Acceleration of Spoken Term Detection Using a Suffix Array by Assigning Optimal Threshold Values to Sub-Keywords

Kouichi Katsurada[1], *Seiichi Miura*[1], *Kheang Seng*[1], *Yurie Iribe*[2], *Tsuneo Nitta*[13]

[1]Toyohashi University of Technology, Toyohashi, Japan
[2]Aichi Prefectural University, Nagakute, Japan
[3]Waseda University, Tokyo, Japan

{katsurada, nitta}@cs.tut.ac.jp, {miura, kheang}@vox.cs.tut.ac.jp, iribe@imc.tut.ac.jp

Abstract

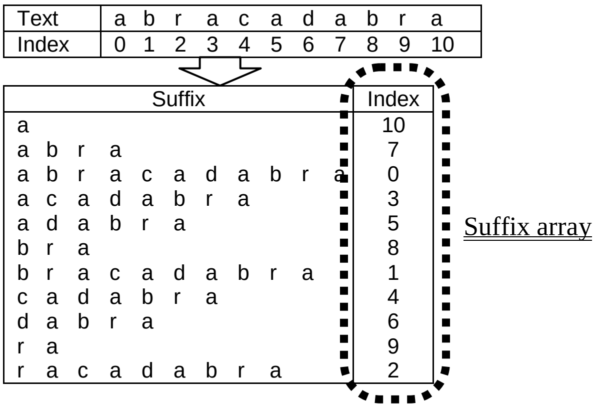
We previously proposed a fast spoken term detection method that uses a suffix array data structure for searching large-scale speech documents. The method reduces search time via techniques such as keyword division and iterative lengthening search. In this paper, we propose a statistical method of assigning different threshold values to sub-keywords to further accelerate search. Specifically, the method estimates the numbers of results for keyword searches and then reduces them by adjusting the threshold values assigned to sub-keywords. We also investigate the theoretical condition that must be satisfied by these threshold values. Experiments show that the proposed search method is 10% to 30% faster than previous methods.

Index Terms: spoken term detection, suffix array, keyword division, threshold value assignment

1. Introduction

Fast spoken term detection has become an essential function in utilizing large-scale speech documents. Many studies have been conducted on this topic, and reasonable progress has been made in detecting keywords from a speech database [3][4]. Recently, studies have focused on search speed [5][6][7] as a priority for large speech/video databases, such as digital archives of TV/radio programs or video sites on the internet. Nevertheless, existing methods are still too slow for useful processing of a 10,000-h speech database.

Recognizing this, we previously proposed a fast spoken term detection method for large-scale speech documents [1][2]. This method reduced search time using a suffix array data structure and other techniques, such as keyword division and iterative lengthening search. Using the proposed method, a keyword is divided into sub-keywords, which are then searched instead of the initial keyword. To guarantee that the search results are unchanged from the original keyword search, we control the search threshold value attached to each sub-keyword according to the number of sub-keywords. However, in this previous proposal, we considered only the case in which the threshold value is assigned equally to each sub-keyword. In this paper, we investigate the theoretical condition that must be satisfied by a threshold value when different values are assigned to each sub-keyword. We also propose a statistical indicator that can help identify optimal threshold values for efficient search.



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

Figure 1: *Example suffix array.*

2. Keyword detection using suffix array

2.1. Structure of suffix array

A suffix array [8] is a data structure that is used to quickly search for a keyword in a text database. We have previously employed it for phoneme-based keyword detection. The data structure holds sorted indexes for all suffixes of phoneme sequences in a database. Figure 1 shows a sample of a suffix array constructed from the character string[1] “abracadabra.” The indexes in the figure represent the positions at which the suffixes start in the string. Because the indexes are sorted alphabetically, we can use an efficient binary search to detect a keyword. Moreover, because the array holds only the indexes, memory requirements are modest.

2.2. Similarity search on suffix array

The original suffix array was designed to support exact string matches. To adapt it for use with speech recognition and speech data, we employed a search algorithm using DP-matching (or dynamic time warping (DTW)) on the suffix array that is proposed by Yamashita et al. [9]. This algorithm regards the 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. Otherwise, the search is terminated at a higher node.

Figure 2 shows an example in which the keyword “bra” is detected within the character string “abracadabra.” In this

[1] In this example, we use a character string instead of a phoneme string to simplify the explanation.

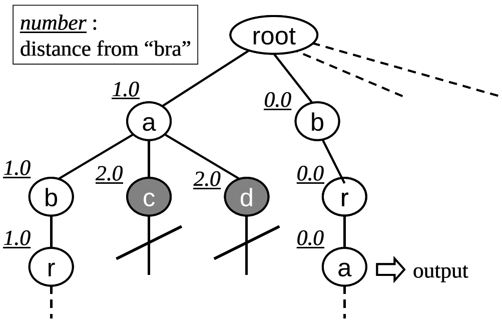


Figure 2: *Similarity search on suffix array.*

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

Figure 3: *Table of distinctive phonetic features.*

example, the threshold is assumed to be 1.0, and edit distance is used to calculate the distance between two strings. In the example, we cut off the descendant nodes of the paths "ac" and "ad" because their distances are greater than the threshold. As a result, "bra," "abra," and some other strings are detected within the threshold. Finally, by referring to the suffix array shown in Figure 1, we output the indexes 8, 1, 7, 0, etc. as results.

In the DP-matching process, we use distinctive phonetic features to calculate the distance between phonemes. These features represent a phoneme using fifteen articulatory features. Figure 3 shows a portion of the relationship between Japanese phonemes and these articulatory features. We used the Hamming distance of these features to calculate the distance between two strings of phonemes.

2.3. Iterative lengthening search

If the threshold is set at a large value, the recall rate of the search will increase because many results are output, whereas the precision rate will decrease and the search time will increase exponentially. On the other hand, if the threshold is a small value, the precision rate will increase and the search time will be small. Considering these characteristics, we employed an iterative lengthening search algorithm for keyword detection. In this search, the threshold is initially set at a small number to output accurate results rapidly, and as the user is checking the former results, the threshold is slowly increased and the search is executed iteratively.

3. Keyword division

3.1. Outline of keyword division and search procedure

In the algorithm described in Section 2.2, all paths that lie within the threshold are temporarily stored in memory while DP-matching is applied. Thus, if the threshold is large, the processing time will increase exponentially according to the depth of the tree. Because the threshold must increase in proportion to the length of the keyword, an exponential increase in processing time will result if the keyword is long. To avoid this problem, we divide a long keyword into short sub-keywords, and searched for them instead of the original keyword.

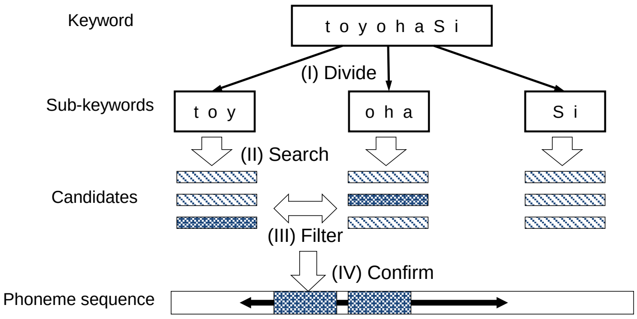


Figure 4: *Outline of keyword search.*

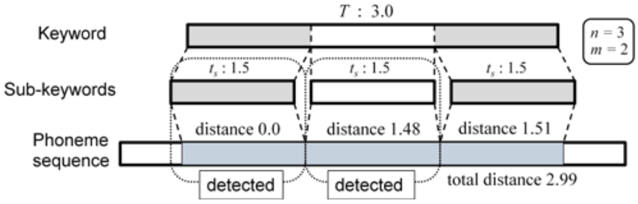


Figure 5: *Example where equation* (2) *is satisfied.*

Of course, the results obtained using sub-keywords, hereafter referred to as the "candidates," may not actually match the results when the original keyword is used. To guarantee that the same results are obtained, we have proposed a search algorithm composed of the following four steps: (I) divide, (II) search, (III) filter, and (IV) confirm. In step (I), the original keyword (threshold value: T) is divided into n sub-keywords with the threshold of t_1 ... t_n. In step (II), n sub-keywords are searched for in the phoneme sequence. In step (III), the phoneme sequences with at least m candidates in adjacent regions are extracted. In step (IV), the phoneme sequences whose distance from the original keyword is less than T are output as the final results.

A remaining issue is how best to determine t_1 ... t_n for the above procedure. In the following section, we discuss the condition that needs to be satisfied by t_1 ... t_n.

3.2. Condition to be satisfied by threshold values

In [2], we presented a condition that ensures that all candidates are detected when the following threshold value is attached to each sub-keyword:

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

where T is the threshold assigned to the original keyword. The keyword is divided into n sub-keywords of which at least m are detected. t_s is then the modified threshold assigned to a sub-keyword. A proof of equation (2) is provided in the Appendix, and Figure 5 provides an example of its application. 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-keywords have distances of 0.0, 1.48, and 1.51. Note that, in this case, t_s becomes 1.5, and two sub-keywords can be detected.

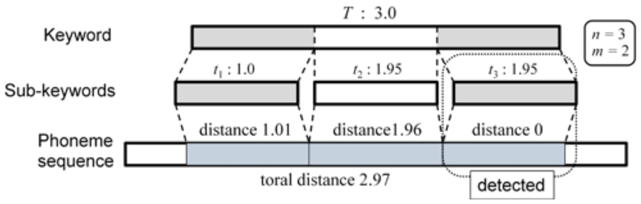


Figure 6: *Example where Inequality* (3) *is not satisfied.*

Table 1: *Search time and number of candidates.*

Average threshold per phoneme	Search time [sec]		Number of candidates	
	(A)	(B)	(A)	(B)
1.0	0.076	0.065	946,313	587,103
1.2	0.161	0.124	2,753,235	1,740,630
1.4	0.433	0.310	5,303,355	4,439,691
1.6	1.252	1.129	16,488,792	16,203,510
1.8	2.671	2.093	40,225,231	31,619,312

In this paper, we extend the above condition so that it can be applied to different threshold values for the sub-keywords. We prove that if the threshold values t_1 ... t_n attached to the sub-keywords satisfy the condition below, we can detect all the candidates. Let $t_{i_1}, \cdots, t_{i_{n-m+1}}$ be an arbitrary combination of $n - m + 1$ threshold values of t_1 ... t_n. Then the following inequality must be satisfied:

$$t_{i_1} + \cdots + t_{i_{n-m+1}} \geq T \tag{3}$$

If $t_1 \leq \ldots \leq t_n$ is satisfied, inequality (3) is equivalent to the following inequality:

$$t_1 + \cdots + t_{n-m+1} \geq T \tag{4}$$

The proof of inequality (3) is given in the Appendix. In the example shown in Figure 5, the two arbitrary t_s values satisfy $t_s + t_s \geq T$. Therefore, this example satisfies inequality (3), and at least two sub-keywords can be detected. On the other hand, Figure 6 shows an example in which inequality (3) is not satisfied. In this example, $t_1 + t_2 = 2.95$ which is less than $T = 3.0$. Therefore we can detect only one sub-keyword. If $t_2 = t_3 = 2.0$, $t_1 + t_2 \geq T$ is satisfied and we can detect two sub-keywords (the second and third). Note that equation (2) is derived from inequality (3) by assigning $t_1 = t_s$, …, $t_n = t_s$ and by changing the inequality to an equation.

In paper [2], we showed m=1 is best for quick search. In addition, total number of threshold values should be minimized to reduce search space and search time. From this discussion, the following equality is derived from inequality (4) for quick search.

$$t_1 + \cdots + t_n = T \tag{5}$$

We attach the threshold values assigned to sub-keywords so as to satisfy the above equation (5).

In previous research, similar arithmetic analysis about the condition satisfied by the threshold values assigned to sub-keywords was performed by Navarro et al. [10]. They also proposed a search method with a suffix array and keyword division. The difference from our method is that they analyzed only the case of m=1 when DP-matching is applied[2] and they assumed that the threshold values are integer. Our discussion in this paper is regarded as an extension of their analysis to arbitral number of m with DP-matching and real number threshold values.

4. Statistic indicator of threshold value assignment

The numbers of candidates obtained in sub-keyword search can differ even when the threshold values are equal. When some sub-keywords output large number of candidates, the confirmation time will increase. By averaging the number of candidates, we can reduce the total and decrease confirmation time. For this purpose, we estimate the number of candidates to be detected and control the threshold value assigned to each sub-keyword so that equal number of candidates are detected in each sub-keyword search.

As our estimate of the number of candidates, we use the number of candidates detected in the last iteration of the iterative lengthening search. Suppose that threshold values t_i’ $(1 \leq i \leq n)$ were assigned to n sub-keywords of an original keyword, and C_i’ candidates $(1 \leq i \leq n)$ were detected in the last iteration. Since the number of candidates is almost proportional to the search space of the suffix array search, the number can be expected to increase exponentially with an increase in threshold value. Therefore, the number of candidates C_i in the current iteration can be estimated as follows:

$$C_i = C_i' e^{a(t_i - t_i')} \tag{6}$$

where C_i’ is the number of candidates detected in the last iteration of the iterative lengthening search, a is a constant, t_i is the threshold value assigned to the sub-keyword in the current iteration, and t_i’ is the threshold value assigned to the sub-keyword in the last iteration. The constant a is preliminarily determined using a statistical method.

To equalize the number of candidates to be detected means that the condition $C_1 = \cdots = C_n$ must be satisfied. Equation (5) should also be satisfied to maintain a speedy search. Based on these conditions, t_i is formally calculated as follows:

$$t_i = \frac{\ln \frac{C_1' \cdot C_2' \cdot \cdots \cdot C_n'}{C_i'^n}}{a \cdot n} + t_i' + \frac{T - T'}{n} \tag{7}$$

where $T' = t_1' + t_2' + \cdots + t_n'$. In the experiment in the next section, the threshold values are assigned to sub-keywords using this formula.

5. Experiment

An experiment was conducted using a PC with a 3.0 GHz Intel Core2Duo processor and 6 GB main memory. The CSJ (Corpus of Spontaneous Japanese) 606-h speech database was used for evaluation. For speech recognition, we used Julius [8] with its default trigram language model (60,000 words) and a speaker independent tri-phone acoustic model. The constant a in equation (6) was statistically determined to be 0.7123 based on preliminary measurements. In the experiment, we compared the following two threshold value assignment methods to show effectiveness of our approach.

A) Threshold value is equally assigned to each sub-keyword.

[2] They analyzed the case of arbitral m when exact matching is applied.

B) Threshold value is controlled by equation (7).

We searched for 30 keywords (10 to 18 phonemes) selected from the NTCIR-9 SpokenDoc test collection. These 30 keywords were enough long to be divided into more than two sub-keywords in our search method.

The average search time per keyword and the total number of candidates detected in search process (II) are shown in Table 1. Note that the search time has a strong correlation with the number of candidates. This result confirms that reducing the number of candidates using the proposed method (B) does indeed help to speed up searches.

Further, note that there were some cases in which we could not estimate the number of candidates correctly. In the DP-matching process, we used distinctive phonetic feature-based distance for substitution between phonemes. But we determined a constant penalty for insertion/deletion of a phoneme. For more accurate estimation, we have to refine equation (6) so that it can deal with these different types of measures.

6. Conclusions

We have investigated the theoretical condition that must be satisfied by threshold values assigned per sub-keyword. We have also proposed a statistical indicator for tuning these threshold values to achieve speedier searches. Experimental results confirm that this indicator is effective for reducing the number of search results detected, and thereby significantly decreasing the search time. In future work we will further optimize the threshold values assigned to sub-keywords by refining the indicator.

7. Acknowledgements

This work has been supported by a Grant-in-Aid for Young Scientists (B) 24700167 2012 and for Scientific Research (B) 22300060 2012 by MEXT, Japan, and the Kayamori Foundation of Information Science Advancement.

8. Appendix

[Proof of Equation (2)]

Let K be a keyword, k_1, …, k_n be sub-keywords, S be a phoneme sequence in the speech database, s_1, …, s_n be sub-sequences, $D \leq T$ be the distance between K and S, and d_i be the distance between k_i and s_i. We assume without loss of generality that d_1, …, d_n is sorted as $d_1 \leq \ldots \leq d_n$. From the relation $D \leq T$, $d_1 + \ldots + d_n \leq T$ is derived. This formula is converted to the following: $d_m \leq T - (d_1 + \ldots + d_{m-1}) - (d_{m+1} + \ldots + d_n)$. From $d_1 \leq \ldots \leq d_n$, d_m is maximized if $d_1, \ldots, d_{m-1}=0$, and $d_{m+1}, \ldots, d_n = d_m$. In this case, the following formula is satisfied: $(n - m + 1)\ d_m \leq T$. To find at least m adjacent candidates under any condition, t_s should be equal to the maximum value of d_m. Thus, equation (2) is derived. ■

[Proof of Inequality (3)]

Inequality (3) is derived from the two lemmas that follow. For these lemmas, let K be a keyword, k_1, …, k_n be sub-keywords, S be a phoneme sequence in the speech database, s_1, …, s_n be sub-sequences, $D \leq T$ be the distance between K and S, and d_i be the distance between k_i and s_i, and assume without loss of generality that d_1, …, d_n are sorted as $d_1 \leq \ldots \leq d_n$. ■

[Lemma 1]

If $t_{i_1} + \cdots + t_{i_{n-m+1}} \geq T$ is satisfied by arbitrary $n - m + 1$ threshold values, there are at least m pairs of $d_{j_k}$and t_{j_k} that satisfy $d_{j_1} \leq t_{j_1}, \cdots, d_{j_m} \leq t_{j_m}$.

[Lemma 2]

If there is an $n - m + 1$ combination of threshold values $t_{i_1}, \cdots, t_{i_{n-m+1}}$ that satisfies $t_{i_1} + \cdots + t_{i_{n-m+1}} < T$, there are some cases where fewer than m pairs of $d_{j_k}$and t_{j_k} satisfy $d_{j_1} \leq t_{j_1}, \cdots, d_{j_m} \leq t_{j_m}$.

[Proof of Lemma 1]

If there are fewer than m pairs of $d_{j_k}$and t_{j_k} that satisfy $d_{j_k} \leq t_{j_k}$, then $n - m + 1$ or more pairs must satisfy the following inequality:

$$d_{j_1} > t_{j_1}, \cdots, d_{j_{n-m+1}} > t_{j_{n-m+1}}$$

From this inequality, $d_{j_1} + \cdots + d_{j_{n-m+1}} > t_{j_1} + \cdots + t_{j_{n-m+1}} \geq T$ is derived. This contradicts D (= $d_1 + \ldots + d_n$) $\leq T$. Therefore, there exist at least m pairs of $d_{j_k}$and t_{j_k} that satisfy $d_{j_1} \leq t_{j_1}, \cdots, d_{j_m} \leq t_{j_m}$. ■

[Proof of Lemma 2]

The inequality $t_{i_1} + \cdots + t_{i_{n-m+1}} < T$ can be represented as $t_{i_1} + \cdots + t_{i_{n-m+1}} + \Delta = T\ (\Delta > 0)$. Consider $d_{i_1}, \cdots, d_{i_n}$ that satisfy the following conditions:

$$d_{i_1} = t_{i_1} + \frac{\Delta}{N}, d_{i_2} = t_{i_2} + \frac{\Delta}{N}, \cdots, d_{i_{n-m+1}} = t_{i_{n-m+1}} + \frac{\Delta}{N},$$
$$d_{i_{n-m+2}} = 0, \cdots, d_{i_n} = 0 \qquad (N \geq n - m + 1)$$

These $d_{i_1}, \cdots, d_{i_n}$ also satisfy the following inequality:

$$\begin{aligned} d_{i_1} + \cdots + d_{i_n} &= d_{i_1} + \cdots + d_{i_{n-m+1}} + d_{i_{n-m+2}} + \cdots + d_{i_n} \\ &= t_{i_1} + \cdots + t_{i_{n-m+1}} + \frac{n-m+1}{N}\Delta \leq T \end{aligned}$$

In this case, $d_1 + \ldots + d_n \leq T$ is satisfied. However, there exist only $m - 1$ pairs of $d_{j_k}$and t_{j_k} that satisfy $d_{i_{n-m+2}} \leq t_{i_{n-m+2}}, \cdots, d_{i_n} \leq t_{i_n}$, since we have $n - m + 1$ pairs of $d_{j_k}$and t_{j_k} that satisfy $d_{i_1} \geq t_{i_1}, \cdots, d_{i_{n-m+1}} \geq t_{i_{n-m+1}}$. ■

9. References

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

[2] Katsurada, K., Sawada, S., Teshima, S. and Nitta, T., “Evaluation of Fast Keyword Detection Using a Suffix Array”, InterSpeech2011, pp.909-912, 2011.

[3] 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.

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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)

[10] Navarro, G., Baeza-Yates, R., Sutinen, E. and Tarhio, J., “Indexing Methods for Approximate String Matching”, IEEE Data Engineering Bulletin, Vol.24, No.4, pp.19-27 (2001).