(제10회 한글 및 한국어 정보처리 학술대회)

# 한글 문장의 자동 띄어쓰기

강 승 식

한성대학교 정보전산학부
136-792 서울특별시 성북구 삼선동2가 389
kang@ham.hansung.ac.kr

# Automatic Word-Segmentation for Hangul Sentences

Kang, Seung-Shik
School of Information and Computer Engineering
Hansung University

**요약**

자동 띄어쓰기는 띄어쓰기가 무시된 한글 문서의 자동색인이나 문자인식에서 줄바꿈 문자에 대한 공백 삽입 문제 등을 해결하는데 필요하다. 이러한 문서에서 공백이 삽입될 위치를 찾아 주는 띄어쓰기 알고리즘으로 어절 블록에 대한 문장 분할 기법과 양방향 최장일치법을 이용한 어절 인식 방법을 제안한다. 문장 분할은 한글의 음절 특성을 이용하여 어절 경계가 비교적 명확한 어절 블록을 추출하는 것이며, 어절 블록에 나타난 각 어절들을 인식하는 방법으로는 형태소 분석기를 이용한다. 4,500여 어절로 구성된 두 가지 유형의 문장 집합에 대하여 제안한 방법의 띄어쓰기 정확도를 평가한 결과 '공백 재현율'이 97.3%, '어절 재현율'이 93.2%로 나타났다.

## 1. 서론

띄어쓰기를 하지 않는 언어의 분석은 문장에서 단어 경계를 인식하는 작업이 선행되어야 하기 때문에 중국어의 형태소 분석은 입력 문장으로부터 단어들을 분리하는 문제가 가장 중요시된다[1]. 일본어는 가다가나와 한자를 혼용하기 때문에 최장일치법(longest-segment method), 최소어절(least-*bunsetsu*) 인식법, 문자 유형에 따라 인식하는 방법 등을 이용하여 단어를 분리하고 있다[2].

한국어는 어절 단위로 띄어쓰기를 하기 때문에 문장 단위의 자동 띄어쓰기의 필요성이 제기되지 않았다. 다만, 복합명사의 경우에 띄어쓰기와 붙여쓰기가 모두 허용되기 때문에 정보검색이나 기계번역 등 일부 응용 분야에서 활용하기 위해 복합명사를 분해하는 알고리즘이 개발되었다[3,4,5].

그런데 최근에는 문자인식에 의해 대량의 정보자료를 입력할 때 줄바꿈 위치에서 공백을 삽입해야 하는지를 판단하는 문제가 발생하고 있다. 줄바꿈 위치의 띄어쓰기 문제는 전자출판 시스템에서 편집된 자료를 재편집하거나 문서편집기로 작성된 문서를 아스키 형태로 저장한 경우에도 필요하다.

특히, 일부 데이터베이스에서는 정보자료를 저장할 때 어떤 항목을 공백없이 저장한 경우가 있는데, 이러한 문서를 자동색인할 때 자동 띄어쓰기가 필수적이다. 이외에도 자동 띄어쓰기는 한글을 입력할 때 띄어쓰기 문제를 자동으로 처리하거나 철자 검사기에서 띄어쓰기 오류는 발견하는데 활용될 수 있다. 뿐만 아니라, 차세대 사용자 인터페이스로서 연속 어절 음성인식 기능이 상용화되면 그 필요성은 매우 증가할 것으로 예상된다.

이러한 필요성에 따라 한글 문장의 띄어쓰기 문제를 '공백 인식 접근법'과 '어절 인식 접근법'이라는 두 가지 관점에서 정의하고, 문장을 어절 블록으로 분할하여 어절 경계를 인식하는 띄어쓰기 알고리즘을 제안한다.

## 2. 기존의 연구 및 문제 정의

한글 문서의 자동 띄어쓰기는 이웃한 두 어절을 붙여쓴 경우에 자동으로 띄워주는 기능과 문장 전체 또는 여러 어절을 모두 붙여썼을 때 각 어절을 자동으로 분리하는 기능이 가능하다. 이웃한 두 어절 사이의 띄어쓰기는 현재 문서편집기에서 활용되고 있으나 자주 틀리는 고정된 유형에 대해서만 처리되는 제약이 있다.

문장 전체나 여러 어절에 대한 자동 띄어쓰기는 심광섭(1996)이 처음으로 시도하였으며, 말뭉치에서 획득한 이웃 음절 사이의 띄어쓰기-붙여쓰기 음절특성을 이용한 통계적 기법을 이용하고 있다[6]. 이 방법은 음절간 띄어쓰기 정보를 획득할 때 사용한 말뭉치의 문서 유형에 따라 유사 분야의 문장들에 대한 성능이 매우 우수하지만 문서 유형에 따라 정확도가 달라지는 특성이 있다.

한글 문장의 띄어쓰기는 음절과 음절 사이에 공백이 삽입되어야 하는지, 그렇지 않은지를 판단 기준으로 하는 '공백 삽입 접근법'과 어절과 어절 사이에 공백이 삽입되므로 문장을 구성하고 있는 어절들을 인식함으로써 어절 경계에 공백을 삽입하는 '어절 인식 접근법'이 가능하다. 띄어쓰기 알고리즘을 구현할 때는 각 접근법에 따라 처리 방법이 달라질 수 있다.

### 2.1 공백 삽입 접근법

$n$개의 음절로 이루어진 문장의 띄어쓰기 문제는 음절과 음절 사이에 공백을 삽입해야 하는지, 그렇지 않은지에 따라 어느 위치에 공백을 삽입할 것인가 하는 문제로 정의된다[1].

$$S_1S_2\cdots S_{i-1}S_iS_{i+1}\cdots S_{n-1}S_n$$

$n$개의 음절로 구성된 문장에서 공백이 삽입될 수 있는 위치는 모두 $(n-1)$개이며, 공백이 삽입된 개수에 따라 가능한 경우의 수는 아래와 같이 계산된다.

공백이 삽입되지 않은 경우 : ${}_{n-1}C_0$
1개의 공백이 삽입되는 경우 : ${}_{n-1}C_1$
2개의 공백이 삽입되는 경우 : ${}_{n-1}C_2$
...
$(n-2)$개의 공백이 삽입되는 경우 : ${}_{n-1}C_{n-2}$
$(n-1)$개의 공백이 삽입되는 경우 : ${}_{n-1}C_{n-1}$

즉, 공백이 삽입될 수 있는 모든 경우의 수는 식 (1)과 같이 $2^{n-1}$가지이다.

$$\sum_{i=0}^{n-1} {}_{n-1}C_i = 2^{n-1} \quad \text{--- 식 (1)}$$

이 모든 경우에 대해 띄어쓰기가 옳은지를 검사하는 문제는 지수함수 시간이 걸리게 된다. 따라서 보다 효율적인 방법으로 공백 삽입이 가능한 $(n-1)$개의 위치에 공백을 삽입할 것인지를 결정하는 방법이 사용된다. 심광섭(1996)은 말뭉치에 나타난 *bigram* 음절 특성을 이용하여 말뭉치에서 추출된 임의의 두 음절 사이에 공백이 삽입될 빈도수와 공백 삽입 확률을 구하고 임계치를 초과한 두 음절 사이에만 공백을 삽입하는 통계적 기법을 취하고 있다.

1) 실제 문장에서는 문장부호와 숫자, 영문자 등 한글 이외의 문자가 포함되기도 한다.

그런데 통계적 기법만을 이용한 띄어쓰기 알고리즘은 정확도가 높지 않으므로 공백 삽입 확률과 함께 형태소 분석기를 이용하여 인식된 어절을 확인하는 방법을 취한다.

### 2.2 어절 인식 접근법

어절 인식을 바탕으로 한 자동 띄어쓰기는 문장내에 출현한 어절을 인식하여 어절과 어절 사이에 공백을 삽입하는 인식론적인 방법이다. 문장을 구성하고 있는 어절들이 인식되면 어절 경계에 공백을 삽입하는데, 어떤 문장이 세 어절로 구성되어 있을 때 두 번째 어절만 인식되더라도 어절의 양쪽 경계에 공백을 삽입할 수 있다. 이 방법에서도 $n$음절 입력 문장에 대해

1개의 어절로 이루어진 경우
2개의 어절로 이루어진 경우
...
$(n-1)$개의 어절로 이루어진 경우
$n$개의 어절로 이루어진 경우

가 가능하며, 모든 경우의 수는 식(1)과 동일한 $2^{n-1}$가지이다.

어절 인식법으로 형태소 분석기를 이용할 때 형태소 분석기 호출 횟수는 1음절어에 대해 $n$번, 2음절어 $(n-1)$번, ..., $n$음절어 1번이므로 형태소 분석기 호출 연산에 대한 알고리즘의 복잡도는 식(2)와 같이 $O(n^2)$이다[2].

$$n+(n-1)+\ldots+2+1 = \frac{n(n+1)}{2} \quad \text{--- 식 (2)}$$

그런데 입력 문장에 나타난 모든 부분문자열(substring)에 대해 옳은 어절인지를 판단하는 것은 비효율적이므로 문장의 한쪽 끝에서부터 순서대로 이웃한 어절을 인식하는 순차적 알고리즘을 사용한다. 이 알고리즘의 복잡도는 퇴각검색(backtracking)을 고려했을 때 $m \cdot O(n)$이다[3]. 어절을 인식하는 방법으로 맞틀오류나 틀맞오류가 전혀 없는 완벽한 형태소 분석기를 사용하더라도

"나는 학생이다", "나는 학생 이다"

와 같이 중의성 문제로 인해 정확도가 낮아지게 된다. 따라서 정확도 향상을 위하여 최장일치법이나 어휘지식을 이용한 중의성 해결 기법이 사용된다.

2) 입력 문장에서 가능한 모든 어절들로부터 문장을 구성하는 어절열을 구하는 방법으로 동적 프로그래밍 기법을 이용하면 알고리즘의 복잡도는 $O(n^3)$이다. 그러나 띄어쓰기 알고리즘에서 비교 연산은 형태소 분석기 호출 연산에 비해 복잡도에 미치는 영향이 매우 적으므로 무시할 수 있다.

3) 각 음절 위치에서 평균 2회의 어절 인식(형태소 분석)이 일어났을 때 $m$=2이다 .

## 3. 띄어쓰기 알고리즘

어절 인식 접근법에 의한 띄어쓰기 알고리즘으로는 순방향, 역방향, 양방향 알고리즘이 가능하다.

### 3.1 순방향-역방향 알고리즘

어절 인식을 위한 순방향(forward) 알고리즘은 입력 문장을 전진 방향(좌에서 우로)으로 진행하면서 형태소 분석기를 이용하여 순차적으로 어절을 인식하는 방법이고, 역방향(backward) 알고리즘은 진행 방향으로 반대로 하여 문장끝에서부터 후진 방향으로 어절을 인식해 나가는 방법이다.

$n$개의 음절로 이루어진 문장 $S_1S_2\cdots S_{i-1}S_iS_{i+1}\cdots S_{n-1}S_n$에서 음절 $S_i$부터 시작되는 어절의 인식은 1음절어 $S_i$, 2음절어 $S_iS_{i+1}$, 3음절어 $S_iS_{i+1}S_{i+2}$ 등 두 가지 이상의 중의성이 발생하는 경우가 발생한다. 이러한 중의성 문제는 길이가 긴 어절을 우선으로 취하는 최장일치법을 적용하고 다음 어절을 인식할 때 문제가 발생했을 때 순서대로 짧은 어절을 취하는 퇴각검색 기법을 이용하여 해결이 가능하다.

그런데 어떤 경우에는 어절 인식 중의성으로 인해 앞어절 인식 오류가 다음 어절을 인식하는데 영향을 주게 되고 다음 어절들이 계속해서 오인식되는 전파오류(triggered errors)가 발생하기도 한다. 대부분의 일반적인 문장에서는 조사나 어미가 오류가 전파되는 것을 차단해 주기 때문에 퇴각검색에 의해 전파오류가 자동적으로 해결되기도 한다.

그러나 드물긴 하지만 아래 예와 같이 '기초지방자치단체' 등 일부 복합명사에서 전파 오류가 심각한 문제를 야기하는 경우도 있다.

기초지 방자 치단 체 행 정전 산화[4]

특히, 어절 인식기가 1음절어를 옳은 어절로 인식하는 것을 허용했을 때 전파오류 문제는 매우 심각하게 발생한다. 국어사전 표제어에 1,700여개의 음절이 사용되고, 사용빈도가 매우 낮은 음절들을 제외할 때 한글 문서에서 주로 사용되는 음절수는 1,300여개로 이중에서 1음절 어휘형태소 개수가 1,000여개나 된다[7]. 즉, 어절인식기(형태소 분석기)는 대부분의 1음절어를 옳은 어절로 인식하게 된다.

따라서 일단 최장 첫 어절이 인식되고 나면 퇴각검색에 의해 옳은 어절을 인식할 가능성은 매우 낮다. 왜냐하면, 다음 어절로 2음절어 이상이 인식되지 않더라도 1음절어를 인식한 후에 다음으로 진행하기 때문이다.

이와 같이 1음절어 인식을 허용하면 대부분의 문장에서 전파오류가 빈번하게 발생할 뿐만 아니라 미등록어를 1음절어 혹은 2음절어로 분해하게 되어 정확도를 저하시키는 요인이 된다. 이러한 문제점을 예방하려면 어절인식기로 하여금 1음절어, 특히 1음절 명사를 인식하지 못하도록 해야 한다. 그러나 '이/그/저', '한/두/세/네', '때/등/중'과 같은 관형사와 의존명사, 1음절 용언 등을 인식하지 못하게 되므로 여전히 문제가 남아 있다.

### 3.2 양방향 최장일치법

양방향(bi-directional) 최장일치법은 순방향 혹은 역방향 알고리즘에서 문제점으로 지적되는 전파 오류와 미등록어로 인한 어절 인식 오류, 1음절어 인식 오류 문제를 최소화하기 위하여 문장의 양쪽끝에서 동시에 진행하는 방법이다[8]. 즉, 순방향 알고리즘과 역방향 알고리즘을 조합하여 문장의 첫부분 혹은 끝부분부터 어절을 인식하다가 더 이상 어절이 인식되지 않으면 다른쪽 끝에서부터 반대쪽으로 어절을 인식한다.

이 방법은 양쪽끝에서 어절 인식이 동시에 진행되므로 전파 오류로 인한 정확도 감소를 줄일 수 있으며, 미등록어의 어절 경계를 인식하기가 용이하다. 또한, 어절을 인식할 때 양쪽 끝에서 최장일치 어절을 우선적으로 취함으로써 어절 인식 오류를 감소시킬 수 있다. 그러나 이 알고리즘에서도 전파 오류가 발생할 수 있으며, 미등록어가 두 개 이상 포함된 문장을 처리하기가 어렵다.

이러한 전파 오류와 미등록어로 인한 오류는 문장의 길이가 긴 경우에 더 자주 발생하기 때문에 띄어쓰기의 단위를 문장 전체가 아니라 어절 블록으로 분할하여 처리함으로써 오류 발생 가능성을 줄일 수 있다.

### 3.3 어절 블록 양방향 알고리즘

일반적으로 한 문장은 10개 이상의 어절들로 구성되기 때문에 어절 인식 중의성과 전파 오류에 의해 오인식된 어절이 다수 발견되어 정확도가 낮아진다. 그러나 양방향 최장일치법을 사용하더라도 어절수 3~4개 정도의 짧은 문장에 대한 띄어쓰기는 어절 인식 중의성이나 전파 오류의 발생 확률이 높지 않기 때문에 비교적 정확하게 어절을 인식하는 것이 가능하다.

따라서 문장의 길이를 어절수 3~4개 정도의 어절 블록으로 분할한 후에 양방향 최장일치법을 적용하면 '어절 블록 양방향 알고리즘'이 효율적이다. 이 알고리즘은 아래 예와 같이 어절을 인식할 때 발생하는 어절 인식 오류가 전파되지 않도록 어절 블록을 설정함으로써 전파 오류를 차단하고, 블록내에서는 양방향 최장일치 알고리즘을 사용한다.

4) '기초지'는 '기초'명사+'이'서술격조사+'지'어미로 분석되고, '전산화'명사는 사전에 수록되지 않은 단어이다.

$[s_1s_2\cdots s_{i-1}s_i][s_{i+1}\cdots s_{j-1}s_j][s_{j+1}\cdots s_{n-1}s_n]$

한 문장을 몇 개의 어절 블록으로 분할할 때 블록과 블록 사이의 경계는 어절 경계와 일치하는 것이 바람직하다. 그러나 특정 어절이 두 블록으로 분할되지 않도록 완벽하게 블록 경계를 설정하기는 쉽지 않다. 따라서 가급적 블록 경계가 어절 경계와 일치하도록 문장을 분할하여 어절 블록에 대한 띄어쓰기를 수행하고, 띄어쓰기 결과를 이용하여 블록 경계에 걸쳐 있는 어절이라고 판단된 것은 수정해 주는 방법을 취한다.

```
Algorithm word_spacing(char sent[])
{
    int i=0; /* 어절블록 시작 인덱스 */
    int j;   /* 어절블록 끝 인덱스 */
    int k, n=strlen(sent);
    int sp_index[SENTSIZE];

    while (j < n) {
        j = wblock_end_index(sent, i, n);
        k = set_spaces(sent, i, j, sp_index);
        i = wblock_begin_index(j, k);
    }
    adjust_word_spaces(sent, sp_index);
}
```

그림 1. 어절 블록 양방향 알고리즘

어절 블록 양방향 알고리즘은 그림 1과 같다. 입력 문장 sent에 대해 어절 블록의 시작 인덱스 $i$의 초기값을 0으로 하고 wblock_end_index에 의해 어절 블록의 끝 인덱스 $j$를 구한다. 어절 블록 sent[$i$]~sent[$j-1$]에 대해 양방향 최장일치법(함수 set_spaces)을 이용하여 어절들을 인식하고 공백이 삽입될 위치를 배열 sp_index[]에 0 또는 1로 표시한다[5].
어절 블록의 끝 어절이 형태소 분석에 실패한 불완전 어절이면 함수 set_spaces는 끝 어절의 시작 위치를 반환하여 다음 어절 블록의 시작 인덱스로 한다. 어절 블록의 끝부분이 어절로 인식된 경우에도 어절 블록의 끝 인덱스가 어절 경계가 일치하지 않아서 한 어절이 두 블록에 걸치는 경우도 있다. 이 경우는 함수 adjust_word_spaces에서 처리한다.

## 4. 어절 블록 및 어절 인식

심광섭(1996)은 *bigram* 음절 특성을 이용하여 어절 블록이 아니라 어절 경계를 인식하는 방법을 사용하고 있다. 이 *bigram* 음절 특성을 이용하고 임계치를 조절함으로써 어절 블록의 경계를 인식하는 방법이 가능하다. 그러나 *bigram* 음절 특성은 기억 공간을 많이 차지하는 단점이 있기 때문에 이와 비슷한 성능으로 어절 블록을 인식할 수 있는 조사/어미 음절 특성을 이용하는 방법을 사용하는 것이 효율적이다[6].
예를 들어, '는'과 '를'은 거의 체언 어절의 끝에 사용되므로 어절 경계일 가능성이 매우 높다. 특히, 많은 어절이 조사나 어미를 동반하고 있으며, '가/고/지' 등을 제외한 조사/어미 음절은 어휘형태소의 첫부분으로 사용될 가능성이 매우 낮다. 이처럼 조사/어미의 음절 특성에 의해 비교적 어절의 끝일 가능성이 높은 곳을 추정함으로써 어절 블록의 경계를 인식할 수 있다.

### 4.1 어절 블록 인식

조사로 사용되는 음절수는 70여개, 어미로 사용되는 음절수가 130여개로 자주 사용되는 음절수 1,300여개에 비해 그 수가 매우 적으므로 문장내에 나타난 조사/어미의 음절 특성을 이용하여 어절 블록의 경계를 인식한다[7]. 먼저, 문장내 각 음절이 조사/어미의 첫음절로 사용되는지에 따라 음절 특성을 부여하는데, 조사/어미의 음절 길이에 따라 $i$-음절 조사의 첫음절은 $j_i$, $i$-음절 어미의 첫음절은 $e_i$로 구분한다[7].

구문분석과$j_1j_2$의$j_1$미분석이$j_1j_3$라$j_1e_1e_2$는$j_1e_1$어$e_1$려$e_1$운문제

이와같이 조사/어미의 첫음절 특성에 의하여 조사/어미의 시작 및 끝 위치를 알 수 있으며 이를 기준으로 어절 블록의 경계를 추정할 수 있다. 이 때 '과의', '이라는어려' 등에서 알 수 있듯이 조사/어미 음절이 2개 이상 연속해서 나타났을 때 어절 블록이 잘못 설정되어 한 어절이 두 어절 블록으로 분할되는 경우가 발생할 수 있다.
이러한 블록 경계 오류를 최소화하기 위하여 빈도가 높은 조사만으로 조사/어미 음절 특성을 가변화시켜

- 조사/어미 첫음절 특성을 그대로 사용
- 빈도가 낮은 조사/어미를 제외
- 고빈도 조사의 첫음절 특성만 사용

에 대한 실험 결과, '빈도가 높은 조사 음절 특

5) sp_index[$i$]가 1이면 문자 sent[$i$] 앞에 공백이 삽입되고, 0이면 공백이 삽입되지 않는다. 또한, sent[$i$] 앞에 공백이 삽입될 가능성이 높지만 확실치 않은 경우에는 1이외의 다른 값을 부여하여 오류를 수정하는 목적으로 사용할 수도 있다.

6) *bigram* 음절 정보의 개수가 7만여개이고 4가지 음절 특성 정보를 수록하면 약 1M bytes의 기억 공간을 차지한다.

7) $j_3$은 3음절 조사의 첫음절로 사용되는 음절을 의미한다. 어미 'ㄴ/ㄹ/ㅁ/ㅂ'은 정보를 부가하지 않으며, '은/는', '이/가' 등이 체언과 결합 제약을 위반하면 조사/어미 특성을 부여하지 않는다.

성'만으로 어절 블록을 구분했을 때 가장 효과가 좋게 나타났다. 고빈도 조사의 첫음절 특성을 이용하여 어절 블록의 끝 위치를 찾는 알고리즘은 다음과 같다.

```
int wblock_end_index(sent, i, n)
char sent[]; int i, n;
{
  int j;
  for (j = i+4; i < n; i += 2)
    if (high_freq_josa(sent[j], sent[j+1))
      return j + n_syl_josa(sent, j);
  return n;
}
```

그림2. 어절 블록의 끝 인덱스 알고리즘

이 알고리즘[8]에서 인자 *i*는 어절 블록의 시작 위치이고 *n*은 입력 문장의 끝 위치이다. 어절 블록의 시작부 처음 두 개의 음절은 고빈도 조사라 하더라도 그 이후에 있는 고빈도 조사를 블록의 끝 위치로 한다[9]. 그 이유는 '휴가를갔다'에서 '휴가'와 '를갔다'로 분할했을 때처럼 한 어절이 두 블록으로 쪼개지는 경우가 자주 발생하기 때문이다.

### 4.2 어절 블록내의 어절 인식

어절 블록내에서 어절들을 인식하는 방법은 형태소 분석을 이용한 양방향 최장일치법을 이용한다. 먼저, 조사 부분을 제외한 어절 블록의 음절수가 4음절 이하인 경우는 음절수에 따라 처리한다. 어절 블록 전체가 형태소 분석 성공이면 하나의 어절로 간주하고, 3음절이면 2+1 또는 1+2, 4음절인 경우는 2+2, 1+3, 3+1로 분할하여 검사한다(그림 3).

조사 부분을 제외하고 5음절 이상인 어절 블록은 역방향 분석을 먼저 시도한다. 역방향 분석은 5/4/3/2 음절에 대해 순서대로 형태소 분석을 시도하여 최장일치 어절을 발견해 나간다. 국어사전의 어휘수나 실제 문서에 출현한 어휘형태소의 길이를 살펴보면 2음절과 3음절이 가장 많다. 또한, 4음절 이상 어휘형태소는 주로 2음절 혹은 3음절 어휘형태소로 구성된 복합어인 경우가 대부분이므로 2음절과 3음절 어휘형태소의 인식이 주가 된다[10].

형태소 분석 성공 어절이 더 이상 발견되지 않으면, 순방향 분석을 같은 방법으로 시도한다. 어절 인식이 끝난 후에도 인식되지 않은 어절은 미등록어로 간주한다.

```
int set_spaces(sent, i, j, sp_index)
char sent[];
int i, j;  /* 어절블록 시작-끝 인덱스 */
char sp_index[]; /* 공백 표시 */
{
  if (is_word(sent, i, j)) return 0;

  if (j-i == 3음절)
    /* 2+1, 1+2 유형 인식 */
  else if (j-i == 4음절)
    /* 2+2, 1+3, 3+1 유형 인식 */
  else
    /* 양방향 최장일치 어절 인식 */

  if (끝어절 == 불완전어절)
    return 끝어절시작인덱스;
}
```

그림 3. 어절 인식 및 공백 표시 알고리즘

### 4.3 어절 인식 오류 교정

어절 인식 오류에는 블록 경계 오류와 블록내에서 어절 인식 오류 등이 있다. 블록 경계 오류 중에서 경계 어절(어절 블록의 마지막 어절)에 대한 형태소 분석이 실패한 것은 다음 어절 블록에 결합하여 어절 인식을 시도함으로써 해결이 가능하다. 그러나 경계 어절에 대한 형태소 분석이 성공한 것은 옳은 어절로 간주되므로 다음 어절 블록에 대한 처리가 끝난 후에 전 블록의 마지막 어절과 후 블록의 처음 블록에 대해 재분석을 시도한다.

어절 블록내에서 어절 인식 오류는 주로 1음절어로 인하여 발생한다. 순방향 분석과 역방향 분석이 끝난 후 1음절이 남으면 1음절어가 인식되지만 그렇지 않은 경우도 발생한다. 인식되지 못한 1음절어 중에서 비교적 빈도가 높은 '이/그/저/한/두/세/네'와 '-을수있다'의 '수'는 독립 어절로 간주하는 것이 좋으나 기타 1음절어들을 인식하는 문제는 더 많은 연구가 요구된다.

어절 인식 과정에서 발생하는 오류 중의 하나는 최장일치 어절을 우선으로 하기 때문에 발생한다. 예를 들어, '수정하여만들었다'를 '수정하여만 들었다'로 분해하는 경우이다. 이러한 유형의 오류 중 많은 경우에 이웃한 두 어절을 비교하여 교정이 가능하다. 즉, 두 어절이 모두 옳은 어절로 인식되었다고 하더라도 1음절씩 좌우로 이동하여 분석했을 때 어휘형태소의 길이가 1인 것보다는 2인 것을 우선으로 한다.

이와 더불어 숫자나 영문자 뒤에 오는 조사/어미는 별도로 인식하여야 하며, 쉼표와 괄호 등 문장부호 또한 별도의 처리 과정이 필요하다.

---

8) 실제로 구현할 때는 고빈도 조사가 2개 연속된 경우와 한 음절 건너에 고빈도 조사가 있는 경우는 2번째 조사를 블록의 끝으로 간주하였다.

9) 어절 블록의 큰 경우에는 전파 오류가 발생할 가능성이 높으므로 조사 부분을 제외한 어절 블록의 크기는 10 음절 정도로 제한하는 것이 좋다.

10) 어휘형태소가 1음절인 것보다는 2음절 이상인 것을 우선으로 취한다.

## 5. 실험 및 비교 평가

본 논문에서 제안한 알고리즘을 구현하여 두 가지 유형의 문장 집합에 대해 실험하였다. 문장 집합 1은 여러 가지 유형의 문서에서 임의로 추출한 문장들이고, 문장 집합 2는 본 논문에서 수식과 표 등 한글 이외의 문자들로 구성된 문장들을 제외한 것이다. 두 실험 데이터의 크기는 표 1과 같다.

표 1. 실험 데이터의 크기

| | line수 | 어절수 | 공백위치수 |
|---|---|---|---|
| 문장집합1 | 150 | 1,882 | 5,790 |
| 문장집합2 | 285 | 2,599 | 7,928 |

표 1에서 line수는 실험문장들의 줄수로 긴 문장은 2줄 이상으로 분할된 것도 있다. 공백위치수는 공백이 삽입될 수 있는 위치의 수로 $n$ 음절로 구성된 문장의 경우에 공백위치수는 $n$-1개이다.

띄어쓰기 실험 결과를 두 가지 측정 기준에 따라 재현율을 계산하였다. '공백 재현율'은 공백이 삽입될 수 있는 모든 음절(또는 아스키 문자) 위치(공백 위치수)에 대하여 띄어써야 할 위치에 공백을 삽입한 것과 붙여써야 할 위치에 공백을 삽입하지 않은 것을 백분율로 계산한 것이다. '어절 재현율'은 문장 집합에 나타난 모든 어절수에 대하여 띄붙오류와 붙띄오류를 제외한 어절수를 계산하였다[11].

표 1의 두 가지 문장집합에 대한 띄어쓰기 실험 결과는 표 2와 같다. '공백 재현율'과 '어절 재현율'은 비교 대상이 되는 모범 답안에 따라 오류의 수가 2배 이상 차이가 날 수 있다. 예를 들어 문장집합2에서 '정보전산학부', '자동색인', '문자인식'의 띄어쓰기 결과는 각각 '정보 전산학부', '자동 색인', '문자 인식'이다. 이와 같이 띄어쓰기와 붙여쓰기가 모두 허용되는 것은 옳은 것으로 간주하였다.

표 2. 띄어쓰기 실험 결과

| | 공백 재현율 | 어절 재현율 |
|---|---|---|
| 문장집합1 | 97.2% | 92.9% |
| 문장집합2 | 97.4% | 93.5% |
| 평 균 | 97.3% | 93.2% |

심광섭(1996)의 통계적 띄어쓰기 알고리즘의 '공백 재현율'이 96.4%이고, '어절 재현율'이 87.2%인데 비해 본 논문에서 제안한 알고리즘의 정확도가 더 높게 나타났다.

실행 속도는 linux 2.0(pentium 120MHz)에서 약 300어절/초였다.

11) 심광섭(1996)은 '공백 재현율'을 '음절 단위 정확도', '어절 재현율'을 '어절 단위 정확도'라 하였다.

## 6. 결론

본 논문에서는 띄어쓰기 문제를 해결하기 위하여 '어절 인식 접근법'을 취했으며, 알고리즘의 복잡도를 줄이고 오류를 최소화하기 위하여 어절 블록을 추출하는 문장 분할 기법을 도입하였다. 또한, 어절을 인식할 때 발생하는 오류를 줄이기 위하여 어절 블록내에서 양방향 최장일치법에 의해 어절을 인식하는 방법을 이용하였다. 어절 블록내 어절 인식결과로 어절 블록의 경계가 잘못 설정된 것은 후처리에서 오류를 교정하여 정확도를 높였다.

본 논문에서 제안한 방법의 띄어쓰기 정확도를 평가한 결과 '공백 재현율'이 97.3%, '어절 재현율'이 93.2%였다. 이 방법은 문서나 문장 유형에 무관하게 적용되기 때문에 유형에 따른 정확도 편차가 적을 뿐만 아니라 어절 블록 인식을 위해 사용되는 조사/어미의 음절 특성은 형태소 분석시에 사용되는 정보를 그대로 활용하므로 기억 공간의 낭비가 적고 실행 효율이 뛰어나다.

## 참고문헌


[1] Chen K. J. and Liu S. H., Word Identification for Mandarin Chinese Sentences, *Proceedings of the 14th International Conference on Computational Linguistics,* pp.101-107, 1992

[2] Nobesawa S., et. el, Segmenting a Sentence into Morphemes using Statistic Information between Words, *Proceedings of the 15th International Conference on Computational Linguistics,* pp.227-233, 1994

[3] 윤보현, 조민정, 임해창, 통계정보와 선호 규칙을 이용한 한국어 복합명사의 분해, 정보과학회 논문지(B), 24권 8호, pp.900-909, 1997

[4] 심광섭, 합성된 상호정보를 이용한 복합명사 분리, 정보과학회 논문지(B), 24권 11호, pp.1307-1317, 1997

[5] 강승식, 한국어 복합명사 분해 알고리즘, 정보과학회 논문지(B), 25권 1호, pp.172-182, 1998

[6] 심광섭, 음절간 상호정보를 이용한 한국어 자동 띄어쓰기, 정보과학회 논문지(B), 23권 9호, pp.991-1000, 1996

[7] 강승식, 음절정보와 복수어 단위정보를 이용한 한국어 형태소 분석, 서울대학교 컴퓨터공학과 박사학위 논문, pp.56-72, 1993

[8] 최재혁, 이상조, 양방향 최장일치법에 의한 한국어 형태소 분석에서의 사전 검색 횟수 감소 방안, 정보과학회 논문지, 20권 10호, pp.1497-1507, 1993