Backtracking-Free Dictionary Access Method for Japanese Morphological Analysis

Hiroshi Maruyama
IBM Research, Tokyo Research Laboratory
maruyama@trl.ibm.co.jp

1 Introduction

Since the Japanese language does not have explicit word boundaries, dictionary lookup should be done, in principle, for all possible substrings in an input sentence. Thus, Japanese morphological analysis involves a large number of dictionary accesses.

The standard technique for handling this problem is to use the TRIE structure to find all the words that begin at a given position in a sentence (Morimoto and Aoe 1993). This process is executed for every character position in the sentence; that is, after looking up all the words beginning at position n, the program looks up all the words beginning at position $n + 1$, and so on. Therefore, some characters may be scanned more than once for different starting positions.

This paper describes an attempt to minimize this 'backtracking' by using an idea similar to one proposed by Aho and Corasick (Aho 1990) for multiple-keyword string matching. When used with a 70,491-word dictionary that we developed for Japanese morphological analysis, our method reduced the number of dictionary accesses by 25%.

The next section briefly describes the problem and our basic idea for handling it. The detailed algorithm is given in Section 3 and Section 4, followed by the results of an experiment in Section 5.

Input sentence: 二十世紀最後の十年に入った。
JMA output:

```
二十世紀:19 最後:19 の:76
十年:19 に:78
入:9 っ:29 た:63 。:100
```

Fig. 1: Sample input/output of JMA

2 Japanese Morphological Analysis

2.1 Grammar

A Japanese morphological analyzer (hereafter called the JMA) takes an input sentence and segments it into words and phrases, attaching a part-of-speech code to each word at the same time. Figure 1 shows a sample input and the output of our JMA.

The grammaticality of a sequence of Japanese words is mainly determined by looking at two consecutive words at a time (that is, by looking at two-word windows). Therefore, Japanese morphological analysis is normally done by using a Regular Grammar (e.g., Maruyama and Ogino 1994). Our JMA grammar rules have the following general form:

state1 → "word" [*linguistic-features*]
state2 `cost=`*cost*.

Each grammar rule has a heuristic cost, and the parse with the minimum cost will be selected as the most plausible morphological reading of the input sentence. A part of our actual gram-

mar is shown in Figure 2. Currently our grammar has about 4,300 rules and 400 nonterminal symbols.

2.2 Dictionary Lookup

While the function words (particles, auxiliary verbs, and so on, totaling several hundred) are encoded in the grammar rules, the content words (nouns, verbs, and so on) are stored in a separate dictionary. Since content words may appear at any position in the input sentence, dictionary access is tried from all the positions n.

For example, in the sentence fragment in Figure 3,

> "大型 (large)" and "大型計算機 (mainframe)"

are the results of dictionary access at position 1. For simplicity, we assume that the dictionary contains only the following words:

> "大型 (large),"
> "大型計算機 (mainframe),"
> "計算機 (computer)",
> "計算機設備 (computing facility),"
> and
> "設備 (facility)." [1]

2.3 Using TRIE

The most common method for dictionary lookup is to use an index structure called TRIE (see Figure 4). The dictionary lookup begins with the root node. The hatched nodes represent the terminal nodes that correspond to dictionary entries. At position 1 in the sentence above, two words, "大型 (large)" and " 大型計算機 (mainframe)," are found.

Then, the starting position is advanced to the second character in the text and the dictionary lookup is tried again. In this case, no word is found, because there are no words that begins with "型" in the dictionary. The starting position is then set to 3 and tried again, and this time the words "計算機 (computer)" and " 計算機設備 (computing facility)" are obtained. [2]

The problem here is that, even though we know that there is "大型計算機 (mainframe)" at position 1, we look up "計算機 (computer)" again. Since "計算機 (computer)" is a substring of "大型計算機 (mainframe)," we know that the word "計算機 (computer)" exists at position 3 as soon as we find "大型計算機 (mainframe)" at position 1. Therefore, going back to the root node at position 3 and trying matching all over again means duplicating our efforts unnecessarily.

2.4 Eliminating Backtracking

Our idea is to use the index structure developed by Aho and Corasick to find multiple strings in a text. Figure 5 shows the TRIE with a pointer called the *fail pointer* associated with the node corresponding to the word "大型計算機 (mainframe)" (the rightmost word in the first row). When a match starting at position n reaches this node, it is guaranteed that the string "計算機" exists starting at position $n+2$. Therefore, if the next character in the input sentence does not match any of the child nodes, we do not go back to the root but go back to the node corresponding to this substring by following the fail pointer, and resume matching from this node. For the input sentence in Figure 3, the dictionary access proceeds as indicated by the dotted line in the Figure 5, finding the words "大型 (large)," "大型計算機 (mainframe)," "計算機 (computer)," and so on. Thus, the number of dictionary node accesses is greatly reduced.

In many Japanese morphological analysis systems, the dictionary is held in the secondary storage, and therefore the number of dictionary

[1] Actual dictionaries also contain "大 (big)," "型 (type)," "計 (measure)," "計算 (compute)," "算 (order)," "機 (machine)," "設 (establish)," and "備 (prepare)."

[2] The fact that "大型計算機 (mainframe)" was found before does not necessarily mean that there is no need to look up "計算機..(computer ...)," because at this point two interpretations, "mainframe facility" and "large computing facility," are possible.

```
文節頭 -> KAGYOU-5DAN [pos=1,kow=doushi] カ行５段;
カ行５段 -> "か" [pos=26,kow=v_infl] 動詞５段未然ナイ接続 cost=300;
動詞５段未然ナイ接続 -> "ず" [pos=64,kow=jodoushi,fe={negative}]
                        "に" [pos=78,kow=setsuzoku_joshi]
                        接続助詞ガ cost=500;
接続助詞ガ -> 文節尾 cost=999;
名詞 -> "" [pos=48,kow=jodoushi] 助動詞ダ語幹 cost=500;
助動詞ダ語幹 -> "だろ" [pos=27,kow=aux_infl] 未然ウ接続 cost=300;
助動詞ダ語幹 -> "だっ" [pos=45,kow=aux_infl] 連用タ cost=300;
```

Fig. 2: Some of the grammar rules

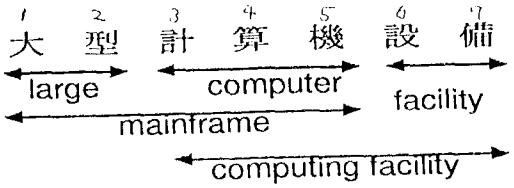

Fig. 3: Dictionary lookup for Japanese morphological analysis

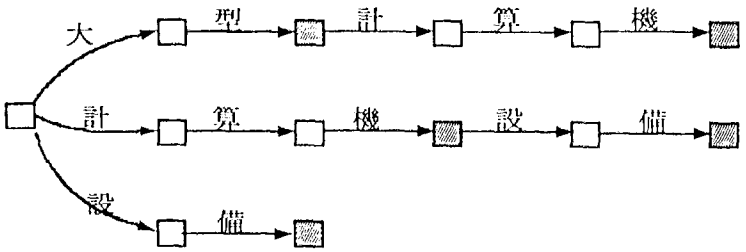

Fig. 4: TRIE index

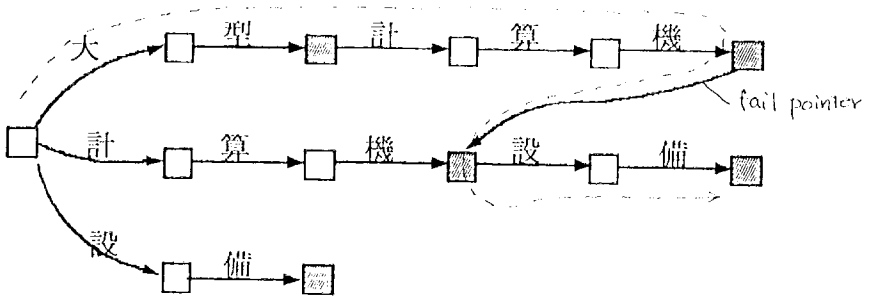

Fig. 5: TRIE structure with *fail pointers*

node accesses dominates the performance of the overall system.

2.5 Other Cosiderations

Theoretically there is a possibility of pruning dictionary lookup by using the state set at position n. For example, if no noun can follow any of the states in the current state set, there is no need to look up nouns. One way to do this pruning is to associate with each node a bit vector representing the set of all parts of speech of some word beyond this node. If the intersection of the expected set of parts of speeche and the possibilities beyond this node is empty, the expansion of this node can be pruned. In general, however, almost every character position predicts most of the parts of speech. Thus, it is common practice in Japanese morphological analysis to look up every possible prefix at every character position.

Hidaka et al. (1984) used a modified B-tree instead of a simple TRIE. Altough a B-tree has much less nodes than a TRIE and thus the number of secondary storage accesses can be significantly reduced, it still backtracks to the next character position and duplicate matching is inevitable.

3 Constructing TRIE with fail pointers

A TRIE index with fail pointers is created in the following two steps:

1. Create a TRIE index, and
2. Calculate a fail pointer of each node in the TRIE.

Since Step 1 is well known, we will describe only Step 2 here.

For each node n, Step 2 gives the value $fail(n)$. In the following algorithm, $forward(n, c)$ denotes the child node of the node n whose associated character is c. If there is no such node, we define $forward(n, c) = \text{nil}$. *Root* is the root node of the TRIE.

2-1 $fail(Root) \leftarrow Root$

2-2 for each node n of depth 1, $fail(n) \leftarrow Root$

2-3 for each depth $d = 1, 2, ...,$

  2-3-1 for each node n with depth d,

    2-3-1-1 for each child node m of n (where $m = forward(n, c)$),
    $fail(m) \leftarrow f(fail(n), c)$.

Here, $f(n, c)$ is defined as follows:

$$f(n,c) = \begin{cases} fail(n) & \text{if } forward(n,c) \neq nil \\ f(fail(n),c) & \text{if } forward(n,c) = nil \\ & \quad \& \; n \neq Root \\ Root & \text{otherwise} \end{cases}$$

If the node corresponds to the end of some word, we record the length l of the word in the node. For example, at the node that corresponds to the end of the word "大型計算機 (mainframe)", $l = 5$ and $l = 3$ are recorded because it is the end of both of the words "大型計算機 (mainframe, $l = 5$)" and "計算機 (computer, $l = 3$)." [3]

Figure 6 shows the complete TRIE with the fail pointers.

4 Dictionary access

The algorithm for consulting the dictionary is quite simple:

1 $n \leftarrow Root$

2 for each character position $i = 1, 2, ...k$,

2-1 while $n \neq Root$ and $forward(n, c_i) =$ nil do $n \leftarrow fail(n)$

2-2 $n = forward(n, c_i)$

2-3 if n is the end of some word(s), output them

where c_i is the character at position i.

5 Experimental results

We applied the TRIE with fail pointers to our 70,491-word dictionary for Japanese morphological analysis (in which the average word length is 2.8 characters) and compared it with a conventional TRIE-based system, using two sets of data: newspapers articles (44,113 characters) and computer manuals (235,104 characters). The results are shown in Table 1.

The tables show both the number of node accesses and the actual CPU time. For the newspaper articles, our method (marked as TRIE w/ FP) had 25% fewer node accesses than than the traditional TRIE and was 27% faster in CPU time. The CPU time was measured with all the nodes in the main memory.

For the computer manuals, the reduction rate was a little larger. This is attributable to the fact that computer manuals tend to contain longer, more technical terms than newspaper articles. Our method is more effective if there are a large number of long words in a text.

[3] This information is redundant, because one can look up every possible word by following the fail pointer. However, if the nodes are in secondary storage, it is worth having the length information within the node to minimize the disk access.

6 Conclusion

We have proposed a new method of dictionary lookup for Japanese morphological analysis. The idea is quite simple and easy to implement with a very small amount of overhead (a fail pointer and an array of length l to each node). For large terminology dictionaries (medical, chemical, and so on), this method will greatly reduce the overhead related to dictionary access, which dominates the efficiency of practical Japanese morphological analyzers. Fast Japanese morphological analyzers will be crucial to the success of statistically-based language analysis in the near future (Maruyama et al. 1993).

References

1. Aho, A. V., 1990: "Algorithms for Finding Patterns in Strings," in Leeuwen, J.V. ed., *Handbook of Theoretical Computer Science, Volume A – Algorithms and Complexity*, pp. 273-278, Elsevier.

2. Hidaka, T., Yoshida, S., and Inanaga, H., 1984: "Extended B-Tree and Its Applications to Japanese Word Dictionaries," (In Japanese) *Trans. of IEICE*, Vol. J67-D, No. 4.

3. Hisamitsu, T. and Nitta, Y., 1991: "A Uniform Treatment of Heuristic Methods for Morphological Analysis of Written

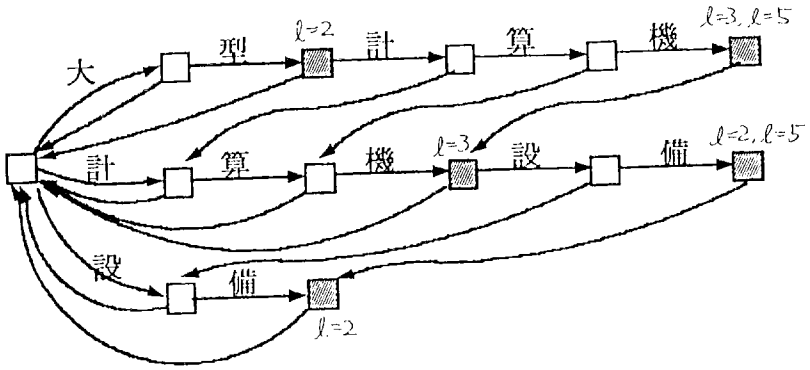

Fig. 6: TRIE index with fail pointers

	TRIE	TRIE w/ FP	Reduction rate
Node accesses	104,118	78,706	25%
CPU time (sec.)	64.77	40.92	27%

(a) 44,113 characters in newspaper articles

	TRIE	TRIE w/ FP	Reduction rate
Node accesses	542,542	383,176	30%
CPU time (sec).	372.47	228.63	28%

(b) 235,104 characters in computer manuals

Table 1: Experimental results

Japanese," *Proc. of 2nd Japan-Australia Joint Workshop on NLP.*

4. Maruyama, H., Ogino, S., and Hidano, M., 1993: "The Mega-Word Tagged-Corpus Project," *Proc. of 5th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-93)*, Kyoto, Japan.

5. Maruyama, H. and Ogino, S., 1994: "Japanese Morphological Analysis Based on Regular Grammar," (In Japanese), *Transactions of IPSJ*, to appear.

6. Morimoto, K. and Aoe, J., 1993: "Two Trie Structures for Natural Language Dictionaries," *Proc. of Natural Language Processing Pacific Rim Symposium (NLPRS '93)*, Fukuoka, Japan.