2011 Seventh International Conference on Computational Intelligence and Security

An Efficient Linear Text Segmentation Algorithm Using Hierarchical Agglomerative Clustering

Ji-Wei Wu
Dept. of Computer Science
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.
jwwu.cs97g@nctu.edu.tw

Judy C.R. Tseng
Dept. of Computer Science and Information Engineering
Chung Hua University
Hsinchu, Taiwan, R.O.C.
judycrt@chu.edu.tw

Wen-Nung Tsai
Dept. of Multimedia and Game Science
Yu-Da University
Miaoli, Taiwan, R.O.C.
tsaiwn@ydu.edu.tw

***Abstract*—Linear text segmentation aims at dividing a long text into several topical segments. It is beneficial to many natural language processing tasks, such as information retrieval and document summarization. In this article, an efficient linear text segmentation algorithm based on hierarchical agglomerative clustering is presented. The proposed linear text segmentation algorithm is implemented without auxiliary knowledge base, parameter setting, and user involvement. Experimental results show that the proposed linear text segmentation algorithm not only provides linear time computational complexity, but also provides comparable segmentation accuracy with several well-known linear text segmentation algorithms.**

Keywords-text segmentation; hierarchical agglomerative clustering, computational intelligence, NLP application

I. INTRODUCTION

The purpose of linear text segmentation is to divide a long text into several segments, each of which corresponds to a topic and consists of consecutive sentences or paragraphs. In other words, the task of linear text segmentation is to identify topic boundaries within a long text. Linear text segmentation algorithms are widely used as an essential step in many natural language processing tasks, such as information retrieval and document summarization. In information retrieval, to segment a long document into distinct topics is useful because only the topical segments relevant to the user's needs are retrieved [1]. It not only provides more accurate information to the user, but also reduces the user's burden to read the whole document. In document summarization, a long document is often divided into topics and then each topic is summarized independently [2]. A text segmentation algorithm is usually applied as the first step of these tasks.

Segmentation accuracy and computational complexity are two critical issues in linear text segmentation algorithm design. In the past, many well-known linear text segmentation algorithms have been proposed [3, 4, 5, 6]. Although it has been proven that these algorithms either improve the segmentation accuracy or provide efficient processing of linear text segmentation, they can hardly deal with both issues at the same time. Some algorithms provide better segmentation accuracy, but suffer from expensive computational complexity; some algorithms provide efficient processing of linear text segmentation, but suffer from lower segmentation accuracy [7].

In recent years, some algorithms, such as *TSF* [7], are proposed to provide high segmentation accuracy in a reasonable processing time. However, most of the algorithms rely heavily on a training phase or manpower to designate parameters used in the algorithms. The training phase is time consuming, while to designate parameters by manpower increases user's burden. Moreover, it is difficult to determine manually suitable parameters.

To tackle the problems mentioned above, a novel efficient linear text segmentation algorithm based on Hierarchical Agglomerative Clustering (HAC) is presented in this article. The proposed linear text segmentation algorithm dynamically designates parameters according to the metadata of a text, and hence neither training data nor user involvement is needed. In other words, the algorithm requires only the given text, no other information or preprocessing is needed. It not only reduces the user's burden, but also increases the utility of the proposed linear text segmentation algorithm. Segmentation accuracy of the proposed linear text segmentation algorithm is evaluated with a most commonly used test collection, created by Choi [3]. Also, the computational complexity of the proposed algorithm is analyzed. Experimental results show that the proposed algorithm not only provides comparable segmentation accuracy with several well-known linear text segmentation algorithms, but also provides a linear time computational complexity without any auxiliary knowledge base, parameter setting, or user involvement.

978-0-7695-4584-4/11 $26.00 © 2011 IEEE
DOI 10.1109/CIS.2011.240

II. Related Works

Existing text segmentation algorithms can be divided into two major categories [3]. The first category is lexical cohesion methods. The second category is multi-source methods. Lexical cohesion methods detect cohesion using word stem repetition, context vectors, entity repetition, semantic similarity, word distance model, and word frequency model. Multi-source methods combine lexical cohesion with other indicators of topic shift, including cue phrase, prosodic features, reference, syntax and lexical attraction using decision trees and probabilistic models.

In the literatures, some efficient linear text segmentation algorithms are proposed, such as *TextTiling* [5]. *TextTiling* is a well-known linear time text segmentation algorithm proposed by Hearst. It uses a sliding window approach to segment a text. The similarities between adjacent blocks within the text are computed to detect topic changes. The computed similarities are smoothed, and used to identify topic boundaries by a cutoff function. Although *TextTiling* is an efficient linear text segmentation method (the complexity is linear time), it suffers from lower segmentation accuracy.

There are some more complex linear text segmentation algorithms, such as *C99* [3]. Sentence-similarity matrix, consists of similarities between all sentences within a text, is frequently adopted in these algorithms. For example, *C99* proposed by Choi [3] segments a text by combining a rank matrix, transformed from the sentence-similarity matrix, and divisive clustering. Some algorithms apply the similarity matrix to detect the optimal topic boundaries by using dynamic programming [4, 6]. Owing to the cost of constructing the sentence-similarity matrix is $O(n^2)$, where n represents the number of sentences in a text, such algorithms suffer from higher computational complexity [7].

To tackle the problems observed above, Kern and Granitzer proposed an efficient linear text segmentation algorithm, called *TSF* [7]. Similar to *TextTiling*, *TSF* identifies topic boundaries using a sliding window. The differences between *TSF* and *TextTiling* are: 1) in *TSF*, a term vector is built per sentence, instead of merging all terms within a block into one term vector in *TextTiling*; 2) both the inner similarity of a block of sentences and the outer similarity of adjacent blocks of sentences are considered when evaluating a potential segmentation position.3) In *TSF* the smoothing happens implicitly by using the average of the sentence similarities, instead of smoothing the similarity between adjacent blocks *in TextTiling*. It has been proven that *TSF* has an $O(n)$ computational complexity, and provides a comparable accuracy when compared with several higher computational complexity algorithms. However, two parameters need to be provided by the user, say the size of block and the threshold to identify candidate topic boundaries. It may increase the user's burden and the parameters provided may not always be suitable to reflect the real metadata.

Based on the above observations, an efficient linear text segmentation algorithm based on Hierarchical Agglomerative Clustering (HAC) [8] will be presented in the next section. The proposed linear text segmentation algorithm provides comparable segmentation accuracy in a linear time complexity, i.e., $O(n)$. Especially, no parameter setting is required.

III. Text Segmentation Based on Hierarchical Agglomerative Clustering (TSHAC)

In this section, an efficient linear text segmentation algorithm (called *TSHAC*), which considers both computational complexity and segmentation accuracy, is proposed. The process of *TSHAC* consists of 4 steps. At first, a long text is preprocessed; tokenization, stopword removal, and stemming are conducted to construct the vocabulary of the text. After text preprocessing, the text can be represented as vectors, each of which represents a sentence within the text. A part of sentence similarities are then computed to construct the sentence-similarity matrix. Finally, the optimal topic boundaries are identified by the proposed algorithm.

A. Text Preprocessing

In general, the process of text preprocessing can be divided into 3 stages: 1) tokenization; 2) stopword removal; 3) stemming. Punctuation is firstly removed from a long text, and the sentences within the text are then converted into a stream of words. Subsequently, generic stopwords are removed. The rest of words are stemmed and regarded as the vocabulary of the long text.

In 2003, Ji et al. [6] devise a new idea of document-dependent stopword removal. In contrast to the fixed set of generic stopwords, document-dependent stopwords are the words that are useful in discriminating among several different documents but are rather harmful in detecting subtopics in a document. Both generic stopword removal and document-dependent stopword removal are adopted in our proposed linear text segmentation algorithm.

B. Text Representation

Given a long text T, a sentence s_j contained in T can be represented as a vector in order:

$$s_j = \{w_{1j}, w_{2j}, ..., w_{ij}, ...w_{kj}\}$$

where w_{ij} represents the word weight of the ith word in s_j; k represents the total number of words in the vocabulary V constructed from the long text. In our proposed algorithm, w_{ij} can be computed by any formula, such as TFxIDF, that the word weight can be suitably represented. As the core contribution of the proposed algorithm is the process of hierarchical agglomerative clustering for linear text segmentation, here we use the same formula as in *TSF* for comparison purpose. The word weight w_{ij} is computed by a variant of TFxIDF [7]:

$$w_{ij} = \frac{\sqrt{termFreq_{ij}}}{\sqrt{tokenCount_j}} (\log(\frac{docCount}{docFreq_i + 1}) + 1),$$

where $termFreq_{ij}$ represents the occurrences of word i in sentence j; $tokenCont_j$ represents the total number of words

in sentence *j*; *docCount* represents the number of documents from a corpus; $docFreq_i$ represents the total number of documents contain word *i*.

C. Sentence-Similarity Matrix Construction

In previous researches, the sentence-similarity matrix is usually constructed by computing the similarities of all pairs of sentences within a text. The similarity of two sentences is usually computed by cosine measurement:

$$Co\sin e(s_i, s_j) = \frac{\sum_k w_{ik} w_{jk}}{\sqrt{\sum_k w_{ik}^2}\sqrt{\sum_k w_{jk}^2}},$$

where s_i represents the *i*th sentence and s_j represents the *j*th sentence in text *T* respectively. As mentioned in Section II, the construction of such a sentence-similarity matrix costs $O(n^2)$, where *n* represents the number of sentences.

To reduce the cost, *TSHAC* computes only the similarities alongside the main diagonal according to the block size, instead of the complete sentence-similarity matrix. At first, the similarities between each sentence and its neighbors are computed. If the similarity between two adjacent sentences is equal to 0, the sentences are divided into different blocks. Hence, text *T* is roughly divided into several blocks, each of which is composed of consecutive sentences. Subsequently, the number of sentences in the largest block is regarded as the block size. The similarities alongside the diagonal of the sentence-similarity matrix are then computed according to the block size.

For example, assume that the block size is 2, only the elements (i, j) under $i = 1, 2, \ldots, 7$ and $j = i + 1, i + 2, i + 3$ (i.e., 2 × block size - 1), where $j \leq 7$, are computed. Each element (i, j) in the sentence-similarity represents the similarity between sentences *i* and *j* (i.e., $Cosine(s_i, s_j)$). Because all the elements in the diagonal of the sentence-similarity matrix are equal to 1 and the sentence-similarity matrix is symmetric, only the upper triangle of the sentence-similarity matrix needs to be computed. An example of the sentence-similarity matrix is shown as Fig. 1.

The construction of the sentence-similarity matrix is similar to *TSF* [7]. However, in the proposed algorithm, the block size is dynamically determined according to the metadata of a text, instead of statically provided by the user.

	1	2	3	4	5	6	7
1		0.7	0.5	0.3			
2			0.5	0.2	0		
3				0.3	0	0	
4					0	0	0
5						0.7	0.6
6							0.5
7							

Figure 1. Upper triangle of the sentence-similarity matrix.

D. Hierarchical Agglomerative Clustering for Linear Text Segmentation

Hierarchical Agglomerative Clustering (HAC) [8] is a bottom-up hierarchical clustering method. It has been successfully applied to many applications. In *TSHAC*, a sentence is used as the basic processing unit. For more efficient process of linear text segmentation, in the first step, a text is roughly divided into several blocks of sentences by the method mentioned in the last subsection. Each block is then regarded as an independent cluster. In each merging stage, the similarities between a cluster and its neighbors are measured by the following formula:

$$Sim(c_i, c_j) = \frac{2 \times \sum_{s_m \in c_i} \sum_{s_n \in c_j} sim(s_m, s_n)}{(|c_i| + |c_j|)^2} - STDEV, \quad (1)$$

where $Sim(c_i, c_j)$ represents the similarities of cluster *i* and cluster *j*. The first part of (1) represents the lexical cohesion. The numerator is the total similarities of sentences from c_i with the sentences from c_j. The denominator is used to normalize the lexical cohesion to the range [0, 1]. The second part of (1) represents the standard deviation of the cluster size (i.e., number of sentences in a cluster) after c_i and c_j are merged into a group. In general cases, the number of sentences in each cluster is similar. Based on this observation, smaller standard deviation of the cluster size is preferred [9]. The standard deviation of the cluster size is computed by the following equation:

$$STDEV = \sqrt{\frac{\sum_{j=1}^{\#cluster} (|cluster_j| - \mu)^2}{\#cluster}},$$

where μ represents the average number of sentences within clusters.

Because the sentences within a text are located in order, only adjacent clusters need to be considered in each merging stage (i.e., only $Sim(c_{i-1}, c_i)$ and $Sim(c_i, c_{i+1})$ need to be computed). The nearest pair of adjacent clusters is then merged into a new cluster. The merging process is repeated until all clusters are merged into one universal cluster. A hierarchical cluster tree is then constructed as shown in Fig. 2.

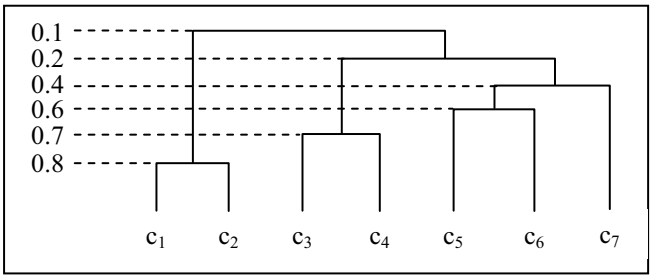

Figure 2. An example of a hierarchical cluster tree.

In order to divide the text into several topical segments, after the hierarchical cluster tree is constructed, the input data objects can be divided into several clusters, each of which represents a topic, by cutting the hierarchical cluster tree at a feasible height. A fitness function is proposed to evaluate each of the potential height hi of the hierarchical cluster tree constructed as follow:

$$f(h_i) = \frac{\sum_{j=1}^{\#seg-1} dissimilarity_j}{\#seg} - STDEV(i) - \frac{\#seg_{len\le 2}}{\#seg} \quad (2)$$

As mention in previous researches [9], when dividing a long text into several topical segments, the sentences within a topical segment should cover the same subtopic. Moreover, sentences among different segments should belong to different subtopics. Therefore, both the average sentence similarity within each segment (*inner similarity*) and the average sentence similarity between two consecutive segments (*outer similarity*) are considered in the fitness function. The *inner similarity* and the *outer similarity* are then combined as *dissimilarity* [7] as follows:

$$dissimilarity_j = \frac{sim_j^{inner} - sim_j^{outer}}{sim_j^{inner}}, \text{ where}$$

$$sim_j^{inner} = \frac{\mu(B_j^{pre}, B_j^{pre}) + \mu(B_j^{post}, B_j^{post})}{2} \text{ and}$$

$$sim_j^{outer} = \mu(B_j^{pre}, B_j^{post}),$$

where μ represents average pair-wise sentence similarities of two blocks; B_i^{pre} represents the block of sentences that precede the potential boundary; B_i^{post} represents the block of sentences that succeed the potential boundary. The higher $dissimilarity_j$ implies higher feasibility of the potential boundary.

On the other hand, the standard deviation of the segment length (as shown in (3)) is also considered in the fitness function. Because the number of sentences in different segment is usually similar [9], a lower standard deviation value of segment size is preferred when we design the fitness function.

$$STDEV(i) = \sqrt{\frac{\sum_{j=1}^{\#seg} (|seg_j| - \mu)^2}{\#seg}} \quad (3)$$

Finally, a topic segment usually contains several sentences that could well describe a topic. To avoid topic segments to be too short, the number of segments which contain less than or equal to 2 sentences is also considered. A lower proportion of short segments is preferred in our fitness function.

E. The Proposed TSHAC Algorithm

The complete process of *TSHAC* is summarized as Fig. 3.

IV. PERFORMANCE EVALUATIONS

To evaluate the performance of *TSHAC*, a publicly available test corpus is adopted. The test corpus was created by Choi [3] and has been commonly used in previous researches. The test corpus consists of 700 samples. A sample is a concatenation of ten text segments. A segment is the first *n* sentences of a randomly selected document from the Brown corpus. The 700 samples are divided into 4 sets according to the range of the number of sentences. Table I shows the statistics of the test corpus, *n* represents the number of sentences.

To compare with several well-known text segmentation algorithms, including *TextTiling* [5], *C99* [3], *U00* [10], *TopSeg02* [11], *AniDiffDynProg03* [6], and *TSF* [7], the commonly used segmentation metric proposed by Beeferman [12] is adopted:

$$p(error|ref,\ hyp,\ k) =$$
$$p(miss|ref,\ hyp,\ different\ segments,\ k)\,p(diff\ segments|ref,\ k) +$$
$$p(false\ alarm|ref,\ hyp,\ same\ segment,\ k)\,p(same\ segment|ref,\ k)$$

The aim of the segmentation metric is to compute the error probability of a randomly chosen pair of words at distance *k* words apart that is inconsistently classified. *ref* represents the true segmentation and *hyp* represents the proposed

The proposed TSHAC algorithm
Input: A long text. Output: Several topical segments.
Step 1. Preprocess a text by tokenization, generic stopword removal, document-dependent stopword removal, and stemming.
Step 2. Determine block size by the method mentioned in Section III-C.
Step 3. Compute the similarities of sentences alongside the diagonal of the sentence-similarity matrix with the block size.
Step 4. Segment the text by the following steps:
Step 4.1. The text is roughly divided into several groups by the method mentioned in Section III-C and each group is regarded as an independent cluster.
Step 4.2. Compute the similarities between adjacent clusters by (1).
Step 4.3. Merge the pair of clusters to achieve the greatest similarity.
Step 4.4. Repeat Step 4.2 and Step 4.3 until only one universal cluster left.
Step 4.5. Cut the hierarchical cluster tree by the predesigned fitness function (shown as (2)) to achieve the highest fitness value.

Figure 3. The porposed TSHAC algorithm.

TABLE I. STATISTICS OF THE TEST CORPUS

Range of *n*	3-11	3-5	6-8	9-11
#samples	400	100	100	100

segmentation. The error probability is composed of the *miss* and the *false alarm* probabilities. The *miss* probability is a conditional probability that the randomly chosen pair of words spans a segment boundary for the true segmentation, but lies in the same segment for the proposed segmentation. The *false alarm* probability is a conditional probability that the randomly chosen pair of words lies in the same segment for the true segmentation, but spans a segment boundary for the proposed segmentation. The lower error probability implies higher segmentation accuracy.

A. Computational Complexity Analysis

As mentioned above, the construction of the complete similarity matrix cost $O(n^2)$, where n is the number of sentences in a text. To reduce the time complexity, in the proposed algorithm, only the similarities alongside the diagonal are computed with the block size. Therefore, the time complexity is $O(nk)$, where n represents the number of sentences and k represents the block size.

In this article, the proposed linear text segmentation algorithm is based on HAC. Unlike the general process of HAC, only the similarities between each cluster and its neighbors are computed at each merging stage. Therefore, in contrast to the general HAC, which takes $O(n^2)$, the proposed *TSHAC* algorithm compute only the similarities between the newly merged cluster and its two neighbors in each merging stage, and hence it takes only $O(n)$. Moreover, the roughly grouping of sentences into blocks will further improve the efficiency of the proposed linear text segmentation algorithm. The actual complexity of the HAC process will be less than or equal to $O(n)$.

According to the above analysis, *TSHAC* is a linear time algorithm of linear text segmentation.

B. Experimental Results

Table II shows the error probability of *TSHAC* and several well-known text segmentation algorithms. Among these algorithms, *TextTiling*, *TSF* and *TSHAC* are linear time algorithms, and the others are non-linear time algorithms [7]. The error probability of the well-known linear text segmentation algorithms are collected from the literatures [3, 7]. The experimental results in Table II are reported when the number of segments is unknown in advance.

From Table II, the proposed linear text segmentation algorithm (i.e., *TSHAC*) outperforms the linear time algorithm, *TextTiling*, and the more complex algorithms, *C99*. *TSHAC* also provides comparable results with other algorithms. Moreover, unlike *TSF*, which requires manpower to designate parameters used, *TSHAC* provides a fully automatic process for linear text segmentation without auxiliary knowledge base, parameter setting, or user involvement.

TABLE II. EXPERIMENTAL RESULTS AND COMPARISON

	3-11	**3-5**	**6-8**	**9-11**
TextTiling	46%	44%	43%	48%
C99	13%	18%	10%	10%
U00	11%	13%	6%	6%
TopSeg	10.74%	7.44%	7.95%	6.75%
AniDiffDynProg03	6.0%	7.1%	5.3%	4.3%
TSF	9.0%	9.3%	6.8%	9.2%
TSHAC	9.84%	9.26%	8.23%	7.54%

V. CONCLUSION AND FUTURE WORKS

The contributions of this article include: 1) Propose a high performance linear text segmentation algorithm based on HAC; 2) The proposed linear text segmentation algorithm achieves comparable segmentation accuracy with efficient processing time; 3) The parameters used in the process of linear text segmentation are automatically designated without auxiliary knowledge base, parameter setting, or user involvement. Therefore, the proposed linear text segmentation algorithm is feasible for segmenting long texts in real time, especially when user involvement is unavailable. Furthermore, the proposed algorithm not only provides topical segments, but also provides hierarchical discourse structure of the text, if necessary. In the future, we plan to implement the proposed linear text segmentation algorithm in some real world applications.

REFERENCES

[1] A. C. Jobbins and L. J. Evett, “Text segmentation using reiteration and collocation”, Proceedings of the COLING-ACL’98, 1998, pp. 614–618.

[2] D. Beeferman, A. Berger, and J. Lafferty, “Text segmentation using exponential models”, In Proceedings of the 2nd Conference on Empirical Methods in Natural Language Processing, 1997, pages 35-46.

[3] F.Y.Y. Choi, "Advances in domain independent linear text segmentation", in Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference, 2000, pp. 26-33.

[4] P. Fragkou, V. Petridis, and A. Kehagias, "A dynamic programming algorithm for linear text segmentation", Journal of Intelligent Information Systems, 23(2), pp. 179-197, 2004.

[5] M.A. Hearst, "TextTiling: Segmenting text into multi-paragraph subtopic passages", Computational linguistics, 23(1), pp. 33-64, 1997.

[6] X. Ji, and H. Zha, "Domain-independent text segmentation using anisotropic diffusion and dynamic programming", In Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, 2003, pp. 322-329.

[7] R. Kern, and M. Granitzer, "Efficient linear text segmentation based on information retrieval techniques", In Proceedings of the International Conference on Management of Emergent Digital EcoSystems, 2009, pp. 167-171.

[8] P. Willett, “Recent trends in hierarchic document clustering: a critical review”, Information Processing & Management, 24(5), pp. 577-597, 1988.

[9] J. W. Wu, J. C. R. Tseng and W. N. Tsai, “A discrete particle swarm optimization algorithm for domain independent linear text segmentation”, 2010 IEEE International Conference on Granular Computing (GrC), 2010, pp. 519-524.

[10] M. Utiyama and H. Isahara, “A statistical model for domain-independent text segmentation”, In annual meeting-association for computational linguistics, volume 39, pages 491-498, 2001.

[11] T. Brants, F. Chen, and I. Tsochantaridis, “Topic-based document segmentation with probabilistic latent semantic analysis”, In Proceedings of the eleventh international conference on Information and knowledge management, 2002, pp. 211-218.

[12] D. Beeferman, A. Berger, and J. Lafferty, "Statistical models for text segmentation", Machine learning, 34(1), pp. 177-210, 1999.