MACH : A Supersonic Korean Morphological Analyzer

Kwangseob Shim
School of Computer Science and Engineering
Sungshin Women's University
Seoul 136-742, KOREA
shim@cs.sungshin.ac.kr

Jaehyung Yang
Department of Computer Engineering
Kangnam University
Kyunggido 449-702, KOREA
jhyang@kangnam.ac.kr

Abstract

This paper introduces a supersonic Korean morphological analyzer named MACH. It analyzes a 1 GB document within 5 minutes on a typical personal computer with a 1 GHz Pentium III CPU. This means that it analyzes about 500,000 words per second. In fact, the analysis speed of MACH is 12-20 times faster than the most famous Korean morphological analyzers. In spite of its supersonic speed, the analysis accuracy of MACH is not degraded at all, compared with any other Korean morphological analyzers.

1 Introduction

Researches on computational analysis for Korean started in the early 1980s. Morphological analyzers, one of the most essential parts in natural language processing systems, were considered easy to develop. A lot of different methodologies were proposed for Korean morphological analysis (S.S.Kang and Y.T.Kim 1994; D.B.Kim et. al. 1994; H.S. Park 1999; S.H.Yang and Y.S.Kim 2000).

At those times, the processing speed of a morphological analyzer was not regarded so important becasue other parts of a natural language processing system were very slow. The only concern was the accuracy of analysis, and that was the only measure for evaluating morphological analyzers.

It had been true until the advent of what is called an internet age. In the internet age, the processing speed is thought more important than the accuracy because a vast number of documents scattered over the internet are waiting for being indexed for information retrieval. Internet is not the only source of the need for a faster morphological analyzer. Besides information retrieval, there are many other applications that need morphological analysis as a minimal requirement. As the size of disks is doubled every year, so is the number of documents stored in them. This is another source of the requirement for a faster morphological analyzer.

Recently, some researchers have made efforts to implement a faster Korean morphological analyzer. Among them, Yang and Kim proposed a high-speed Korean morphological analysis method that was based on pre-analyzed partial words (S.H.Yang and Y.S.Kim 2000). They shifted the computational overhead such as segmentation of morphemes to an off-line pre-analysis dictionary building stage, in order to reduce the number of run-time dictionary lookups. As a result, they could attain the implementation of a high-speed morphological analyzer. They reported that their morphological analyzer processed a 1 GB document in 172 minutes on a personal computer with a 333 MHz Pentium II CPU.

In this paper, MACH, an acronym that stands for Morphological Analyzer for Contemporary Hangul, is introduced. As the name implies, MACH is a morphological analyzer that flies at a supersonic speed. For example, MACH analyzes a 1 GB document within 5 minutes on a personal computer with a 1.0 GHz Pentium III CPU. This means MACH is about 12 times faster than Yang and Kim's morphological analyzer, the latest state-of-the-art high-speed analyzer[1]. In fact, the processing speed of MACH is 12-20 times faster than that of the most famous Korean morphological analyzers. In spite of its supersonic speed, the accuracy of MACH is also comparable to that of other analyzers.

2 Problems with Previous Methods

2.1 Candidate Generation and Filtering

In Korean, whatever looks like a word is rather called an *eojeol*. Though a content word itself can be an *eojeol*, it is more often the case that an *eojeol* is made up of a content word and one or more function words such as *josa* (postposition) or *eomi* (ending)[2]. For a given *eojeol*, a morphological analyzer for Korean generates all possible combinations of a content word and one or more function words with an appropriate part-of-speech tag sequence.

The morphological analysis methods that have long been used are to generate morphologically possible analysis candidates from a given *eojeol*, and then cross out those that prove false in that there are no such words in Korean. Consider an *eojeol gal*. Morphologically, it can be realized in four different ways as shown in (1).

(1)	a. *ga*/stem + *l*/eomi	(addition)
	b. *gal*/stem + *l*/eomi	(deletion)
	c. *gad*/stem + *l*/eomi	(alternation)
	d. *gah*/stem + *l*/eomi	(alternation)

Stems in (1a) and (1b) are true, whereas those in (1c) and (1d) are false. The proof of falsehood for each candidate needs dictionary lookups, which is time-consuming. Kang proposed several heuristics that turned out somewhat effective in reducing the number of false candidates (S.S.Kang and Y.T.Kim 1994; S.S.Kang 1995). However, false candidates are still generated even with the heuristics.

Candidates are generated by applying morphological transformation rules such as addition, deletion and alternation. Since morphological transformation accompanies the breakdown of a syllable to a character level, the composite code system has been considered to be the most appropriate for morphological analysis for Korean[3]. The problem is that the complete code system is much more prevailing in real texts. This problem has been treated in such a simple way as introducing a code conversion module into a morphological analyzer. That is, input strings in complete code system are converted to strings in composite code system, and then morphological analysis is performed on them to produce internal analysis results. They are converted back to the original complete code system to get the external analysis results.

Although this approach seems to solve the problem anyhow, there is a bit of inefficiency in that the whole input strings are converted back and forth whereas only a small portion of input strings are subject to the morphological transformation. There had been no arguments for adopting a code conversion module until Yang and Kim suggested a method for bypassing code conversion by using pre-analyzed partial words (S.H.Yang and Y.S.Kim 2000).

2.2 Sequential Analysis

A morphological analyzer is supposed to generate all possible analyses for a given *eojeol*. For example, consider *yi·reul*[4]. A typical morphological analyzer for Korean would generate analyses as shown in (2)[5]. These analyses are generated one by one, in sequence. It would cost many CPU clocks in generating one analysis. There have been many efforts to reduce the number of CPU clocks consumed in morpho-

[1]For comparison, the execution time is divided by 3 on the assumption that the execution speed would be proportional to CPU's clock speed.

[2]For explanation, only simple *eojeol*s will be considered in this paper. The proposed method is also applicable to real *eojeol*s, which are formed in a lot more complex way than described here.

[3]In the complete code system, a unique 2-byte code is assigned to each syllable. In the composite code system, the 2-byte area is divided into three parts, each for the leading consonant, a vowel, and the optional trailing consonant, respectively.

[4]Dots are used to distinguish syllables in an *eojeol*.

[5]In each analysis, a string in italic type represents a Korean word, whereas a two-letter symbol in upper case represents a part-of-speech tag. NN, NP, NX, NU, VI, VT, AJ, JO and EM represent noun, pronoun, auxiliary noun, numeric noun, intransitive verb, transitive verb, adjective, *josa* (postposition) and *eomi* (ending), respectively.

logical analysis (S.S.Kang and Y.T.Kim 1994; S.S.Kang 1995).

(2) a. *yi*/NN + *reul*/JO
b. *yi*/NP + *reul*/JO
c. *yi*/NX + *reul*/JO
d. *yi*/NU + *reul*/JO
e. *yi·reu*/VI + *l*/EM
f. *yi·reu*/VT + *l*/EM
g. *yi·reu*/AJ + *l*/EM

What about the number of analyses? If the number of analyses could be reduced by half, the morphological analysis speed would be doubled. However, as mentioned above, since a morphological analyzer is supposed to generate all possible analyses, it is impossible to reduce the number of analyses.

Let's take a look at the example (2) again. A morpheme *yi* has four different part-of-speech tags. Although NN, NP, NX and NU are different in some point, they share common properties as a noun. They do not inflect and they usually accompany JO. In fact, they are ramified from a common category *cheon*. Another morpheme *yi·reu* has three different part-of-speech tags VI, VT and AJ that share common properties. They are also ramified from a common category *yongon*. Words of this category always accompany EM.

Considering these facts, we will merge several analyses into just one analysis as shown below.

(3) a. *yi*/{NN, NP, NX, NU} + *reul*/{JO}
b. *yi·reu*/{VI, VT, AJ} + *l*/{EM}

This shows that it is possible to do morphological analysis in quasi parallel rather than in sequence. The number of analyses in (3) is only one-third of those in (2). Analysis speed will be enhanced as much.

3 Proposed Method

3.1 Dictionary

We can avoid false candidate generation by listing all possible inflected forms in a dictionary. Assume that *gal*, an inflected form of both *ga* and *gal*, is listed in a dictionary as shown below.

(4) **gal** *ga*/{VI, VX} + *l*/{EM}
gal/{VT} + *l*/{EM}

With this kind of dictionary, only one dictionary lookup is sufficient to find all possible combinations of content and function words. Compare this with the candidate generation and filtering approach where four dictionary lookups are needed in addition to candidate generation process. Since there are several thousand different function words in Korean, however, it is not a good idea to list all possible combinations of content and function words in a dictionary.

It is not always possible to separate function words from preceding content words at syllable boundaries. Some function words start with a dangling consonant that cannot be a syllable by itself. In that case, the dangling consonant is melted into the last syllable of a preceding content word. Consider the following examples.

(5) a. *pi·da* $\Rightarrow$ *pi*/{VI, VT} + *da*/{EM}
b. *pin·da* $\Rightarrow$ *pi*/{VI, VT} + *n·da*/{EM}

In (5a), the separation is done at a syllable boundary. In (5b), the separation is not done at a syllable boundary because the given *eojeol pin·da* is realized by combining a content word *pi* with a function word *n·da* that has a dangling consonant *n*. With the complete code system, it has been considered impossible to take out a dangling consonant from the last syllable of a content word. For this reason, existing morphological analyzers adopted a code conversion module.

Fortunately, some specific consonants such as *n*, *l*, *m*, *b* and *ss*. Therefore, there are only 5 combinations for a Korean intransitive verb *ga*, as shown in (6).

(6) a. *gan* : N_CMBD
b. *gal* : L_CMBD
c. *gam* : M_CMBD
d. *gab* : B_CMBD
e. *gass* : SS_CMBD

We can list all combinations of content words and those consonants[6]. By doing so, we can eliminate code conversion modules.

Our dictionary is defined as a list of sextuples as shown below. Each sextuple will be called dinfo here after.

[6]These combinations are automatically generated through simple morphological processing.

(7) (<entry>, <base>, <tagset>, <form>, <tagset_condition>, <form_condition>)

<entry> represents an entry in our dictionary that is used as a search key. <entry> may be a base or a combination with a dangling consonant. <base> is the base form of <entry>. <tagset> stands for a set of part-of-speech tags that <base> assumes. <form> indicates the inflection form of <entry>, and takes either BASE to indicate that <entry> is a base form, or X_CMBD to indicate that <entry> is combined with a dangling consonant X where X is one of N, L, M, B and SS. A function word or a *cheon* that does not inflect in itself always takes BASE as the value of <form>. Finally, <tagset_condition> and <form_condition> specify part-of-speech tags and inflection form of a word, if any, that adjoins to the left of <entry>[7].

The following shows part of our dictionary for two *yongon*s *sa* and *sal*. Among 5 different combined forms, only two of them are shown below.

(8) a. (*sa*, *sa*, {VT}, BASE, {AD}, BASE)
b. (*san*, *sa*, {VT}, N_CMBD, {AD}, BASE)
(*san*, *sal*, {VI,VT,AJ}, N_CMBD, {AD}, BASE)
c. (*sal*, *sa*, {VT}, L_CMBD, {AD}, BASE)
(*sal*, *sal*, {VI,VT,AJ}, L_CMBD, {AD}, BASE)
(*sal*, *sal*, {VI, VT, AJ}, BASE, {AD}, BASE)

In this example, (8a) explains that a transitive verb (VT) *sa* is a base form and only an adverb (AD) in base form can adjoin to the left of *sa*. (8b) explains that *san* is not a base form, but a form realized by combining a base *sa* with a consonant *n* or by combining a base *sal* with the same consonant. In this case, only an adverb in base form can adjoin to the left of *san*. (8c) can be interpreted in a similar way.

The following is another part of our dictionary for some typical function words, that is, *eomi*s that combines with *yongon*s.

(9) a. (*deun*, *deun*, {EM}, BASE, {VI, VT, AJ, AX, VX}, BASE)
b. (*da*, *n·da*, {EM}, BASE, {VI, VT, AJ, AX, VX}, N_CMBD)
c. (*da*, *da*, {EM}, BASE, {VI, VT, AJ, AX, VX}, BASE)
d. (*gga*, *l·gga*, {EM}, BASE, {VI, VT, AJ, AX, VX}, L_CMBD)

(9a) explains that VI, VT, AJ, AX or VX in its base form can adjoin to the left of *deun*. (9b) explains, as N_CMBD indicates, that VI, VT, AJ, AX or VX combined with a consonant *n* can adjoin to the left of *da*. (9c) means that only VI, VT, AJ, AX or VX in its base form can adjoin to the left of *da*. Finally, (9d) shows that VI, VT, AJ, AX or VX combined with a consonant *l* can adjoin to the left of *gga*.

3.2 Analysis

This subsection describes our morphological analysis method. Given an *eojeol*, the problem of morphological analysis can be reduced to the problem of searching one or more paths that satisfy conditions assigned at each node of a search space. The conditions are fed from our dictionary (<tagset_condition> and <form_condition>).

Before giving an algorithmic description of our analysis method, several examples are given here. Let us start with an *eojeol sal·deun*. The <entry> in (9a) matches with *sal·deun*, and the <entry> in (8c) matches with *sal·deun*. As the condition fields in (9a) indicates, a *yongon* (such as VI, VT, AJ, AX or VX) in base form can adjoin to the left of *deun*. The third dinfo in (8c) satisfies this condition. Therefore, *sal·deun* is anlayzed as follows.

(10) *sal*/{VI, VT, AJ} + *deun*/{EM}

As anohter example, let us analyze an *eojeol san·da*. The <entry> in (9b) and (9c) matches with *san·da*, and then the <entry> in (8b) matches with *san·da*. (9b) indicates that a *yongon* combined with a consonant *n* can adjoin to the left of *da*. In (8b), both dinfos satisfy this requirement. (9c) indicates that only a *yongon* in base form can adjoin to the left of *da*. None of (8b) satisfies this condition. Therefore, *san·da* is analyzed as follows.

(11) *sa*/{VT} + *n·da*/{EM}
sal/{VI, VT, AJ} + *n·da*/{EM}

Finally, let us take an example with an *eojeol sal·gga*. In fact, this *eojeol* is analyzed in

[7] In this paper, right-to-left analysis is assumed. Left-to-right analysis is also possible in a similar way.

the same way as described above. The <entry> in (9d) matches with *sal·gga*, and then the <entry> in (8c) matches with *sal·gga*. As (9d) indicates, a *yongon* combined with a consonant *l* can adjoin to the left of *gga*. In (8c), the first two dinfos satisfy this condition, while the third dinfo does not because it is not an *l*-combined form. Therefore, *sal·gga* is analyzed as follows.

(12) *sa*/{VT} + *l·gga*/{EM}
sal/{VI, VT, AJ} + *l·gga*/{EM}

Note that only two dictionary lookups are sufficient to analyze an *eojeol* in the above examples. These are artificial and the situation may be a bit more complex with a full-fledged dictionary, and thus the number of dictionary lookups in our approach will increase accordingly. However, the number of dictionary lookups will be much smaller than that required in so-called candidate generation and filtering approach. Quantitive details is given in the next section.

No code conversion module is needed in our approach. Moreover, analysis is done in quasi parallel. Only two analysis results are generated in (12), whereas a total of 4 different analysis results should be generated with a conventional sequential analysis method. All these factors contribute to building a supersonic morphological analyzer.

The algorithmic description of the proposed method is given in Figure 1. analyze_word($s_a s_{a+1} \cdots s_b$, *rtagset*, *rform*) returns all possible morphological analyses for the given *eojeol* $s_a s_{a+1} \cdots s_b$. lookup_dictionary ($s_a s_{a+1} \cdots s_b$) returns a list of dinfos whose <entry> matches to $s_p s_{p+1} \cdots s_b$ where p is any integer between a and b. If p is not equal to a, the given *eojeol* is divided into $s_a s_{a+1} \cdots s_{p-1}$ and $s_p s_{p+1} \cdots s_b$, and the analysis goes on for $s_a s_{a+1} \cdots s_{p-1}$[8]. *rtagset* and *rform* specify the adjacency conditions to be satisfied by $s_p s_{p+1} \cdots s_b$. $r_{a(p-1)}$ indicates morphological analysis results for $s_a s_{a+1} \cdots s_{p-1}$. d^l_{pb} is one of the dinfos that lookup_dictionary($s_a s_{a+1} \cdots s_b$) returns. tagset(), form(), tagset_condition() and form_condition() are macros that return an appropriate dinfo field. In the algorithm, base(d^l_{pb})/*ctag* does not mean any division, but

[8]As mentioned earlier, a right-to-left analysis is assumed in this paper.

```
procedure analyze_word($s_a s_{a+1} \cdots s_b$, $rtagset$, $rform$)
begin
 $r \leftarrow \emptyset$
 $d_{pb} \leftarrow$ lookup_dictionary($s_a s_{a+1} \cdots s_b$)
 for $\forall\, d^l_{pb} \in d_{pb}$ do
   $ctag \leftarrow tagset(d^l_{bp}) \cap rtagset$
   if $ctag \neq \emptyset \wedge$ form($d^l_{pb}$) $\equiv rform$
      then do
      if $|s_a s_{a+1} \cdots s_{p-1}| > 0$ then do
         $r_{a(p-1)} \leftarrow$ analyze_word($s_a s_{a+1} \cdots s_{p-1}$,
                        tagset_condition($d^l_{pb}$),
                        form_condition($d^l_{pb}$))
         if $|r_{a(p-1)}| > 0$ then do
            $r \leftarrow r \cup r_{a(p-1)} \odot$ base($d^l_{pb}$)/$ctag$
         end_if
      else do
         $r \leftarrow r \cup$ base($d^l_{pb}$)/$ctag$
      end_if
   end_if
 end_for
 return $r$
end_procedure
```

Figure 1: Morphological Analysis Algorithm

means an analysis to be built that looks like, for example, *sal*/{VI, VT, AJ}. The operator $\odot$ represents the Cartesian product.

Now, let us follow the algorithm with examples. First, consider a Korean *eojeol* *ga·neun·de*. For the analysis of this, analyze_word(*ga·neun·de*, {AD, NN, NX, JO, EM}, BASE) is called. lookup_dictionary(*ga·neun·de*) returns the following dinfos.

(13) a. (*neun·de*, *neun·de*,
{EM}, BASE, {VI, VT, VX}, BASE)

b. (*de*, *n·de*, {EM}, BASE, {AJ, AX}, N_CMBD)

c. (*de*, *de*, {NX}, BASE, {NN}, BASE)

As the for loop in Figure 1 suggests, each dinfo is considered in sequence.

Firstly, (13a) is considered and the given *eojeol ga·neun·de* is divided into *ga* and *neun·de*. The tagset of (13a) is EM and its form is BASE, which conform to the given adjacency condition, that is, {AD, NN, NX, JO, EM } and BASE. From the adjancency condition fields in (13a), only VI, VT or VX in its base form can adjoin to the left of *neun·de*. Therefore, analyze_word(*ga*, {VI, VT, VX}, BASE) is called again. From the dictionary, the following dinfos are retrieved.

(14) a. (*ga*, *ga*, {JO}, BASE, {NN, NX}, BASE)
b. (*ga*, *ga*, {VI, VX}, BASE, {AD}, BASE)
c. (*ga*, *ga*, {NN}, BASE, {NN}, BASE)

Among them, only (14b) satisfies the new adjancency condition. So, analyze_word(*ga*) returns *ga*/{VI, VX}, and thus *ga*/{VI, VX} + *neun·de*/{EM} is stored in *r*.

Secondly, (13b) is considered in the same way. With this, the given *eojeol* is divided into *ga·neun* and *de*. The tagset of (13b) is EM and its form is BASE, which conform to the given adjacency condition, too. Only AJ or AX coupled with a dangling consonant *n* can adjoin to the left of *de*. Therefore, analyze_word(*ga·neun*, {AJ, AX}, N_CMBD) is called again. From the dictionary, only one dinfo is retrieved as shown below.

(15) (*ga·neun*, *ga·neul*, {AJ}, N_CMBD, {AD}, BASE)

This dinfo satisfies the new adjacency condition. So, analyze_word(*ga·neun*) returns *ga·neul*/{AJ}, and then *ga·neul*/{AJ} + *n·de*/{EM} is added to *r*.

Finally, (13c) is considered. With this, the given *eojeol* is divided into *ga·neun* and *de*. As the condition field of dinfo indicates, only NN can adjoin to the left of *de*. So, analyze_word(*ga·neun*, {NN}, BASE) is called again. As we have seen in (15), there is only one dinfo for *ga·neun*, and its tagset does not match with the adjacency condition, that is, NN . Therefore, the analysis fails and thus null is returned. Final analysis results for *ga·neun·de* are stored in *r*. They are shown below.

(16) *ga*/{VI, VX} + *neun·de*/{EM}
ga·neul/{AJ} + *n·de*/{EM}

As we have seen above, morphological analysis is done merely by calling the procedure analyze_word() recursively. In fact, the procedure is not called exhaustively. There are several conditions to determine whether the analysis should go further or not. These conditions are too specific to be described in this paper.

4 Evaluations

As described in the previous section, our approach eliminated the need of code conversion and candidate generation and filtering. Moreover, the analysis is done in quasi parallel. All

test set	size (MB)	size (*eojeol*s)	source
A	7.55	1,098,316	theses
B	9.74	1,172,216	newspapers
C	8.00	1,193,939	encyclopedia

Table 1: Test Sets

test set	elapsed time (sec)	speed (*eojeol*s/sec)	speed (sec/MB)
A	2.06	533,163	0.273
B	2.67	439,032	0.274
C	2.58	462,767	0.323
A+B+C	7.25	477,858	0.287

Table 2: Analysis Speed

these factors contribute to enhancing the processing speed of MACH. In this section, we will describe the performance of MACH.

For the performance evaluation, many text documents were collected from different sources. They were classified into three groups according to their sources. Table 1 shows the size and source of each test sets.

The evaluation was performed on a LINUX platform. As MACH is designed for a higher speed, the analysis speed should be the first thing to be evaluated. Table 2 shows the analysis speed of MACH, measured on a personal computer with a 1.0 GHz Pentium III CPU. As we know from Table 2, MACH spends only 4.9 minutes in analyzing a 1 GB text. This shows that MACH is about 12 times faster than Yang and Kim's analyzer, the latest state-of-the-art high-speed analyzer (S.H.Yang and Y.S.Kim 2000). In fact, the analysis speed of MACH is 12-20 times faster than that of the most famous Korean morphological analyzers.

As the volume of text to be indexed increases, there was a proposal to use a noun extraction system for faster indexing. Because noun extraction is much simpler than full morphological analysis, extraction is done much faster. The extraction speed was reported to be 43,000 *eojeol*s per second on a personal computer with a 450 MHz Pentium III CPU (D.G.Lee 2000). If the CPU's clock speed considered, MACH, a full-fledged morphological analyzer, is 5 times

test set	average # of dic lookups	average # of proc calls
A	2.42	2.94
B	2.66	3.08
C	2.53	3.10
A+B+C	2.54	3.04

Table 3: Dictionary Lookups & Procedure Calls

faster than the simple noun extraction system.

Table 3 shows an average number of dictionary lookups and the number of function calls for `analyze_word()`. Kang reported in his dissertation that the average number of dictionary lookups was 5.70 when syllable information was used to reduce the number of dictionary lookups (S.S.Kang 1993). Our figure was only 2.54 on the average as shown in Table 3.

The effect of quasi parallel analysis was evaluated too. Yang and Kim's morphological analyzer produced about 2.07 million analyses for the test set A, while MACH generated about 1.28 million analyses for the same test set. As mentioned in the previous section, the reduction in the number of analyses contributes to the implementation of faster analyzers.

Finally, the accuracy[9] of MACH was evaluated. For accuracy evaluation, 1,000 *eojeol*s were randomly selected from each test set. The accuracy averaged 98.8%. Kang reported the accuracy to be 99.4% (S.S.Kang 1993), while Yang and Kim reported 98.6% (S.H.Yang and Y.S.Kim 2000). Since no accuracy reports in the literature were based on the same data set, the comparison is not so meaningful. Nevertheless, we can say that the accuracy of MACH is comparable to that of other morphological analyzers for Korean. Since MACH has just developed, its accuracy is expected to be enhanced after fixing the probable errors in our dictionary.

5 Conclusions

In this paper, a supersonic Korean morphological analyzer called MACH was introduced. It analyzes around 500,000 *eojeol*s per second on a personal computer with a 1 GHz Pentium III CPU. That is, it analyzes a 1 MB document within 0.3 second and a 1 GB document within 5 minutes. In fact, the speed of MACH is 12-20 times faster than that of the most famous Korean morphological analyzers. The analysis speed of MACH seems to be close to that of an optiomal Korean morphological analyzer. In spite of the supersonic speed, the accuracy of MACH is not degraded at all, compared with other Korean morphological analyzers.

[9]Morphological analyzers for Korean are supposed to generate all possible analyses. Resolving ambiguities is often regarded as a tagging task. The term *accuracy* in this paper refers to the generation of all possible analyses.

References

Seung-Shik Kang, "Korean Morphological Analysis using Syllable Information and Multi-Word Unit Information," *PhD dissertation*, Seoul National University, 1993.

Seung-Shik Kang and Yung Taek Kim, "Syllable-based Model for the Korean Morphology," *The 15th International Conference on Computational Linguistics*, pp.221-226, 1994.

Seung-Shik Kang, "Morphological Analysis of Korean Irregular Verbs Using Syllable Characteristics," *Journal of the Korea Information Science Society: Software and Application*, Vol.22, No.10, pp.1480-1487, 1995.

Deok-Bong Kim et. al., "A Two-level Morphological Analysis of Korean," *The 15th International Conference on Computational Linguistics*, pp.535-539, 1994.

Do-Gil Lee et. al., "Korean Noun Extraction Using Exclusive Segmentation Information and Post-Noun Morpheme Sequences," *The 12th Conference on Hangul and Korean Information Processing*, pp.19-25, 2000.

Hyun S. Park, "Integrating Phrase Structure Grammar Rules with Spelling Rules for Morphological Analysis of Korean," *Proceedings of the 18th International Conference on Computer Processing of Oriental Languages*, pp.485-490, 1999.

Seung Hyun Yang and Young-Sum Kim, "A High-Speed Korean Morphological Analysis Method based on Pre-Analyzed Partial Words," *Journal of the Korea Information Science Society: Software and Application*, Vol.27, No.3, pp.290-301, 2000.