etc. Marcue created a set of more than 450 cue phrases[11-13]. Also, Simon H Corston-Oliver describes a set of linguistic cues that can be identified in a text as an evidence of discourse relations[14] as well as to segment the text.

NEED FOR TEXT SEGMENTATION

In order to automatically build the valid text structure of an arbitrary text, we need only to determine the elementary units of that text. Therefore, an accurate determination of the elementary units of a text is the

Corresponding Author: Muhammad Shoaib, Department of Computer Science & Engineering, University of Engineering and Technology, Lahore, Pakistan, Tel: +92+333-4214364

most important task. Using cue phrases is one of the best ways to determine elementary units[15]. According to Litman and Hirschberg, Cue phrases are words, phrases, or linguistic expressions that may directly and explicitly mark the structure of a discourse[16].

The main cause to divide text into elementary units is to eliminate the complexity of large grammatical sentences. Finding the elementary units at early stages carries many advantages as follows:

* Elementary units provide an easy to handle Natural Language Programming
* Elementary units of text help to extract relations more easily.
* Elementary units enhance the efficiency of a complete process, as smaller text units are easier to handle with respect to their larger equivalents.

Hearst introduced the Text Tilling algorithm[16]. This algorithm segments the text into multiple paragraphs of coherent discourse units. As Hearst's approach segments at the paragraph level, this is not suitable for the applications like Information Retrieval.

Kozima gives the approach of a "Lexical Cohesion profile" to keep track of the semantic cohesiveness of words in a text within a fixed length window[17]. Kozima uses a semantic network to provide knowledge about related word pairs. The network is trained automatically using a language specific knowledge, generalizing it by applying it to a window of text and finding the cohesiveness of successive text windows in a document and hence finding the boundaries of the text segments.

Reynan[18] present the graphically motivated segmentation technique called dot plotting .It uses the simplified notion of lexical cohesion. It exclusively depends upon on word repetition to find tight regions of topic similarity.

Grosz and Sidner proposed the Grosz and Sidner's Theory (GST), in which, they named linguistic textual units as Discourse Segments (DS), which are used in construction of discourse structure. Although GST provided an idea of the DS or elementary units, but they never explained as a particular methodology[6].

Daniel Marcu proposed a comprehensive methodology of Shallow Processing to decompose given text into elementary units and named them as Text Spans[4]. He used cue phrases, comma and parenthetical as a basic tool. He also elaborated the idea to handle the duel behavior of cue phrases (sentential and discourse usage) like and or. Unfortunately he missed much other punctuation, which can play a vital role in the segmentation of text into elementary units and his algorithm is not as efficient as it should be also it has not provided an automated solution to text segmentation

In this study, we presented novel scalable and robust text segmentation technique, which cover all the deficiencies present in the algorithm of Daniel Marcue.

PROPOSED ALGORITHM FOR TEXT SEGMENTATION

The accuracy and efficiency can be achieved by using the following proposed algorithms, which is providing the stub parts of RST for tree development. The get Text Span procedure serves as the main starting point for this proposed solution, this procedure takes input the string to be processed and calls the makeParagraphArray procedure to divide text into paragraphs with the help of utility procedure getParagrph and afterwards the output is sent to makeSentenceArray which divides the paragraphs into sentences with the help of utility procedure getSentence. And each sentence is analyzed by analyseTextSpan procedure, which provides the core functionality to extract elementary units:

Input: Unstructured text
Output: The Elementary Units
Processing:

Step 1:

Procedure Name	**getTextSpan**
Input	String to Process
Output	ArrayList of Elementary Units
Process	This Procedure takes input the string to be processed and calls the makeParagraphArray procedure to divide text into paragraphs and afterwards the output is sent to makeSentenceArray which divides the paragraphs into sentences. And each sentence is analyzed by analyseTextSpan procedure, which provide the core functionality to extract elementary units.

```
getTextSpans(string st)
        st = handleMultiWordCuePhrases(st)
        aParagraphs = makeParagraphArray(st)
        for(i=0 to aParagraphs.Count-1)
        aSentences=makeSentenceArray((string)(aParagraphs[i]));
                for(j=0 to aSentences.Count-1)
                        index=-1;
                        iStart=0;
                        sSentence=aSentences[j]
                        while(index <sSentence.Length-1)
                                sTkn = getNextToken(sSentence, index)
                                if( sTknis not NULL)
```

```
                                        if( sTkn is delim  OR sTkn is cuePhrases))
                                                sUTS=analyseTextSpan(sSentence,
                                                        iStart, index, sTkn, sNextTkn)
                                        if(sUTS is Null and sTK is cuePhrase)
                                                sLastCuePhrase = sNextTkn;
                                            else if (sUTS is not NULL)
                                            if (sLastCuePhrase is not NULL)
                                                sCuePhrase=sLastCuePhrase
                                                sLastCuePhrase=""
                                                End if
                                                if(sTkn.IndexOfAny(delim)>-1)
                                                        sPunctuation = sTkn;
                                                else
                                                sLastCuePhrase = sTkn;
                                                End if
                                                if(sNextTkn!="")
                                                sPunctuation=sNextTkn;
                                                        tn = new
                                                        CTextNode(sUTS, iUID,
                                                        j, i, sPunctuation,
                                                        sCuePhrase);

                                                        aTextSpans.Add(tn);
                                                        ++iUID;
                                                End if
                                        End if
                                End if
                        End While
                End For
        End For
    return aTextSpans
End getTextSpans
```

Step 2:

Procedure Name	**analyseTextSpan**
Input	Sentence to Process Starting Index Current Index Token to Analyze Next Token
Output	Elementary Unit
Process	This Procedure takes input the Sentence to be processed and provide the core functionality to extract elementary units; this procedure uses getNextToken procedure to tokenize the input.

```
analyseTextSpan(string sSentence, By Reference int iStart, By Reference int index, string sTkn, By Reference string sNextTkn)
    sNTkn = getNextToken(sSentence, ref iNext).Trim();
    if(sTkn.IndexOfAny(parentheticals)>-1)
        if(bParenthetical==false)
                sTS= sSentence.Substring(iStart, index-iStart-sTkn.Length)
                bParenthetical=true
                iStart=index+1
                index++
        else
                sTS= sSentence.Substring(iStart, index-iStart-sTkn.Length+1)
                iStart=index+sTkn.Length
                index+=sTkn.Length
                bParenthetical=false
                End if
    else if(bParenthetical==true)
                        return sTS;
    else if(sSentence.Substring(iStart,index-iStart+ sTkn.Length ).Trim()==sTkn &
    CString.IndexOfAny(sTkn,cuePhrases)>-1)
                sNextTkn=sTkn;
                return sTS;
        else if(sTkn=="," && sNTkn.ToUpper()!="AND" && sNTkn.ToUpper()!="OR")
            sTS= sSentence.Substring(iStart, index-iStart+sTkn.Length)
             iStart=index+sTkn. Length
             index++
        else if(sTkn=="," && (sNTkn.ToUpper()!="AND" || sNTkn.ToUpper()!="OR"))
              return sTS;
        else if(CString.IndexOfAny(sTkn.ToUpper(),cuePhrases)>-1 && sNTkn.IndexOfAny(delim)>-1 )
```

```
        sTS= sSentence.Substring(iStart, iNext-iStart+sNTkn.Length)
                                iStart=iNext+sNTkn. Length
                                index=iNext+1
                                sNextTkn=sNTkn
    else if(CString.IndexOfAny(sTkn.ToUpper(),cuePhrases)>-1)
        If (index-estate-sTkn. Length>-1)
                sTS= sSentence.Substring(iStart, index-iStart-sTkn.Length);
        else
                sTS= sSentence.Substring(iStart, index+sTkn.Length)
                iStart=index-sTkn.Length
                index+=sTkn. Length
    else if(sTkn.IndexOfAny(delim)>-1)
        sTS= sSentence.Substring(iStart, index-iStart+sTkn.Length)
        iStart=index+sTkn. Length
        index++
    End if
    return sTS
End analyseTextSpan
```

Step 3:

Procedure Name	**makeParagraphArray**
Input	String to Process
Output	ArrayList of Paragraphs
Process	This Procedure takes input the string to be processed and divide text into paragraphs with the help of procedure getParagraph.

```
makeParagraphArray(string st)
    Set index=-1
While (index < St. Length-1)
        Set stkn = getParagraph(st index)
        if( stkn is not NULL )
                aParagraphs.Add(stkn)
        end if
    End While
    return aParagraphs
End makeParagraphArray
```

Step 4:

Procedure Name	**getParagraph**
Input	String to Process Starting Index of new Paragraph
Output	Paragraph String
Process	This Procedure takes input the string to be processed and return the next paragraph on the basis of new line criteria.

```
getParagraph(string st, by Reference int index)
    if (not End of String)
    do
        if(no new line found)
                inc index by 1
                Set sTP=st
        else if( st.IndexOf(newLine, index+1)>0)
                Set sTP = st.Substring(from index to new line)
                Set index = st.IndexOf(next newLine)
        else
                sTP=st.Substring( index+1)
                index += st.Length- st.LastIndexOf(newLine);
        End if
    while((sTP.IndexOf(newLine) > 0 OR sTp is not NULL ) AND index<st.Length-1)
    End if
    return sTP
End getParagraph
```

Step 5:

Procedure Name	**makeSentenceArray**
Input	String to Process in form of Paragraph
Output	ArrayList of Sentences
Process	This Procedure takes input the paragraph to be processed and divide text into sentences with the help of procedure getSentence.

```
makeSentenceArray(string st )
```

```
    Set index=-1
    while(index < st.Length-1)
        stkn = getSentence(st, index)
        if( stknis not NULL )
            aSentences.Add(stkn)
    return aSentences
End makeSebtenceArray
```

Step 6:

Procedure Name	**getSentence**
Input	String to Process in form of Paragraph Starting Index of new Sentence
Output	Sentence String
Process	This Procedure takes input the string to be processed and return the next sentence on the basis of new sentence criteria.

```
getSentence(string st, by Reference int index)
    if (not end of String)
    do
        if(st.IndexOfAny(newSentence,index+1)==0)
            inc in index by 1
            set sTP=st
        else if( st.IndexOfAny(newSentence, index + 1) > 0)
            sTP=st.Substring(index + 1, nextSentence)
            index = st.IndexOfAny(newSentence, index + 1)
        else
        sTP=st.Substring( index + 1).Trim();
        index += st.Length - st.LastIndexOfAny(newSentence);
    while( sTP.Trim().Length < 1 && index<st.Length-1);
    return sTP
End getSentence
```

Step 7:

Procedure Name	**getNextToken**
Input	String to Process Starting Index of Token
Output	Token String
Process	This Procedure takes input the string to be processed and return the next token which can either be a simple word, a cue phrase or a punctuation.

```
getNextToken(string st, by Reference int index)
    set sTkn=""
    if (index<st.Length-1)
        do
            if(st.IndexOfAny(delim,index+1)==0)
                inc index by 1
                sTkn=""
            else if( st.IndexOfAny(delim, index + 1) > 0)
                if (first char is delim)
                    sTkn=st.Substring(first char)
            else
                sTkn=st.Substring(next char)
            End if
            index = st.IndexOfAny(delim, index+1 )
            if (sTkn is not NULL)
                dec index by 1
            End if
        else
            sTkn=st.Substring( index )
            index = last index of delim
        End if
    while( sTkn is NULL AND Not End of String)
  return sTkn
End getNextToken
```

COMPUTATIONAL EVALUATIONS

Although the complexity of the proposed solution seems to be exponential from general mathematical representation:

$$\text{Complexity of Algorithm} = \sum_{i=0}^{P-1}\sum_{j=0}^{Sp-1}\sum_{k=0}^{Us-1}\text{Operations}$$

Where:

P = Total Number of Paragraphs in giving Text

S = Total Number of Sentences in respective Paragraph

U = Total Number of Elementary Units in respective Sentence

But with the analysis of the results it yields a linear relationship with following mathematical representation

Complexity of Algorithm = $\sum_{k=0}^{N-1}$ Operations

Where:
N = Total Number of Elementary Units in giving Text

The analytic study shows that our algorithm is more efficient for text segmentation.

CONCLUSION

We have presented a technique of segmenting the text into elementary units, which will help us to form a rhetorical tree. The text spans will be used for extracting relations. The pseudocode presented has been implemented in the programming language C# and results are promising. The results are being verified on different types of the text as well as mathematically it has been proved that the execution flow of our proposed algorithm is linear which is more efficient than exponential algorithms.

Future work: Currently we are working on the extension of our proposed weight assignment approach and considering both keywords and the RST relationships of a collection for the purpose of indexing and referring it to as this indexing technique as composite dynamic indexing technique. The output of the technique demonstrated in the paper will be used for RST relation based tree constructing whose node will contain a text segment and relations. The next paper will demonstrate this technique. These techniques will finally be used for indexing technique in the IR Systems.

REFERENCES

1. Carlson, L., J. Conroy, D. Marcu, D. O'Leary, M.E. Okurowski, A. Taylor *et al.*, 2001. An empirical study of the relation between abstracts, extracts and the discourse structure of texts. Proc. Document Understanding Conf. (DUC-2001), New Orleans, Louisiana.
2. Bateman, J. and K.J. Rondhuis, 1997. Coherence relations: Towards a general specification. Discourse Processes, 24: 3-49.
3. Corston, S. and M. Cardoso de Campos, 2000. Automatically recognizing the discourse structure of a body of text. United States Patent 6,112,168, Microsoft Corporation.
4. Mann, W.C. and S.A. Thompson, 1998. Rhetorical structure theory: Towards a functional theory of text organization. The J. Text, 8: 243-28.
5. Shoaib, M. and A. Shah, 2005. A dynamic weight assignment approach for IR systems. Ist Intl. Conf. Computer and Commun. Technology., IEEE, Pakistan.
6. Grosz B. and S. Candace, 1986. Attention, intentions and the structure of discourse. Computational Linguistics.
7. Kintsch and van Dijik, 1978. The structure of Discourse. Computational Linguistics.
8. Afantenos, S., V. Karkaletsis and P. Stamatopoulos, 2005. Summarization of medical documents: A survey. Artif. Intell. in Med., 33: 157-177.
9. Shah, A. and M. Shoaib, 2005. Sources of irrelevancy in information retrieval systems. Intl. Multi Conf. In Computer Sci. & Computer Engg., USA.
10. Shoaib, M. and A. Shah, 2005. Remote information retrieval using a cell phone. Intl. Multi Conf. In Computer Sci. & Computer Engg., USA.
11. Marcu, D., 1996. Building up rhetorical structure trees. Proc. 13th Natl. Conf. Artif. Intell., USA., 2: 1069-107.
12. Marcu, D., 1997. The rhetorical parsing of natural language texts. Proc. 35th Ann. Meeting of the Assoc. for Computational Linguistics (ACL-97), pp: 96-103.
13. Marcu, D., 2000. The theory and practice of discourse parsing and summarization. Proc. 35th Ann. Meeting of the Assoc. Computational Linguistics (ACL-97), pp: 96-103.
14. Simon, H. and Corston-Oliver, 1998. Computing representations of the structure of written discourse. Technical Report MSR-TR-98-15, Microsoft Research, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052.
15. Burstein, J., D. Marcu and K. Knight, 2003. Finding the WRITE stuff: Automatic identification of discourse structure in student essays. IEEE Intelligent Systems, 18: 32-39.
16 Redeker, G., 2000. Coherence and Structure in Text and Discourse. W. Black and H. Bunt (Eds.), Abduction, Belief and Context in Dialogue. Studies in Computational Pragmatics, John Benjamins, Amsterdam and Philadelphia, pp: 233-263.
16. Hearst, M., 1994. Multi-paragraph segmentation of expository text. 32nd Ann. Meeting of the Assoc. for Computational Linguistics.
17. Kozima, H., 1993. Text segmentation based on similarity between words. Proc. ACL'93, Ohio.
18. Reynar, J., 1994. An automatic method of finding topic boundaries. 32nd Ann. Meeting of the Assoc. for Computational Linguistics.