Fast Keyword Detection Using Suffix Array

Kouichi Katsurada, Shigeki Teshima, Tsuneo Nitta

Toyohashi University of Technology, Toyohashi, Japan

{katurada, nitta}@tutkie.tut.ac.jp, teshima@vox.tutkie.tut.ac.jp

Abstract

In this paper, we propose a technique for detecting keywords quickly from a very large speech database without using a large memory space. To accelerate searches and save memory, we used a suffix array as the data structure and applied phoneme-based DP-matching. To avoid an exponential increase in the process time with the length of the keyword, a long keyword is divided into short sub-keywords. Moreover, an iterative lengthening search algorithm is used to rapidly output accurate search results. The experimental results show that it takes less than 100ms to detect the first set of search results from a 10,000-h virtual speech database.

Index Terms: keyword detection, large-scale speech database, suffix array, iterative lengthening search

1. Introduction

Keyword detection is an essential issue in effectively utilizing a speech database. Considerable research has been conducted on this issue, and a reasonable performance has been achieved in detecting the correct keywords from a speech database [1][2]. Recently, some studies focused on the search speed [3][4][5] because a quick search is important when 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 need a large memory space to hold temporal index data; furthermore, the existing methods cannot be used for a 10,000-h speech database. This paper provides fast keyword detection for a large speech database without using a large memory space.

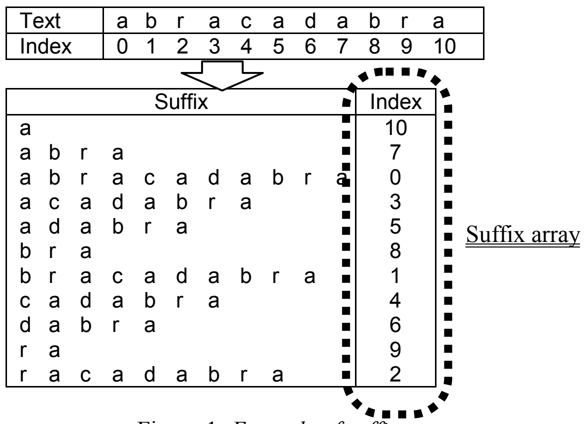
We employed a suffix array, which enables a fast binary search and decreased memory usage, as a data structure and applied phoneme-based DP-matching (or DTW: Dynamic Time Warping) to detect a keyword from the array. Even though the suffix array enables binary search, a DP-matching-based similarity search causes an exponential increase in processing time with length of a keyword. To avoid the problem, we split a long keyword into short sub-keywords during the search. Moreover, an iterative lengthening search algorithm is introduced to rapidly output accurate results. These techniques make it possible to show search results quickly.

In the following sections, we explain the outline of a suffix array and search algorithm using DP-matching. Then, we present keyword division and an iterative lengthening search algorithm for a faster search. After evaluating our approach, we provide our conclusion and suggest future research.

2. Keyword matching using a suffix array

2.1. Structure of suffix array

A suffix array [6] is a data structure that is used for quickly searching for a keyword from 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



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: *Example of suffix array.*

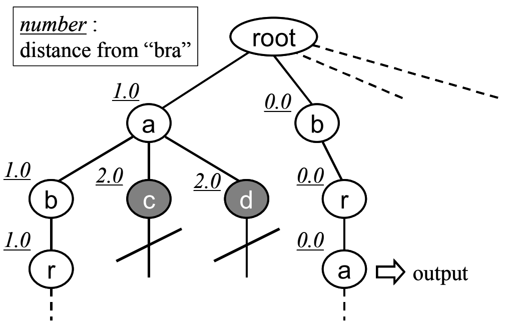


Figure 2: *Similarity search on a suffix array.*

shows a sample of a suffix array constructed from a character string [1] "abracadabra." Indexes in the figure represent the position at which the suffixes start in the string. Since the indexes are sorted by dictionary order of suffixes, we can use a binary search to detect a keyword. Moreover, a large memory space is not required because the array holds only the indexes.

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 some technique for a similarity search together with a suffix array. For this purpose, a search algorithm using DP-matching on the suffix array is proposed [7]. 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. One of the reason we employed suffix array is to use this similarity search and pruning algorithm that cannot be applied to the other structure such as hash table, and so on.

[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: *Distinctive phonetic feature table.*

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 the results by referring to the suffix array shown in Figure 1.

3. Keyword detection from speech database

3.1. Distinctive phonetic features

Before starting the keyword search, a speech database is transformed into a sequence of phonemes by means of Large Vocabulary Continuous Speech Recognition (LVCSR) or other methods. To apply DP-matching to the sequence of phonemes, the distinctive phonetic feature-based distance is introduced. These are articulatory features that represent a phoneme by using 15 features such as plosive, affricative, and so on. Figure 3 shows a fragment of the relationship between phonemes and distinctive phonetic features. We used the hamming distance of these features to calculate the distance between two strings of phonemes.

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

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

where a_i is a phoneme in a keyword $a_1a_2 \dots a_K$; b_j is a phoneme in a speech database; $P_{i,j}$ is the distance between $a_1a_2 \dots a_i$ and $b_1b_2 \dots b_j$; and $d(a_i, b_j)$ is the hamming distance calculated from the articulatory features of a_i and b_j. Although we currently use the hamming distance for simplicity, we will replace it with weighted distances based on recognition error rate of articulatory features.

3.2. Keyword division

According to the algorithm shown 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, memory usage (and process time) will increase exponentially according to the depth of the tree. Since 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 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.

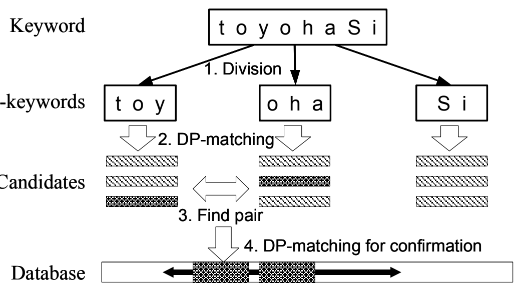


Figure 4: *Outline of keyword search.*

Even though the above division reduces the process time, a large number of candidates can be detected. To reduce the process time, we modified the threshold by using the following equation so as to detect at least two candidates at the same time if the original keyword is within the threshold.

$$t' = \frac{p}{p-1} t \quad (2)$$

In the above equation, t' is the modified threshold per phoneme, p is the number of division, and t is the original threshold. Note that t' is a threshold per phoneme. The total threshold for a sub-keyword is the multiplied value by the number of phonemes. If p=2, the threshold per phoneme is twice the original threshold according to equation (2). In this case, the total threshold of sub-keywords is same value as that of the original keyword. This division has no meaning for a fast search. For this reason, a division is executed if the original keyword is divided into at least three sub-keywords.

Figure 4 illustrates the outline of a keyword search. The search algorithm is summarized as follows.

1. If the keyword is long, divide it into more than three sub-keywords.
2. Search the sub-keywords on the suffix array by DP-matching, and find candidates.
3. Find pairs of candidates with different sub-keywords close to each other.
4. Detect the final results from the pairs by using DP-matching again.

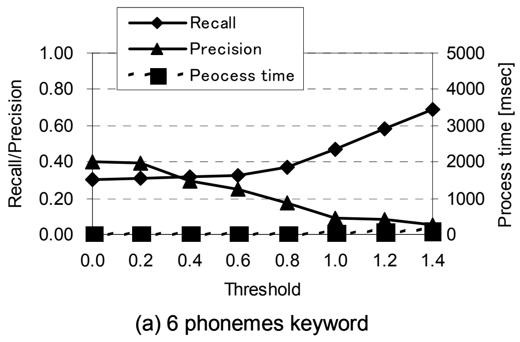
3.3. Iteratively lengthening the search

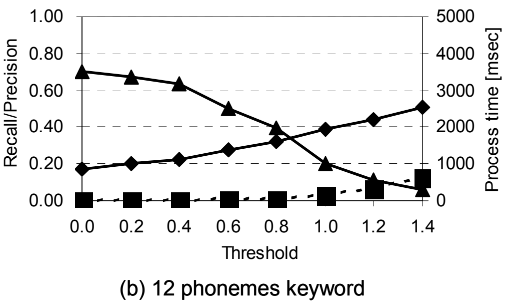
If the threshold is set at a large value, the recall rate of the search will increase because many results are output, whereas the precision rate will decrease and the search time will increase exponentially. On the other hand, if the threshold is a small value, the precision rate will increase and the 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.

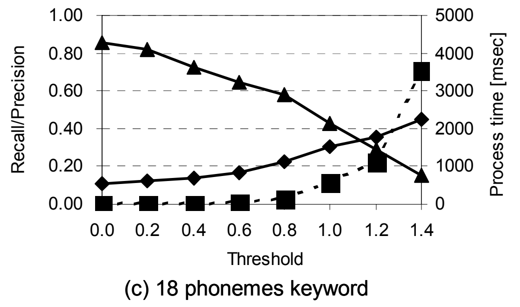
4. Evaluation

4.1. Experimental setup

Experiments were carried out on a PC with an Intel PentiumD 2.8GHz processor and a 4GB main memory. The speech database is 390-h male speakers' lecture speeches from CSJ (Corpus of Spontaneous Japanese). For speech recognition, we used Julius [8] with its default trigram language model (60,000 words) and a speaker independent acoustic model.







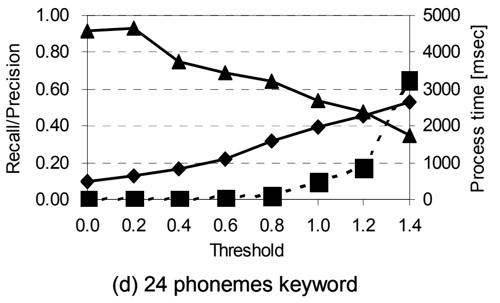


Figure 6: *The number of phonemes in a sub-keyword.*

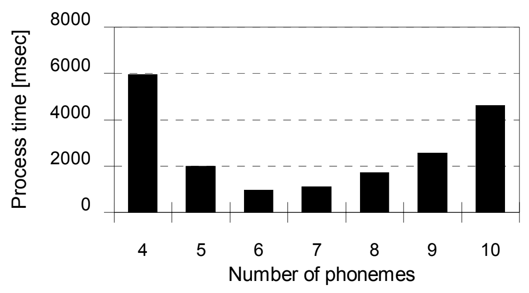


Figure 5: *The number of phonemes in a sub-keyword.*

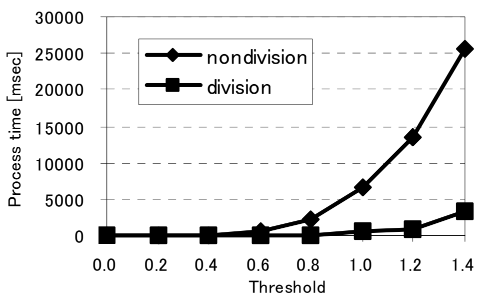


Figure 7: *Effect of keyword division.*

The correct rate and accuracy of phonemes were 71% and 66%, respectively. The size of the suffix array was 52 MB. The average distance between phonemes was 5.7; therefore, if the threshold is set at 1.0, we can allow one phoneme error per 5.7 phonemes.

We carried out a preliminary experiment to find the optimum number of phonemes in a sub-keyword. In the experiment, the keywords containing 6 to 30 phonemes were used. We searched 10 keywords per size and measured the processing time. The threshold was set at 1.0. As shown in Figure 5, the process time is the shortest when the number of phonemes is six. Therefore, we used this size for the following experiments.

4.2. Evaluation of search performance and process time

We evaluated the search performance (recall rate and precision rate) and process time with keywords containing 6, 12, 18, and 24 phonemes. Figure 6 shows the results of searching 100 keywords. In Figure 6, (a), (b), (c), and (d) represent the results of 6, 12, 18, and 24 phoneme keywords, respectively. The horizontal axis in each graph represents the threshold of the search, the left vertical axis represents the performance, and the right vertical axis represents the processing time. The results show that the processing time is less than 10 ms when the threshold is 0.0 or 0.2 for any number of phonemes. At the same time, a high precision rate is obtained especially in the results for (b), (c), and (d). These results show the iterative lengthening search works well for long keywords.

Even though the threshold is 1.4, for which the recall rate is around 50% in all results, the process time is between 100 ms and 3,500 ms. This means that half of the results to be obtained are definitely output to a user while he/she is checking the former results.

The process times of (c) and (d) rapidly increase according to the threshold. This is because keyword division is applied to the keywords of (c) and (d); this occurs in combination with an increase in the threshold per phoneme according to equation (2). Nevertheless, keyword division reduces the process time. We compared the processing time when keyword division is applied with that when it is not applied. Figure 7 shows the results when the keywords contain 24 phonemes. It shows that the processing time is about 1/7 times the original time if division is applied; thus, we confirmed that keyword division works well for fast searches.

4.3. Comparison between suffix array and continuous DP-matching

Figure 8 shows the process time required when using our approach and when using continuous DP-matching without using a suffix array. The results for continuous DP-matching were obtained by searching the database from beginning to

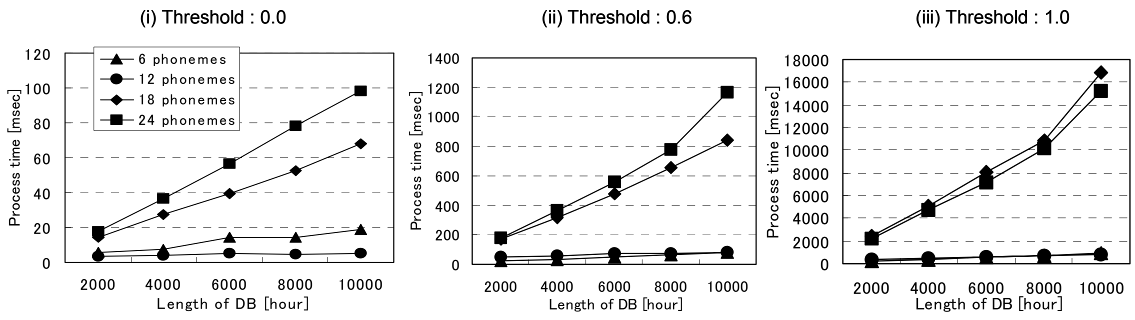


Figure 9: *Change in process time according to the size of speech database.*

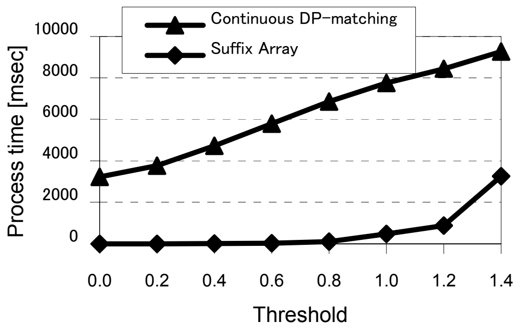


Figure 8: *Comparison between continuous DP-matching and suffix array.*

end with conventional continuous DP-matching. As shown in Figure 8, a long time is required even if the threshold is a small value. On the other hand, in our approach, a binary search works well if the threshold values are small. This is desirable for an iterative lengthening search because the first several results can be quickly shown to a user. However, if the threshold is a large number, continuous DP-matching might be faster than our approach. We are planning to combine our approach with conventional continuous DP-matching to search keywords rapidly.

4.4. Process time on a very large speech database

For evaluating the process time on a large speech database, we constructed a virtual speech database using Mainichi Newspaper articles from 1997 to 2000. The articles were converted into strings of phonemes by using a morphological analyzer MeCab [9] and a dictionary. We constructed a scale of 2,000-h to 10,000-h phoneme strings virtually and measured the process time. In Figure 9, (i), (ii) and (iii) show the process time when the thresholds are 0.0, 0.6, and 1.0, respectively.

Even though a binary search was applied to the suffix array, the processing time for 18 and 24 keywords increased in proportion to the size of the database. This is because the processing time of DP-matching for confirmation is considerably more than that of a search on a suffix array. Since the number of candidates increases in proportion to the size of database, the processing time increases accordingly.

5. Conclusions

This paper presented fast keyword detection technique for a 10,000-h speech database by using suffix array. The results show that a suffix array and keyword division work well for rapidly detecting the first several sets of results. This is desirable for an iterative lengthening search that shows the results to a user incrementally from the accurate ones. However, the process time increases exponentially according to the threshold of the search, unlike the case when conventional continuous DP-matching is used by itself. These results suggest that it is necessary to investigate a combination of our approach and conventional continuous DP-matching to develop an optimum method for fast keyword detection. Future study will focus on how to combine these approaches appropriately.

6. Acknowledgements

This work was supported by the Telecommunications Advancement Foundation and the Grant-in-Aid for Young Scientists (B) 20700156 2009 from MEXT - Japan.

7. References

[1] 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.

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

[3] 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.

[4] Thambiratnam, K. and Sridharan, S., "Dynamic Match Phone-Lattice Searches For Very Fast And Accurate Unrestricted Vocabulary Keyword Spotting", ICASSP 2005, vol.1, pp.465-468, 2005.

[5] Kokiopoulou, E., Frossard, P. and Verscheure, O., "Fast Keyword Detection with Sparse Time-Frequency Models", ICME'08, pp.1081-1084, 2008.

[6] 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.

[7] 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)

[8] 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.

[9] Kudo, T., "MeCab: Yet Another Part-of-Speech and Morphological Analyzer", http://mecab.sourceforge.net/, accessed on April 9, 2009. (In Japanese)