A Fast Morphological Analysis Using the Extended AC Machine for Oriental Languages[1]

Kazuaki Ando, Kimihiro Iwasaki, Masao Fuketa and Jun-ichi Aoe

Department of Information Science & Intelligent Systems
University of Tokushima
2-1 Minami-Josanjima-Cho
Tokushima-Shi 770-8506
Japan

e-mail: {ando,aoe}@is.tokushima-u.ac.jp

Abstract. This paper presents a fast morphological analysis for oriental languages by extending an Aho and Corasick's pattern matching machine. Our method is a simple and efficient algorithm to find all possible morphemes in an input sentence and in a single pass, and it stores the relations of grammatical connectivity of adjacent morphemes into the output functions. Therefore, the costs of checking connections between the adjacent morphemes can be reduced by using the connectivity relations. Furthermore, the construction method of the relations of grammatical connectivity is described. Finally, the proposed method is verified by a theoretical analysis, and an experimental estimation is supported by the computer simulation with a 100,267 words dictionary. From the simulation results, it turns out that the proposed method was 49.9% faster (CPU time) than the traditional trie approach. As for the number of candidates for checking connections, it was 25.5% less than that of the original morphological analysis.

Key words: morphological analysis, oriental language, dictionary lookup, trie structure, AC machine, grammatical connectivity

1 Introduction

An intelligent natural language interfaces enable users to communicate with the computer in English, Japanese or other human languages. Morphological analysis [ABE86, AKI94, KUR94, LEE97, MAR94, MOR96, SAN94] is the first step of natural language processing in the applications of natural language interfaces such as Information Retrieval [AOE91], Database Queries [KAP84], Expert Systems and so on. In general, the morphological analysis means segmentations of the input sentence into words (morphemes) and attachments of part-of-speech to them. Therefore, although morphological analysis for European languages, especially for English, plays only a minor role in a natural language processing system, in the analysis of oriental languages

[1]This work was supported by the Grant-in-Aid of the Ministry of Education, Science and Culture, Japan.

such as Japanese, Chinese and Korean it plays an important role because oriental languages are agglutinative languages, that is the language do not have explicit word boundaries between the words [ABE86, AKI94, KUR94, MAR94, MOR96, SAN94].

The procedure of morphological analysis of oriental languages consists of two steps. The first is to detect all possible morphemes, which are the smallest meaningful units, in a given input sentence. The second is to find the possible connections between adjacent morphemes by using a connection cost or probability based on the grammaticality [ABE86, AKI94, SAN94]. In the first step, the morphological analysis involves a large number of dictionary lookup. In general, a well-known technique for dictionary lookup is to use a trie structure [AOE91, AOE96, KUR94]. The trie is a tree structure in which each transition corresponds to a key character in the given keys set and common prefixes of keys can be shared. Therefore, the trie can search all keys made up from prefixes in an input string without the need of scanning the structure more than once. However, it is not so effective to use the trie for the morphological analysis [KUR94, MAR94, MOR96]. In order to detect all possible substrings in a given input sentence, the dictionary access must be tried repeatedly at each character position in the input sentence. Therefore, some characters may be scanned more than once for different starting positions and the number of dictionary accesses is increased. In the second step, the morphological analysis checks grammatical connectivity between adjacent words in order to find all possible connections [ABE86, SAN94]. This grammatical connectivity can be easily checked by using a grammatical table [ABE86]. However, this process requires considerable cost to check the grammatical connectivity, because it includes some checks of unnecessary connections, for example, checking connection between NOUN and CONJUGATION, since many words as part of speech have different grammatical interpretations. In order to achieve a fast morphological analysis, the mentioned problems should be solved.

This paper proposes a high speed morphological analysis of oriental languages by extending a pattern matching machine based on Aho and Corasick machine (called AC machine) [AHO75]. The proposed method is a simple and fast algorithm to find all possible substrings in an input sentence, and during only a single scan. Moreover, since the proposed method stores relations of grammatical connectivity of adjacent words into the output functions, the cost of checking connections between the adjacent words can be reduced by using the connectivity relations.

In the following sections, our ideas are described in detail. In Section 2, we describe the dictionary lookup method using a trie structure for the morphological analysis. Section 3 presents the high speed morphological analysis by extending the AC machine. Section 4 shows the theoretical analysis, and the experimental evaluations verified by the computer simulations with a 100,267 words dictionary. Finally, the results are summarized and the future research is discussed.

2 Dictionary Lookup Method using Trie in the Morphological Analysis

Morphological analysis of oriental languages is very different from that of English [ABE86, AKI94, KUR94, MAR94, MOR96, SAN94], because the languages do not have explicit word boundaries between the words as shown Fig. 1. Therefore, in order

to find the most suitable word boundary, the morphological analysis must detect all possible substrings in a given input sentence in the first place. This process is one of the most important tasks of morphological analysis of oriental languages, since the wrong segmentation causes serious errors in the later analysis such as syntactic and semantic analysis [AKI94].

English : How lucky you are.

→ How / lucky / you / are / . /

Japanese : anatahanantekouunnanda = How lucky you are.

(a-na-ta-ha-na-n-te-ko-u-u-n-na-n-da)

→ a / na / ta / ha / na / n / te / ko / u / u / n / na / n / da /

ana / ta / ha / na / n / te / ko / u / u / n / na / n / da /

anata / ha / na / n / te / ko / u / u / n / na / n / da /

⋮

anata / ha / nante / kouun / na / n / da /

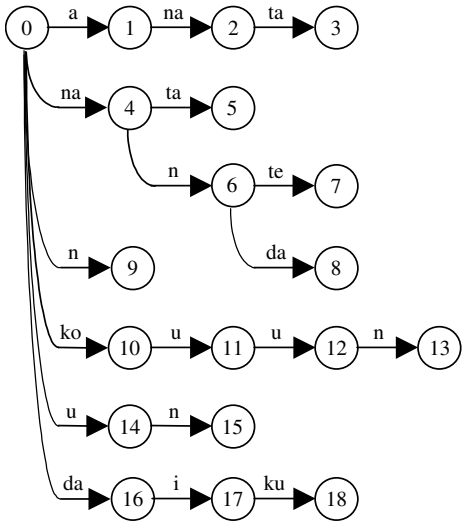
Figure 1: Difference in word boundary between English and Japanese.

In this process, the morphological analysis involves a large number of dictionary accesses. In general, a well-known method for dictionary lookup is to use a trie structure [AOE91, AOE96, KUR94]. The trie is a tree structure in which each transition corresponds to a character of the keys in the presented key set K. In the trie, a path from the root (initial state) to a leaf corresponds to one key in K. This way, the states of the trie correspond to the prefixes of keys in K.

The following is introduced for formal discussions:
1) S is a finite set of states, represented as a positive number.
2) I is a finite set of input symbols, or characters.
3) *goto* is a function from $S \times I$ to $S \cup \{fail\}$, called a goto function.
4) *output* is a function from S to morphological information, called an output function.
5) The state number of the trie is represented as a positive number, where the initial state in S is represented by the number 0.

A transition labeled with 'a' (in I) from r to n indicates that $goto(r, \text{'a'}) = n$. The absence of a transition indicates *failure* (*fail*). Fig. 2 shows an example of a trie for the set K = {"ana", "anata", "na", "nata", "nante", "nanda", "n", "ko", "kouu", "kouun", "u", "un", "da", "dai", "daiku"}. In this paper, for convenience of explanation, Japanese characters are described roman letters and a transition label is represented by the characters corresponding to the Japanese syllables. For example, retrieval of key "anata" is performed by traversing transitions $goto(0, \text{'a'}) = 1$, $goto(1, \text{'na'}) = 2$ and $goto(2, \text{'ta'}) = 3$, sequentially, and this time the key "ana"($=output(2)$) and "anata"($=output(3)$) are obtained.

In the morphological analysis, all possible substrings must be detected in order to find the most suitable word boundary. The following shows a dictionary lookup algorithm using a trie structure.



(a) The goto function.

s	*output*(*s*)
2	{ana} (hole)
3	{anata} (you)
4	{na}(name)
5	{nata}(hatchet)
7	{nante}(how)
8	{nanda}(what)
9	{n}(auxiliary verb)
10	{ko}(arc)
12	{kouu}(rainfall)
13	{kouun}(luck)
14	{u}(auxiliary verb)
15	{un}(destiny)
16	{da}(auxiliary verb)
17	{dai}(stand)
18	{daiku}(carpenter)

(b) The output function.

Figure 2: An example of TRIE.

Algorithm 1 : A dictionary lookup algorithm using a trie structure.
Input : A sentence $TEXT = c_1c_2...c_n$, where each c_i, for $1 \leq i \leq n$, is an input character, a goto function *goto* and output function *output*.
Output : The morphological information of all possible substrings in a given input sentence *TEXT*.
Method :

Step 1-1 : { Initialization }
$i \leftarrow 1$;
Step 1-2 : { Change of the starting position }
$state \leftarrow 0$; $j \leftarrow i$;
Step 1-3 : { State transitions }
$state \leftarrow goto(state, c_j)$;
if $state = fail$ **then goto** Step 1-5;
if $output(state) \neq \phi$ **then print** $output(state)$;
Step 1-4 : { Operation control }
$j \leftarrow j + 1$;
if $j \leq n$ **then goto** Step 1-3;
Step 1-5 : { Operation control }
$i \leftarrow i + 1$;
if $i \leq n$ **then goto** Step 1-2;

The trie is a very common structure for dictionary access. However, it is not so effective to use the trie for morphological analysis of oriental languages, because the dictionary access must be tried from every character position in the input sentence, in order to detect all possible substrings in a given input sentence. Therefore, some characters may be scanned more than once for different starting positions and the number of unnecessary dictionary accesses is increased.

Consider the following input sentence (see Fig.2).

$TEXT$ = "kouunnanda (ko-u-u-n-na-n-da)" (It is lucky for me.)

The morphological analysis tries to find all possible words starting with "ko"(i=1). Then, the starting position is advanced to the second character in $TEXT$ and the dictionary access is repeated. The dictionary access is executed repeatedly until the end of input.

Step 1 : $i = 1$("ko"); The goto function fails at the character "na"; Keys "ko", "kouu" and "kouun" are found.
Step 2 : $i = 2$("u"); The goto function fails at the next character "u"; A key "u" is found.
Step 3 : $i = 3$("u"); The goto function fails at the character "na". Keys "u" and "un" are found.
Step 4 : $i = 4$("n"); The goto function fails at the character "na". A key "n" is found.
Step 5 : $i = 5$("na"); Keys "na" and "nanda" are found.
Step 6 : $i = 6$("n"); The goto function fails at the next character "da". A key "n" is found.
Step 7 : $i = 7$("da"); A key "da" is found and the process is finished.

As shown above, the character "u"(i=2), "u"(i=3), "n"(i=4), "na"(i=5), "n"(i=6) and "da"(i=7) were scanned two, three, three, four, two and three times, respectively, that is, the dictionary lookup was repeated 7 times.

3 Morphological Analysis Using the AC Machine

3.1 Dictionary Lookup

It was observed in the preceding section that morphological analysis using the trie involves a large number of dictionary accesses. In this section, in order to solve this problem, an efficient string pattern matching machine is used. Here a finite state string pattern matching machine based on the AC machine [AHO75, MAR94] locates all occurrences of any of a finite number of keywords in a text string.

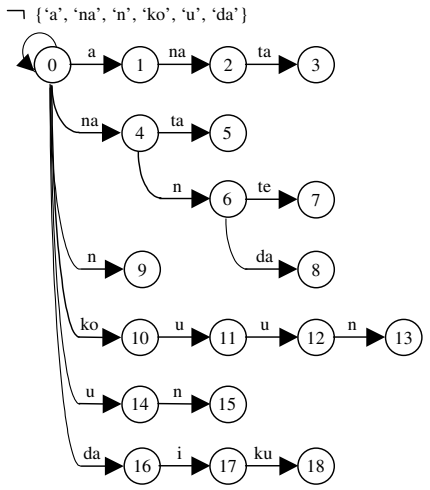
Let $KEY = \{k_1, k_2, ..., k_k\}$ be a finite set of strings which we shall call key and let $TEXT$ be an arbitrary string which we shall call the text string. The AC machine is a program which takes as input the text string $TEXT$ and produces as output the morphological information and the locations in $TEXT$ at which keys of KEY appear as substrings. The AC machine is constructed as a finite set of states S. Each state is represented by a number. One state (usually 0) is designated as the initial state.

The behavior of the AC machine is defined by the next three functions:

goto function $goto : S \times I \rightarrow S \cup \{fail\}$,
failure function $f : S \rightarrow S$,
output function $output : S \rightarrow A$, morphological information.

The function $goto$ maps a set consisting of a state and a character into a state or the message $fail$. The function f maps a state into a state. The failure function is constructed whenever the goto function reports the message $fail$. Certain states are designated as output states which indicate that a set of keys has been found. The function output formalizes this concept by associating a set of keys (possible empty) with each state.

Fig. 3 shows the functions used by the AC machine for the set of keys $K = \{$"ana", "anata", "na", "nata", "nante", "nanda", "n", "ko", "kouu", "kouun", "u", "un", "da", "dai", "daiku"$\}$. Here, $\neg$ {'a','na','n','ko','u','da'} denotes all input characters other than 'a','na','n','ko','u','da'. The directed graph in Fig. 3(a) represents the goto function and the dotted line represents the failure function. For example, the transition labeled 'a' from state 0 to state 1 indicates that $goto(0, \text{'a'}) = 1$. The absence of transition indicates *fail*. The AC machine has the property that $goto(0, \text{'}\sigma\text{'}) \neq fail$ for all input symbols σ. A dictionary lookup algorithm using the AC machine is summarized below.



(a) The goto function.

s	*output*(*s*)
2	{ana, **na**} (hole, **name**)
3	{anata} (you)
4	{na}(name)
5	{nata}(hatchet)
6	{**n**}(**auxiliary verb**)
7	{nante}(how)
8	{nanda,**da**}(what,**auxiliary verb**)
9	{n}(auxiliary verb)
10	{ko}(arc)
11	{**u**}(**auxiliary verb**)
12	{kouu,**u**}(rainfall,**auxiliary verb**)
13	{kouun,**un**}(luck,**destiny**)
14	{u}(auxiliary verb)
15	{un,**n**}(destiny,**auxiliary verb**)
16	{da}(auxiliary verb)
17	{dai}(stand)
18	{daiku}(carpenter)

(b) The output function.

$f(2)=4, f(6)=9, f(8)=16, f(11)=14, f(12)=14, f(13)=15, f(15)=9,$
$f(0)=f(1)=f(3)=f(5)=f(7)=f(9)=f(10)=f(14)=f(16)=f(17)=f(18)=0;$

(c) The failre function.

Figure 3: An example of the AC machine.

Algorithm 2 : A dictionary lookup algorithm using the AC machine.
Input : A sentence $TEXT = c_1c_2...c_n$, where each c_i, for $1 \leq i \leq n$, is an input character, an AC machine with goto function *goto*, failure function *f*, and output function *output*.
Output : The morphological information of all possible substrings in a given input sentence *TEXT*.
Method :

Step 2-1 : { Initialization }
$state \leftarrow 0$;
$i \leftarrow 1$;
Step 2-2 : { State transitions }
if $goto(state, c_i) \neq fail$ **then goto** Step 2-3;
$state \leftarrow f(state)$;
goto Step 2-2;
Step 2-3 : { Output operation }

$state \leftarrow goto(state, c_i)$;
if $output(state) \neq \phi$ **then print** $output(state)$;
Step 2-4 : { Operation control }
$i \leftarrow i + 1$;
if $i \leq n$ **then goto** Step 2-2;

Consider the behavior of the AC machine that uses the functions in Fig. 3 to process the text string "kouun"(lucky). Since $goto(0, \text{'ko'}) = 10$, the AC machine enters state 10, advances to the next input symbol and emits $output(10)$, indicating that it has found the key "ko". Similarly, since $goto(10, \text{'u'}) = 11$, $goto(11, \text{'u'}) = 12$, and $goto(12, \text{'n'}) = 13$, the AC machine finds the $output(11)$(="u"), $output(12)$(="kouu" and "u") and $output(13)$("kouuni" and "un") respectively and enters state 13.

As shown above, the AC machine can find all possible substrings in an input sentence, scanned only once. Therefore, the AC machine is the most advantageous method for the morphological analysis.

3.2 Connection Check

In the second step of morphological analysis for oriental languages, the grammatical connectivities between adjacent words which were obtained by the dictionary lookup are checked in order to find the most suitable word boundary [ABE86, SAN94]. This grammatical connectivity can be easily checked by using a grammatical table which is described in a matrix of the connectivity between two words. However, this process requires considerable cost, because it must check the relation of all parts of speech which the preceding word and the following word have, and the unnecessary checks are included in those checks since many words have different kinds of parts of speech.

Let us now consider the Japanese words written in the syllabic alphabet called Hiragana. For example, suppose that Hiragana character 'ka' has 14 kinds of parts of speech and 'i' has 19 kinds of parts of speech in our dictionary for the morphological analysis as shown Figure 4. As for checks of grammatical connectivities between the preceding character 'ka' and the following character 'i' ("kai" means a shellfish, a floor etc.), it involves 266 ($=14 \times 19$) kinds of checks. However, these checks includes 126 ($=14 \times 9$) kinds of unnecessary checks such as checking a grammatical connectivity between NOUN and CONJUGATION. Such kinds of parts of speech, represented as bold types in Fig. 4, are called unconnection candidates. In other words, the unconnection candidate is a part of speech of the following word which is not connectable to all parts of speech of the preceding word. The problem of how to reduce the number of unnecessary checks should be solved in order to achieve a fast morphological analysis.

Thus, we extend one of the features of the AC machine in which information of substring are stored in one pass from initial state to terminal state by the failure function. By using this feature, the grammatical connectivities between the adjacent substrings which are included in one pass can be checked in advance, and the results can be stored into each output function as the unconnection candidate when the dictionary of morphological analysis is constructed. Therefore, if the unconnection candidate is available throughout the execution of the morphological analysis, the number of checking unnecessary connections can be reduced. For example, concerning

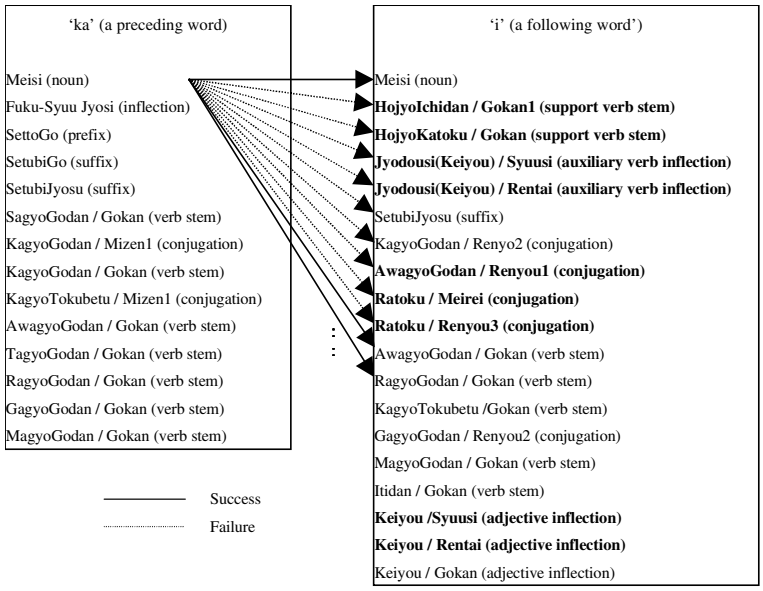
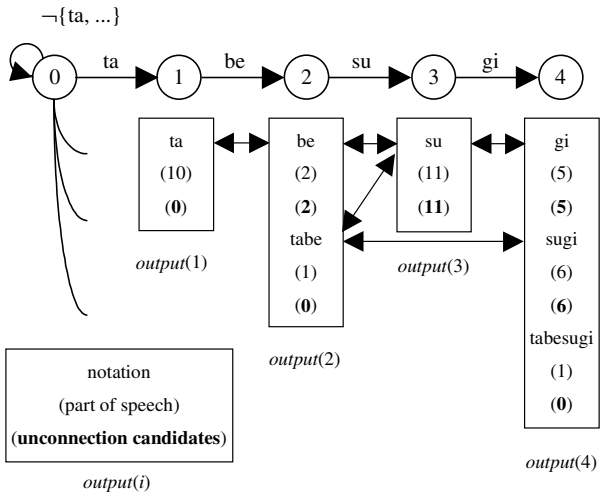


Figure 4: The connection checking between 'ka' and 'i'.

a pass "ta-be-su-gi" in Fig. 5, the relation between 'ta' and 'be', 'tabe' and 'su', and 'tabe' and 'sugi' can be checked in advance.

Consider the approaches using Directed Acyclic Word Graph (DAWG) or Finite State Automaton (FSA). In general, the DAWG and FSA have states with more than one outgoing transition after a state with two or more incoming transitions. Thus, there is no guarantee that there exists a subset of states in them with a one-to-one correspondence between the outputs and the states in that subset. Therefore, the DAWG and FSA cannot keep information of unconnection candidates correctly [AOE96]. But, by using the AC machine, information of unconnection candidates can be attached to the corresponding output state uniquely.



(a) The goto function and the output function.

Figure 5: The connection checking for the pass "tabesugi".

Algorithm 3 summarizes the method for checking all unconnection candidates in

one pass, and the following variables and functions are utilized:
Variable *queue* : stores the states;
Variable *candidate* : stores a set of unconnection candidates returned by a function CCheck();
Function CCheck(*state1*, *state2*) : checks the grammatical connectivity between the adjacent words in *output*(*state1*) and *output*(*state2*) and it returns a set of unconnection candidates;
Function Update(*state*, *candidate*) : updates the *output*(*state*) on the basis of the *candidate*;

Algorithm 3 : Algorithm for checking all unconnection candidates in one pass.
Input : A key $KEY = c_1c_2...c_n$ registered in the AC machine, where each c_i, for $1 \leq i \leq n$, is an input character and the AC machine.
Output : All unconnection candidates in one pass.
Method :

Step 3-1 : { Initialization }
$state \leftarrow 0$; $queue \leftarrow empty$; $i \leftarrow 1$;
Step 3-2 : { Storing states }
$state \leftarrow goto(state, c_i)$;
if $output(state) \neq empty$ **then** $queue \leftarrow queue \cup state$;
Step 3-3 : { Operation control }
$i \leftarrow i + 1$;
if $i \leq n$ **then goto** Step 3-2;
Step 3-4 : { Getting a state }
if $queue = empty$ **then** the process is terminated.
let *pre_state* be the next state in the *queue*;
$queue \leftarrow queue$ - $\{pre\_state\}$;
$tmp \leftarrow queue$;
Step 3-5 : { Connection checking }
if $tmp = empty$ **then goto** Step 3-6
let *next_state* be the next state in the *tmp*;
$tmp \leftarrow tmp$ - $\{next\_state\}$;
Un-Cand $\leftarrow$ ConnectionCheck(*pre_state*, *next_state*);
UpdateOutput(*next_state*, Un-Cand);
goto Step 3-5;
Step 3-6 : { Operation control }
goto Step 3-4;

Since all unconnection candidates in the dictionary can be checked by repeating the Algorithm 3 after the dictionary based on the AC machine is constructed, the number of checking unnecessary connections can be reduced by using the unconnection candidates.

In the example of the pass "ta-be-su-gi" in Fig. 5, when checking connections between the word starting with 'ta' and the following words, it involves 37(= 10(ta) × 2(be) + 1(tabe) × 11(su) + 1(tabe) × 6(sugi)) kinds of checking by using our grammatical table. On the other hand, if the unconnection candidates are available, it is not necessary to check the grammatical connectivity, because all parts of speech

of 'be', 'su' and 'sugi' are unconnection candidates, that is, it is 0(=10×0+1×0+1×0). Therefore, the most suitable word boundary for the "tabesugi" is the only last position of the word, that is, "tabesugi/".

As shown the above, in some cases, the word boundary can be detected by using the unconnection candidate without checking the grammatical connectivity during the execution of the morphological analysis.

4 Evaluation

4.1 Theoretical Evaluation

Let n be the length of key. The precise complexity of algorithms presented depends on the data structures, so theoretical analysis is first discussed under the following assumptions:
1) The time complexity of confirming one transition, that is, a goto function is $O(1)$.
2) The time complexity of a failure function is $O(1)$.
3) The time complexity of a output function is $O(1)$.

Consider the time complexity of dictionary lookup. As for a trie, it is clear that the time complexity of retrieving a key is $O(n)$ [AOE91, AOE96]. However, in morphological analysis, since it must detect all possible substrings in a given input sentence, the number of the dictionary access depends on the length of the input sentence. Therefore, the time complexity becomes $O(n^2+n)(=O((n+1)n/2)$. On the other hand, the time complexity for dictionary lookup of the proposed method is $O(n)(=O(2n\text{-}1))$ [AHO75].

Next, consider the time complexity of construction. Suppose that k is the total number of length of the keys. Concerning the trie, the time complexity is $O(k)$ because it is proportional to the total length of keys. The time complexity for construction of the AC machine is $O(k)$ [AHO75]. The cost of the algorithm 3 depends on the function ConnectionCheck and the length of key. Let p be the average number of parts of speech of preceding words and let f be the average number of parts of speech of following words. Then, since the time complexity of the ConnectionCheck is $O(pf)$, the time complexity of the algorithm 3 becomes $O(n+pf(n^2\text{-}n))$ in the worst case, because the cost of for-loop in algorithm 3 is $O(n)$ and the cost of while-loop is $O((pf)(n\text{-}1)n/2)$. From above observation, the time complexity for constructing the proposed method becomes $\mathrm{O}(s(l+pf(l^2\text{-}l)))$, where s is the total number of keys and l is the average length of keys.

Consider the effectiveness of the unconnection candidate. By using this candidate during the execution of the morphological analysis, the time complexity of the ConnectionCheck becomes $O(p(f\text{-}c))$, where c is the average number of the unconnection candidates which are stored in the output functions.

4.2 Experimental Evaluation

For experimental evaluation the following methods have been implemented on DELL OptiPlex XMT5120 (Pentium 120MHz) and they have been written in the C language.

(1) The dictionary lookup using a trie represented by a list structure.

(2) The proposed dictionary lookup method represented by a list structure.

In order to observe the effect of the proposed method, the following materials were used:

Dictionary : A 100,267 words dictionary of a Kana-to-Kanji translation system; The maximum length of keys is 19 and the average length of keys is 4.2. Note that Hiragana character in the Dictionary Source and the Input Text requires two bytes.

Inputs : News papers articles (2.7MByte); The total number of sentences is 39,339 and the average number of characters of the sentences is 72.6.

Table 1 shows the obtained simulation results. From these results, it turns out that the number of state transitions of the proposed method is 45.5% less than that of the trie method, and the proposed method is 49.9% faster (CPU time) than the trie one. However, the dictionary size of the proposed method is larger than that of a trie approach.

As for the candidates for checking connections, the total number of candidates is reduced from 5,637,007 to 4,014,625 by the unconnection candidates. This is 25.5% less than that of common morphological analysis. This means that the number of checking unnecessary connections can be reduced.

From the whole experimental observations, we can say that the proposed method is more practical than that of the trie. Although it requires more memory spaces, a fast morphological analysis can be achieved by using the proposed method.

Table 1: Simulation Results.

	Trie method	Our method
Dictionary Size (Mbyte)	8.3	32.5
State Transtions	14.444,170	8,016,514
Retrieval Time (sec.)	43.34	21.95

Conclusions

This paper has proposed a high speed morphological analysis by the AC machine. The proposed method is a simple and fast algorithm to find all possible substrings in an input sentence, and during a single scan, and it stores the connectivity relation of adjacent words into the output functions as the unconnection candidates. Therefore, if the unconnection candidates are available during the execution of the morphological analysis, the number of checking unnecessary connections can be reduced. Since these features depends on the passes, the proposed method have a good effect if there are a large number of long words in input text.

The efficiency of the proposed algorithm has been algorithm by the theoretical analysis, and the experimental evaluation was supported by the computer simulation with a 100,267 words dictionary. From the results, it turns out that the proposed method was 49.9% faster (CPU time) than the traditional trie approach. As for the number of candidates for checking connections, it was 28.8% less than that of the original morphological analysis by using the unconnection candidates.

As a future extension to this work, we are considering an implementation of morphological analysis system using the Multi-attribute AC machine [AND96] and the proposed method based on Double-Array Structure.

References

[ABE86] Abe, M. Ooshima, Y., Yuura, K., and Takeichi, N.: A Kana-Kanji Translation System for Non-Segmented Input Sentences Based on Syntactic and Semantic Analysis, Proceedings of the 10th International Conference on Computational Linguistics, 1986, pp.280-pp.285.

[AHO75] Aho, A.V., and Corasick, M.J.: Efficient String Matching : An Aid to Bibliographic Search, Communications of the ACM, Vol.18, No.6, 1975, pp.333-340.

[AKI94] Akiba, T., Tokunaga, T., and Tanaka, H.: An Extension of LangLAB for Japanese Morphological Analysis, Proceedings of the International Workshop on Sharable Natural Language Resources, 1994, pp.36-42.

[AND96] Ando, K., Shishibori, M., and Aoe, J.: An Efficient Multi-Attribute Pattern Matching Machine, Proceedings of the Prague Stringology Club Workshop'96, 1996, pp.1-14.

[AOE91] Aoe, J.: Computer Algorithms: Key Search Strategies, IEEE Computer Society Press, 1991.

[AOE96] Aoe, J., Morimoto, K., Shishibori, M., and Park, K.H.: A Trie Compaction Algorithm for a Large Set of Keys, IEEE Transactions of Knowledge and Date Engineering, Vol.8, No.3, June. 1996, pp.476-491.

[KAP84] Kaplan, S.J.: Designing a Portable Natural Language Database Query System, ACM Transactions on Database Systems, Vol.9, No.1, March.1984, pp.1-29.

[KUR94] Kurohashi, S., Nakamura, T., Matsumoto, Y., and Nagao, M.: Improvements of Japanese Morphological Analyzer JUMAN, Proceedings of the International Workshop on Sharable Natural Language Resources, 1994, pp.22-28.

[LEE97] Lee, G., Lee, J.H., B.-C., Kim, B.C., and Lee, Y.: A Viterbi-based morphological analysis for speech and natural language integration, Proceedings of the 17th International Conference on Computer Processing of Oriental Languages, Vol.1, 1997, pp.133-138.

[MAR94] Maruyama, H.: Backtracking-Free Dictionary Access Method for Japanese Morphological Analysis, Proceedings of the 15th International Conference on Computational Linguistics, 1994, pp.208-213.

[MOR96] Mori, S.: High Speed Morphological Analysis using DFA, Technical report of IEICE of Japan, NLC96-23, 1996, pp.17-23. (in Japanese)

[SAN94] Sano, H., Kawada, R., and Hasimoto, M.: Morphological Grammar Rules : An Implementation for JUMAN, Proceedings of the International Workshop on Sharable Natural Language Resources, 1994, pp.29-35.