
Machine Translation **16:** 151–174, 2001.

Learning-based Intrasentence Segmentation for Efficient Translation of Long Sentences

SUNG-DONG KIM, BYOUNG-TAK ZHANG and YUNG TAEK KIM
Department of Computer Engineering, Hansung University, Samsun-dong Sungbuk-gu, Seoul, Korea
E-mail: {sdkim@hansung.ac.kr;btzhang,ytkim}@comp.snu.ac.kr

Abstract. Long-sentence analysis has been a critical problem in machine translation because of its high complexity. Intrasentence segmentation has been proposed as a method for reducing parsing complexity. This paper presents a two-step segmentation method: (1) identifying potential segmentation positions in a sentence and (2) selecting an actual segmentation position amongst them. We have attempted to apply machine learning techniques to the segmentation task: "concept learning" and "genetic learning". By learning the "SegmentablePosition" concept, the rules for identifying potential segmentation positions are postulated. The selection of the actual segmentation position is based on a function whose parameters are determined by genetic learning. Experimental results are presented which illustrate the effectiveness of our approach to long-sentence parsing for MT. The results also show improved segmentation performance in comparison to other existing methods.

Key words: concept learning, genetic algorithm, intrasentence segmentation, parsing, segmentation-appropriateness function, version space

1. Introduction

With the popularization of World Wide Web, the need for machine translation (MT) systems is growing rapidly. Parsers for practical MT systems should be able to analyze various types of sentences, and a large number of rules are required to parse such sentences. The computational complexity of parsing with such complex grammars becomes more difficult as the length of the sentence increases. For example, for a longer sentence, the possibility that the sentence has an exact match in the translation archive is less. Example-based MT systems will also become less flexible (Cranias et al., 1994). In idiom-based MT (Lee, 1993; Yoon, 1994), long-sentence parsing is also difficult, because, as the length of the sentence increases, more resources are spent during the idiom-recognition phase. A parser is often unable to analyze long sentences due to their complexity, although they have no grammatical error (Nasukawa, 1995). It is a well-known fact that there is a trade-off between parsing efficiency and parsing coverage. For parsers of practical MT systems, a method is required that improves parsing efficiency without sacrificing parsing coverage.

This paper addresses the problem of reducing parsing complexity by "intrasentence segmentation", which is distinguished from "intersentence segmentation". Intersentence segmentation generally refers to text categorization (Beeferman et al., 1999; Passonneau and Litman, 1997) or sentence-boundary identification (Palmer and Hearst, 1997; Reynar and Ratnaparkhi, 1997). Most context-free parsing algorithms have $O(n^3)$ parsing complexity in terms of time and space, where n is the length of a sentence (Tomita, 1986). Our work is motivated by the fact that parsing becomes faster and more efficient if the length of a sentence decreases. Intrasentence segmentation plays the role of a preliminary step to a chart-based, context-free parser in English–Korean MT.

In English–Korean MT, it is difficult to get accurate results due to the different cultural environments and major structural differences between the languages. We take an idiom-based translation approach to get more natural translations. Under this approach, the parser is a translation-adapted parser rather than a simple syntactic analyzer. Idioms are recognized prior to syntactic analysis, and part of a sentence for an idiom takes an edge in the chart. While parsing long sentences, an ambiguity of the idiom's range may generate more edges than the number of words included in the idiom. This increases parsing complexity a great deal. Intrasentence segmentation is required for the above translation-adapted parser in English–Korean MT.

Intrasentence segmentation consists of two steps: (a) identifying potential segmentation positions in a sentence and (b) selecting an actual segmentation position amongst them. It must possess the following features. An accurate segmentation is important because wrong segmentation leads to a wrong parse tree or parsing failure. In addition, segmentation coverage must be high so that the intrasentence segmentation may be applicable to practical systems. Considering the above requirements, we have applied two machine learning techniques to the intrasentence segmentation problem: "concept learning" and "genetic learning". The learning approach makes it possible to build a more flexible and adaptable intrasentence segmentation system. Experiments show that its accuracy and coverage outperform existing techniques, and that it is effective in parsing long sentences due to its improved parsing efficiency. The method is now incorporated into a commercial English–Korean MT system, E-Tran2000.

The following section discusses related work for intrasentence segmentation and Section 3 describes our approach. The concept-learning task and the process of identifying potential segmentation positions are described in Section 4. Section 5 presents segmentation-appropriateness functions. A genetic learning algorithm used in determining the function's parameters is also described. In Section 6, the segmentation performance of the proposed method is presented, with the degree of contribution to efficient parsing by segmentation. We also compare our approach with other intrasentence segmentation techniques. Section 7 draws conclusions and presents scope to future work.

2. Related Work

Several studies have been carried out to reduce parsing complexity by using intrasentence segmentation. Abney (1991, 1995) introduces chunks that correspond in some way to prosodic patterns and a chunk parsing technique. His idea is inspired by the "performance structures" of Gee and Grosjean (1983). The typical chunk consists of a single content word surrounded by a constellation of function words, matching a fixed template. Chunks are the non-recursive cores of major phrases, i.e., NP, VP, PP, AJP, and they can be regarded as the plausible phrase structures. Abney (1991) has also discussed a rule-based chunker and Chen and Chen (1997) describe a probabilistic chunker. Chunk structures can be described by context-free grammar, but it is difficult to describe the relationships between chunks, because they are similar to performance structures rather than syntactic structures. Furthermore, the order in which chunks occur is much more flexible than the order of words within chunks.

In English–Chinese MT, pattern rules are used to partition an input long sentence into meaningful segments (Li et al., 1990). Each segment is parsed and translated separately and the results of all segments are combined to generate the corresponding Chinese sentence. In addition to the augmented context-free rules, rules for patterns are constructed and they are complementary to each other. In this approach, only long English sentences covered by the manually constructed patterns can be parsed efficiently.

Kim and Ehara (1994) proposed a method of partitioning a long Japanese sentence into several shorter sentences in Japanese–English MT. They used a multi-layered pattern-matching method to search for partition points. The partition increased the parsing rate from 60% to 78% and the translation rate from 25.4% to 31.2%. Subject for the short sentences lacking in subject due to the partition must also be looked into. This method also needs manually constructed partition patterns and a study of long sentence types.

Lyon and Dickerson (1995, 1997) exploit the fact that declarative sentences can almost always be segmented into three concatenated sections (pre-subject, subject, predicate), which can reduce the complexity of parsing English sentences. Though a constituent in one section may have dependent links to elements in other sections, once the three sections have been located, each can be partially processed simultaneously. Pattern-matching capabilities of neural networks are used to locate syntactic constituents of the declarative sentences. This approach is useful for simple sentences containing a subject and a predicate. Long sentences are generally coordinate sentences resulting from the combination of several simple sentences with coordinate conjunctions, or complex sentences consisting of a main clause and several subordinate clauses, which have more than a subject and a predicate.

In English–Korean MT, long English sentences are segmented according to sentence patterns specifying the construction formats of long sentences (Kim and Kim, 1995). Sentence patterns specify the rules for combining the parsed results

of all segments to build a full sentence parse tree. This method improves parsing efficiency about 30% in time and 58% in space. But, constructing sentence patterns requires much human effort, and coverage of the sentence patterns is only 36%.

A rule-based segmentation approach has been proposed to increase the coverage of segmentation (Kim and Kim, 1997). Hand-written rules are used to search segmentation positions, and they have their own segmentation scores. Selection of an actual segmentation position is based on the score, which is manually refined during segmentation tests for long sentences. This method improves segmentation coverage better than the sentence pattern approach. However, the construction of segmentation rules and the refinement of segmentation score are also labor-intensive tasks.

In sum, existing methods for intrasentence segmentation tend to require a great deal of human effort and show limited segmentation coverage. In order to construct segmentation rules automatically and improve the segmentation coverage, we take a machine learning-based approach.

3. Our Approach

3.1. LEARNING-BASED SEGMENTATION

Learning-based segmentation is accomplished in two steps. First, potential segmentation positions in an input sentence are found. Then, an actual segmentation position is determined, and the sentence is segmented at that position. This process continues until the size of each segment of the sentence is within a threshold length, which is determined such that the parser can accommodate a sentence of that length without facing difficulties from parsing complexities. There are several candidate positions for segmentation, all of which may add a parsing overhead. Therefore, selection of the best segmentation position is required, which comprises a two-step segmentation, combined with a search for candidate positions. Figure 1 shows the overall flow of the learning-based segmentation.

The problem of searching for potential segmentation positions can be viewed as a classification problem. Each word in a sentence is classified into two classes, namely "segmentable" and "non-segmentable". In this paper, we learn a function that classifies a word in a sentence into one of these two classes. When the learned function has only two possible values as output, the function is called a "concept", and learning in this case is called "concept learning" (Dean et al. 1995:180–195; Mitchell, 1997: 20–51). In concept learning, the learning program is presented with a set of training examples, labeled "positive" or "negative" depending on whether the example belongs to the concept or not. The objective is to learn a function that correctly labels any example. The function can be expressed in a variety of ways as a set of rules, a procedure, or a network of computing elements modeling neurons. We represent the classification function by sets of if–then rules.

Decision-tree learning provides a practical method for concept learning and for learning other discrete-valued functions. Decision-tree learning algorithms such as

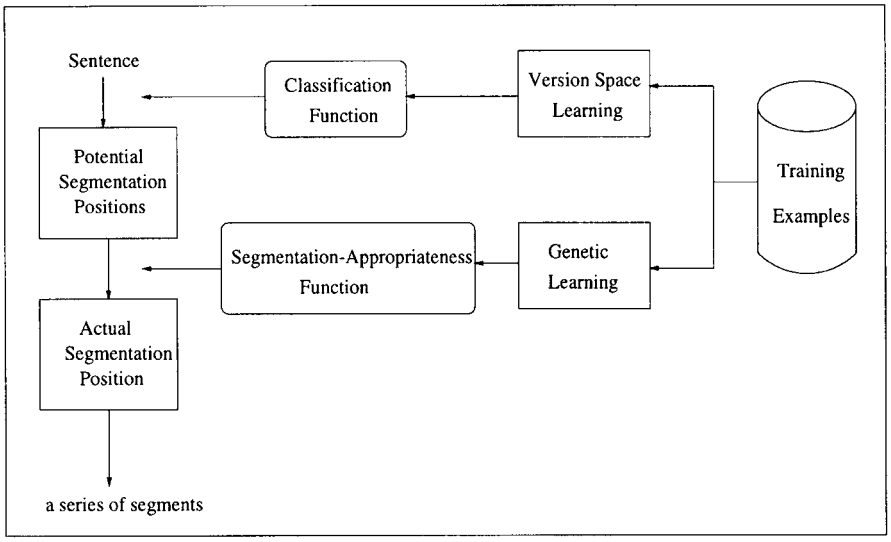

Figure 1. Learning-based segmentation.

ID3 (Quinlan, 1986), ASSISTANT (Cestnik et al., 1987), and C4.5 (Quinlan, 1993) have been widely used in classification problems. Learned trees can be represented as sets of if–then rules. Decision-tree algorithms search a completely expressive hypothesis space, and search incompletely through this space, from simple to complex hypotheses, until a termination condition is met. In our problem, training examples are represented as attribute–value pairs and include attributes for a specific word. The number of possible values for those attributes is as large as that of the vocabulary (which in our case is about 80,000 items): in such cases, decision trees become so large that building such trees becomes impractical. It is computationally not feasible to search hypotheses in such a large hypothesis space, so we restrict the space to include only conjunctions of attribute values.

A version-space learning algorithm, developed as part of a strategy for the search for good generalizations (Mitchell, 1977, 1982), searches an incomplete hypothesis space, and searches this space completely, finding every hypothesis consistent with the training data. It outputs a compact representation of the set of consistent hypotheses through generalizing hypotheses by examining training data without explicitly enumerating all of its members.

Given a training text with segmentation positions manually labeled, a target concept, "word at which a sentence can be segmented" is learned. The learned function is then used in finding potential segmentation positions in a sentence. In Section 4, we describe the version-space learning algorithm in detail.

Selection of an actual segmentation position is based on a "segmentation-appropriateness function", which is a linear sum of variables corresponding to factors that affect the selection. The function coefficients, which are weights

of variables, must be determined such that the function can select the best segmentation position.

The problem of determining weights can be formulated as a search for a highly fit weight vector in a space of possible weight vectors. Genetic algorithms provide a way of searching for this best-fit weight vector.

The problem can be formulated as a problem of function optimization, and genetic algorithms perform quite well in optimization. Also, they can optimize complex and varied functions by changing the problem representation that defines and limits the space of possible solutions. In Section 5, we present three kinds of segmentation appropriateness function. When the number of parameters changes, genetic algorithms are well adapted.

Genetic algorithms are based on processes that can be found in natural evolution. Like evolution, a genetic algorithm operates on a population of individuals that represent potential solutions to a given problem. It then seeks to produce better (more fit) individuals (solutions) by combining the best of the existing ones. Using a "survival of the fittest" tactic, it weeds out the bad and tends to produce more of the good individuals. It produces not only more of the good solutions but better and best solutions. This is because the best traits of parent individuals are combined through a "crossover" operation to produce superior offspring. With only crossover operations, genetic algorithms may get stuck to local minima as network models, such as multilayered perceptrons, make local changes and find local optimal solutions. In addition to recombination by crossover, genetic algorithms use a "mutation" operation to keep from getting stuck at good but non-optimal solutions. Section 5 gives a detailed description of the genetic algorithm.

3.2. SEGMENT-BY-SEGMENT PARSING

This section gives a perspective on the process of building full-sentence parse trees. This paper, however, concentrates on the segmentation method and does not give a detailed description of the process.

We consider a sentence to be a concatenation of a series of segments. Therefore, a sentence can be segmented into several segments. A segment is defined as a correspondent to a phrase or a clause in a sentence, and a context-free chart parser analyzes it. This is similar to the "chunk-by-chunk" parsing strategy (Abney, 1991, 1995, 1996), where a simple context-free grammar is quite adequate to describe the structure of the chunks. The relationship between segments can also be described by context-free grammar, while the relationship between chunks is mediated more by lexical selection than by rigid templates. The segments are parsed from right to left; when a left segment is analyzed, the parsed result of its right-adjacent segment takes an edge in a chart. Edges correspond to a phrase or a clause, while others to words. After the first segment is analyzed, full-sentence parse trees are built.

4. Learning of Rules for Segmentable Position Classification

This section describes the segment and the potential segmentation position called "segmentable position". The concept of "SegmentablePosition" is learned using a corpus annotated with segmentation positions. Through the concept learning, all hypotheses consistent with training examples are found from which the rules are constructed that classify words into segmentable or non-segmentable.

4.1. SEGMENTS AND SEGMENTABLE POSITIONS

A sentence is constructed by the combination of words, phrases, and clauses under well-defined grammatical rules. A sentence can be segmented into several shorter segments, which correspond to the constituents of the sentence. A "segment" is the block of words in a sentence that corresponds to a phrase or a clause, that is, segments correspond to the nonterminal symbols of the context-free grammar that is frequently used to describe natural language. Table I shows the kinds of segments. Figures 2 and 3 show a tree structure of a sentence and a segment structure for the same sentence, respectively. The latter illustrates our view that a sentence is a concatenation of series of segments.

Table I. Syntactic categories and related segments

Syntactic category	Segment
SENT (sentence)	s^{SENT}
NP (noun phrase)	s^{NP}
VP (verbal phrase)	s^{VP}
PP (prepositional phrase)	s^{PP}
AVP (adverbial phrase)	s^{AVP}
AJP (adjective phrase)	s^{AJP}
RLCL (relative clause)	s^{RLCL}
SUBCL (subordinate clause)	s^{SUBCL}

The position of a word is called a "segmentable position" if it is a starting point of a specific segment or a boundary between two segments. It can also be a constituent boundary and a starting position of a phrase or a clause. Even a word may be classified into different types according to its context in the sentence or other factors.

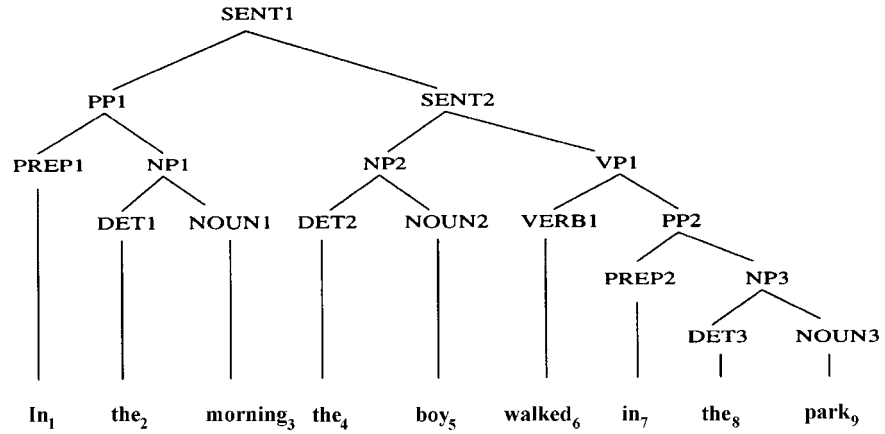

Figure 2. Tree structure for *In the morning the boy walked in the park.*

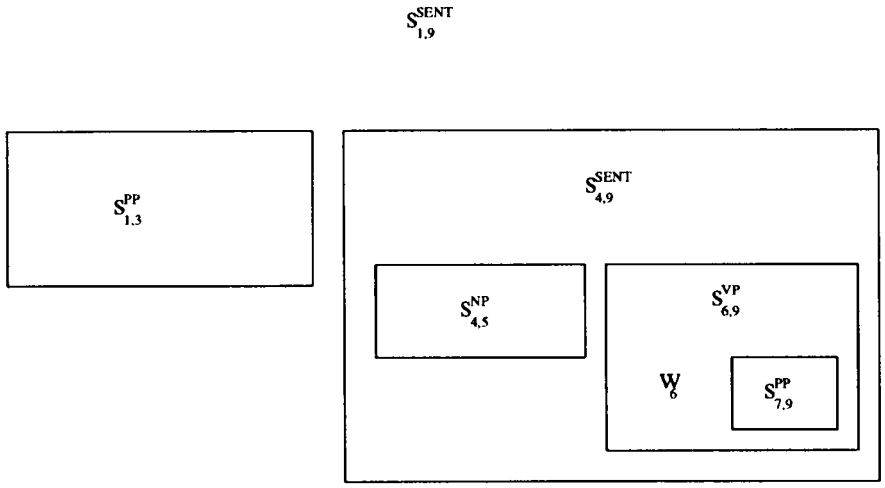

Figure 3. Segment structure.

4.2. LEARNING THE SEGMENTABLE POSITION CONCEPT

4.2.1. *Lexical Context*

From a corpus with annotations on segmentation positions, we construct lexical contexts that serve as training examples for concept learning. A "lexical context" of a word includes a five-word window: two to the left and two to the right of the word and the word itself. It also includes the parts of speech of these words and subcategorization information (which indicates whether a word can have an object clause) for two words to the left, and the position value. A position where a word is located in a sentence is considered when deciding a segmentation position. A "region" is defined as a sequentially ordered block of words in a sentence, and

a "sentence" is a concatenation of regions viewed at a higher level. The position value $posi\_v$ of the ith word w_i is calculated as (1)

$$posi\_v = \lceil \frac{i}{n} \times R \rceil, \tag{1}$$

where n is the number of words in a sentence and R represents the number of regions in the sentence.[1] $posi\_v$ represents the region in which a word is present. Thus, the lexical context of a word is represented by 13 attributes. Figure 4 shows the structure of the lexical context of w_i, the ith word in a sentence.

$sp?$	w_i	w_{i-2} w_{i-1} w_{i+1} w_{i+2}	p_{i-2} p_{i-1} p_{i+1} p_{i+2}	s_cat_{i-2}	s_cat_{i-1}	$posi\_v$

Figure 4. The structure of a lexical context.

There are two kinds of the lexical context: "active context" for a word tagged with a segmentation position and "inactive context" for other words. Examples of a sentence from a corpus, an active and an inactive context are shown in Figure 5. The symbol "#" in the sentence represents a segmentation position. Therefore,

Of course his parents became terribly worried **#when** they saw what was happening to Atzel.

(**1 when** terribly worried they saw ADV ADJ PRON VERB 0 1 2)
(**0 they** worried when saw what ADJ CONJ VERB PRON 1 0 2)

Figure 5. Examples of a training data and active/inactive contexts.

the active context of the word *when* includes the value 1 for attribute $sp?$ and also the following: Two words to the left of *when* (*terribly* and *worried*) and parts of speech of each word (ADV VERB), two words to the right (*they* and *saw*) and parts of speech (PRON VERB), subcategorization information for two words to the left (0 1), and position value (2).

4.2.2. *Version-space Learning*

The problem of finding segmentable positions is formulated as the problem of generating functions that classify words as segmentable or non-segmentable. The functions' outputs have two possible values which are represented by a set of if–then rules in this paper. That is, we learn the concept of "SegmentablePosition", as the word at which the sentence can be segmented. Figure 6 shows the concept-learning task in this paper. Through the learning process, all hypotheses that are consistent with the observed training examples are found. We consider only conjunctive hypotheses in this paper. This makes the hypotheses less expressive, but it

makes the learning computationally practical by restricting the possible hypothesis space. The version-space learning algorithm searches this space completely, finding every hypothesis consistent with the training examples, active contexts in this case. It maintains a compact representation of a set of consistent hypotheses by generalizing the training examples (Zhang and Kim, 1990). It also incrementally refines this representation as each new example is encountered.

- Given:
  - Instances X: lexical contexts, each described by 12 attributes except sp? in Figure 4
  - Hypotheses H: described by a conjunction of constraints on the attributes
  - Target concept c: "SegmentablePosition": $X \rightarrow \{0,1\}$
  - Training examples D: a set of active contexts
- Determine:
  - A set of hypotheses h such that $h(x) = c(x)$ for all $x \in X$, where $h \in H$

Figure 6. The "SegmentablePosition" concept-learning task.

Version space is the set of all hypotheses that are consistent with training examples. We represent the version space as a graph called a "version graph". In the version-graph representation, a node corresponds to a consistent hypothesis and there exist the relations *more_general_than* and *more_specific_than* between parent and child nodes, as defined in Definition 1.

Definition 1: $more\_general\_than$, $more\_specific\_than$ relation
Let h_i and h_j be boolean-valued functions defined over X. h_i is *more_general_than* h_j and h_j is *more_specific_than* h_i ($h_i >_g h_j$) iff
$(\forall x \in X)[((h_j(x) = 1) \rightarrow (h_i(x) = 1)) \;\&\; ((h_i(x) = 1) \not\rightarrow (h_j(x) = 1))]$

A version graph corresponds to a classification rule. The root node acts as the most general boundary, called the "general boundary" of a rule and the terminal node as the most specific one called the "specific boundary". Before describing the version-space learning algorithm, the *join* operation $\oplus$ needs to be defined as in Definition 2.

Definition 2: *join* operation $\oplus$
Let $(a_1, \ldots, a_n)$, $(b_1, \ldots, b_n)$, and $(c_1, \ldots, c_n)$ be lexical contexts.

$$(a_1, \ldots, a_n) \oplus (b_1, \ldots, b_n) = (c_1, \ldots, c_n)$$

$$c_i = \begin{cases} * & \text{if} \quad a_i \neq b_i \\ a_i & \text{if} \quad a_i = b_i \end{cases}$$

where $*$ is a wild-card value accepting any value.

Under the assumption that the attribute values common to at least two segmentation positions influence a word to be a segmentation position, the *join* operation is useful in identifying such attributes of the active context. An attribute without a

wild-card value as a result of the *join* operation is considered to affect whether a word is segmentable or not. Figure 7 shows the version-space learning algorithm. Version-spaces are generated for those words tagged as segmentation positions in a corpus. For an efficient rule generation, the algorithm is run word by word for all words that are labeled as segmentation position in a corpus. Thus, the algorithm receives a set of active contexts of a word and outputs a set of rules for classifying that word. A hypothesis is generated by the *join* operation in a corpus. We attempt to generalize across words and parts of speech by applying the $join$ operation on active contexts. A hypothesis specifies a condition of a segmentable position and takes a node in a version graph where $more\_general\_than$ and $more\_specific\_than$ relations are kept.

- Input: a set of active lexical context of a word w_i, D_{w_i}.
- Output: a set of rules for a word w_i, R_{w_i}.

1. $R_{w_i} = \emptyset$
2. Do for each lc_i and lc_j ($lc_i, lc_j \in D_{w_i}$ and $i \neq j$)
   (a) $hypothesis = lc_i \oplus lc_j$
   (b) Do for each $r_k \in R_{w_i}$
      (i) determine the relation between $hypothesis$ and $r_k : g$
      (ii) if ($hypothesis <_g r_k : g$) then
         if ($r_k : s == NULL$) then $r_k : s \leftarrow hypothesis$
         else insert $hypothesis$ into a path from $r_k : g$ to $r_k : s$
      (iii) if ($hypothesis >_g r_k : g$) then $r_k : g \leftarrow hypothesis$
   (c) if ($hypothesis$ is $conflict$ with all $r_k \in R_{w_i}$ or $R_{w_i} = \emptyset$) then
      - generate new rule r' as follows: $r' : g \leftarrow hypothesis$
      - $R_{w_i} \leftarrow R_{w_i} \cup r'$

Figure 7. Version-space learning algorithm. A specific boundary of a rule r_k is expressed as $r_k : s$ and a general boundary as $r_k : g$. The insertion must satisfy the $more\_general\_than$ and $more\_specific\_than$ relations. If a relation is neither $more\_general\_than$ nor $more\_specific\_than$, it is a "conflict" relation.

The algorithm differs from the general version-space learning algorithm as follows. First, we use only positive examples, i.e. active contexts. This is because we aim at getting a compact representation of all active contexts, from which we construct the rules for segmentable-position classification. Second, we have the *conflict* relation and the associated action. Due to the restricted expressive power from the biased hypotheses space, all consistent hypotheses cannot be represented as only one version graph. When a conflict relation occurs, a new version graph is generated. Therefore, several version graphs may be constructed for each word. Third, we allow a node to exist between a general boundary and a specific boundary to reduce the generalization error.

4.3. IDENTIFICATION OF SEGMENTABLE POSITIONS

Figure 8 shows a concrete example of the rules for segmentable-position classification. The version graph consists of six nodes: one general boundary g_1, two specific boundaries s_1, s_2, and three intermediate nodes n_1, n_2, n_3. An arrow indicates the *more_general_than* relation and is directed from a more specific hypothesis to a more general one. Each hypothesis consists of 12 attributes except *sp*? in Figure 4. A symbol "*" represents a generalized term for words and "–" for part of speech and subcategorization information. Identification of segmentable positions is performed with the *inferable* operation as defined in Definition 3.

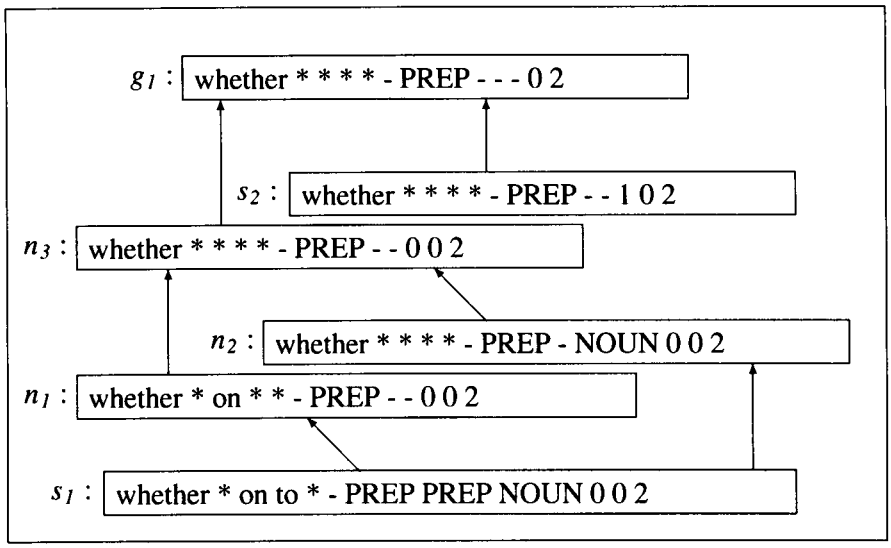

Figure 8. A rule represented as a version graph.

Definition 3: *inferable* operation $\vdash$
If a lexical context lc_{w_i} of a word w_i satisfies the following condition for a rule r_{w_i}, then lc_{w_i} is *inferable* from the rule r_{w_i}: $r_{w_i} \vdash lc_{w_i}$.

$$\{(r_{w_i} : n_l,\ r_{w_i} : n_m) \,|\, r_{w_i} : n_m \geq {}_g lc_{w_i} \quad \text{and} \quad r_{w_i} : n_l \geq {}_g lc_{w_i}\} \neq \emptyset$$

where $r_{w_i} : n_l$ and $r_{w_i} : n_m$ are two nodes on a path from a general boundary $r_{w_i} : g$ to a specific boundary $r_{w_i} : s$ of the rule r_{w_i}, and $(r_{w_i} : n_l,\ r_{w_i} : n_m)$ is an edge connecting two nodes.

If two nodes on a path in a version graph are in a *more_general_than* relation with a lexical context, the lexical context is said to be "inferable" from the rule. Words are identified as segmentable positions, whose lexical context is inferable from the rules for segmentable-position classification.

5. Selection of Segmentation Positions

This section introduces segmentation-appropriateness functions that give a score to each segmentable position. Several factors must be considered while selecting the best segmentation position that would not cause parsing failures and improve parsing efficiency to the greatest possible extent.

5.1. FACTORS CONSIDERED

Though the parsing complexity can be reduced by segmenting an input sentence, wrong segmentation may occur, which causes parsing failures.[2] A segment is "safe" when there is a syntactic category symbol N^P dominating the segment and the segment can be combined with adjacent segments using correct grammar. Figure 9 shows an unsafe and a safe segmentation for the sentence (2). In Figure 9, (a) is an unsafe segmentation because the second segment cannot be analyzed into one syntactic category, resulting in parsing failure. In the safe segmentation (b), the first segment corresponds to an NP/VP and the second to a verb phrase so that we can get a correct analysis result.

(2) The students who study hard will pass the exam.

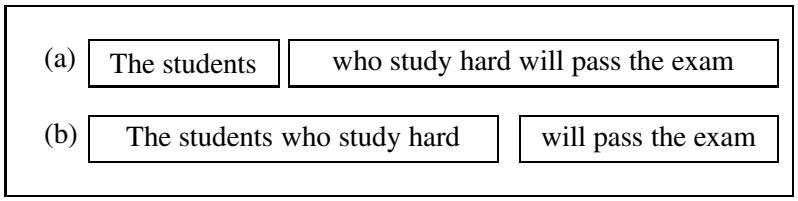

Figure 9. Examples of unsafe/safe segmentation in English-Korean translation

The purpose of segmentation is to reduce parsing complexity while guaranteeing correct parsing. We consider three kinds of probability and two values of segment size for the purpose. Every rule is associated with a probability which can be thought of as a degree of belief in the rule and represents the appropriateness of the segmentable position identified by the rule. To calculate the probability, we need to count the number of lexical contexts that are covered by the rule. The probability of a rule r is calculated as in (3),

$$p(r) = \frac{AC_r}{AC_r + IC_r}, \tag{3}$$

where AC_r (IC_r) is the number of active (inactive) contexts inferable from r. The probability of a word w is also considered and is calculated as in (4),

$$p(w) = \frac{|w|}{W}, \tag{4}$$

where $|w|$ indicates the number of times w is tagged as segmentation position, and W is the total number of words so tagged. Equation (4) reflects how many times a word w was used as a segmentation position in a corpus. It also represents the tendency of a word to be a segmentation position. Each word has a different role in a sentence and some words are more often used as the beginning word of a phrase or a clause than others. Therefore, segmentation at such words may be safer. The probability given in (4) ignores the influence of the position of the word on its being a segmentation position. A word has different possibilities of being a segmentation position according to its position in a sentence. We consider a regional probability $p(w^i)$ calculated as in (5),

$$p(w^i) = \frac{|w^i|}{|w|}, \tag{5}$$

where w^i represents the word w in the ith region of a sentence.

A sentence is segmented into two "child segments" at each segmentation. We consider the size of the child segments to improve parsing efficiency. Parsing complexity increases exponentially with the length of the sentence to be parsed, and the bigger segment has an influence upon the whole parsing complexity. Therefore, a smaller size of the bigger segment is preferred for efficient parsing. The size difference of child segments is also considered for efficient parsing. The bigger-sized segment dominates parsing complexity. The generation of similar-sized segments is desirable for achieving more improved parsing efficiency.

5.2. SEGMENTATION APPROPRIATENESS FUNCTION

We present the function that gives an appropriateness score to each segmentable position. The function must be constructed to select the best segmentation position, which results in safe segments and an improvement of parsing efficiency to the greatest extent possible. The function is a linear function called the "segmentation appropriateness function" (SAF). The SAF includes five variables corresponding to each factor considered in Section 5.1. Here we define three functions: the first function (6) is the first-order polynomial, the second (7) is the second-order polynomial with only second-order terms, while the third (8) is a combination of two previous functions. In each case, w_i and v_i are the ith weight and the ith variable respectively.

$$f_1(v_1, \ldots, v_5) = w_1v_1 + w_2v_2 + w_3v_3 + w_4v_4 + w_5v_5 \tag{6}$$

$$\begin{aligned} f_2(v_1, \ldots, v_5) = {} & w_1{v_1}^2 + w_2{v_2}^2 + w_3{v_3}^2 + w_4{v_4}^2 + w_5{v_5}^2 + \\ & w_6v_1v_2 + w_7v_1v_3 + \ldots + w_{15}v_4v_5 \end{aligned} \tag{7}$$

$$f_3(v_1, \ldots, v_5) = f_1 + f_2 \tag{8}$$

The weights must be determined so that the function may select the best segmentation position. The process of weight determination can be formulated as a

search for a highly fit individual weight vector in a space of all possible weight vectors. Genetic algorithms provide a way to search for the best-fit weight vector. In a genetic algorithm, hypotheses are often described by bit-strings. The search for an appropriate hypothesis begins with a population of initial hypotheses. Members of the current population give rise to the next generation population by means of a "reproduction" procedure, where two individuals selected in the previous step are allowed to mate to produce an offspring. Genetic operators like *crossover* and *mutation* facilitate this mating. Crossover is the process by which the bit-strings of two parent individuals combine to produce two child individuals, while mutation produces small random changes to the bit-string by choosing a single bit at random, then changing its value. At each reproduction step, the hypotheses in the current population are evaluated relative to a given measure of fitness. The fittest hypotheses are selected probabilistically as seeds for producing the next generation. The algorithm operates through iterative updating of the population until a certain termination condition is met.

For the problem of determining weights of the SAF, a genetic algorithm "chromosome" is a vector consisting of five weights. We use two kinds of chromosomes that are represented by bit-strings. Figure 10 shows the structure of each chromosome.

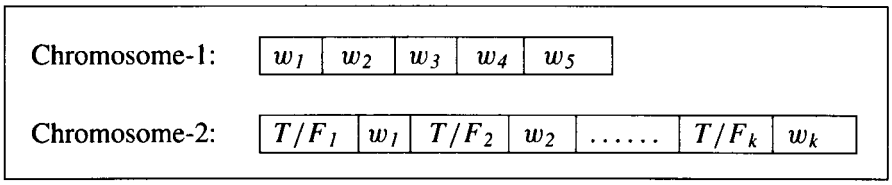

Figure 10. The structures of chromosomes represented by bit-strings.

In Figure 10, w_i is a weight value associated with a variable, and the T/F values take a value of 1 or 0, representing the inclusion or absence of each term when calculating the appropriateness score for each of the segmentation positions. The T/F values are introduced to help fast learning. Though theoretically, some weight values can reach 0 using Chromosome-1, it requires a long time to reach 0. With each chromosome structure, we apply a genetic algorithm to determine the weights of the above three functions. With each iteration, all individuals of a population are evaluated according to the fitness function (9),

$$fitness(a) = \frac{satis(a)}{TD} \tag{9}$$

where $satis(a)$ inidcates the number of training data satisfied by a out of the total number TD of training data. We adopt ranking selection (Schwefel, 1995) to generate the next generation from the current one. This assigns selection probabilities solely on the basis of the rank i of individuals, ignoring absolute fitness values.

The best μ individuals are assigned a selection probability of μ^{-1} while the rest are discarded (10).

$$p_s(a_i) = \begin{cases} \frac{1}{\mu}, & 1 \leq i \leq \mu \\ 0, & \mu < i \leq \lambda \end{cases} \tag{10}$$

where λ is the population size. The termination criterion is the "generational predicate": the maximum number of generations is prescribed. We designate the best individual that has ever appeared in any generation of the population as a result of the genetic algorithm. After running the genetic algorithm 10 times, we take the best individual with the highest fitness value as the weight vector for the SAF. Table II shows the control parameters for the genetic algorithm and Figure 11 describes the adopted genetic algorithm.

Table II. The control parameters for the genetic algorithm

Population size λ		200
Selection parameters	Ranking threshold μ	120
Reproduction parameters	Probability of crossover operation r	0.4
	Probability of mutation operation m	0.2
Termination condition parameter	Maximum number of generations	100
Selection method		Ranking selection
Spousal selection method		Ranking selection

1. From the population P_t at time t, select μ individuals according to the selection probability in (10).
2. With the set of the selected individuals, do the followings $\frac{\lambda}{2}$ times to generate population of next generation P_{t+1}:
   (a) Select two individuals randomly.
   (b) Produce two offsprings a_1, a_2 by applying the *crossover* operation at probability r.
   (c) Apply the *mutation* operation to a_1, a_2 from step (2b) at probability m.
   (d) Add all offsprings from step (2c) to new generation P_{t+1}.

Figure 11. The genetic algorithm

Amongst the segmentable positions, one position with the highest score is selected as a segmentation position (11),

$$sp_* = \arg\max_{sp_i \in D} SAF(sp_i) \tag{11}$$

where sp_* means the selected segmentation position and D is a set of segmentable positions sp_i.

6. Experiments

6.1. CONSTRUCTION OF CORPUS AND RULES FOR SEGMENTABLE POSITION CLASSIFICATION

Our corpus consists of a training portion and a test portion. We extracted sentences with more than 15 words (but excluding sentences with commas) from four different domains. The training portion was used to generate rules for segmentable position classification. From the *Wall Street Journal*, 3,000 sentences were annotated with segmentation positions manually, to serve as the training portion. We also set aside 300 training sentences from the *Wall Street Journal* for genetic learning.

Segmentation performance was evaluated using-test portion that consisted of 3,600 sentences. Test domains were *Byte* magazine, the *Washington Post*, high-school English texts, and the *Wall Street Journal*. 900 sentences were extracted from each domain.

Lexical contexts were generated using 3,000 sentences from the *Wall Street Journal*. We ended up with 45,611 lexical contexts, including 5,375 active and 40,236 inactive lexical contexts. As a result of learning the "SegmentablePosition" concept, 360 version graphs were generated with 9,002 nodes, which correspond to rules for segmentable-position classification.

6.2. SEGMENTATION PERFORMANCE

Segmentation performance is measured in terms of coverage and accuracy. Coverage is the ratio of the number of actually segmented sentences to the number of segmentation target sentences that are longer than α words, where α is a fixed constant distinguishing long sentences from short ones. Accuracy is evaluated in terms of the safe segmentation ratio. They are calculated as in (12) and (13)

$$coverage = \frac{S_{seg}}{S}, \tag{12}$$

$$accuracy = \frac{S_{safe}}{S_{seg}}, \tag{13}$$

where S is the total number of sentences, S_{seg} is the number of actually segmented sentences, and S_{safe} the number of sentences with safe segmentation. The "SC value" is also defined (14) to express the degree of contribution to efficient parsing by intrasentence segmentation. It is not a measure of improved parsing efficiency but the ratio of the sentences that can benefit from intrasentence segmentation. When a long sentence is not segmented or is segmented at unsafe segmentation positions, the sentence is called a "segmentation-error sentence".

$$SC = 1 - \frac{S_e}{S_t}, \tag{14}$$

where S_e is the number of segmentation-error sentences and S_t the number of segmentation target sentences. A sentence longer than α words is considered as

Table III. Comparison of segmentation performance

Methods	Coverage (%)	Accuracy (%)	*SC* value
Baseline 1	100.0	79.5	0.795
Baseline 2	99.0	84.0	0.836
Rule-based method	85.2	86.5	0.703
CL-GA method 1	99.0	87.2	0.853
CL-GA method 2	99.0	86.4	0.845
CL-GA method 3	99.0	88.9	0.870

a segmentation target sentence, where α is set to 12 for our experiments. Table III shows the comparison of segmentation performance.

The proposed method for intrasentence segmentation is called the Concept Learning – Genetic Algorithm (CL-GA) method, and there are three kinds according to the segmentation appropriateness function used: equation (6) is used in method 1, equation (7) in method 2, and equation (8) in method 3. In Table III, the baseline 1 method selects the segmentation positions as follows. First, the words tagged as segmentation position more than five times are identified as segmentable positions. Second, the selection is based on the word probability value in equation (4). This method considers no contextual information and empirically selects segmentation positions. In the baseline 2 method, segmentable positions are identified in the same way as that of the CL-GA method. The selection of segmentation positions is based only on the rule's probability value in the equation (3). In the rule-based method (Kim and Kim, 1997), manually constructed rules are used for finding segmentable positions, and the selection of a segmentation position is based on the manually refined scores. In comparing the baseline 1 method with the others, accuracy is observed to depend on the context. The CL-GA method shows 99.0% coverage, which is superior to the rule-based method, and it also performs better than the Rule-based method in terms of accuracy. In the CL-GA method, segmentation appropriateness functions result in a different accuracy performance.

Table IV shows the segmentation performance of the proposed CL-GA method 3, and the domains of test data classify the performance. The *SC* value for the sentences from the same domain as the training data is 0.873. It is 0.860 for sentences from the high-school English text, 0.833 for sentences from *Byte* magazine, and 0.9 for sentences from the *Washington Post*. Although the *SC* values slightly differ for each test domain, the average value is 0.87, and there is little difference between the values for each domain. This suggests that the proposed method for intrasentence segmentation can be applicable irrespective of the domain.

Table IV. Segmentation performance of CL-GA method 3

Domain	Coverage (%)	Accuracy (%)	*SC* value
Wall Street Journal	99.3	87.9	0.873
English text	100.0	86.0	0.860
Byte magazine	98.6	84.5	0.833
Washington Post	98.0	91.8	0.900
Total	99.0	87.9	0.870

6.3. ANALYSIS OF SEGMENTATION ERRORS

Segmentation errors can be classified into three types. The first type is "unsafe segmentation". Figure 12 shows an example of an unsafe segmentation for the sentence in (15).

(15) The developers created the objects as 3-D models and then rendered them into bit maps from various angles.

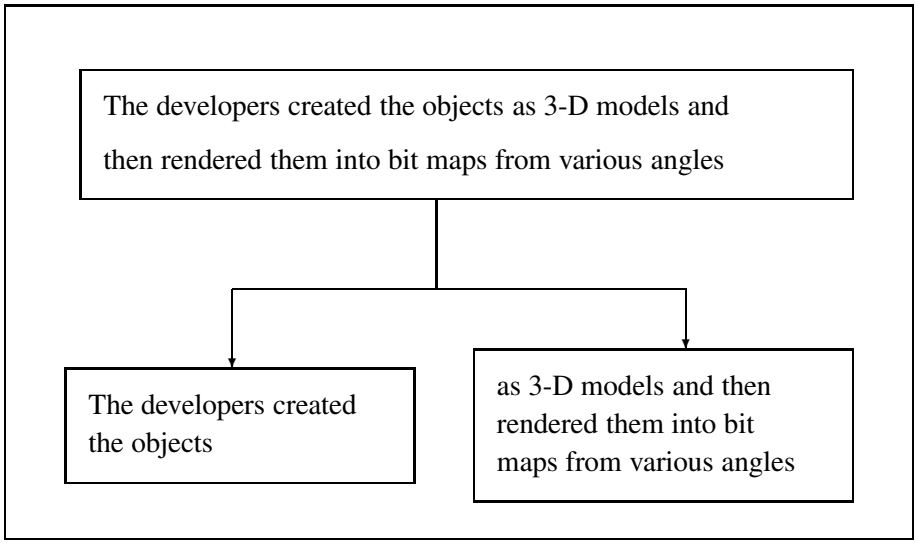

Figure 12. Example of unsafe segmentation.

In the figure, the second segment cannot be analyzed as a phrase or a clause, which causes parsing failure. Second, there can be cases in which a target sentence for segmentation is not segmented. For example, the sentence in (16) was not segmented, though its length is 17.

(16) These two objects are the result of the cooperation of the virtual-memory manager and an external pager.

Third, there could be cases in which better segmentation positions[3] exist but a sentence is not segmented at those positions, though the sentence was segmented at safe positions. Figure 13 shows a safe segmentation for the sentence (17). At first, the sentence is segmented at *as*, resulting in two segments of size 13 and 5. The first segment is again segmented, so the sentence results in the combination of three segments. Figure 14 shows a better segmentation for the sentence. The sentence is segmented at *they*, resulting in two segments of size 7 and 11, which can be analyzed without further segmentation. In terms of segmentation, a single segmentation has a smaller chance of error than two or more segmentations.

(17) Fully 10 percent of those polled indicated they had made a final decision as to their presidential choice.

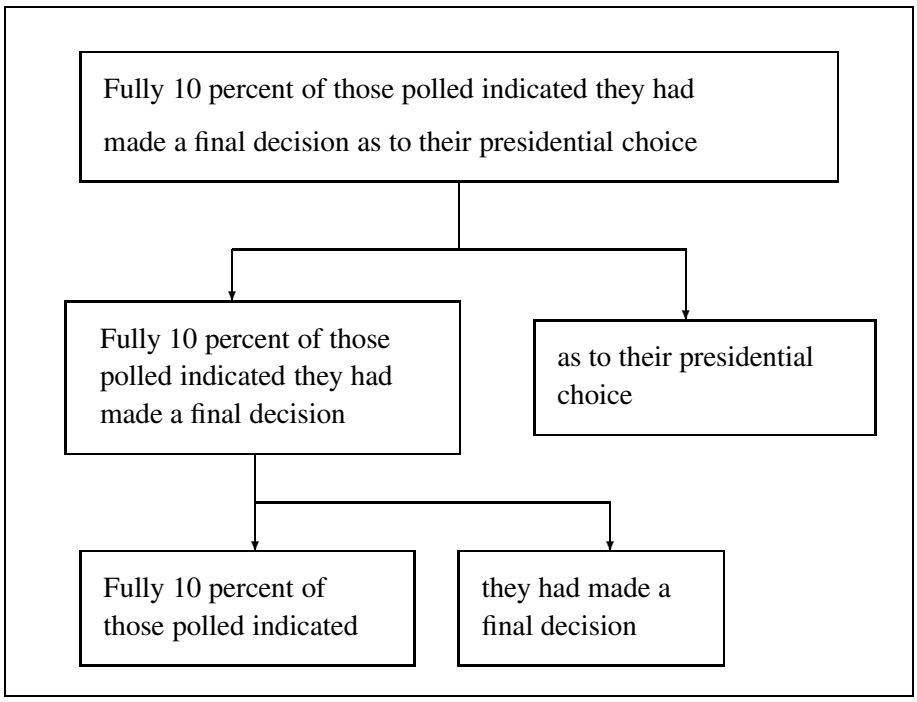

Figure 13. Example of safe segmentation 1.

The first type causes incorrect analysis or parsing failure, and the second does not reduce the parsing complexity so that the parsing of the sentence may not complete. Therefore, only the first two types of error are included in segmentation errors in Section 6.2.

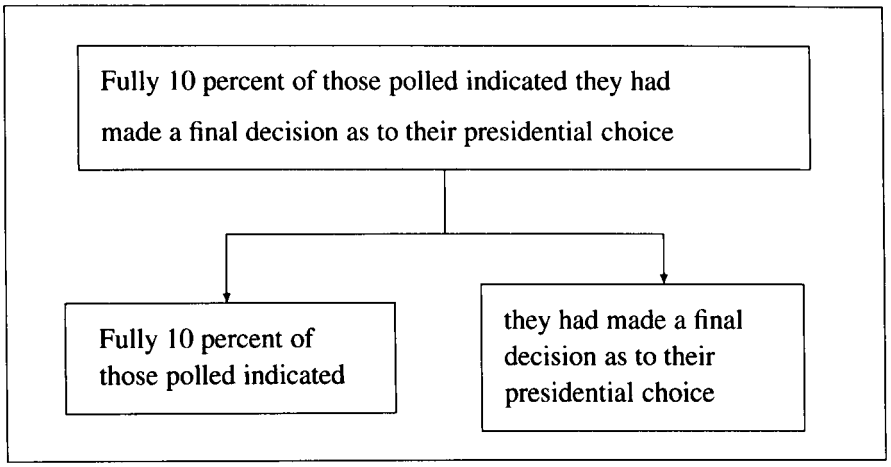

Figure 14. Example of safe segmentation 2.

6.4. PARSING EFFICIENCY

Parsing efficiency is generally measured by time and memory spent in parsing. In English–Korean MT, an idiom-based chart parser spends a lot of time and memory in analyzing sentences with more than 20 words. Even worse, some sentences may not parse due to memory and time limitations. Efficiency improvement therefore is compared for the parsing of the sentences with more than 15 and less than 20 words. An Ultra-Sparc 30 machine was used for these comparisons. Efficiency improvements are measured as in (17) and (18),

$$EI_{time} = \frac{t_{unseg} - t_{seg}}{t_{unseg}} \times 100, \tag{17}$$

$$EI_{memory} = \frac{m_{unseg} - m_{seg}}{m_{unseg}} \times 100, \tag{18}$$

where t and m are the amount of time and memory used during parsing, $unseg$ without segmentation and seg with segmentation. Table V summarizes the results. The experiment was performed using 1,200 sentences, 300 sentences from each of the four domains. By segmenting long sentences into several manageable-sized segments, we could get computational advantages in terms of both time and memory. Though wrong segmentation causes wrong results, the improved efficiency by intrasentence segmentation can surpass such disadvantages.

7. Conclusion and Future Work

In this paper we have presented two machine-learning techniques for segmenting long sentences. These have reduced the computational complexity of the chart parser, which was adopted for our idiom-based English–Korean MT system, but

Table V. Comparison of analysis results with/without segmentation

Domain	Time (sec)		Improvement (%)	Memory (MB)		Improvement (%)
	With	Without		With	Without	
Wall Street Journal	2.32	6.35	63.5	2.50	4.91	49.0
English text-book	1.92	5.57	65.6	2.01	4.21	54.7
Byte magazine	2.61	7.83	66.7	2.04	4.00	49.0
Washington Post	2.94	8.87	66.9	2.18	4.96	56.0
Total	2.44	7.16	65.9	2.18	4.59	51.8

had difficulties in accommodating long sentences. The two learning techniques are concept learning and genetic learning. Concept learning outputs a compact representation of the set of rules, and the learning speed allows for a large set of training data. It makes rule acquisition and rule maintenance simple. In order to select the best segmentation position, SAFs have been proposed with five variables that correspond to each factor affecting the selection. The weights associated with variables are optimized by genetic algorithm. The genetic algorithm finds an optimal weight vector for the function that results in better segmentation performance, and it performs such a function optimization quite well. Also, it is well adapted to change of function parameters.

In experiments with 3,600 test sentences extracted from four different domains, about 87% of the sentences benefitted from intrasentence segmentation. Experimental results have also shown that segmentation performance is little affected by the nature of the test domains. With the help of the proposed intrasentence segmentation, parsing efficiency is improved by 66% in time and 52% in space. These results are better than those obtained by the method using sentence patterns (Kim and Kim, 1995). These features suggest that the proposed method is applicable to practical English–Korean MT and provide a perspective for parsing long sentences by reducing the parsing complexity.

Future work can be pursued in three directions. First, studies on recovery mechanisms from unsafe segmentation before parsing will be necessary since unsafe segmentations may cause parsing failures. Second, using negative examples in version-space learning can be considered. Using only positive examples in version-space learning, the acquired rules may classify a word with a negative lexical context as segmentable. With negative examples, a higher accuracy can be obtained by removing such rules that misclassify a word. Third, parsing-control mechanisms that exploit the characteristics of segmentation positions and the parallelism among segments should be studied. This will further enhance parsing efficiency.

Notes

[1] R is a heuristically set value and was set to 4 based on the assumption that there are 4 regions in a sentence: two regions for pre-subject and subject sections and two for predicate section. This reflects the fact that a sentence generally consists of pre-subject, subject and predicate and that the predicate section becomes long as modifiers (relative clause, prepositional phrase, and etc.) are added.

[2] Parsing failure means that there is no complete sentence structure or the wrong parse tree is generated, after parsing a sentence.

[3] They are the positions that can result in more similar-sized segments or reduce the number of intrasentence segmentation.

References

Abney, Steven: 1991. 'Parsing by Chunks', in Robert Berwick, Steven Abney and Carol Tenny (eds), *Principle-Based Parsing*, Dordrecht, Kluwer Academic Publishers, pp. 257–278.

Abney, Steven: 1995. 'Chunks and Dependencies: Bringing Processing Evidence to Bear on Syntax', in Jennifer Cole, Georgia M. Green and Jerry L. Morgan (eds) *Computational Linguistics and the Foundations of Linguistic Theory*, Stanford, CA, CSLI Publications, pp. 145–164.

Abney, Steven: 1996. 'Partial Parsing via Finite-State Cascades', *ESSLLI'96 Workshop on Robust Parsing Workshop*, Prague, Czech Republic.

Beeferman, D., A. Berger and J. Lafferty: 1999. 'Statistical Models for Text Segmentation', *Machine Learning* **4**, 177–210.

Cestnik, B., I. Kononenko, and I. Bratko: 1987. 'ASSISTANT-86: A Knowledge-Elicitation Tool for Sophisticated Users', in I. Bratko and N. Lavrac (eds) *Progress in Machine Learning*, Wilmslow: Sigma Press.

Chen, Kuang-Hua and Hsin-Hsi Chen: 1997. 'A Hybrid Approach to Machine Translation System Design', *Computational Linguistics and Chinese Language Processing* **23**, 241–265.

Cranias, Lambros, Harris Papageorgiou and Stelios Piperidis: 1994. 'A Matching Technique in Example-Based Machine Translation', *COLING 94: The 15th International Conference on Computational Linguistics*, Kyoto, Japan, pp. 100–104.

Dean, Thomas, James Allen and Yiannis Aloimonos: 1995. *Artificial Intelligence: Theory and Practice*. Amsterdam: Benjamin/Cummings Publishing Company.

Gee, James Paul and François Grosjean: 1983. 'Performance Structures: A Psycholinguistic and Linguistic Appraisal', *Cognitive Psychology* **15**, 411–458.

Kim, Sung Dong and Yung Taek Kim: 1995. 'Sentence Analysis Using Pattern Matching in English–Korean Machine Translation', *ICCPOL '95: International Conference on Computer Processing of Oriental Languages*, Honolulu, Hawaii, pp. 199–206.

Kim Sung Dong and Kim Yung Taek: 1997. Hyo-eul-juk-in yeoung-u gu-moon boon-seok-eul wui-han moon-jang boon-hal [Sentence Segmentation for Efficient English Syntactic Analysis], *Han-kook Jung-bo-gwa-hak-hoy Non-moon-ji (Journal of Korea Information Science Society)* **24**, 884–890.

Kim, Yeun-Bae and Terumasa Ehara: 1994. 'A Method for Partitioning of Long Japanese Sentences with Subject Resolution in J/E Machine Translation', *ICCPOL '94: International Conference on Computer Processing of Oriental Languages*, Taejon, Korea, pp. 467–473.

Lee Ho Suk: 1993. Young-hasn gi-gye-byen-yeouk-eul wui-han mal-moong-chi-e gi-ban-han byeon-hwan-sa-jun-wy ja-dong goo-chuk [Automatic Construction of Transfer Dictionary based on the Corpus for English–Korean Machine Translation], PhD thesis, Seoul National University.

Li, Wei-Chuan, Tzusheng Pei, Bing-Huang Lee and Chuei-Feng Chiou: 1990. 'Parsing Long English Sentences with Pattern Rules', *COLING–90: Papers presented to the 13th International Conference on Computational Linguistics*, Helsinki, Vol. 3, pp. 410–412.

Lyon, Caroline and Bob Dickerson: 1995. 'A Fast Partial Parse of Natural Language Sentences Using a Connectionist Method' *Seventh Conference of the European Chapter of the Association for Computational Linguistics*, Dublin, Ireland, pp. 215–222.

Lyon, Caroline and Bob Dickerson: 1997. 'Reducing the Complexity of Parsing by a Method of Decomposition', *International Workshop on Parsing Technology*, Boston, pp. 215–222.

Mitchell, Tom M.: 1977. Version Spaces: An Approach to Concept Learning. PhD thesis, Stanford University.

Mitchell, Tom M.: 1982. 'Generalization as Search', *Artificial Intelligence* **18**, 20–51.

Mitchell, Tom M.: 1997. *Machine Learning*. New York: McGraw Hill.

Nasukawa, Tetsuya: 1995. 'Robust Parsing Based on Discourse Information: Completing partial parses of ill-formed sentences on the basis of discourse information', *33rd Annual Meeting of the Association for Computational Linguistics*, Cambridge, Mass., pp. 39–46.

Palmer, David D. and Marti A. Hearst: 1997. 'Adaptive Multilingual Sentence Boundary Disambiguation', *Computational Linguistics* **23**, 241–265.

Passonneau, Rebecca J. and Diane J. Litman: 1997. 'Discourse Segmentation by Human and Automated Means', *Computational Linguistics* **23**, 103–139.

Quinlan, J. R.: 1986. 'Induction of Decision Trees', *Machine Learning* **1**, 81–106.

Quinlan, J. R: 1993. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Reynar, Jeffrey C. and Adwait Ratnaparkhi: 1997. 'A Maximum Entropy Approach to Identifying Sentence Boundaries', *Fifth Conference on Applied Natural Language Processing*, Washington, DC, pp. 16–19.

Schwefel, Hans-Paul: 1995. *Evolution and Optimum Seeking*. New York: Wiley.

Tomita, Masaru: 1986. *Efficient Parsing for Natural Language*, Dordrecht, Kluwer Academic Publishers.

Yoon, Sung Hee: 1994. 'Efficient Parser to Find Bilingual Idiomatic Expressions for English-Korean Machine Translation', *ICCPOL '94: International Conference on Computer Processing of Oriental Languages*, Taejon, Korea, pp. 455–460.

Zhang, Byoung-Tak and Yung-Taek Kim: 1990. 'Morphological Analysis and Synthesis by Automated Discovery and Aquisition of Linguistic Rules', in *COLING-90: Papers presented to the 13th International Conference on Computational Linguistics*, Helsinki, Vol. 2, pp. 431–436.