Evaluation of Fast Spoken Term Detection Using a Suffix Array

Kouichi Katsurada, Shinta Sawada, Shigeki Teshima, Yurie Iribe, Tsuneo Nitta

Toyohashi University of Technology, Toyohashi, Japan

{katsurada, nitta}@cs.tut.ac.jp, {sawada, teshima}@vox.cs.tut.ac.jp, iribe@imc.tut.ac.jp

Abstract

We previously proposed [1] fast spoken term detection that uses a suffix array as a data structure for searching a large-scale speech documents. In this method, a keyword is divided into sub-keywords, and the phoneme sequences that contain two or more sub-keywords are output as results. Although the search is executed very quickly on a 10,000-h speech database, we only proposed a variety of matching procedures in [1]. In this paper, we compare different varieties of matching procedures in which the number of phonemes in a sub-keyword and the required number of sub-keywords to be contained in a search result are different. We also compare the performance and the process time of our method with typical spoken term detection using an inverted index.

Index Terms: spoken term detection, large scale speech document, suffix array, keyword division, iterative lengthening search

1. Introduction

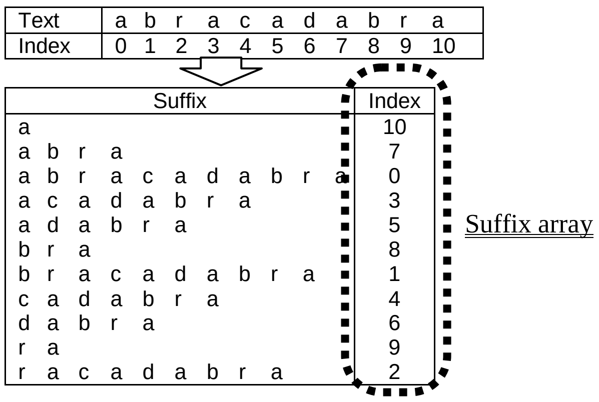
Fast spoken term detection is essential in effectively utilizing large-scale speech documents. A considerable number of studies have been conducted on this topic, and reasonable performances have been achieved in detecting keywords from a speech database [2][3]. Recently, some studies have focused on search speed [4][5][6] because quickness is important when a search is executed on very large speech/video databases such as the digital archives of TV/radio programs or video sites on the internet. However, most existing methods are not fast enough on a 10,000-h speech database.

For this situation, we proposed fast spoken term detection for large-scale speech documents [1]. It reduces search time by employing a suffix array as the data structure and introducing other techniques such as keyword division and an iterative lengthening search. However, in our previous paper [1], we only provided a variety of matching procedures and evaluated its performance. In this paper we conduct some experiments under various conditions in which the number of phonemes in a sub-keyword and the required number of sub-keywords to be contained in a search result are different. We also compare our method with typical spoken term detection using an inverted index.

2. Keyword detection using a suffix array

2.1. Structure of the suffix array

A suffix array [7] is a data structure that is used for quickly searching for a keyword in a text database. We employ it for phoneme-based keyword detection. It holds sorted indexes of all suffixes of the phoneme string in a database. Figure 1 shows a sample of a suffix array constructed from the character string[1] "abracadabra." Indexes in the figure represent the position at which the suffixes start in the string. Because the indexes are sorted by the dictionary order of suffixes, we can use a binary search to detect a keyword. Moreover, large memory space is not required because the array holds only the indexes.



Text	a	b	r	a	c	a	d	a	b	r	a
Index	0	1	2	3	4	5	6	7	8	9	10

Suffix	Index
a	10
a b r a	7
a b r a c a d a b r a	0
a c a d a b r a	3
a d a b r a	5
b r a	8
b r a c a d a b r a	1
c a d a b r a	4
d a b r a	6
r a	9
r a c a d a b r a	2

Suffix array

Figure 1: *An example suffix array.*

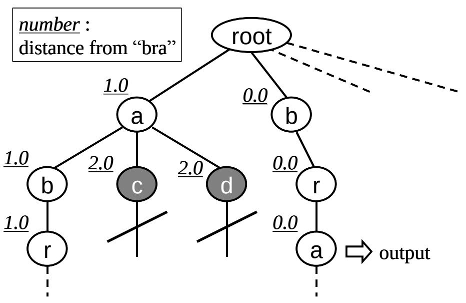


Figure 2: *Similarity search on a suffix array*

2.2. Similarity search on a suffix array

In the case of speech data, we are unable to ignore recognition errors. Since the original suffix array is intended to search for an exact string, we need to introduce a technique for a similarity search together with the suffix array. For this purpose, a search algorithm using DP-matching (or Dynamic Time Warping (DTW)) on the suffix array is proposed [8]. This algorithm regards a suffix array as a tree, and DP-matching is applied to all paths from the root of the tree. If the distance between a keyword and a path is not more than a threshold value, the path is output as a search result, but if the distance is more than a threshold at some node, the search is terminated at the node.

Figure 2 shows an example in which the keyword "bra" is detected from the character string "abracadabra." In this example, the threshold is assumed to be 1.0 and the distance between different characters is defined as 1.0. In the example, the descendant nodes of the paths "ac" and "ad" are cut off because their distances are greater than the threshold. As a result, "bra," "abra," and some other strings are detected within the threshold. Finally, indexes 8, 1, 7, 0, etc., are output as results by referring to the suffix array shown in Figure 1.

[1] In this example, we use a character string instead of a phoneme string for simplicity of explanation.

	a	i	u	e	o	k	s	...
low	-	+	+	-	-	-	-	
high	+	-	-	-	-	+	-	
plosive	-	-	-	-	-	+	-	
affricative	-	-	-	-	-	-	-	
:								

Figure 3: *Table of distinctive phonetic features*.

3. Keyword detection from a speech database

3.1. Distinctive phonetic features

Before starting the keyword search, a speech database is transformed into a phoneme sequence by means of Large Vocabulary Continuous Speech Recognition (LVCSR) or some other methods. To apply DP-matching to the phoneme sequence, distinctive phonetic feature-based distance is introduced. The distinctive phonetic features represent a phoneme using fifteen articulatory features such as plosive and affricative. Figure 3 shows a fragment of the relationship between phonemes and articulatory features. We used the hamming distance of these features to calculate the distance between two strings of phonemes.

The definition of distance used in DP-matching is given by the following equation:

$$P_{i,j} = \min \begin{cases} P_{i-1,j} + D \\ P_{i-1,j-1} + d(a_i, b_j) \\ P_{i,j-1} + I \end{cases} \tag{1}$$

In this equation, a_i is a phoneme in a keyword $a_1a_2 \ldots a_K$; b_j is a phoneme in a speech database; $P_{i,j}$ is the distance between $a_1a_2 \ldots ai$ and $b_1b_2 \ldots b_j$; D and I are the deletion and insertion penalties; and $d(a_i, b_j)$ is the hamming distance calculated from the articulatory features of a_i and b_j.

3.2. Keyword division

According to the algorithm described in Section 2.2, all paths within the threshold are temporarily stored in the memory while DP-matching is applied. Therefore, if the threshold is large, process time will increase exponentially according to the depth of the tree. Because the threshold increases in proportion to the length of the keyword, an exponential increase in process time will result if the keyword is long. To avoid this problem, a long keyword is divided into short sub-keywords, which are then searched for on the array instead of the original keyword. Of course, the results obtained by using sub-keywords, hereinafter called the candidates, may not actually match the results when the original keyword is used. Thus, to confirm the validity of the candidates, DP-matching process is repeated.

Even though the above division reduces the process time, a large number of candidates can be detected. To reduce the candidates, we proposed to try detecting at least two adjacent candidates of different sub-keywords on the phoneme sequence in paper [1]. Figure 4 illustrates the outline of a keyword search. The search algorithm is summarized as follows.

(I) Divide the keyword into sub-keywords.

(II) Search for the sub-keywords in the suffix array and find candidates.

(III) Filter the candidates by detecting adjacent candidates.

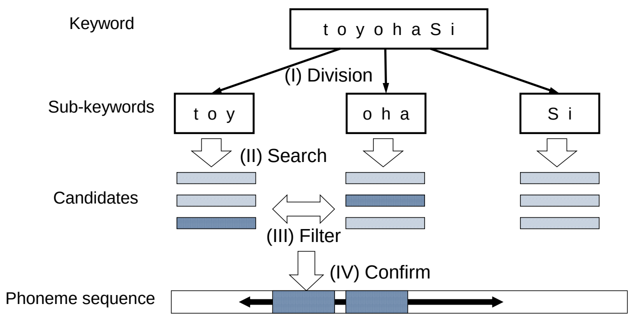


Figure 4: *Outline of keyword search*.

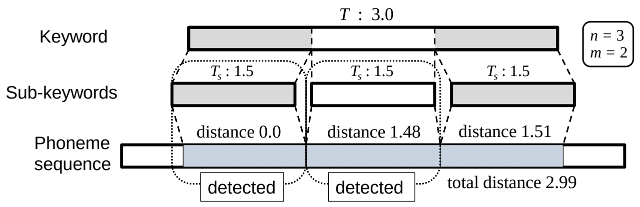


Figure 5: *An example of threshold modification*.

(IV) Confirm the validity of the candidate by DP-matching.

In this paper, we generalize this method so as to detect at least m candidates of n sub-keywords at the same time. For this purpose, we modify the threshold assigned to each sub-keyword by using the following equation.

$$T_s = \frac{T}{n - m + 1} \tag{2}$$

In the above equation, T is the threshold assigned to the original keyword. The keyword is divided into n sub-keywords and at least m of them are detected. T_s is the modified threshold assigned to a sub-keyword. This modification and the following validity confirmation process guarantee that the same results will be detected in any conditions. Figure 5 shows an example. In this figure, $T = 3.0$, $n = 3$, $m = 2$, and there is a sequence whose distance from the keyword is 2.99. The sub-strings corresponding to the sub-keyword have distances of 0.0, 1.48, and 1.51. In this case T_s becomes 1.5, and two sub-keywords are detected.

4. Experiments

4.1. Experimental setup

Experiments were carried out on a PC with a 3.4 GHz Intel Core i7-2600 processor and 8 GB of main memory. The CSJ (Corpus of Spontaneous Japanese: 604 hours) and the CSJ Spoken Term Detection test collection [9] are used as a data set to evaluate the effectiveness of the proposed method. For speech recognition, we used Julius [10] along with its default speaker independent acoustic model. The language model is constructed from 2,525 lectures from the CSJ corpus (27,000 words), which is specified by the test collection. The correct rate and accuracy of the phonemes were 75% and 71%, respectively.

4.2. Basic search performance

First, we compare the performance of our method with a method using the phoneme 3-gram inverted index that is utilized in Kanda's very fast keyword detection scheme [4]. In this experiment, 50 OOV words specified in the CSJ STD test collection are used.

Table 1. *Performance of our method*.

Threshold	0.0	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8
Recall rate (%)	3.00	3.67	20.0	31.7	38.7	58.7	70.3	78.0	86.3	92.3
Precision rate (%)	50.0	60.0	49.0	38.5	22.2	11.6	4.12	0.373	0.014	0.002
TWV score	0.0300	0.0366	0.198	0.304	0.331	0.0753	-0.721	-5.37	-23.6	-79.6

Table 2. *Performance of phoneme 3-gram inverted index-based method*.

Threshold	100	90	80	70	60	50	40	30	20	10	5
Recall rate (%)	3.00	4.00	5.67	11.0	18.0	33.0	39.3	53.7	68.3	78.0	81.0
Precision rate (%)	50.0	60.0	34.2	41.5	22.1	13.3	1.92	0.831	0.027	0.003	0.001
TWV score	0.0300	0.0400	0.0563	0.108	0.171	0.288	0.0978	-0.261	-6.89	-33.5	-66.9

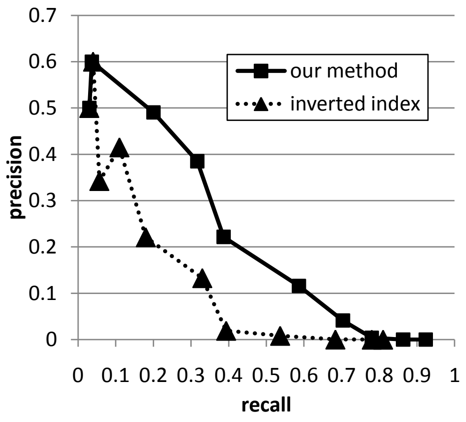


Figure 6. *Precision-recall curve of our method and inverted index-based method.*

Tables 1 and 2 list the recall rate, precision rates, and the Term Weighted Value (TWV) score [2] of our method and the method using the phoneme 3-gram inverted index, respectively. In Table 1, the threshold indicates the average threshold per phoneme (i.e., if the number of phonemes in a keyword is 24, and T = 24, then the average threshold is 1.0.). In Table 2, all phoneme 3-grams of the keyword are searched for in the phoneme sequence using exact matching, and as in step (III) of our method, the adjacent 3-grams are found. The percentage of numbers of adjacent 3-grams to be detected is given as a threshold value. Figure 6 illustrates the recall-precision curve of the keyword search.

These results indicate that our method is slightly better than the method using the phoneme 3-gram inverted index. The discrepancy is caused by the difference in the search method. In our method, DP-matching and a distinctive phonetic feature can evaluate similarities better than the inverted index-based method, which uses an exact matching of phoneme 3-grams.

4.3. Ideal number of phonemes in a sub-keyword

In paper [1], the number of phonemes in a sub-keyword is fixed at six from the results of preliminary experiments. In this paper, we examine the details of process time and investigate the ideal number of phonemes in a sub-keyword. Table 3 shows the results in which the number of phonemes in a keyword is 24, and the numbers of phonemes in the sub-keywords are 4 to 12. The columns of the table, from left to right, show the number of phonemes in a sub-keyword (value of n), the processing times in steps (I), (II), (III), and (IV), and the total processing time. The number of adjacent candidates m in formula (2) is set at 1, and the threshold is set at 1.0 per phoneme.

As shown in the table, if there are few phonemes in a sub-keyword, the processing time of step (II) is short and that of step (IV) is long. However, if the number of phonemes is

Table 3. *Processing time when the number of phonemes in a sub-keyword is different*.

Number of phonemes in sub-keyword	Step(I) (ms)	Step(II) (ms)	Step(III) (ms)	Step(IV) (ms)	Total time (ms)
4 (n = 6)	0	31	16	1,248	1,295
5 (n = 4)	0	63	15	1561	1,639
6 (n = 4)	0	63	0	187	250
7 (n = 3)	0	124	0	163	287
8 (n = 3)	0	140	0	47	187
9 (n = 2)	0	671	0	125	796
10 (n = 2)	0	812	0	31	843
11 (n = 2)	0	671	0	0	671
12 (n = 2)	0	858	0	0	858

Note: The original keyword is "koNSumaKarakurarisutiQku."

Table 4. *Details of the processing time under each condition*.

Conditions	Step(I) (ms)	Step(II) (ms)	Step(III) (ms)	Step(IV) (ms)	Total time (ms)
Condition (a)	0	11,294	0	0	11,294
Condition (b)	0	47	0	187	234
Condition (c)	0	188	124	437	749
Condition (d)	0	2,184	3,339	0	5,523
Condition (e)	0	60,996	111,852	32	172,880

larger, the processing time for step (II) lengthens but that of step (IV) is shortened. This is due to the fact that the search space narrows if n is a large value (that is, T_s is a small value), but many candidates will be detected because the sub-keywords are short, which increases the confirmation time in step (IV). Table 3 indicates that the number of phonemes is best set at 6 to 8 for an efficient search.

4.4. Ideal number of sub-keywords to be detected

To investigate the affect of m in formula (2) and effectiveness of keyword division, we compare the following five conditions.

(a) Keywords are searched for without division.

(b) Keywords are divided and m is set at 1.

(c) Keywords are divided and m is set at 2.

(d) Keywords are divided and m is set at 3.

(e) Keywords are divided and m is set at 4.

In conditions (b) through (e), the thresholds assigned to sub-keywords are modified according to formula (2). The length of the sub-keywords is set to 6-phonemes based on the results in section 4.3. If the original keywords are not long enough to be divided into several sub-keywords, they are divided into the highest possible number of sub-keywords, and the highest possible numbers of sub-keywords are detected (e.g., if the length of the keyword in condition (e) is 12, it is divided into two sub-keywords, and two keywords are detected.). Figure 7 shows the recall-processing-time curve for each condition. It shows the average processing time when

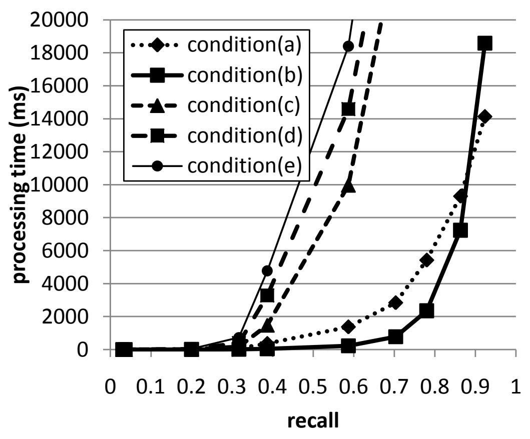


Figure 7. *Processing time in conditions (a) through (e).*

detecting 50 OOV keywords. Table 4 shows the detailed processing time in detecting the phoneme string "koUSumaKarakutarisutiqku" (24 phonemes).

We can easily confirm that keyword division works well for speeding up keyword detection by comparing conditions (a) and (b) in Figure 7. As shown in Table 4, we can also confirm that the search time (step (II)) dramatically decreases by applying keyword division. Even though a confirmation time (step (IV)) is required under condition (b), the total processing time is considerably reduced.

However, we were unable to confirm the effect of detecting adjacent candidates as shown in Figure 7, which shows that condition (b) is the fastest among conditions (b) through (e). As shown in Table 4, step (II) only requires 47 ms under condition (b), while step (IV) needs 187 ms. On the other hand, in the case of condition (e), step (II) requires more than 60,000 ms. The reason for the expansion in the processing time is that T_s in formula (2) under condition (e) is four-times larger than that under condition (b). This tendency of expansion is also observed in other keywords. Therefore, condition (c) through (e) could not reduce the search time effectively.

4.5. Comparison of search speed with the other approaches

In this section, we compare the speed of our method with two approaches: continuous DP-matching, and the phoneme 3-gram inverted index used in Kanda's method [4]. Continuous DP-matching is a very simple method for acquiring the distances between the keyword and the sub-sequences in a phoneme sequence by calculating the distances from the beginning to the end of the sequence, continuously. Our method employs condition (b) described above that is faster than the other conditions.

Figure 8 illustrates the results. The figure shows that the processing time of DP-matching is almost constant in any conditions and that of inverted index is almost constant when the recall rate is low. On the other hand, the processing time of our method is very short when the recall rate is low (that is, the threshold is small and the precision rate is high). This is because a DP-matching based approach cannot reduce the processing time since it calculates the distance from the beginning to the end of the phoneme sequence. At the same time, an inverted index based approach also cannot reduce the time as it first detects all 3-grams on the sequence, and then finds adjacent 3-grams, as in step (III) of our method. The phoneme 3-grams are very short, and a lot of candidates are found, which increases the processing time. Our method avoids such overhead by directly reducing the search space by

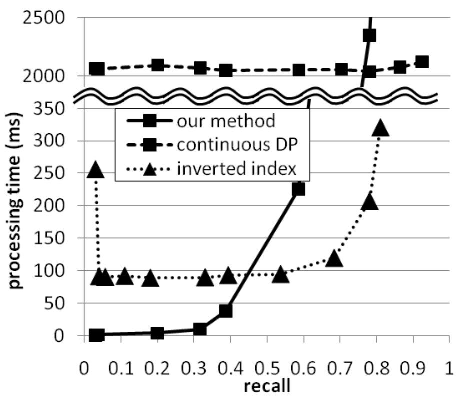


Figure 8. *Comparison of process time between our method and the other methods.*

applying DP-matching on the suffix array when threshold values are small.

5. Conclusions

This paper evaluated the fast keyword detection technique using a suffix array. The results show that a suffix array and keyword division work well for rapidly detecting the results without spending a large memory space. This characteristic is desirable for utilizing large scale speech documents on the internet, call center, broadcast station and so on. The remaining study is to combine some conditions to improve precision rate, and to enlarge the speech database size to 100,000-h scale.

6. Acknowledgements

This work was supported by the Grant-in-Aid for Young Scientists (B) 20700156 2010 and the Grant-in-Aid for Scientific Research (B) 22300060 2010 from MEXT, Japan.

7. References

[1] Katsurada, K., Teshima, S. and Nitta, T., "Fast Keyword Detection Using Suffix Array", InterSpeech2009, pp.2147-2150, 2009

[2] Fiscus, J., Ajot, J., Garofolo, J. and Doddington, G., "Results of the 2006 Spoken Term Detection Evaluation", SIGIR'07 Workshop in Searching Spontaneous Conversational Speech, 2007.

[3] Smidl, L. and Psutka, J.V., "Comparison of Keyword Spotting Methods for Searching in Speech", InterSpeech2006, pp.1894-1897, 2006.

[4] Kanda, N., Sagawa, H., Sumiyoshi, T., and Obuchi, Y., "Open-vocabulary keyword detection from super-large scale speech database", IEEE MMSP 2008, pp.939-944, 2008.

[5] Pinto, J., Szoke, I, Prasanna, S. R. M. and Hermansky, H., "Fast Approximate Spoken Term Detection from Sequence of Phonemes", SIGIR '08 Workshop, pp.28-33, 2008.

[6] Wallace, R., Vogt, R. and Sridharan, S., "Spoken term detection using fast phonetic decoding", ICASSP'09, pp.2135-2138, 2009.

[7] Manber, U. and Myers, G., "Suffix arrays: A new method for on-line string searches", SIAM J. Computation, vol.22, no.5, pp.935-948, 1993.

[8] Yamasita, T. and Matsumoto, Y., "Full Text Approximate String Search using Suffix Arrays", IPSJ SIG Technical Reports 1997-NL-121, pp.23-30, 1997. (In Japanese)

[9] Itoh, Y. et al., "Constructing Japanese Test Collections for Spoken Term Detection", InterSpeech2010, pp.677-680, 2010.

[10] Kawahara, T., Lee, A., Takeda, K., Itou, K. and Shikano, K., "Recent Progress of Open-Source LVCSR Engine Julius and Japanese Model Repository", ICSLP2004, pp.688-691, 2004.