Utilization of Suffix Array for Quick STD and Its Evaluation on the NTCIR-9 SpokenDoc Task

Kouichi Katsurada
Toyohashi Univ. of Tech.
1-1 Hibarigaoka, Tempaku-cho
Toyohashi 441-8580, JAPAN
+81-532-44-6884
katsurada@cs.tut.ac.jp

Koudai Katsuura
Toyohashi Univ. of Tech.
1-1 Hibarigaoka, Tempaku-cho
Toyohashi 441-8580, JAPAN
+81-532-44-6884
katsuura@vox.cs.tut.ac.jp

Yurie Iribe
Toyohashi Univ. of Tech.
1-1 Hibarigaoka, Tempaku-cho
Toyohashi 441-8580, JAPAN
+81-532-44-6638
iribe@imc.tut.ac.jp

Tsuneo Nitta
Toyohashi Univ. of Tech.
1-1 Hibarigaoka, Tempaku-cho
Toyohashi 441-8580, JAPAN
+81-532-44-6890
nitta@cs.tut.ac.jp

ABSTRACT
We propose a technique for detecting keywords quickly from a very large speech database without using a large-sized memory. For acceleration of search and saving the use of memory, we employed a suffix array as a data structure and applied phoneme-based DP-matching to it. To avoid exponential explosion of process time with the length of a keyword, a long keyword is divided into short sub-keywords. Moreover, iterative lengthening search algorithm is used for outputting the accurate search results fast.

Categories and Subject Descriptors
H.3.3 [**Information Search and Retrieval**]: Search process.

General Terms
Experimentation.

Keywords
Spoken term detection, large scale speech document, suffix array, keyword division, iterative lengthening search.

Team Name
NKI-lab.

Subtasks
Spoken Term Detection.

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 [1][2]. Recently, some studies have focused on search speed [3][4][5] 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 very large speech database.

For this situation, we propose a fast spoken term detection for large-scale speech documents [6][7]. In our approach, suffix array is employed as a data structure for quick search and phoneme-based DP-matching is used to deal with OOV words and recognition errors. Even though suffix array enables quick search, DP-matching-based similarity search brings about exponential explosion of process time with length of a keyword. To avoid the problem, we split a long keyword into short sub-keywords when searching a keyword. Moreover, iterative lengthening search algorithm is introduced to output accurate results fast. These techniques make it possible to show search results quickly to a user.

2. KEYOWRD DETECTION USING A SUFFIX ARRAY

2.1 Structure of the suffix array
A suffix array [8] 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.

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

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

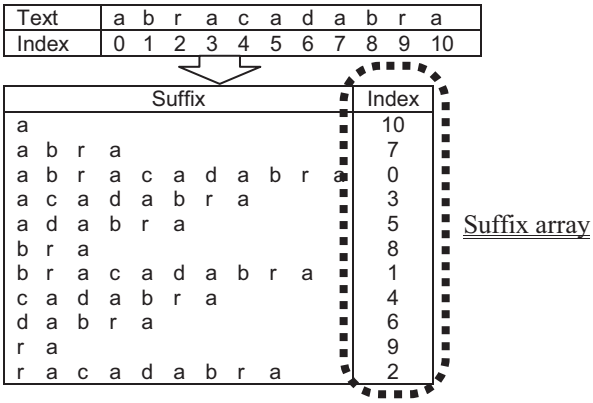

Figure 1: *An example suffix array*.

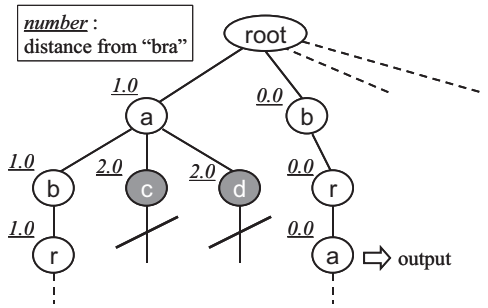

Figure 2: *Similarity search on a suffix array*

algorithm using DP-matching (or Dynamic Time Warping (DTW)) on the suffix array is proposed [9]. 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.

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.

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

Figure 3: *Table of distinctive phonetic features*.

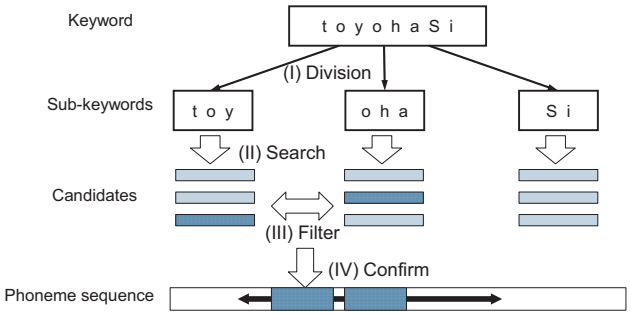

Figure 4: *Outline of keyword search*.

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

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

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 [6]. 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.

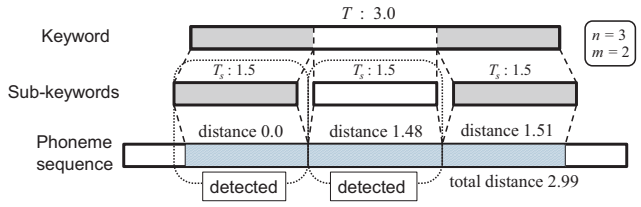

Figure 5: *An example of threshold modification*.

(III) Filter the candidates by detecting adjacent candidates.

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

In paper [7], we generalized 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. Equation (2) is derived as follows: Let K be a keyword; k_1, …, k_n, sub-keywords; S, a phoneme sequence in the speech database; s_1, …, s_n, sub-sequences; $D \leq T$, the distance between K and S; d_i, the distance between k_i and s_i. We assume d_1, …, d_n is sorted as $d_1 \leq \ldots \leq d_n$ without loss of generality. From the relation $D \leq T$, $d_1 + \ldots + d_n \leq T$ is derived. This formula is converted into the following formula: $d_m \leq T - (d_1 + \ldots + d_{m-1}) - (d_{m+1} + \ldots + d_n)$. From $d_1 \leq \ldots \leq d_n$, d_m is maximized if d_1, …, $d_{m-1}=0$, and d_{m+1}, …, $d_n = d_m$. In this case, the following formula is satisfied: $(n - m + 1)\, d_m \leq T$. In order to find at least m adjacent candidates under any condition, T_s should be equal to the maximum value of d_m. Therefore, equation (2) is derived.

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.

3.3 Iterative lengthening search

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

4. EVALUATION

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. We evaluated our method on the syllable-based transcription of the reference automatic transcriptions provided by the NTCIR-9 STD working group.

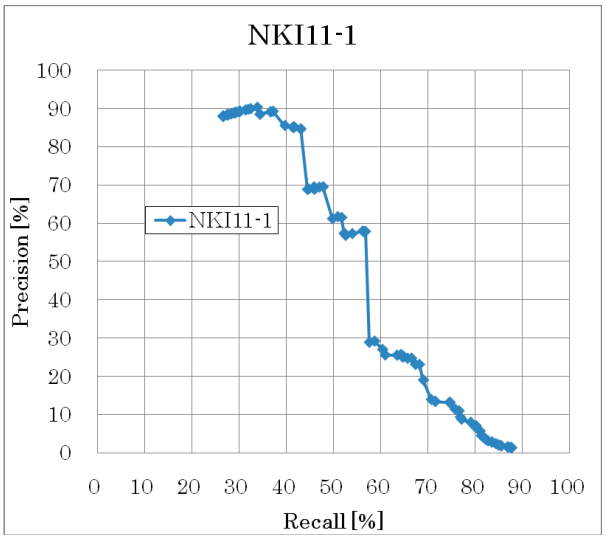

Figure 6: *Precision-recall curve of our method and baseline on CORE lectures*.

In this experiment, we divide a keyword into six-phoneme-sub-keywords and detect one of them (i.e. $m = 1$ in equation (2)) because this setting is confirmed to be best for detecting keyword quickly in paper [7]. Deletion and insertion penalty in equation (1) is set as 3.0. The score is calculated by the following equation:

$$score = \frac{1}{T/l^{3/2} + 1} \tag{3}$$

where l is the length of keyword. This equation converts the search threshold into the score between 0.0 and 1.0. We attached binary decisions "yes" to the results whose score is 0.89 or more. This score is obtained by the preliminary experiment.

4.2 Experimental results on CORE and ALL lectures

Index size and memory usage in the CORE experiment were 5.7MB and 33MB, while those in the ALL experiment were 84MB and 455MB. Figures 6 and 7 illustrate the precision-recall curves of our method on CORE and ALL lectures[2]. MAP score on CORE lectures was 0.684, and that on ALL lectures was 0.339. The result of our method on CORE lectures overtakes the baseline, while that on ALL lectures was below it [11]. This is unexpected result because both of them use DP-matching in the searching process. The difference is caused by the definition of distance used in the DP-matching process. In the DP-matching process, our method employs distinctive phonetic features, while the baseline uses phoneme-based edit distance. It brings about the difference in performance between two methods.

The notable characteristic of our method is that it can detect the keywords considerably faster than the other methods. Tables 1 and 2 show the processing time of our method. The F-measure of the CORE experiment is maximized when the score is 0.886; in

[2] We submitted two sets of results of both CORE and ALL experiments. However, we show only a result of each experiment because the difference is just the number of search results contained in a set.

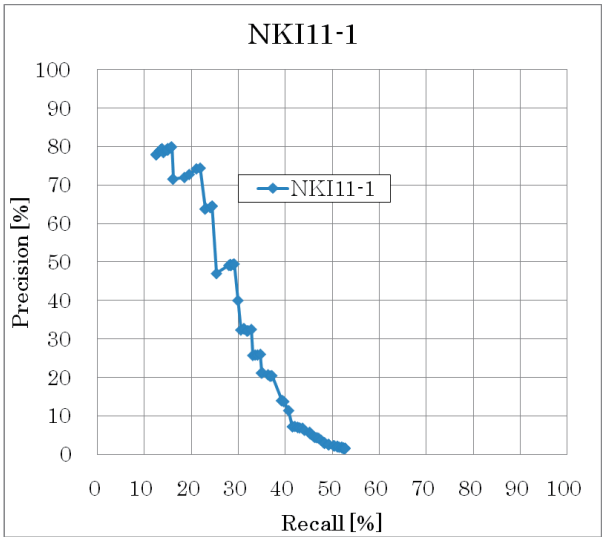

Figure 7: *Precision-recall curve of our method and baseline on ALL lectures.*

that case, it takes 0.62ms to get the results. In the ALL experiment, F-measure is maximized when the score is 0.884, and the results are obtained in 3.44ms.

5. CONCLUSIONS

This paper evaluated the fast keyword detection technique using a suffix array on the document provided by NTCIR-9 STD working group. The results show that a suffix array and keyword division work well for rapidly detecting the results. 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 Scientific Research (B) 22300060 2010 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.

Table 1: *Processing time of our method in the CORE experiment.*

Score	1.00	0.96	0.92	**0.886**	0.88	0.84	0.80
Recall	26.5	27.7	33.8	**43.0**	44.4	53.9	65.6
Precision	88.0	88.4	90.3	**84.6**	68.8	57.3	24.7
F-measure	40.8	42.1	49.2	**57.0**	54.0	55.5	35.9
Time [ms]	0.00	0.00	0.00	**0.62**	0.96	2.18	10.92

Table 2: *Processing time of our method in the ALL experiment.*

Score	1.00	0.96	0.921	**0.884**	0.881	0.841	0.799
Recall	12.6	13.8	21.9	**29.1**	29.9	36.3	43.7
Precision	78.0	79.5	74.5	**49.6**	40.1	20.8	6.92
F-measure	21.7	23.5	33.9	**36.7**	34.3	26.5	11.9
Time [ms]	0.32	0.32	1.28	**3.44**	3.42	15.94	73.98

[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] 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.
[5] Wallace, R., Vogt, R. and Sridharan, S., "Spoken term detection using fast phonetic decoding", ICASSP'09, pp.2135-2138, 2009.
[6] Katsurada, K., Teshima, S. and Nitta, T., "Fast Keyword Detection Using Suffix Array", InterSpeech2009, pp.2147-2150, 2009.
[7] Katsurada, K., Sawada, S., Teshima, S., Iribe, Y. and Nitta, T., "Evaluation of Fast Keyword Detection Using a Suffix Array", InterSpeech2011, pp.909-912, 2011.
[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] 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.
[11] Akiba, T., Nishizaki, H., Aikawa, K., Kawahara, T. and Matsui, T., "Overview of the IR for Spoken Documents Task in NTCIR-9 Workshop", 2011.