A reestimation algorithm for probabilistic dependency grammars

SEUNGMI LEE and KEY-SUN CHOI

Department of Computer Science, Center for Artificial Intelligence Research, Korea Advanced Institute of Science and Technology, 373-1 Kasung-Dong YuSung-Gu Taejon 305-701 Korea
e-mail: leesm,kschoi@world.kaist.ac.kr

(Received 23 June 1997; revised 25 September 1998)

Abstract

A probabilistic parameter reestimation algorithm plays a key role in the automatic acquisition of stochastic grammars. In the case of context-free phrase structure grammars, the inside-outside algorithm is widely used. However, it is not directly applicable to Probabilistic Dependency Grammar (PDG), because PDG is not based on constituents but on a head-dependent relation between pairs of words. This paper presents a reestimation algorithm which is a variation of the inside-outside algorithm adapted to probabilistic dependency grammar. The algorithm can be used either to reestimate the probabilistic parameters of an existing dependency grammar, or to extract a PDG from scratch. Using the algorithm, we have learned a PDG from a part-of-speech-tagged corpus of Korean, which showed about 62.82% dependency accuracy (the percentage of correct dependencies) for unseen test sentences.

1. Introduction

Many efforts have been made to induce grammars automatically by using a corpus of a vast size. Corpus-based stochastic grammar induction has many advantages over the manual construction of grammar, such as the simple acquisition and extension of linguistic knowledge, the easy treatment of ambiguities by virtue of its innate scoring mechanism, and the fail-soft reaction to ill-formed or extra-grammatical sentences.

Most of the works for corpus-based stochastic grammar acquisition concentrate on context-free Phrase Structure Grammars (PSG). In those works, the inside-outside algorithm (Jelinek, Lafferty and Mercer 1990) is extensively used as a reestimation algorithm of PSG. The algorithm learns a grammar by iteratively adjusting the probabilities of rules to minimize the entropy of the training corpus.

Lari and Young (1990) and Chen (1995) use the inside-outside algorithm for unsupervised learning of PSGs, and apply the PSG to language modeling. Lari and Young (1990) initially construct CNF grammar rules by enumerating all possible rules with limited nonterminals and terminals, and then reestimate the rules by the inside-outside algorithm. Chen (1995) employs a greedy heuristic search in the induction of CNF grammar, and retraîns the probabilities of the rules using the inside-outside algorithm.
If a bracketed corpus is available, the corpus can provide useful information in learning a more linguistically-plausible grammar. Black, Lafferty and Roukos (1992) modify the inside-outside algorithm slightly to make it reestimate a grammar using only the parses that are consistent to treebank parses. Using the algorithm, they train a grammar which was initially written by linguistic experts. Pereira and Schabes (1992) extend the inside-outside algorithm so that it can learn a grammar from a partially bracketed corpus. Both works have resulted in a more linguistically qualitative grammar, but they require a (partially) bracketed corpus, a prerequisite that cannot be fulfilled easily, in general. In an effort to cope with this difficulty, Marcken (1995) suggests that incorporating mutual information between phrase heads into the inside-outside algorithm should result in a linguistically plausible grammar without a bracketed corpus, mentioning that head-driven grammatical formalisms like dependency grammars or link grammars could be better suited to the automatic induction of grammar.

In case of Dependency Grammar (DG), it can be divided into broadly two kinds: DG that specifies word order; and DG that does not. Most of the efforts towards automatic learning of stochastic dependency grammar aim at DG with word order. DG with word order represents the rules in a form such as

\[ X(Y_1 Y_2 \ldots Y_n) \]

which means that \( X \) governs (is head of) words \( Y_1 \ldots Y_n \) which occur in the given order where \( X \) is to occupy the position of * (Gaifman 1965). This form of dependency grammar was proved to be weakly equivalent to phrase structure grammar by Gaifman (1965). Since this form of dependency grammar can be viewed as a highly restricted form of context-free phrase structure grammar, where the constituents are without nonterminal labels, it can be reestimated by the inside-outside algorithm directly, as was done by Carroll and Charniak (1992).

Carroll and Charniak (1992) have experimented on the learning of dependency grammar with word order using the inside-outside algorithm. They use grammar rules in the form

\[ X \rightarrow \alpha X \beta, \]

where \( X \) is a terminal, \( \alpha \) and \( \beta \) are possibly empty strings of nonterminals, and for each terminal \( T \) in the language, there is exactly one corresponding nonterminal, \( \tilde{T} \). The rule means that \( X \) is the head of the other nonterminals in \( \alpha \) and \( \beta \). For each sentence in a corpus sorted in increasing sentence length, they construct such rules as necessary and retrain the resulting grammar iteratively using the inside-outside algorithm.

A similar grammar formalism to DG with word order is link grammar. Link grammar was introduced by Sleator and Temperley (1991), and the probabilistic parameter reestimation algorithm was developed by Lafferty, Sleator and Temperley (1992). Link grammar resembles dependency grammar in that it describes a language by linking lexical units, and does not include any constituent categories, while it differs from DG in that it has no concept of any head-dependent relation. Link grammar defines several ‘connectors’, by which two words can be linked, that are
A reestimation algorithm for PDGs

similar to the functional roles of head-dependent relations in DG. Every word \( W \) in link grammar is correlated with one or more lists of left, right connectors that represent a usage of the word, such as

\[(l_{m}, l_{m-1}, \ldots, l_{1}), (r_{1}, r_{2}, \ldots, r_{n})\],

where the lists of left (\( l_{i} \)) and right-connectors (\( r_{i} \)) are ordered, implying that the words to which \( l_{i}, l_{i+1}, \ldots \) are connected are decreasing in distance to the left of \( W \), and the words to which \( r_{i} \) are connected are decreasing in distance to the right of \( W \). As can be seen from the description of a word, link grammar resembles dependency grammar with word order which explicitly specifies the order of dependent words. DG with word order is appropriate to describe languages with a somewhat fixed word order such as English and Chinese. For languages with much variability in word order, though, such as Korean and Russian, DG without word order is appropriate.

Dependency grammar without word order represents the rules in the form

\[ x \rightarrow y(f) \]

which means that word \( x \) governs (is head of) word \( y \), and the functional role of the relation is \( f \). For example, the head-dependent relations in the sentence ‘I gave him a book’ can be described as

- \( \text{gave} \rightarrow \text{I} \) (subject),
- \( \text{gave} \rightarrow \text{him} \) (indirect object)
- \( \text{gave} \rightarrow \text{book} \) (direct object), and
- \( \text{gave} \rightarrow \text{a} \) (determiner).

As shown, the dependency rules do not specify the order of dependent words. It specifies only the head-dependent relations between pairs of words and the functional roles of the relations. To our knowledge, there has not been any attempt to automatically learn and/or to reestimate the probabilistic parameters of this form of dependency grammar. The inside-outside algorithm cannot be used directly for this form of grammar, because the grammar is not based on constituents but on inter-word dependencies. Using the inside-outside algorithm, the probabilities of the constituents are reestimated, but not the probability of each inter-word head-dependent relation in the constituents.

In this paper, we propose a reestimation algorithm for probabilistic dependency grammar without word order\(^1\). The reestimation algorithm is a variation of the inside-outside algorithm adapted to PDG. We define the non-constituent objects complete-link and complete-sequence as the basic units for dependency structures on which the inside-outside probabilities are developed. The probabilities of inter-word dependencies are reestimated through computation of the inside-outside probabilities of the objects. The reason why we define a novel structure of complete-link and complete-sequence is that they give us a framework of viewing dependency structures from the viewpoint of inter-word dependencies, not from constituents. The algorithm has \( O(n^{3}) \) time complexity, where \( n \) is the number of words in the sequence to be

\(^{1}\) In the following, we denote probabilistic dependency grammar without word order by PDG for short.
processed. It can be used either for the learning of probabilistic dependency grammars from scratch, or for reestimating the probabilistic parameters of an existing dependency grammar.

The rest of the paper is organized as follows. Section 2 defines the probabilistic model of the dependency grammar we use, and the non-constituent objects complete-link and complete-sequence. Section 3 describes the reestimation algorithm based on the non-constituent objects, and section 4 shows some experimental results on learning the probabilistic dependency grammar of Korean. Finally, section 5 concludes this paper.

2. Dependency grammar

Dependency grammar describes a language with a set of head-dependent relations between words in the language. In general, a functional role is assigned to a head-dependent relation, and specifies the syntactic and/or semantic relation between the head and the dependent, such as modifiee-modifier, predicate-argument, etc. However, in this paper we use the minimal definition of dependency grammar with only a head-dependent relation, and without functional roles. Thus, the rules are represented in the form 

\[ x \rightarrow y, \]

which means that word \( x \) is the head of word \( y \).

A sentence is analysed by establishing dependency links between pairs of words in the sentence. A dependency analysis \( \mathcal{D} \) is a set of inter-word dependencies which satisfy the following requirements: (1) every word in the sentence except for the head word of the sentence has its head in the sentence; (2) every word can have only one head; and (3) there is neither crossing nor a cycle of dependencies.

An example of \( \mathcal{D} \) of the sentence ‘I gave him a book’ is depicted in Fig. 1, with arrows pointing from the head to dependent words. For structural generality, we assume that there is always a special purpose tag, EOS (End of Sentence) at the end of a sentence, and the tag has the head of the sentence as its own dependent (‘gave’ in Fig. 1).

The following five head-dependent relations participate in the dependency analysis of Fig. 1:

- EOS \( \rightarrow \) gave
- gave \( \rightarrow \) I
- gave \( \rightarrow \) him
- gave \( \rightarrow \) book
- book \( \rightarrow \) a

2.1 Definition: Complete-link and complete-sequence

Here we define the non-constituent objects complete-link and complete-sequence that represent partial \( \mathcal{D} \)s for substrings. They are used as building blocks to construct overall \( \mathcal{D} \)s, and are the basic units on which the inside-outside probabilities are defined in section 3.
A set of head-dependent relations on a word sequence $w_{i,j}$ is a complete-link\(^2\) when the following conditions are satisfied:

- There is $w_i \rightarrow w_j$ or $w_i \leftarrow w_j$ exclusively.
- Every inner word has its head in the word sequence.
- Neither crossing nor a cycle of dependency relations exists in the set.

A complete-link has direction. A complete-link on $w_{i,j}$ is said to be ‘rightward’ if the outermost relation is $w_i \rightarrow w_j$, and ‘leftward’ if the relation is $w_i \leftarrow w_j$. A basis complete-link is defined on a string of two adjacent words, $w_{i,i+1}$, either as $w_i \rightarrow w_{i+1}$ or $w_i \leftarrow w_{i+1}$. Figure 2(a) is a rightward complete-link, and both figures 2(b) and 2(c) are leftward ones. Figure 2(c) is an example of a basis complete-link.

A complete-sequence is a sequence of zero or more adjacent complete-links that have the same direction. A complete-sequence also has direction, which is determined by the direction of its component complete-links. If a complete-sequence is composed of leftward complete-links, the complete-sequence is leftward. A rightward complete-sequence is composed of rightward complete-links. A basis complete-sequence is a zero sequence of complete-links, and is defined on a string of one word. In figure 3, figure 3(a) is a rightward complete-sequence composed of four rightward complete-links, and figure 3(b) is a leftward one composed of one leftward complete-link. Figure 3(c) is a complete-sequence composed of zero complete-links. It can be both leftward and rightward. For the sake of readability, we draw dashed lines, one for each component complete-link, which together constitute the complete-sequence.

The word ‘complete’ means that linking the head-dependent relations on the inner words is completed, and consequently there is no need to process them further. From now on, we use $L_{r}(i,j)$ for a rightward complete-link on $w_{i,j}$ and $L_{l}(i,j)$ for a leftward complete-link on $w_{i,j}$.

\(^2\) We use $w_i$ for the $i$th word in a sentence and $w_{i,j}$ for the sequence of words between $w_i$ and $w_j$ ($i < j$).
complete-link on \( w_{i,j} \). Likewise, we use \( S_r(i,j) \) and \( S_l(i,j) \) for a rightward complete-sequence and a leftward complete-sequence on \( w_{i,j} \), respectively. Subscripts \( r \) and \( l \) represent the direction; \( r \) is for rightward and \( l \) is for leftward.

Then, any complete-link on \( w_{i,j} \) can be viewed as the following combination:

- \( L_r(i,j) := \{ w_i \rightarrow w_j, S_r(i,m), S_l(m+1,j) \}, i \leq m < j \)
- \( L_l(i,j) := \{ w_i \leftarrow w_j, S_r(i,m), S_l(m+1,j) \}, i \leq m < j \)

Otherwise, the set of dependencies does not satisfy the conditions of no crossing, no cycle and no multiple heads, and thus is no longer a complete-link. Figure 4 shows the abstract rightward/leftward complete-links for \( w_{i,j} \), and the basis complete-links. A double-slashed link means a complete-sequence.

Similarly, any complete-sequence on \( w_{i,j} \) can be viewed as the following combination\(^8\):

- \( S_r(i,j) := \{ S_r(i,m), L_r(m,j) \}, i \leq m < j \)
- \( S_l(i,j) := \{ S_l(i,m), L_l(m,j) \}, i \leq m < j \)

Figure 5 shows the abstract rightward/leftward complete-sequences on \( w_{i,j} \), and the basis complete-sequences.

Then any dependency analysis of a sentence can be constructed by a recursive combination of complete-links and complete-sequences from bottom up. Figure 6 shows an abstract representation of \( \mathcal{D} \) of an \( n \)-word sentence, \( w_{1,n} \). When \( w_k(1 \leq k \leq n) \)

---

\(^8\) All the unique \( S_r \)s on \( w_{i,j} \) can be found by enumerating the combinations of \( S_r(i,m) \) with \( L_r(m,j) \), for \( i \leq m < j \). The same set of \( S_r \)s can be found by the combinations of \( L_r(i,m) \) with \( S_l(m,j) \), for \( i < m \leq j \). However, allowing the two kinds of combinations will find redundant \( S_r(i,j) \)s. This is the same for the case of \( S_l(i,j) \). So we restrict \( S_r(i,j) \) and \( S_l(i,j) \) to the above combinations to prevent multiple constructions of the same complete-sequences.
is the head of the sentence any $D$ of the sentence can be represented by a $S_i(1, EOS)$ uniquely by the assumption that there is always a dependency relation from $EOS$ to the head word of the sentence, $w_k \leftarrow EOS$. As an example, figure 7 shows the gradual construction of the $D$ in Fig. 1. From the top of figure 7, complete-links and complete-sequences are combined into larger ones until they construct together the overall $D$. The $D$ is represented uniquely by the $S_i(1, EOS)$. The construction sequence in figure 7 is not identical to the actual processing sequence. In each phase, one or more phases are incorporated into one for the sake of convenience.

### 2.2 Probabilistic model

Here we describe the probabilistic model of the simple dependency grammar. The probability of a sentence is the sum of the probabilities of all $D$s the sentence can have. The probability of a $D$, in turn, is approximated to the product of the probabilities of all the dependency relations in the analysis:

$$p(w_{1,n}) = \sum_D p(D, w_{1,n})$$

$$\cong \sum_D \prod_{x \rightarrow y \in D} p(x \rightarrow y),$$

where $p(x \rightarrow y)$ is estimated as

$$p(x \rightarrow y) = p(y \mid x)$$

$$= \frac{C(x \rightarrow y)}{\sum_z C(x \rightarrow z)},$$

and thus

$$\sum_y p(x \rightarrow y) = 1.$$

Then, the probability of a sentence $w_{1,n}$ in terms of the probabilities of a complete-link and a complete-sequence is defined as

$$p(w_{1,n}) = \sum_D p(D, w_{1,n})$$

$$\cong \sum_D p(S_i(1, EOS)),$$

where

$$p(L_i(i,j)) = p(w_j \rightarrow w_i) p(S_i(i,m)) p(S_{i+1}(m+1,j)),$$

$$p(L_i(i,j)) = p(w_i \leftarrow w_j) p(S_i(i,m)) p(S_{i+1}(m+1,j)),$$

$$p(S_i(i,j)) = p(S_i(i,m)) p(L_i(m,j)),$$

$$p(S_i(i,j)) = p(S_i(i,m)) p(L_i(m,j)),$$

and the basis probabilities are

$$p(L_i(i,i+1)) = p(w_i \rightarrow w_{i+1})$$

$$p(L_i(i,i+1)) = p(w_i \leftarrow w_{i+1}),$$

$$p(S_i(i,i)) = p(S_i(i,i)) = 1.$$
Fig. 7. Gradual construction of an overall $\mathcal{G}$.

3 Reestimation algorithm

The reestimation algorithm is a variation of the inside-outside algorithm (Brown, Della Pietra, deSouza, Lai and Mercer 1992) adapted to dependency grammar. In this section, we first define the inside-outside probabilities of complete-links and complete-sequences, and then describe the reestimation algorithm based upon them.

In the following, we use the notations of $\beta'_l$, $\beta'_r$, $\beta''$, and $\beta''_r$ for the inside probabilities of $L_r$, $L_l$, $S_r$, and $S_l$, respectively. Also, $\alpha'_l$, $\alpha'_r$, $\alpha''$, and $\alpha''_r$ are used for the outside probabilities of each of $L_r$, $L_l$, $S_r$, and $S_l$. The superscripts $l$ and $s$ indicate complete-link and complete-sequence, and the subscripts $r$ and $l$ represent the directions rightward and leftward, respectively.

3.1 Inside probabilities: $\beta'_l$, $\beta'_r$, $\beta''$, and $\beta''_r$

The inside probabilities of $L_r(i,j)$, $L_l(i,j)$, $S_r(i,j)$ and $S_l(i,j)$ are the probabilities that the words in the word sequence $w_{i,j}$ are linked together, satisfying the construction
A reestimation algorithm for PDGs

requirements of $L_r, L_l, S_r$ and $S_l$, respectively. In the pictorial views of the inside probabilities accompanying equations (13)–(16), the grey partitions indicate the inside variations on construction of the corresponding sub-complete-links and sub-complete-sequences:

\[
\beta'(i,j) = p(w_{i,j} | L_r(i,j))
\]
\[
= \sum_{m=1}^{j-1} p(w_{i,j} \to w_i) \beta'(i,m) \beta'(m+1,j).
\] (13)

\[
\beta'(i,j) = p(w_{i,j} | L_l(i,j))
\]
\[
= \sum_{m=1}^{j-1} p(w_{i,j} \leftarrow w_i) \beta'(i,m) \beta'(m+1,j).
\] (14)

\[
\beta'(i,j) = p(w_{i,j} | S_r(i,j))
\]
\[
= \sum_{m=1}^{j-1} \beta'(i,m) \beta'(m+1,j).
\] (15)

\[
\beta'(i,j) = p(w_{i,j} | S_l(i,j))
\]
\[
= \sum_{m=1}^{j-1} \beta'(i,m) \beta'(m+1,j).
\] (16)

The basis probabilities are as follows:

\[
\beta'(i,i+1) = p(w_i \to w_{i+1}), \quad (17)
\]
\[
\beta'(i,i+1) = p(w_i \leftarrow w_{i+1}), \quad (18)
\]
\[
\beta'(i,i) = \beta'(i,i) = 1, \quad (19)
\]
\[
\beta'(1,EOS) = p(w_{1,n}). \quad (20)
\]

3.2 Outside probabilities: $\alpha'_r, \alpha'_l, \alpha'_s$ and $\alpha'_t$

The outside probabilities of $L_r(i,j), L_l(i,j), S_r(i,j)$ and $S_l(i,j)$ are the probabilities that the word sequences outside of $w_{i,j}, w_{i,i-1}$ and $w_{j+1,n+1}$ are linked together, satisfying the
requirements of $D$ outside of the $L_r, L_l, S_r$ and $S_l$, respectively. In the figures for the outside probabilities which accompany equations (21)–(24), the grey partitions indicate the inside variations and the light grey partitions indicate the outside variations.

$$\alpha'_r(i,j) = p(w_{i,i-1}, L_r(i,j), w_{j+1,n})$$
$$= \sum_{h=1}^i \beta'_r(h,i) \alpha'_r(h,j). \tag{21}$$

$$\alpha'_l(i,j) = p(w_{i,i-1}, L_l(i,j), w_{j+1,n})$$
$$= \sum_{h=1}^i \beta'_l(h,i) \alpha'_l(h,j). \tag{22}$$

$$\alpha'_s(i,j) = p(w_{i,i-1}, S_r(i,j), w_{j+1,n})$$
$$= \sum_{h=1}^{n+1} [\beta'_s(h,k) \alpha'_s(i,k)$$
$$+ p(w_i \rightarrow w_k) \beta'_s(j+1,k) \alpha'_s(i,k)$$
$$+ p(w_i \leftarrow w_k) \beta'_s(j+1,k) \alpha'_s(i,k)]. \tag{23}$$

Every word except EOS has its head in the word sequence. There is no multiple head, no link cycles and no crossing links.
A reestimation algorithm for PDGs

\[ \alpha_s(i, j) = p(w_{1,i-1}, S(i,j), w_{j+1,n}) \]

\[ = \sum_{k=j+1}^{i-1} \beta_s(j,k) \alpha_s(i,k) \]

\[ + \sum_{h=1}^{i-1} [p(w_h \rightarrow w_i) \beta_i(h,i-1) \alpha_i(h,j) \]

\[ + p(w_i \leftarrow w_j) \beta_i(h, i-1) \alpha_i(h,j)] \]. (24)

The basis probability for the outside probability computations is

\[ \alpha_i(1, EOS) = 1. \] (25)

The inside probabilities (\(\beta'_i, \beta'_l, \beta'_r\) and \(\beta'_s\)) are computed in bottom-up and left-to-right manner in the CYK-chart’s point of view. The outside probabilities (\(\alpha'_i, \alpha'_l, \alpha'_r\) and \(\alpha'_s\)) are computed top-down and right-to-left using the inside probabilities computed in advance (figure 8).

### 3.3 Reestimation

Training proceeds as follows:

1. Initialize dependency grammar. Enumerate all the possible pairs of words in the training corpus and initialize the probabilities of the tentative dependency relations.
2. Compute the initial entropy of the training corpus using the initial grammar.
3. Analyse the training corpus with current grammar, and calculate the occurrence frequency of each dependency relation.
4. Compute new probabilities of dependency relations based on the calculated frequencies

\[ p_{new}(w_x \rightarrow w_y) = \frac{C(w_x \rightarrow w_y)}{\sum_{w_z} C(w_x \rightarrow w_z)} \]. (26)
5. Compute a new entropy of the training corpus using the modified grammar.
6. Repeat steps 3 through 5, while (previous entropy − new entropy) > ε.

Given a training corpus, the initial grammar is just a list of all pairs of unique words in the corpus. The initial pairs represent the tentative head-dependent relations, and the initial probabilities of the pairs can be given randomly. The training starts with the initial grammar. The algorithm can be used to reestimate the probabilistic parameters of an existing dependency grammar. In that case, step 1 is omitted and the existing grammar is regarded as the initial grammar.

The occurrence frequency of each dependency relation, \( C(w_i \rightarrow w_j) \), is calculated as follows. In the following expression, \( O_{\text{cr}}(w_i \rightarrow w_j, \mathcal{D}, w_{1:n}) \) is 1 if the dependency \( w_i \rightarrow w_j \) is used in the \( \mathcal{D} \), and 0 otherwise:

\[
C(w_i \rightarrow w_j) = \sum_{\mathcal{D}} p(\mathcal{D} \mid w_{1:n}) O_{\text{cr}}(w_i \rightarrow w_j, \mathcal{D}, w_{1:n})
\]

\[
= \frac{1}{p(w_{1:n})} \sum_{\mathcal{D}} p(\mathcal{D}, w_{1:n}) O_{\text{cr}}(w_i \rightarrow w_j, \mathcal{D}, w_{1:n})
\]

\[
= \frac{1}{p(w_{1:n})} \sum_{\mathcal{D}} p(\mathcal{D}, w_{1:n}, w_i \rightarrow w_j)
\]

\[
= \frac{1}{p(w_{1:n})} p(w_{1:n}, w_i \rightarrow w_j)
\]

\[
= \frac{1}{p(w_{1:n})} \sum_{i,j=1}^{j-1} p(w_{1:n}, L_i(i,j), S_i(i,m), S_i(m+1,j))
\]

\[
= \frac{1}{p(w_{1:n})} \sum_{i,j=1}^{j-1} p(w_{1:i-1}, w_{i,m}, w_{m+1,i}, w_{j+1:n}, L_i(i,j), S_i(i,m), S_i(m+1,j))
\]

\[
= \frac{1}{p(w_{1:n})} \sum_{i,j=1}^{j-1} p(w_{1:i-1}, L_i(i,j), w_{j+1:n})
\]

\[
p(S_i(i,m), S_i(m+1,j) \mid L_i(i,j))
\]

\[
p(w_{i,m} \mid S_i(i,m))
\]

\[
p(w_{m+1,j} \mid S_i(m+1,j))
\]

\[
= \frac{1}{p(w_{1:n})} \sum_{i,j=1}^{j-1} \alpha_l(i,j) p(w_i \rightarrow w_j) \beta_l(i,m) \beta_l^*(m+1,j)
\]

\[
= \frac{1}{p(w_{1:n})} \alpha_l(i,j) \beta_l(i,j). \quad (27)
\]
Similarly, the occurrence frequency of \( w_i \leftarrow w_j \), \( C(w_i \leftarrow w_j) \) is computed by

\[
\frac{1}{p(w_{1,n})} \alpha(i,j) \beta(i,j).
\]

(28)

The algorithmic complexity of the PDG reestimation algorithm is a function of input size \( n \), the length of the input word sequence(s). The execution time of a program in terms of input parameter \( n \), \( T(n) \), is determined by the most complex parts of the program. The most complex fragment of the reestimation algorithm is the computation parts of the inside and outside probabilities. Inside computation consists of four sequential computations of \( \beta'_i, \beta'_i, \beta'_i \) and \( \beta'_i \). Likewise, the outside computation consists of four sequential computations of \( \alpha', \alpha', \alpha' \) and \( \alpha' \). Thus, the execution time of the algorithm is

\[
T(n) = T_{\beta'_i}(n) + T_{\beta'_i}(n) + T_{\beta'_i}(n) + T_{\beta'_i}(n) + T_{\beta'_i}(n) + T_{\beta'_i}(n) + T_{\beta'_i}(n).
\]

We show the computation times for one of the inside probabilities and one of the outside probabilities, \( T_{\beta'_i}(n) \) and \( T_{\beta'_i}(n) \).

The inside computation of \( L_i \), \( \beta'_i \), is computed for all substrings of \( w_{1,n} \). So the equation of the inside probability of \( L_i \) defined by equation (14) can be shown as

\[
\sum_{i=1}^{n} \sum_{q=1}^{n} \beta'_i(i,m) \beta'_i(m+1,j), \quad j = i+q.
\]

(29)

The outer summation (loop) is executed \( n \) times, and the inner summation is executed \( j-i \) times, i.e. \( q \) times. So the running time, \( T_{\beta'_i}(n) \), can be expressed as follows:

\[
\sum_{i=1}^{n} \sum_{q=1}^{n} q = n \left( \frac{n(n+1)}{2} \right),
\]

(30)

which is \( O(n^2) \). The other inside probabilities, \( \beta'_i, \beta'_i, \beta'_i \) are computed similarly.

The outside computation of \( S_i \), \( \alpha'_i \), is computed for all substrings of \( w_{1,n} \). So the equation of the outside probability of \( S_i \) defined by equation (24) can be shown as

\[
\sum_{i=1}^{n} \sum_{q=1}^{n} \left\{ \sum_{k=1}^{n} \beta'_i(i,k) \alpha'_i(i,k) + \sum_{h=1}^{i-1} \left[ p(w_{h} \leftarrow w_{i}) \beta'_i(h,i-1) \alpha'_i(h,j) \right. \right. \\
\left. \left. + p(w_{h} \leftarrow w_{i}) \beta'_i(h,i-1) \alpha'_i(h,j) \right] \right\}, \quad j = i+q.
\]

(31)

The first inner loop of the left brace (\{) is computed \( n-j+1 \) times, i.e. \( n-i-q+1 \) times, and the second inner loop is computed \( 2(i-1) \) times. So the running time, \( T_{\alpha'_i}(n) \), can be expressed as follows:

\[
\sum_{i=1}^{n} \sum_{q=1}^{n} ((n-i-q+1) + 2(i-1)) = \sum_{i=1}^{n} \sum_{q=1}^{n} (n+i-q-1)
\]

\[
= n^2 + n \sum_{i=1}^{n} i - n \sum_{q=1}^{n} q - n^2
\]

\[
= n^3 - n^2,
\]

(32)
which is also $O(n^3)$. The other outside probabilities, $\alpha'_r$, $\alpha'_l$ and $\alpha'_i$ are computed similarly, so the time complexity of the algorithm is $O(n^3)$.

4 Experiments

We aim at an automatic grammar learning environment in which we can easily acquire a grammar appropriate to a specific domain, given a text corpus of the domain. The proposed reestimation algorithm was developed to be used as the basis of such an automatic learning environment. Thus, we experimented on how effective grammar can be learned from scratch by the algorithm, when there is no pre-existing grammar.

The PDG reestimation algorithm was implemented in the C language, and the experiments were done on a SPARC 1000 workstation. The algorithm was trained and tested on Korean sentences extracted from the KAIST corpora. The experiments were done on parts of speech, not on words, because we could not get a sufficiently large corpus with a limited number of vocabularies. However, if such a corpus is available, the algorithm can be applied in learning lexical dependency grammar without any change to the algorithm.

Here, we describe some characteristics of the Korean language. First, in Korean a word consists of a content morpheme and zero or more functional morphemes which represent the grammatical role of the word. Thus a Korean word is normally associated with a series of parts of speech, one for each corresponding morpheme in the word. For the sake of convenience of the experiment, we assume that the representative part of speech of a word is the part of speech of the last morpheme of the word.

Secondly, Korean is a partially free-ordered language, in which a head word is always placed to the right side of its dependent words. Because of this characteristic, the rightward structures of $S_r$ and $L_r$ are never constructed in a Korean dependency.

---

6 *KAIST* (Korean Advanced Institute of Science and Technology) corpora have been under construction since 1994 under the *Korean Information Base System construction (KIBS)* project. It is supported by the Ministry of Culture and Sports, Korea. The KAIST corpora consist of a raw text corpus (45,000,000 words), a POS tagged corpus (6,750,000 words), and a tree tagged corpus (30,000 sentences), at present. For our experiments, we extracted each train/test set from the POS tagged corpus.
A reestimation algorithm for PDGs

Fig. 10. Training corpus entropy.

structure. Only $S_r$, $L_r$ and null $S_r$ (zero sequence of $L_r$) are taken into account, so an $L_r(i,j)$ is always composed of null $S_r(i,i)$ and $S_r(i+1,j)$, for arbitrary $i$ and $j$ ($1 \leq i < j \leq n$). The abstract dependency structures for Korean are shown in figure 9.

We randomly extracted 3509 training sentences with 40,808 words and 409 test sentences with 4662 words from the portion of the economy domain of the KAIST POS-tagged corpus. The POS-tagged corpus had been annotated using a tag-set of 55 parts of speech. Thus, the initial dependency grammar was constructed by enumerating all possible pairs of parts of speech over the tag-set $\mathcal{F}$, resulting in 3080 dependency rules. The initial probabilities of the dependencies can be given randomly. In this experiment, however, we set them as being equal to eliminate any possible bias in the initial values, assuming uniform distribution of the probabilities:

$$p(x \rightarrow y) = \frac{1}{|\mathcal{F}|}, \text{ thus}$$

$$\sum_y p(x \rightarrow y) = 1.$$  \hspace{1cm} (33)

Figure 10 shows the gradual decreasing in training corpus entropy (bits/word) with training iterations. The proposed reestimation algorithm converged on the (local) minimum entropy at the fifteenth iteration, and thus learned a stable PDG. The same experiments on training texts of a different domain and different size resulted in similar entropy convergences. This may be because the reestimation was done on inter-POS dependencies. Unlike lexical entities, parts of speech are fixed in number, regardless of the training corpus size, and the use of parts of speech is less sensitive to the domain of the training corpus. If the reestimation were to have been done on the dependencies between lexical entities, the convergence of the entropy may have varied, depending on the domain and the size of the training corpus.

To evaluate the parsing accuracy of the learned grammar, we constructed a treebank for the held-out 409 test sentences by hand, and used a best-first parser to
find the most likely parse of each test sentence according to the grammar. As criteria for parsing accuracy, we used the dependency accuracy and bracketing accuracy scores of the parses yielded by the grammar. We define the dependency accuracy as the proportion of correct dependencies in the analyses. To parse a sentence by dependency grammar is to find the correct dependencies between words in the sentence, constructing a $D$ satisfying the linking requirements. Thus, the dependency accuracy score shows the degree of effectiveness of a dependency grammar. The bracketing accuracy (Pereira and Schabes 1992) score is the proportion of phrases in those analyses that does not cross the brackets in the treebank. It is frequently used as a parsing accuracy measure for phrase structure grammars. However, the bracketing accuracy is a subsidiary measure for dependency grammar, because dependency parsing is not precisely to find the correct constituents, but to find the correct dependencies between words. Dependency analysis is closely related to constituents, though, so correct dependencies imply correct constituents. We therefore tested the bracketing accuracy measure, too. The trained grammar resulted in a 62.82\% dependency accuracy and a 56.40\% of bracketing accuracy, as shown in Table 1.

To see the effect of the size of the training corpus on the learning of grammar and the performance of the grammar acquired, we made 36 partial training sets which include the first $s$ sentences in the whole training set, for $s$ ranging from 100–3509

---

### Table 1. Test set evaluation

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of sentences</td>
<td>409</td>
</tr>
<tr>
<td>Average sentence length</td>
<td>11.4</td>
</tr>
<tr>
<td>Range of sentence lengths</td>
<td>3–21</td>
</tr>
<tr>
<td>Dependency accuracy</td>
<td>62.82%</td>
</tr>
<tr>
<td>Bracketing accuracy</td>
<td>56.40%</td>
</tr>
</tbody>
</table>

---

Fig. 11. Test corpus entropy variation with the size of training corpus.
sentences. For all the partial training sets, the initial dependency grammar consisted of 3080 rules (a set of all possible pairs of parts of speech) with equal probabilities. We reestimated the initial grammar for the partial training sets. For each of the 36 learned grammars, we tested the test corpus entropy, parsing accuracy and size of grammar, i.e. the number of acquired dependencies.

Figure 11 shows that the test corpus entropy decreases accordingly as the size of the training corpus increases. Figure 12 shows the average parsing accuracy of the learned grammars in terms of both dependency accuracy and bracketing accuracy. Initially, increasing the size of the training corpus improves the accuracy of grammar, but with more than 1000 training sentences, the accuracy does not steadily improve, even though the test corpus entropy continues to go down. It shows that a slightly lower entropy does not always imply a more accurate grammar. In the case of learning inter-POS dependencies, which is relatively simpler than learning inter-word
dependencies, the algorithm can learn the best grammar from a small-sized training corpus of about 1000 sentences.

Figure 13 shows that the number of automatically acquired dependency rules increases accordingly as the size of the training corpus increases. The number of rules, however, converges on about 1300 rules, which is a reduction of about 59% in the size of the initial grammar. However, as shown in figure 12, the correctness of the grammar learned does not improve as a proportion to the size of grammar. The grammar of about 1100 rules trained from 1000 training sentences showed the best parsing accuracy.

For comparison, we extracted the inter-POS dependency grammar from the manually constructed treebank. The treebank grammar consisted of 303 dependency rules, and showed an average dependency accuracy of 70.69% and a bracketing accuracy of 67.50%, which is a little better than the automatically acquired grammar. The trained grammar has far more rules, and was less correct than the treebank grammar. In spite of the lower accuracy of the learned grammar as compared to the
A reestimation algorithm for PDGs
269

treeland grammar, in cases when there is no available treeland and the construction
of a treeland costs too much, the automatic learning can be an appropriate choice.

Finally, figure 14 shows an example dependency analysis of a Korean sentence and
the corresponding portion of the trained grammar. The analysis was yielded by a best-
first parser according to the grammar. Below each Korean word, its romanization\footnote{The romanization of Korean words conforms to the rules of ISO TC46/SC2.},
its meaning in English, and the representative part of speech are written in order. In
the depiction of grammar, each row represents a dependency rule, where the left-hand
side of the arrow is the head, the right-hand side is the dependent, and the probability
of the dependency is shown in parentheses.

5 Conclusion

In this paper we have proposed a reestimation algorithm applicable to Probabilistic
Dependency Grammar (PDG). The reestimation algorithm is a variation of the
inside-outside algorithm adapted to PDG. The non-constituent objects complete-link
and complete-sequence are defined as basic units for dependency structures, and the
inside-outside probabilities of the objects are developed. The algorithm has $O(n^3)$
time-complexity in the length of the word sequence to be processed, and can be used
for corpus-based stochastic PDG induction. By experiments on Korean, we have
shown that the reestimation algorithm converged to a local minimum, and thus
learned a stable PDG with an average dependency accuracy of 62-82\% for unseen test
sentences.

The experiment can be extended in several ways. First, we would like to extend to
lexical dependency grammar. Though we experimented with the algorithm on
learning inter-POS dependency grammar, dependency grammar can be fully effective
and meaningful when dependency is defined between lexical entities, not between
parts of speech, because parts of speech lose much useful information by generalizing
the specificity of lexical entities. The reestimation algorithm can be applied to training
lexical dependency grammar, without any modifications. In that case, the initial
grammar can be composed of all pairs of words occurring in the same sentence of the
training corpus.

Secondly, since the reestimation algorithm is sensitive to the initial grammar, we
would like to try to compose a better initial grammar from part of speech
information. For example, initially deciding the inter-POS dependencies roughly by
hand and then mapping the information to a POS-tagged training corpus would
construct a better initial lexical dependency grammar than simply enumerating all
possible pairs of words in the training corpus. The rough inter-POS dependencies
need not cover the entire parts of speech; only some portion of them would be helpful.

Thirdly, in the case of Korean, a word can be segmented into two parts – a content
part (a content morpheme) and a functional part (zero or more functional
morphemes). So, dependency between words $w_d$ and $w_h$ can be defined as $C_{R_{C_h}} C_{C_{w_d}^{F_h}}$, rather than as $w_d \rightarrow w_h$, where $C$ and $F$ represent the content and functional parts of
the words, respectively. Instead of regarding the whole word as the lexical entity on which a dependency relation is defined, separating the contentfunctional parts of a word will generate a more generalized and broad coverage lexical dependency grammar with a slight modification in the reestimation algorithm.

Finally, the proposed PDG reestimation algorithm can be applied to language modeling, which is adaptive to the new domain of texts. Given a training corpus of a new domain, the reestimation algorithm can automatically acquire a PDG appropriate to the domain, and based on the PDG, a language modeling module can be easily constructed with a smoothing technique. PDG-based language modeling is expected to have a better performance than \( n \)-gram-based modeling, because compared to \( n \)-gram, which is just a sequence of words, the dependency grammar can represent long-distance dependencies through hierarchical structures from the head-dependent relations between words.

References


