Vol. 39 No. 4 Transactions of Information Processing Society of Japan Apr. 1998

Regular Paper

New Indices for Japanese Text: A New Word-based Index of Non-segmented Text for Fast Full-text-search Systems

NAOHIKO NOGUCHI,[†] YUJI KANNO,[†] MITSUAKI INABA[†] and KAZUAKI KURACHI[††]

In this paper, we propose a new type of word-based index, the *complete maximal word index*, which is suitable for text of continuous sequence languages such as Japanese and Chinese. This index solves the problems encountered in applying the word-based index file method to a full-text retrieval system for such types of text. The proposed word-based index ensures retrieval with no false dismissals for arbitrary search strings and very fast retrieval even for longer search strings, while its size is small in relation to other types of index such as the n-gram-based index. A formal definition of the *complete maximal word index* is given, and its generating algorithm and searching algorithm are described. Some experimental results are presented to demonstrate that this approach is in fact effective and practical. The proposed index is also promising in that it can be naturally incorporated into inexact-match retrieval models such as probabilistic and vector space model, because it contains word-based information such as word frequencies and word distributions.

1. Introduction

To make word-based indices for text of continuous sequence languages such as Japanese and Chinese, in which words are not delimited by spaces, it is first necessary to segment the text into a sequence of words as accurately as possible. Although research on word segmentation methods for Japanese text has a long history, a perfect segmentation method for arbitrary text has not yet been developed. In fact, the task is impossible in principle, because the entire set of words that could be used in a text cannot be predicted, and unregistered words cannot be pinpointed with an ordinary dictionary. Therefore, use of such a word-based index of Japanese text for a fast full-text retrieval system that has to handle arbitrary search strings would lead to a certain amount of false dismissals, and the recall rate of the entire retrieval system would depend heavily on the accuracy of the word segmentation method it used.

Current studies of fast full-text retrieval systems for Japanese text follow other approaches that do not use word segmentation methods. One example is the character-based or n-gram-based signature file method[4),10),13)], which uses the fact that Japanese text has a large character set. Basically, this method uses a 1-gram signature file to filter out the non-relevant text. Ogawa[10)] uses a 2-gram signature file as well, in order to speed up the whole retrieval process. With this kind of approach, a significant but limited speedup of the retrieval process can be achieved, because it has to be used with a full-text scanning process when retrieval with no false drops is required.

Another route is the n-gram-based index (inverted) file method. Kikuchi[7)] uses a 1-gram index file for *kanji* characters and a 2-gram index file for *kana* character sequences. These index files also contain posting records to check the adjacency of arbitrary 1-grams and 2-grams. This approach ensures retrieval with no false dismissals for arbitrary search strings, as well as very fast retrieval, because it is a sort of inverted file method. However, it involves some problems. One of these is the size of the index file, which may be several times larger than the original text, because it has to include posting records of each occurrence of a 1-gram or 2-gram. Another problem is the retrieval speed. It has reported that the longer the search string is, the slower the search speed will be.

In this paper, we propose a new type of word-based index, which we will call the *complete maximal word index*[6),8)], that is suitable for text of continuous sequence languages. The proposed index can solve the above-mentioned problems, since it ensures retrieval with no false

† Multimedia Systems Research Laboratory, Matsushita Electric Industrial Co., Ltd.
†† Systems Integration Business Center, Matsushita Electric Industrial Co., Ltd.

dismissals for arbitrary search strings and very fast retrieval even for longer search strings, while its size is relatively small. The remaining sections of the paper are organized as follows: In Section 2, we explain the specific problems that arise when we try to apply the naive word-based index file method to Japanese text. We then introduce the concept of a *complete maximal word index* in a formal setting. Section 3 describes algorithms for generating and searching an index. Implementation issues are mentioned in Section 4, and some experimental results are given in Section 5. The final section summarizes the paper.

2. Basic Ideas

2.1 Problems and Solutions

Several types of problem are encountered when a *naive* word-based index method is applied to a full-text retrieval system for Japanese text.

Problem 1. Segmentation failure causes false dismissals in retrieval.

In general, for ordinary Japanese text, a vast number of possible word segmentations exist; thus, the segmentation process is nondeterministic in nature. While some of the segmentation methods developed to date, such as the *longest match method*, the *minimum morpheme number method*, and the *minimum connective-cost method*$^{5)}$, have achieved very high precision, it can never reach 100 percent (we call this **the problem of non-determinacy**). Moreover, a text always contains words that are not included in the dictionary used by the segmentation process. This poses a serious problem for the segmentation method (we call it **the problem of unregistered words**). Because of the above problems, it is not possible to achieve *no-false-dismissal* retrieval with a word-based index of Japanese text.

Problem 2. Word-based indices can only handle *words* as search strings (we call this **the problem of searching for an arbitrary string**).

It is true that, with a word-based index, the search strings must be strings defined as *words*. Whereas for English text, *words* are simply defined as character strings separated by spaces, for Japanese text, which contains no explicit word delimiters, words must be defined in some dictionary, which is then used by the word segmentation process. Since ordinary Japanese text contains many compound words—that is, words consisting of a sequence of component words—the dictionary often contains the component words but not the compound word. In such cases, it is not possible to search for the compound word by using the index. The dictionary may also contain the compound word but not the component words. In such a case, it is not possible to search for the component words. Moreover, as mentioned earlier, a text always contains new words, which the system has to handle as search strings.

The above problems seem inevitable if word-based indices of Japanese text are used for a full-text retrieval system, but if we give up attempting to segment the text, then we will arrive at a simple solution. The basic idea of this solution can be stated as follows:

> Word segmentation methods presuppose that a text has only one correct segmentation. This might be true, but we need not cling to it if the purpose of using a word-based index is to ensure fast retrieval with no false dismissals. What is needed is not a word segmentation of the text, but merely a sequence of words occurring in the text that covers the whole text.

The simplest way to realize this idea is as follows:

1. Include every single character in the dictionary.
2. Register all the words that occur in the text, together with their occurrence positions, in the index.

We will call this type of index a *complete index* of the text. The *complete index* of a text is uniquely defined in relation to a dictionary.

Let us take as an example a text containing a passage that begins "全日本学生選手権に出場する選手は...", and a dictionary containing entries such as "全日", "日本", "本学", "学生", "生", "選手", "選手権", "に", "出場", "する", "は". It is possible that the word segmentation process segments it into a sequence of words, "全日", "本学", "生", "選手権", "に", "出場", "する", "選手", "は" while omitting "日本" and "学生". If the word-based index contains only these words for the portion of the text, it is not possible to search for "日本" or "学生" in this passage. Here we can see **the problem of non-determinacy**. It is also not possible to search for "全日本", which is not a word, if we use this index. Here we can see **the problem**

of searching for an arbitrary string.

These problems can, however, be resolved by using the *complete index*. The *complete index* of this portion of the text comprises every word occurring in the text, together with its occurrence position. Thus, after adding every single character to the dictionary, the *complete index* contains:

("全", 1), ("全日", 1), ("日", 2),
("日本", 2), ("本", 3), ("本学", 3),
("学", 4), ("学生", 4), ("生", 5),
("選", 6), ("選手", 6), ("選手権", 6),
("手", 7), ("権", 8), ("に", 9),
("出", 10), ("出場", 10), ("場", 11),
("す", 12), ("する", 12), ("る", 13),
("選", 14), ("選手", 14), ("手", 15),
("は",16).

Here, the item ("全日", 1) means that the occurrence position of the word "全日" in the text is 1.

It is obvious that we can search for every word in this portion of the text. Thus **the problem of non-determinacy** can be resolved. Moreover, we can search for arbitrary strings by using this index. For the search string "全日本", we have to check the adjacency of the occurrence positions of the word "全日" and "日本". In the *complete index*, since we have an adjacent sequence of words ("全日", 1), ("日本", 2), we get the occurrence position "1" of the string "全日本". It is likely that the search string is not even a "compound word" in the usual sense. For example, think of "権に出" or "生選" as a search string. But it is possible to take those strings as "quasi-compound words," because the dictionary contains every single character as a word. For the string "権に出", there is a sequence of component words, "権", "に", "出". Thus, we can use the *complete index* to search for such strings. As a result, **the problem of searching for an arbitrary string** is resolved. It is not difficult to understand that **the problem of unregistered words** can also be resolved. Since every single character is included as a word, the *complete index* must cover the whole text. Thus, it contains enough information to locate every string occurring in the text.

While all the problems mentioned can be resolved by using a *complete index* of the text, the *complete index* is so simple that it has some defects. One of them is the redundancy of the information it stores. In the above example, the *complete index* includes ("選", 6), and ("選手", 6) as well as ("選手権", 6). But what is intuitively clear is that the information of ("選", 6) and ("選手", 6) is not necessary, because "選" and "選手" occur within the word "選手権" in this portion of the text. This means that we can omit those words that are included by other words in the text.

Now we propose a new type of index, which we will call a *complete maximal word index* 6),8), that removes the redundant information from the *complete index*. If we omit all the elements that are included in other elements from the above-mentioned *complete index*, then we have only:

("全日", 1), ("日本", 2), ("本学", 3),
("学生", 4), ("選手権", 6), ("に",9),
("出場", 10), ("する", 12), ("選手", 14),
("は",16).

With this kind of index, we can find all the occurrence positions of a certain word W by searching for W and its extension words in the index. For example, if the search string is "選手", we first find all extension words of "選手" in the dictionary; in this case, we have "選手" and "選手権". Next, we search the index with those words, and get ("選手権", 6) and ("選手",14) as the occurrence positions of the string "選手" in this portion of the text. We can construct a search algorithm for arbitrary strings, as well; the details of the algorithm will be given in Section 3.

If the dictionary contains only all the 2-grams, the *complete maximal word index* will be identical to the usual 2-gram-based index. Obviously, we can resolve the above-mentioned problems with this 2-gram index, but the *complete maximal word index* using an ordinary dictionary has several advantages:

(1) **Index size**
The number of posting records the 2-gram index would have to contain would be equal to the number of characters in the text. However, if we assume that the average character length of all the maximal words in the text is N, the number of posting records in the *complete maximal word index* would be roughly $1/N$ of the number of characters in the text. This means that the size of the *complete maximal word index* would be roughly $1/N$ of the size of the 2-gram index.

(2) **Search speed**
In general, most real search strings are "words" or "sequences of words" (com-

pound words) in the usual sense. With the 2-gram index, we have to use all the occurrence positions of each 2-gram in a search string. But with the *complete maximal word index*, we only need all the occurrence positions of each component word in a search string, the number of which is usually less than the number of occurrence positions of each 2-gram. This means that a significant speed-up can be achieved when a search string is relatively long.

Some comparisons between these two indices will be given in Section 5.

2.2 Formalization

In this section, we give a formal definition of the *complete maximal word index*.

First, we suppose the following:

- A document is a finite character string.
- A dictionary is a finite set of arbitrary character strings. When a character string is an element of some dictionary *DICT*, we call the character string a *word* in *DICT*.
- When *str* is a character string, we write *str* $[i : j]$ to denote a substring of *str* that begins at the i-th character of *str* and ends at the j-th character of *str*.
- We write $|str|$ to denote the length (number of characters) of the character string *str*.

Next, we define several *extension relations* between *words* in a dictionary.

Definition 1 (extension relation between word and string). Let a be a word in a dictionary *DICT* and b be a character string. Then

- a is an *extension word* of b
  $\Leftrightarrow b$ is a substring of a,
- a is a *prefix word* of $b \Leftrightarrow a = b[1 : |a|]$,
- a is a *prefix extension word* of b
  $\Leftrightarrow a[1 : |b|] = b$,
- a is a *postfix word* of b
  $\Leftrightarrow a = b[|b| - |a| + 1 : |b|]$, and
- a is a *postfix extension word* of b
  $\Leftrightarrow a[|a| - |b| + 1 : |a|] = b$.

For word-based indices such as we are discussing, the posting record for each occurrence of a dictionary word has to be registered. Therefore, the unit of information of the index (we will call it the *index item*) has to be of the form (w, n), such that w is a word and n is its occurrence position in the text.

Definition 2 (index item). *Index items* of a document *DOC* with respect to a dictionary *DICT* are possible 2-tuples of the form (w, n), where

- w is a word in *DICT*,
- n is a character position of *DOC*, and
- w is a *prefix word* of $DOC\,[n : |DOC| - n + 1]$.

An index of a document is then defined as a set of index items.

Definition 3 (index). An *index* of a document *DOC* with respect to a dictionary *DICT* is a finite set of *index items* of *DOC* with respect to *DICT*.

An *extension relation* among *index items* is defined as follows:

Definition 4 (extension relation between index items). Let $I_1 = (w_1, n_1)$, $I_2 = (w_2, n_2)$ be *index items* of a document *DOC* with respect to a dictionary *DICT*. Then,

I_1 is an *extension index item* of I_2
$\Leftrightarrow$ (w_1 is an *extension word* of w_2) $\wedge$
$(n_1 \leq n_2) \wedge (n_1 + |w_1| \geq n_2 + |w_2|)$.

It is obvious that the *extension relation* among *index items* of some document is a partial order relation of which the maximal element is naturally defined. Thus, we arrive at a concept of *maximal index items* in an *index* with respect to this *extension relation*.

Definition 5 (maximal index item). Let *DOC* be a document, *DICT* be a dictionary, and *IND* be an *index* of *DOC* with respect to *DICT*. Then,

An *index item* in *IND* is a *maximal index item*
$\Leftrightarrow$ it has no *extension index items* in *IND* except for itself.

The *complete index*, our first solution to the problems, is naturally defined as follows:

Definition 6 (complete index). The *complete index* of a document *DOC* with respect to a dictionary *DICT* is a set of all possible *index items* of *DOC* with respect to *DICT*.

Under these definitions, we finally arrive at the definition of a *complete maximal word index*.

Definition 7 (complete maximal word index). The *complete maximal word index* of a document *DOC* with respect to a dictionary *DICT* is the set of all *maximal index items* in the *complete index* of *DOC* with respect to *DICT*.

We have one and only one *complete maximal word index* of *DOC* with respect to *DICT*. From this definition, the following claims are deduced:

Claim 1. Let *IND* be the *complete maximal word index* of a document *DOC* with respect to a dictionary *DICT*. Then, for each occurrence

```
0   procedure make_index($DOC$,$DICT$,$IND$)
1   add every single character as a word to $DICT$, forming $DICT'$
2   begin
3       $tail$ := 1; $l$ := $|DOC|$
4       for $head$=1 to $l$ do begin
5           $W$:={$w$ | $w$ is a prefix word of $DOC[head : l]$ in $DICT'$ }
6           $l\_prefix$ := the longest word in $W$;
7           if $head + |l\_prefix| > tail$ then begin
8               register ($l\_prefix, head$) to $IND$;
9               $tail := head + |l\_prefix|$;
10          end;
11      end;
12  end { of make_index };
```

Fig. 1 Algorithm for generating a complete maximal word index.

position n in DOC of a word w in $DICT$, either (a) or (b) holds:

(a) $(w, n) \in IND$.

(b) $\exists(w', n') \in IND$ s.t. (w', n') is an *extension index item* of (w, n).

This claim means that when we search for a word w in a text by using the *complete maximal word index*, every occurrence of w in a text is registered in IND as of the form (w, n) or of the form (w', n'), where w' is an extension word of w.

Claim 2. Let *IND* be the *complete maximal word index* of a document *DOC* with respect to a dictionary *DICT*, and let W be some set of words in $DICT$. We define the set of possible occurrence positions PP of W as follows:

$$PP(W) = \{\ p \mid ((w,n) \in IND) \wedge (w \in W) \wedge (n \leq p < n + |w|)\ \}.$$

We also define the set of *extension words* of a string *str* as follows:

$$E(str) = \{\ w' \mid w' \text{ is an } \textit{extension word} \text{ of } str \text{ in } DICT\ \}.$$

Then, all the occurrence positions of w in DOC are included in $PP(E(w))$.

This claim means that when we search for a word w in a text with the *complete maximal word index*, all we have to do is to find all the occurrence positions of all the *extension words* of w in the index.

3. Algorithms

3.1 Algorithm for Generating a Complete Maximal Word Index

The algorithm for generating a *complete maximal word index* for a document *DOC*, using a dictionary *DICT*, is shown in **Fig. 1**.

In line 6, the algorithm finds the longest word l_prefix of all the possible words that begin at each character position *head* in the text. Then at line 8, it registers the *index item* $(l\_prefix, head)$ in the index if the *index item* has no *extension index items* in the index.

It is easy to prove that each registered *index item* is *maximal*, and that all the *maximal index items* are indeed registered by this algorithm. Thus, this algorithm produces the *complete maximal word index.*

3.2 Algorithm for Searching for Arbitrary Strings with the Complete Maximal Word Index (Outline)

While the process of searching for words is very simple, it might become slightly complicated if the search string is not a word. We will therefore give an outline of the searching algorithm here.

Let *DOC* be a document, *DICT* be a dictionary, IND be the *complete maximal word index* of DOC with respect to $DICT$, and *str* be the search string. First, we define several words, sets of words, and numbers relative to *str*, for convenience in the following discussion.

$$L(str) = \{\ w \mid w \text{ is a } \textit{postfix extension word} \text{ of } str \text{ in } DICT\ \},$$
$$R(str) = \{\ w \mid w \text{ is a } \textit{prefix extension word} \text{ of } str \text{ in } DICT\ \},$$
$$p(str) = \text{the longest of all the } \textit{prefix words} \text{ of } str,$$
$$s(str) = \text{the longest of all the } \textit{postfix words} \text{ of } str,$$
$$i = |p(str)|,\ j = |s(str)|,\ n = |str|, \text{ and } m = n - j + 1.$$

As explained in Section 2.2, when the search string *str* is a word in $DICT$, what the algorithm has to find is $PP(E(str))$. (This is a direct result of **Claim 2.**) To pinpoint the exact occurrence positions of *str*, it has to examine $PP(E(str))$ and adjust the occurrence position of each *extension word* in $E(str)$ to

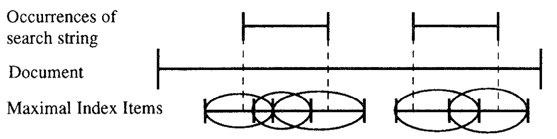


Fig. 2 Situation in which the search string is not a word.

the real occurrence position of *str*☆. We write $P(E(str))$ as a set of all occurrence positions of *str*, each element of which is adjusted from the occurrence position of each word in $E(str)$. Then, we write P_0 to denote all the exact occurrence positions of *str*.

$$P_0 = P(E(str))^{☆☆}$$

When *str* is not a word, the algorithm has to find all the possible sequences of *index items* from *IND*, each of which covers *str*. As shown in **Fig. 2**, some occurrence of a search string might be covered by a sequence of three *index items* in *IND*, while another occurrence might be covered by a sequence of two *index items.*

Such a possible sequence can be divided into two parts: one of them covering the prefix of *str*, and the other covering the postfix of *str*. We call these parts of the sequence the *left-hand extension sequence* and the *right-hand extension sequence.* All the occurrence positions of *str* can be calculated from those parts of the sequence. That is, the occurrence positions of *str* must be the positions of the adjacent pair of the *left-hand extension sequence* and the *right-hand extension sequence.* We will represent this fact as follows:

$$PE(str,a,b) = PL(str,a) \cap PR(str,b).$$

Here, $PE(str,a,b)$ is the set of occurrence positions of *str* that are the positions of the adjacent pair of the *left-hand extension sequence*, which as a whole is the *postfix extension* of $str[1:a]$, and the *right-hand extension sequence*, which as a whole is the *prefix extension* of $str[b:n]$. $PL(str,a)$ and $PR(str,b)$ are the normalized positions of the *left-hand extension sequence* and the *right-hand extension sequence.* **Figure 3** shows this situation.

$PL(str,a)$ and $PR(str,b)$ can be calculated as follows:

☆ When *str* (for example, "abc") has some *extension words* that contain iterative occurrences of *str* (for example, "abcdabc"), one occurrence of the extension word could produce multiple occurrence positions of *str*.

☆☆ We represent $P(X)$ for any string X or set of strings X as the set of all occurrence positions for the string or set of strings, each occurrence position of which is adjusted to the beginning point of the search string.

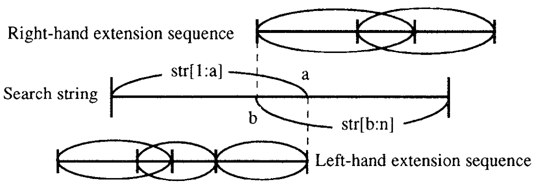


Fig. 3 Adjacent pair of extension sequences.

$$PL(str,k) = \begin{cases} P(L(str[1:k])) & \text{when } k=i, \\ P(L(str[1:k])) \cup \\ \quad (P(s(str[1:k])) \cap \\ \quad\quad pl(\max(i, k-|s(str[1:k])|), k)) & \text{when } i<k<n, \end{cases}$$

$$PR(str,k) = \begin{cases} P(R(str[k:n])) & \text{when } k=m, \\ P(R(str[k:n])) \cup \\ \quad (P(p(str[k:n])) \cap \\ \quad\quad pr(k, \min(m, k+|p(str[k:n])|))) & \text{when } 1<k<m, \end{cases}$$

where

$$pl(t,e) = \textstyle\bigcup_{t \le l < e} PL(str,l),$$
$$pr(t,e) = \textstyle\bigcup_{t < l \le e} PR(str,l).$$

Here again, $P(L(str[1:k]))$, $P(s(str[1:k]))$, $P(R(str[k:n]))$, and $P(p(str[k:n]))$, are the set of occurrence positions, which are adjusted to the real occurrence positions of *str*.

$PE(str,a,b)$ has to be calculated for all possible a and b. Thus we define P_1 as follows:

$$P_1 = \textstyle\bigcup_{a,b} PE(str,a,b),$$

where $i \le a < n$, $1 < b \le m$, $a \le b-1$.

Finally, the occurrence positions P of the non-word string *str* can be calculated as the union of P_0 and P_1.

$$P = P_0 \cup P_1.$$

This is an outline of the searching algorithm. With this algorithm, we can obtain every occurrence position of arbitrary strings systematically.

4. Implementation Issues

4.1 Configuration of the Index File

As explained in Section 3.2, it is often necessary to find all the *extension words* of a certain substring of a search string. L, R, and E, which are defined in Sections 2.2 and 3.2, correspond to such an operation. The structure of the index file should ensure fast retrieval of such *extension words.* **Figure 4** shows the configuration of the index file. It contains an **occurrence word table** in which the words in the *complete maximal word index* are sorted in lexicograph-

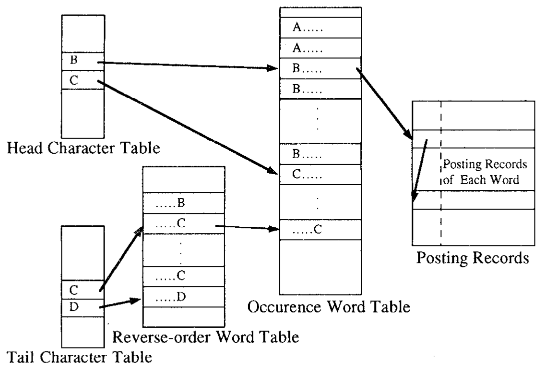


Fig. 4 Configuration of the index file.

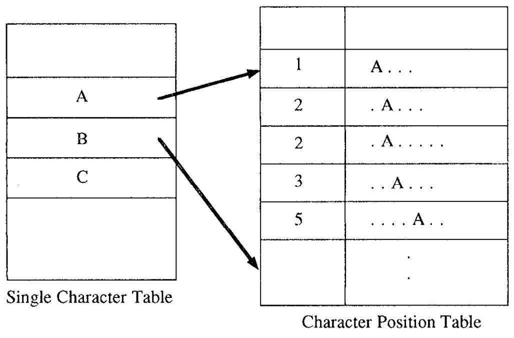


Fig. 5 Configuration of the additional structure.

ical order together with the posting records for each word. It also contains the **head character table**, each character of which points to the first word in the **occurrence word table** that begins with that character. This structure enables us to find all the *prefix extension words* of an arbitrary string quickly. It also contains a **tail character table** and a **reverse-order word table** in order to ensure fast retrieval of all the *postfix extension words* of an arbitrary string.

Figure 5 shows an additional structure to ensure fast retrieval of *extension words* in general. The **single character table** contains every character, with a pointer to the portion of the **character position table** in which all the dictionary words including that character are listed. The **character position table** contains all the words that include a character, together with the position of the character in each word. This structure enables us to find all the *extension words* of an arbitrary string.

Although the number of *index items* in the *complete maximal word index* is much smaller than the number of the *complete index*, the size could be very large, because the index file contains their posting records as well. In order

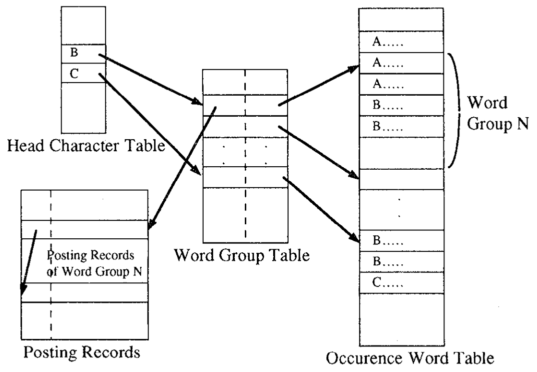


Fig. 6 Configuration of the index file (low-frequency words).

to compress the index file, the posting records are registered in ascending order, and the differences among subsequent postings, instead of the postings themselves, are registered. Since the difference is usually much smaller than the posting itself, the posting records can be compressed. But for the posting records of low-frequency words, it is possible that the differences among subsequent postings are relatively big, which results in a low compression rate. In order to avoid such results, the words in the **occurrence word table** are divided into two sets, a set of high-frequency words and a set of low-frequency words, and the index for a set of low-frequency words is constructed as shown in **Fig. 6**. Here we have the **word group table** as well, each entry of which corresponds to a group of low-frequency words. Each element of the **posting records** contains postings of all the words in a group.

Those configurations are chosen to ensure a compact index and fast retrieval of *extension words*.

4.2 Revision of the Dictionary

In the n-gram-based index file method, the search speed decreases when the length of the search string is greater than n. The same thing happens in the *complete maximal word index* file method when the search string is longer than a word: that is, when it is a compound word.

As explained in Section 3.2, in order to search a compound word, it is first necessary to find every occurrence position of each component word, then to check the adjacency of all the occurrence positions of each component word. For example, in order to search for "東日本", which is not a word in the dictionary, we have to search for "東" and "日本", and then check the adjacency of all the occurrence positions of "東" and all the occurrence positions of "日本". The search time depends on the sum of the numbers of occurrences of "東" and "日本". Typical problematic cases are those in which the component word is very frequent but the compound word is not. In the above example, it is likely that the component words "東" and "日本" are indeed frequent but the compound word "東日本" is not. A relatively long search time will be needed for search strings of this type.

In this problem, we can improve the situation drastically by revising the dictionary. The basic idea of the revision is as follows:

For each word that is very frequent in the text, add some extension words to the dictionary as dummy words.

The problem is that ordinary text contains some very frequent words. If we can remove such words from the *maximal word index*, then the search time will be improved. One way of doing this is to add to the dictionary all the strings consisting of the frequent word and an arbitrary single character. In the above example, a set of new words,

{ あ東, い東,...., 東あ, 東い,....},

is added to the dictionary. If we use the revised dictionary to create a *complete maximal word index*, the occurrence positions of "東" are split into the occurrence positions of { あ東, い東,...., 東あ, 東い,....}. This revision drastically reduces the frequency of the word "東", thus eventually reducing the search time for a search string containing "東".

We will see some effects of this revision in Section 5.3.

5. Experiments

An experimental system was implemented on a SUN SS20 workstation (CPU: SuperSPARC x 2/50 MHz, 128 MB main memory). To evaluate the performance of the system, we used data sets taken from newspaper articles, the largest of which contained about 400 MB of text.

We conducted two experiments in order to compare the *complete maximal word index* with the usual n-gram-based index.

5.1 Size of the Index File

Figure 7 shows the size of the *complete maximal word index* generated by the system. The horizontal axis represents the ratio of the number of high frequency words to the total number of words registered in the index. The vertical axis represents the ratio of the index size to the original document size.

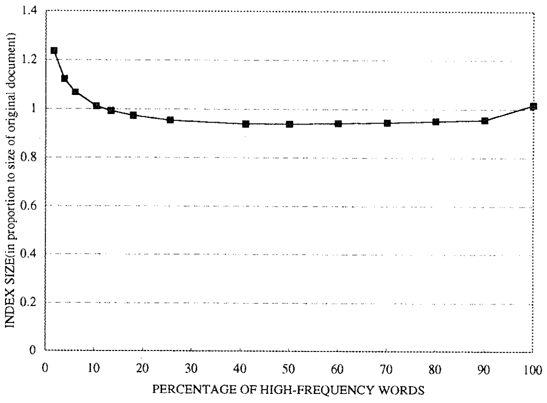


Fig. 7 Index size.

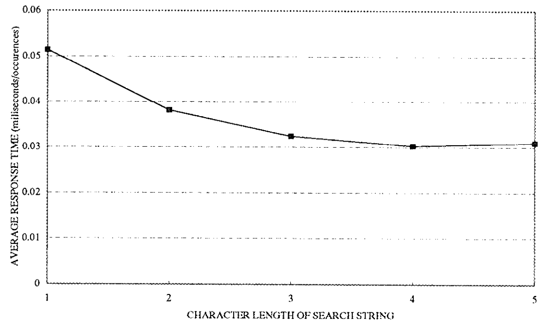


Fig. 8 Average response time for searching for words.

The index size varies according to how many words are registered as high-frequency words. It is shown that for a high-frequency word ratio of between 20 percent and 90 percent, the index is smaller than the original text, which is a considerable improvement over the size of an n-gram-based index.

5.2 Response Time for Searching for Words

Figure 8 shows the response time for searching for words in the dictionary. Words were classified according to their character length. We measured the average response time for each class of words. The horizontal axis represents such classes of words with length 1 to 5, and the vertical axis represents the average response time for searching for one occurrence of each word in these classes of words.

As we expected, the average response time is constant, which means that it depends only on the number of occurrences of the search word and not on the length of the word. With the n-gram-based index, the response time would increase in proportion to the length of the search string. With the *complete maximal word in-*

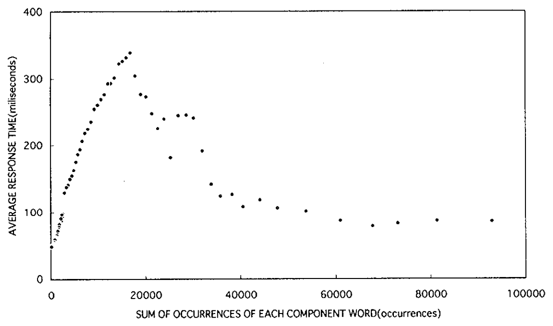


Fig. 9 Average response time for searching compound words.

dex, the response time for searching for an arbitrary string is much improved provided the string is a word in the dictionary. When we use a very equipped dictionary in which almost all the search strings are registered as words, the proposed index would greatly improve the entire performance of the retrieval system.

5.3 Response Time for Searching for Compound Words

Figure 9 shows the response time for searching for compound words each of which comprises two words. Before the experiment, a revision of the dictionary, which we described in Section 4.2, for top 300 high-frequency words has been done.

In searching for compound words, the response time would roughly be the sum of the search time for each component word. From Fig. 8, it appears that the response time for a component word is proportional to the number of occurrence in the text. Thus it can be logically inferred that the response time for searching for a compound word is proportional to the sum of the numbers of occurrences of its component words.

For this experiment, compound words were first classified so that each compound word in one class had almost the same sum of the numbers of occurrences of its component words. The response times for all compound words in a class were then averaged. In Fig. 9, the horizontal axis represents classes of compound words, depicting the sum of the numbers of occurrences of each component word. We could predict that the dots should be plotted in a line. In fact, in the left-hand part of this graph, the dots are roughly in line☆. But in the right-hand part, they drop away suddenly and then level out gradually. Here we see the effect of the revisions of the dictionary that we mentioned in Section 4.2. Each compound word in the classes plotted here has a very frequent word as one of its components. But the number of occurrences of such frequent words decreases as a result of the revision, as shown in the right-hand side of the graph.

As stated in Section 2.1, we added every single character to the dictionary before we constructed the index, so that every search string can be seen as a compound word. Thus, the proposed index is practical even for searching for compound words (which is identical to searching for arbitrary strings), if this kind of revision of the dictionary is done correctly.

6. Conclusion

We have proposed a new type of word-based index, called a *complete maximal word index*, suitable for texts of continuous sequence languages such as Japanese and Chinese. This index solves the problems of using the current word-based index file method in full-text retrieval systems for such types of text.

The formal definition of a *complete maximal word index* was given, and its generating algorithm and searching algorithm were then described.

We described some experimental results and confirmed that this approach is in fact effective and practical. The conventional *n*-gram-based index file method has several problems as regards its index size and search speed. We showed that our approach can offer a solution to these difficulties.

IR systems based on inexact-match retrieval models, such as probabilistic or vector space models, are now in practical use. Those systems have a relevance ranking or relevance feedback function that uses word frequency information in the stored text. The proposed method is promising in that it can be applied to such systems without any computation, because the index is word-based, which means that it contains word frequency information☆☆.

☆ As in Fig. 8, all the words that included low-frequency words as well as high-frequency words were tested. In the case of low-frequency words, the response time has a certain overhead, so the average response time would be somewhat larger than the exact value. That is why the average time per occurrence depicted in Fig. 8 is somewhat larger than the value we can calculate from Fig. 9.

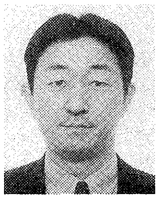
☆☆ Some reports state that n-gram based information is also useful for such inexact match retrieval models[2),11)].

Acknowledgments We would like to express our thanks to anonymous referees for their helpful comments on earlier version of this paper.

References

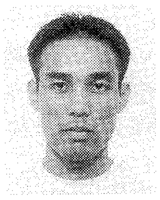
1) Frakes, W.B. and Baeza-Yates, R.A. (Eds.): *Information Retrieval: Data Structures and Algorithms*, Prentice-Hall, Englewood Cliffs, NJ (1992).
2) Fujii, H. and Croft, W.B.: A Comparison of Indexing Techniques for Japanese Text Retrieval, *Proc. 16th ACM SIGIR Conference*, pp.237–246 (1993).
3) Gonnet, H.G., Baeza-Yates, R.A. and Snider, T.: New Indices for Text: PAT Trees and PAT Arrays, in Ref. 1), pp.66–82 (1992).
4) Hatakeyama, A., et al.: Implementation of Software Text Search Machine (in Japanese), Special Interest Group Notes of the IPSJ, Vol.FI25, pp.19–26 (1992).
5) Hisamitsu, T. and Nitta, Y.: A Generalized Algorithm for Japanese Morphological Analysis and Comparative Estimation of Some Heuristics (in Japanese), *Trans. IEICE*, Vol.J77-D-II, No.5, pp.959–969 (1994).
6) Inaba, M., Kurachi, K., Noguchi, N. and Kanno, Y.: New Indices for Japanese Text: Their implementation, experiments and evaluation (in Japanese), *Proc. 50th Annual Convention IPS Japan*, pp.4-43–4-44 (1995).
7) Kikuchi, C.: A Fast Full-Text Search Method for Japanese Text Database (in Japanese), *Trans. IEICE*, Vol.J75-D-I, No.9, pp.836–846 (1992).
8) Kurachi, K., Inaba, M., Noguchi, N. and Kanno, Y.: New Indices for Japanese Text: The principle of making an index and searching index (in Japanese), *Proc. 50th Annual Convention IPS Japan*, pp.4-41–4-42 (1995).
9) Nomoto, M. and Noguchi, N.: A Ranking Strategy Incorporating Document Structure and Cooccurrence (in Japanese), *Proc. 52nd Annual Convention IPS Japan*, pp.4-203–4-204 (1996).
10) Ogawa, Y.: A New Character-based Indexing Method using Frequency Data for Japanese Documents, *Proc. 18th ACM SIGIR Conference*, pp.121–129 (1995).
11) Ogawa, Y.: An Efficient Document Ranking Retrieval Method Using n-gram-based Signature Files, *Trans. IPS Japan*, Vol.38, No.11, pp.2286–2297 (1997).
12) Salton, G. and McGill, M.J.: *Introduction to Modern Information Retrieval*, McGraw Hill, New York (1983).
13) Tomohiro, S., et al.: Bibliotheca/TS: Japanese full-text retrieval system (in Japanese), *Online Kensaku*, Vol.13, No.3, pp.142–147 (1992).

(Received September 9, 1996)
(Accepted February 2, 1998)

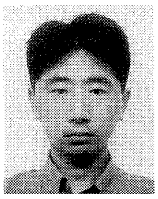


Naohiko Noguchi was born in 1960. He received his M.E. degree from Tokyo Univ. in 1985. He has been working in Matsushita Electric Industrial, Co., Ltd. since 1985 and has been engaging in research on natural language processing, discourse understanding, semantics and pragmatics of natural language, and information retrieval. Since 1990 until 1992 he had been a visiting researcher of Stanford University. He is a member of IPSJ, JCSS, ACL, ACM, and AAAI.

Yuji Kanno was born in 1960. He received his M.S. degree from Tohoku Univ. in 1985. He has been working in Matsushita Electric Industrial, Co., Ltd. since 1985 and has been engaging in research and development on natural language processing, expert systems, string matching algorithm, and information retrieval. He is also interested in Shougi playing algorithm. He is a member of IPSJ.



Mitsuaki Inaba was born in 1967. He received his B.E. degree from Tokyo Univ. in 1991 and his M.E. degree in 1993. He has been working in Matsushita Electric Industrial, Co., Ltd. since 1993 and has been engaging in research on information retrieval and in development of text retrieval systems. He is a member of IPSJ.



Kazuaki Kurachi was born in 1968. He received his B.E. degree from University of Electro-Communications in 1991. He has been working in Matsushita Electric Industrial, Co., Ltd. since 1991 and has been engaging in research on information retrieval. He is a member of IPSJ.