Preference Dependency Grammar (PDG): Sentence Analysis Method Based on Integrated Multilevel Preference and Constraint
選好依存文法：多層の選好／制約知識を統合した文解析方式

Hideki Hirakawa
平川　秀樹

A dissertation
in
Information Science and Technology

Presented to the Faculties of the University of Tokyo in Partial Fulfilment of the Requirements
for
the Degree of Doctor of Information Science and Technology

October 2007

Professor Hiroshi Nakagawa
Supervisor of Dissertation

Abstract

Natural language processing involves very complicated and hard-to-formalize design issues because it has to treat a wide range of different kinds of linguistic knowledge, such as morphological, syntactic, semantic and contextual, with regularity and exceptionality. Various kinds of grammar and knowledge representation frameworks and their processing algorithms are proposed in each linguistic level. Morphological processing is one of the most developed and established technologies in the area of natural language analysis (NLA). Semantic processing needs more future researches. The syntactic layer, which is a bridge to the semantic layer, has been studied intensively for years and various approaches have been proposed in the fields of computational linguistics and linguistics.

Phrase structure syntax (Chomsky, 1956) and dependency syntax (Tesnière, 1969) are two major syntactic theories, and phrase structure and dependency structure are widely used for the syntactic representation of sentences. These structures are considered to show different dimensions of the sentence structure and can be used for compensating each other. However, insufficient efforts have been made for the integrated use of these syntactic structures, especially in dependency analysis research.

This thesis proposes a novel dependency analysis method called "Preference Dependency Grammar (PDG)," which adopts multilevel architecture utilizing the morphological structure, phrase structure, and dependency structure representations. Each of the representations is a kind of packed shared data structure that encompasses all possible sentence interpretations in its interpretation space. This PDG architecture is introduced based on the following design principles obtained through discussions on the NLA framework, which utilizes multilevel linguistic representations with respect to preference and constraint knowledge.

(a) Avoiding over pruning as well as suppressing combinatorial explosion as much as possible
(b) Adopting effective pruning by applying possible constraints in the lower level
(c) Enabling optimum search in the uppermost level to utilize various levels of preference knowledge

PDG is more advantageous than traditional dependency analysis methods in that it can handle POS ambiguities in conjunction with dependency ambiguities and can incorporate more detailed descriptions for both preference and constraint knowledge for the dependency structure. The core technologies of PDG for enabling these features are a new data structure "dependency forest" and a new algorithm "graph branch algorithm," which are the main contributions of this thesis.

The dependency forest is a new packed shared data structure for representing a set of dependency trees with their preference scores. The dependency forest consists of the dependency graph, the constraint matrix and the preference matrix. The multilevel preferences and constraints are integrated into the preference matrix and the constraint matrix of a dependency forest. The dependency forest has a complete and sound mapping to the corresponding phrase structure forest. Because of this feature, the phrase structure grammar (CFG grammar) can function as a filter for the dependency structures for an input sentence, and the POS ambiguities retaining all possible POS sequences can be introduced to dependency analysis. This thesis gives the proof of the completeness and soundness of the dependency forest.

The dependency forest provides a precise definition of a set of dependency trees because the constraint matrix can express co-occurrence restrictions between two arbitrary dependency relations. This flexibility enables PDG to handle non-projective dependencies and the single valence occupation constraint. On the other hand, the preference matrix, which can express preferences for two arbitrary dependency relations, enables the integrated use of tree-local information (preference on dependency relation) and string-local information (preference on word sequence).

This thesis proposes a new search method called the "graph branch algorithm" for dependency forests. This algorithm searches for the best dependency tree with respect to the preference matrix and the constraint matrix based on the branch and bound principle. The DP-based algorithm, widely used in the optimum tree search task, cannot be applied to the dependency forest search due to its high description abilities.

This thesis finally reports the experimental results using a prototype PDG system for examining the various aspects of the PDG framework including the dependency forest, the graph branch algorithm and the effect of the multilevel knowledge integration using the prototype PDG grammar.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Professor Hiroshi Nakagawa for his constant guidance and support in completing this thesis. I am also sincerely grateful to the committee members: Professor Jun-ichi Tsujii, Professor Masato Takeichi, Professor Kokichi Sugihara, Associate Professor Kumiko Tanaka, and Assistant Professor Takashi Ninomiya for providing valuable feedback and suggestions. I would also like to thank Associate Professor Ko Sakai from the School of Institute of Mathematics, University of Tsukuba, for his suggestions on the proof of the completeness and soundness of the dependency forest.

The people in the Toshiba Research and Development Center, Satoshi Kinoshita, Akira Kumano, Kazuhiro Kimura, Kenji Ono, Tatsuya Izuha, Kazuo Sumita, and Ryohei Orihara helped me with their discussions and cooperation. I would like to thank the people in Toshiba Solution Company, Etsuo Ito, Yumiko Yoshimura, and Miwako Shimazu for their help in preparing the grammar, dictionary and data.

I would like to thank Professor Shin-ya Amano of the Shonan Institute of Technology, Professor Hiroyuki Kaji of the University of Sizuoka, Professor Taizo Miyachi of Tokai University, and Yumiko Okada for encouraging this thesis research. I also appreciate the comments from the referees of my related papers submitted to the Transaction of Information Processing Society of Japan and the Journal of Natural Language Processing.

Finally, and most importantly, thanks to my parents and my family for providing their unwavering support during and before this work.

Contents

Chapter 1

Introduction

The final goal of natural language sentence analysis is to benefit human kind by making computers understand the meanings of sentences. The basis of natural language analysis (NLA), i.e., linguistics and computational linguistics, consists of layers of theories such as morphology, syntax, semantics and context. The purpose of sentence analysis is to obtain mapping from an input sentence to a correct interpretation in an appropriate linguistic layer. This is performed by identifying the sentence structure by applying various kinds of linguistic and real world knowledge.

NLA systems adopt the linguistic layer structure explicitly or implicitly. There are various types of applications in each linguistic layer. Information retrieval systems utilize morphological analysis for comparing inflected words. Machine translation systems analyze source sentences to obtain their syntactic and/or semantic representations to transform them to the target language sentences. Dialogue systems require the contextual or intentional structures for the utterance from a user. Interpretations in some linguistic layer are naturally considered to be intermediate structures between the corresponding structures in its lower and upper linguistic layers. Morphological structures bridge the input sentence to the syntactic structures, which bridge them to the semantic structures. The ability of an NLA system is basically determined by the expressive abilities of the sentence interpretation, knowledge description power and quantity adopted by the NLA system. Therefore, the most important and fundamental issues of the NLA system design are what kinds of knowledge in linguistic layers are described and how they should be applied properly.

Syntactic layer has been studied intensively for years and various approaches have been proposed in computational linguistics as well as in linguistics. Phrase structure syntax (Chomsky, 1956) and dependency syntax (Tesnière, 1969) proposed in the same era are two major syntactic theories and the phrase structure and the dependency structure are widely used as syntactic representation for sentences. These structures are considered to show different dimensions of the sentence structure and can be used for compensating each other. However, insufficient efforts have been made for research in this area, especially in computational linguistics.

The goal of this thesis is to discuss the NLA frameworks that utilize multilevel linguistic representations and to propose a novel dependency analysis method that integrates the morphological structure, phrase structure, and dependency structure representations. As described below, the integration of multilevel preference and constraint knowledge is the most basic issues in multilevel NLA system*1 design. The remainder of this chapter describes the traditional approaches for the two major syntactic frameworks and the contributions of this thesis.

1.1 Background to the Research

Phrase structure (or constituency) syntax (Chomsky, 1956) and dependency syntax (Tesnière, 1969) are two major syntactic frameworks in linguistic and computational linguistics; that is, these are two major interpretation description schemes (or data structures) for representing the syntactic structures of sentences. This section describes the phrase structure and dependency structure schemes and the traditional approaches for integrating these two representation schemes.

1.1.1 Phrase Structure and Dependency Structure

Phrase structure grammars describe the structure of a sentence in terms of constituency relations on the words of the phrases of the sentence. Each word in the sentence has its POS (part of speech). Phrases are represented as a sequence of POSs or phrasal labels (non-terminal labels or symbols) each of which defines a set of possible sequence of phrases. The set of the phrase structure relations that can be defined on a sentence forms a tree, known as the phrase structure tree.

Dependency grammars describe the structure of a sentence in terms of binary head-modifier (also known as dependency) relations on the words of the sentence. A dependency relation is an asymmetric relation between a word called the governor (head, parent) and a word called the dependent (modifier, daughter). A word in the sentence can play the role of the governor in several dependency relations, i.e., it can have several dependents; however, each word can play the role of the modifier exactly once in a majority of dependency grammar frameworks. One particular word does not play the role of the modifier in any relation, and this is named the root. The set of the dependency relations that can be defined on a sentence form a tree, known as the dependency tree (Lombardo and Lesmo, 1996).

Fig.1.1 shows the phrase structure tree and dependency tree for the sentence "Time flies like an arrow." The phrase structure explicitly represents phrases (nonterminal nodes), structural categories (nonterminal labels), and possibly some functional categories (grammatical functions). On the other hand, the dependency structure represents head-dependent relations (directed arcs*2), functional categories (arc labels), and possibly some structural categories (POS) (Nivre and Sandra, 2006). The phrase structure follows a horizontal organization principle: it combines

*1 NLA system with more than one sentence interpretation data structures.

*2 There are two conventions to represent the direction of dependency relations. The source and the target of an arrow shows the dependent node and the governor node, respectively, in this thesis.

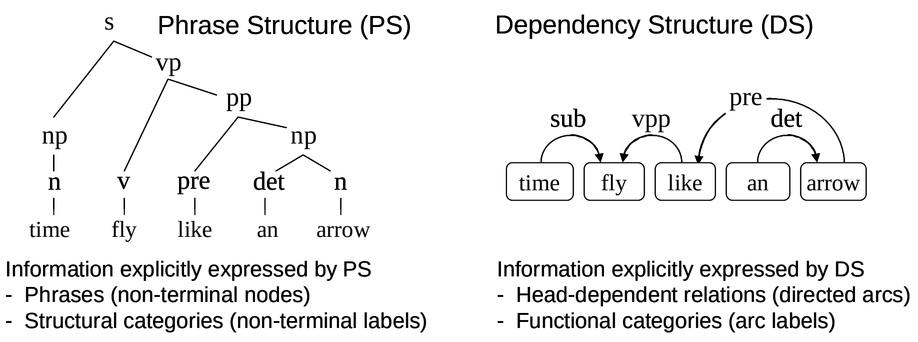

Phrase Structure (PS)
s
vp
pp
np
np
n
v
pre
det
n
time
fly
like
an
arrow
Information explicitly expressed by PS
- Phrases (non-terminal nodes)
- Structural categories (non-terminal labels)
Dependency Structure (DS)
sub
vpp
pre
det
time
fly
like
an
arrow
Information explicitly expressed by DS
- Head-dependent relations (directed arcs)
- Functional categories (arc labels)

Fig.1.1 Phrase structure and dependency structure

the constituents into phrases (larger structures) until the entire sentence is formed. On the other hand, dependency is an asymmetrical relation between a head and a dependent, i.e., it follows the vertical organization principle (Kruijff, 2001).

Context free grammar (CFG) has been studied in depth and adopted as the computational basis of the phrase structure scheme. The context free grammar G is formally defined by the following four components.

$G =< V_t, V_n, P, S >$

V_t : finite set of terminal symbols

V_n : finite set of nonterminal symbols

P : finite set of rewriting rules

S : finite set of start symbols ($S \subset V_n$)

On the other hand, there is no established standard for the formal representation of the dependency grammar framework. This thesis categorizes the existing dependency grammar frameworks into three dependency models, i.e., the Tesniere model, the single dependency model, and the lexical rule model[*3].

(1) Tesniere model

The Tesniere model of dependency grammar is a formal grammar framework (Gaifman, 1965; Hays, 1964) based on the grammatical theory known as dependency grammar (DG), which was proposed by the French linguist Tesniere (Tesnière, 1969). Researches on parsing algorithms (Lai and Huang, 1994; Lombardo and Lesmo, 1996; Courtin and Genthial, 1998; Lombardo and Lesmo, 1998) and the analysis of the grammatical equivalence between CFG and DG (Gaifman, 1965; Abney, 1994) have been conducted based on this model.

The dependency grammar G of the Tesniere model is defined as follows (Lombardo and Lesmo, 1996):

$G =< S, C, W, L, T >$

[*3] These are not generally established terms.

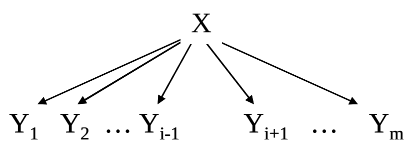


Fig.1.2 Dependency rule in the Tesniere model

W	: a finite set of symbols (vocabulary of words in a natural language)
C	: a set of syntactic categories (preterminals, in constituency terms)
S	: a non-empty set of root categories ($C \supseteq S$)
L	: a set of category assignment rules of the form $X : x$, where $X \in C$, $x \in W$
T	: a set of dependency rules of the form $X(Y_1\ Y_2\ ...\ Y_{i-1}\ \#\ Y_{i+l}\ ...\ Y_m)$ where $X \in C$, $Y_1 \in C$, ... $Y_m \in C$, and # is a special symbol that does not belong to C.

A tree resulting from the dependency rules is essentially an ordered tree of depth one, wherein the nodes are labeled as shown in Fig.1.2. A dependency rule defines the simultaneous existence of multiple dependency relations in order. This is somewhat similar to a CFG rewriting rule that defines the simultaneous existence of multiple phrases or words in order.

The dependency tree for a sentence $x(= a_1 a_2 .. a_p \in W*)$ should satisfy the following conditions[*4]:

(a) The nodes are the symbols $a_i \in W (l \leq i \leq p)$.
(b) The tree has to be covered by a proper set of grammar rules.
(c) The tree satisfies the projectivity condition[*5] with respect to the order in x.
(d) The root is a unique symbol as such that $A_s : a_s \in L$ and $A_s \in S$.

(2) Single dependency model

The single dependency model is basically an analytic grammar model that generates dependency trees for a given sentence. This model covers many dependency parsers such as "kakari-uke"[*6] analyzers (Yoshida, 1972; Shudo et al., 1980; Hitaka and Yoshida, 1980; Ozeki, 1986; Nakagawa and Ito, 1987; Matsunaga and Kohda, 1988; Hirakawa and Amano, 1989a; Kurohashi and Nagao, 1994; Ozeki, 1994; Hirakawa, 2001; Kudo and Matsumoto, 2005), dependency parsers (Covington, 1990; Kubon, 2001; Yamada and Matsumoto, 2003; Nivre and Scholz, 2004; McDonald et al., 2005), and CDG (constraint dependency grammar) parsers (Maruyama, 1990; Harper et al., 1999; Wang and Harper, 2004).

In this model, dependency grammar is defined by two components, i.e., a set of dependency

[*4] Refer to (Lombardo and Lesmo, 1996) for the detailed formal definition

[*5] The projectivity condition consists of two conditions, i.e., "no cross dependency exits" and "no dependency covers the top node" (Mel'cuk, 1988). The second condition is unnecessary when a special root node is introduced at the top or end of a sentence. Dependency structures which violate the projectivity condition are called "non-projective" structures.

[*6] Kakari-uke is a type of dependency relation. The details are explained in Section 1.3.

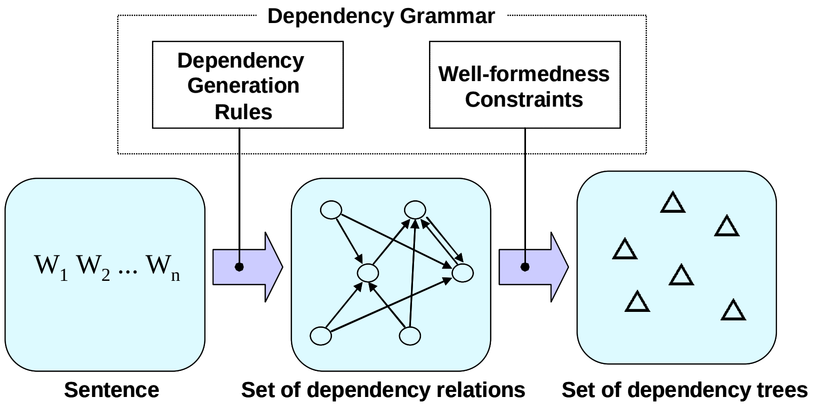

Dependency Grammar
Dependency Generation Rules
Well-formedness Constraints
W₁ W₂ ... Wₙ
Sentence
Set of dependency relations
Set of dependency trees

Fig.1.3 Single dependency model

generation rules G, which generates a set of dependency relations between two nodes (words) and a set of well-formedness constraints C, which defines well-formed dependency trees. As shown in Fig.1.3, a set of dependency relation arcs are obtained by applying G to an input sentence (word sequence $W_1, W_2, ...W_n$). A set of possible dependency trees for the sentence is defined as a set of dependency trees such that each consists of the subset of the generated dependency relation arcs and satisfies the well-formedness constraints C. This framework can distinguish sentences from non-sentences with respect to the grammar (G and C). It can also generate all possible grammatical sentences and their grammatical structures by combining a module that generates all possible word sequences. Therefore, the single dependency model is a type of grammar for languages.

G can be defined as a function that returns a set of dependency pieces for the given two words. Here, "dependency piece" is defined as a triple $< DN, GN, A >$, where DN is a dependent node, GN is a governor node, and A is an arc between the two nodes. This is represented in the form "$DN \xrightarrow{A} GN$." There are several types of functions according to the types of nodes and arcs. For example, one type of function may produce simple dependency pieces such as "time $\longrightarrow$ fly" and "time $\longleftarrow$ fly" from the words "time" and "fly." Another type of function may return dependency pieces such as "time/n $\xrightarrow{\text{subj}}$ fly/v," "time/n $\xrightarrow{\text{nmod}}$ fly/n" and "time/v $\xleftarrow{\text{obj}}$ fly/n," where *subj*, *obj*, and *nmod* represent subject, object, and nominal modification relations, respectively. Chapter 4 provides a more detailed discussion on the node and arc types for dependency structures.

The definition of the well-formedness constraints C prescribes various types of dependency grammars. The most well-known well-formedness constraints are the axioms of the well-formedness of the dependency structure, as defined by Robinson (1970).

(a) One and only one element is independent.
(b) All others depend directly on some element.
(c) No element depends directly on more than one other. (unique head)

(d) If element A depends directly on element B and some element C intervenes between them (in linear order of string), then C depends directly on A or on B or some other intervening element. (projectivity)

Some different versions of dependency structures are obtained by changing the well-formedness conditions. As described in (Kruijff, 2002), if the "unique head" constraint defined above is relaxed, the dependency structures form graphs instead of trees. This type of dependency grammar allows dependents to have multiple heads (Johnson et al., 1985; Hudson, 1984; Hudson, 1991). Relaxing the "projectivity constraint" leads to a non-projective dependency grammar (Covington, 1990; Kubon, 2001; McDonald et al., 2005). These types of general constraints are insufficient to define a proper set of sentences of some natural language. CDG allows arbitrary unary and binary constraints for describing detailed well-formedness constraints.

(3) Lexical rule model

The lexical rule model is a dependency grammar framework where the dependency structure is constructed by combining the partial dependency patterns defined in the lexicons. Nasr (2000) proposed a dependency parsing algorithm combining the partial dependency trees corresponding to the words in a sentence using a graph stack mechanism. Mertens (2002) proposed a chart-parser-based method for constructing the dependency structure for a sentence by combining the basic partial dependency structures in lexicons. Link grammar constructs dependency structures based on the partial connection patterns defined in lexicons (Sleator and Temperley, 1991; Grinberg et al., 1995; Lafferty et al., 1992)[*7].

1.1.2 Relation between Phrase Structure and Dependency Structure

It is sometimes pointed out that the merit of the dependency syntax over the phrase structure is that the dependency structure has the immediate mapping on the predicate-arguments structures, i.e., the semantic structures needed for the next stage of interpretation (Sgall et al., 1986; Mel'cuk, 1988; Hudson, 1991) and is not necessary to "read off" head-modifier or head-complement relations from a tree (Covington, 1990). On the other hand, the phrase structure syntax can express the construction rules related to the word or phrasal order naturally, which is not explicitly represented by the dependency relation.

The phrase and dependency structures are not competing representations; instead, they describe different aspects of the sentence structures (Kruijff, 2002; Nivre and Sandra, 2006). Kruijff mentioned that "A phrase-structure tree is closely related to a derivation, whereas a dependency tree rather describes the product of a process of derivation. Usually, given a phrase-structure tree, we can get very close to a dependency tree by constructing the transitive collapse of headed

[*7] Link grammar is not considered as an instance of dependency grammar by its creators, and it departs from the traditional view of dependency by using undirected links; however, the representations used in link grammar parsing are similar to the dependency representations in that they consist of words linked by binary relations (Nivre, 2005).

structures over non-terminals." Further, "Constituency and dependency are not adversaries, they are complementary notions. Using them together we can overcome the problems that each notion has individually." From the linguistic viewpoint, Kodama (1987) discussed the linguistic information required for obtaining the sentence interpretation in the context of dependency grammars and positioned the dependency structure as a bridge for combining or integrating the syntactic information and the semantic information.

A sentence has a set of possible syntactic interpretations in general, and consequently has a set of corresponding phrase structure interpretations (trees) and the dependency structure interpretations (trees). If phrase structure trees and dependency structure trees for a sentence are different representations for the syntactic interpretations of the sentence, there should be consistent correspondences between these two different kinds of syntactic trees. Since syntactic grammars define the syntactic structures of a sentence, there should be some consistent mapping between phrase structure grammar and the dependency grammar if both of them define the syntactic structures of sentences.

Gaifman (1965) studied the equivalence between CFG and the Tesniere model DG. As shown in Section 1.1.1, the grammar rule formalism of the Tesniere model DG is similar to that of CFG. There are two types of equivalence relations defined between the two grammars. The two grammars are called "weakly equivalent" if the set of strings defined by them are equivalent. They are called "strongly equivalent" if the set of sentence structures generated by them are equivalent. A definition for the equivalence between the sentence structures of the two grammars is necessary for checking the strong equivalence between the two grammars. Gaifman adopted the concept of "ramification" to check the equivalence between a phrase structure tree and a dependency tree. Ramification is a parenthesized structure that represents information that is similar to the phrase boundary. Procedures are outlined for obtaining the ramification from a phrase structure tree and a dependency tree and checking their equivalence. Gaifman proved that CFG and DG were weakly equivalent, i.e., there exists a DG that is weakly equivalent to a given CFG and vice versa. On the other hand, there exists a CFG that is strongly equivalent to a given DG; however, the inverse has not been proven to be true. Although a detailed explanation is not provided here, the condition for CFG to be strongly equivalent to DG is that "a phrase structure system[*8] is equivalent to some d-system[*9] iff its degree is 0 or 1" (Gaifman, 1965). A CFG grammar with a recursive derivation has an infinite degree. This condition is very strong and Gaifman's discussion disproved the strong equivalence between CFG and DG.

Abney (1994) pointed out a problem in Gaifman's framework and studied the equivalence between CFG and DG using a revised framework. Gaifman's mapping method for obtaining the ramification from a dependency tree may produce multiple results due to the lack of information. To resolve this mapping ambiguity, Abney assumed that the heads of phrases were predetermined. Abney assumed a headed CFG (HCFG) and then discussed the equivalence be-

[*8] This is equivalent to CFG

[*9] This implies the Tesniere model DG

tween CFG and DG derived from this HCFG based on Gaifman's framework. The derived CFG is known as a "characteristic grammar" and the derived DG is known as a "projection grammar." The result shows that the characteristic and projection grammars are not equivalent, i.e., there exist HCFGs that have equivalent characteristic grammars and different projection grammars, and inversely, there exist HCFGs that have equivalent projection grammars and different characteristic grammars.

As described in Nivre (2005), these results on the equivalence between CFG and DG have been mentioned to explain the relative lack of interest in dependency grammars within natural language processing. If the strong equivalence between CFG and DG is disproved, a complete formal mapping between the phrase and dependency structures of sentences cannot be constructed. Discussions by Gaifman and Abney have at least two important premises. First, the discussed dependency grammar is limited to the Tesniere model DG. Other dependency grammar frameworks are not discussed. Second, as already mentioned by Gaifman (1965), the criterion for the equivalence between the phrase structure tree and dependency tree, i.e., ramification, is natural for the former but not for the latter. The criteria for the equivalence between these two structures should be a basic and important issue in discussing the equivalence between CFG and DG.

The discussion on the equivalence or correspondence between the phrase structure grammar and dependency grammar does not fall within the scope of this thesis; however, this thesis presents a method for creating not one-to-one but consistent correspondences between a set of phrase structure trees and dependency trees for a sentence as described in Chapter 3.

1.2 Phrase Structure Analysis

As described in Section 1.1.1, CFG is established as a basis of phrase structure grammar to obtain the phrase structures for a sentence. Efficient CFG parsing algorithms such as CKY, Early, Chart, and LR algorithms are widely used. In the 1980s, the framework for attaching arbitrary processing codes to CFG grammar rules was developed on the basis of the logic programming language Prolog (Colmerauer et al., 1973; Clocksin and Mellish, 1984), such as DCG (Definite Clause Grammar) (Pereira and Warren, 1980), and BUP (Bottom Up Parser) (Matsumoto et al., 1983). This mechanism enables a more detailed grammar description by introducing extra constraints referring to various kinds of grammatical and/or semantic features, and structure building function (Dahl and McCord, 1983). The unification operation[*10] in Prolog played an important role in grammar description. In conjunction with the unification framework, linguistic investigations resulted in new grammar frameworks, such as FUG (Functional Unification Grammar) (Kay, 1984), LFG (Lexical Functional Grammar) (Kaplan, 1989; Riezler et al., 2002), PATR-II (Shieber et al., 1983), GPSG (Generalized Phrase Structure Grammar) (Gazdar et al.,

[*10] Operation to make equivalent two terms with or without variables by assigning appropriate values to the variables, or operation attempting to make a one-time assignment of contents to the variables for a set of logical equations.

1985), HPSG (Head-driven Phrase Structure Grammar) (Pollard and Sag, 1994; Tsuruoka et al., 2004), and CCG (Combinatory Categorical Grammar) (Steedman, 2000; Clark and Curran, 2003)*[11]. These are called unification grammars or lexical unification grammars because they introduce lexical information such as linguistic features and subcategorization information. A set of equations that represent linguistic structure and/or constraints are generated from a phrase structure tree for a sentence. The interpretation of a sentence is well-formed (or grammatical) if and only if these equations have proper variable assignments. Unification grammars provide much more detailed and lexicalized linguistic constraints compared with the skeleton CFG framework. Unification grammar parsers are called deep parsers because they generate deep and full sentence structures.

The elaboration of grammar rules provides more opportunities to obtain correct sentence interpretations. However, this is not sufficient because natural language sentences generally have plausible well-formed interpretations as well as implausible interpretations. Ambiguity resolution is indispensable for obtaining the most plausible interpretation from grammatical interpretations. Disambiguation is performed by assigning a preference degree for each of the available interpretations and choosing the best one among them. The knowledge assigning this preference degree is called preference knowledge. Intensive studies on the disambiguation method utilizing statistics from corpora began from the late 1980s to 1990s. The so-called corpus-oriented methods provide a disambiguation mechanism by means of three components, i.e., a statistical model that defines the plausibility of a sentence interpretation, a method for learning parameters from corpora and a method for decoding (or computing) the best interpretation for a sentence from among its possible interpretations. PCFG (Probabilistic CFG) is proposed for a CFG framework (Jelinek et al., 1992). PCFG consists of the probabilistic model based on the probabilities of CFG rules that are obtained by the inside/outside algorithm and the algorithm similar to the Biterbi algorithm for computing the most plausible phrase structure tree in the parse forest of an input sentence.

One significant improvement on the corpus based method is obtained by introducing the lexical information to the probabilistic model of the PCFG (Carroll and Charniak, 1992; Eisner, 1996a; Charniak, 1995; Charniak, 1997; Collins, 1999; Charniak, 2000; Bikel, 2004). Such a method is known as the lexicalized PCFG. The head of phrase (or phrase head)*[12] plays an important role in lexicalized PCFGs. Charniak (1995) reported the significant improvement of parse accuracy by introducing head information such as POS of head, parent's head, grandparent's head, and rule selection by head information into the probabilistic model. Collins (1999) introduced history-based lexicalized CFG (Head-Driven Statistical Model) based on the so-called history-based parsing method (Black et al., 1992) and proposed a bottom-up chart parser based on some probabilistic models. Based on this method, Bikel (2004) analyzed that lexical information such as lexical relations and sub-categorization information were effective for improving the parsing

---

*[11] CCG is not CFG but has a close relation to CFG.

*[12] "the head of phrase" is defined as "an element with X category in X bar theory" (Chomsky, 1970) or "the element that determines the syntactic function of the whole phrase" or simply "most important word of phrase"

accuracy.

Research on feature structure grammars (Abney, 1997) promoted the researches on lexical stochastic unification grammars such as HPSG (Oepen et al., 2002; Toutanova and Manning, 2002), CCG (Clark and Curran, 2003), and LFG (Johnson et al., 1999; Riezler et al., 2002; Kaplan et al., 2004). Lexical dependency information is also utilized as features of the maximum entropy model (Bouma et al., 2001).

One approach for utilizing the dependency information in phrase structure analysis is to utilize the output from some independent dependency analyzer. Sagae et al. (2007) incorporates the output from a shallow dependency parser as a hard dependency constraint or soft dependency constraint to improve the accuracy of the target HPSG deep parser. Mapping between the phrase structure and dependency structure is obtained through an intermediate HPSG structure.

As shown above, the CFG-based approach has achieved higher sentence accuracy by introducing frameworks for more precise constraint knowledge and sophisticated preference knowledge. Lexical relations including the dependency relation are widely introduced to lexicalized PCFG and improved parsing accuracy (Bikel, 2004). Recent studies on phrase structure oriented parsing systems (Bouma et al., 2001; Charniak and Johnson, 2005; Sagae et al., 2007) show the tendency for utilizing lexical dependency relations for improving the parsing accuracy.

1.3 Dependency Structure Analysis

Although the Tesniere model dependency grammar (Tesnière, 1969; Gaifman, 1965; Hays, 1964) was proposed as a formal grammar framework in the 1960s, researches on dependency analysis systems for Tesniere model were conducted relatively recently (Lai and Huang, 1994; Lombardo and Lesmo, 1996; Courtin and Genthial, 1998; Lombardo and Lesmo, 1998). A considerably greater number of studies have been conducted within the framework of the single dependency model. In particular, Japanese grammar and the Japanese analysis system based on kakari-uke grammar has been studied (Hashimoto, 1946; Yoshida, 1972; Shudo et al., 1980; Hitaka and Yoshida, 1980; Nakagawa and Ito, 1987; Matsunaga and Kohda, 1988), where a sentence structure is represented by a set of kakari-uke (dependency) relations between two linguistic units called “bunsetsu,” which is a sequence of morphemes containing at least one contents word. Kakari-uke grammar has a well-formedness axiom: the “dependent always locates to the left of its governor (no backward dependency).” Kakari-uke grammar is a kind of dependency grammar with this axiom peculiar to Japanese language in addition to the axioms by Robinson (1970). Kakari-uke parsing algorithms including the stack-based algorithm and DP based algorithm are proposed (Shudo et al., 1980; Hitaka and Yoshida, 1980; Nakagawa and Ito, 1987; Matsunaga and Kohda, 1988; Ozeki, 1986; Ozeki, 1994; Kurohashi and Nagao, 1994). Katoh and Ehara (1989) proposed a DP-based dependency parsing algorithm allowing backward dependency, i.e., an algorithm for general dependency grammar with Robinson’s axiom, by extending the algorithm proposed by Ozeki (1986).

As described in Section 1.1.1, CDG is a kind of single dependency model grammar. Constraints dependency grammar G, which determines a set of possible assignments of a given sentence, is formally defined by the following four components (Maruyama, 1990).

$G =< \Sigma, R, L, C >$
Σ : finite set of terminal symbols
R : finite set of role-ids
L : finite set of labels
C : constraint that an assignment A should satisfy

C is a set of arbitrary unary and binary constraints for describing detailed well-formedness constraints (Maruyama, 1990; Harper et al., 1999). CDG adopts the eliminative parsing method where sentence analysis is defined as a constraint satisfaction problem (CSP) for all possible interpretations of a sentence[*13]. CDG generates a dependence graph which encompasses all possible dependency trees by assuming all dependency relations between every two nodes (or words) in an input sentence. The constraints in C are propagated over the network by the constraint propagation mechanism (Waltz, 1975; Montanari, 1976) to eliminate ill-formed dependency interpretations. The original CDG parser (Maruyama, 1990) is extended to support the simultaneous analysis of sentences with multiple alternative lexical categories (POS ambiguity) and features (Harper and Helzerman, 1995).

The treatment of preference knowledge in dependency analysis, as well as in phrase structure analysis, is studied in both the heuristic approach (Bouma et al., 2001; Hirakawa, 2001) and corpus-based approach (Carroll and Charniak, 1992; Collins, 1996; Eisner, 1996b; Eisner, 1996c; Lee and Choi, 1997). Eisner (1996b) proposed a dependency parsing algorithm which analyses a whole sentence as a non-constituent span based on the DP algorithm similar to the CKY parsing algorithm and Eisner (1996c) examined four probabilistic models[*14]. Eisner's third model (Model C called the "generative model" or "edge factored model") defines the probability of a dependency tree based on the probabilities of dependency arcs in the tree corresponds to the preference priority and is widely used in the single dependency model. Lee and Choi (1997) presented an unsupervised learning method based on the inside-outside algorithm and a decoding method similar to Eisner's parsing algorithm. As is the case in the probabilistic CFG research field, maximum entropy models for dependency parsing are proposed (Stolcke et al., 1997; Chelba et al., 1997). Moreover, the probabilistic model is introduced in the CDG framework (Wang and Harper, 2004).

Recently, intensive researches on dependency analysis have been conducted on the data driven dependency parsing framework, and the Conference on Computational Natural Language Learning (CoNLL) 2007 has been devoted to dependency parsing. In this CoNLL-X shared task on dependency parsing, there are two dominant models for data-driven dependency parsing (Buchholz and Marsi, 2006; McDonald and Nivre, 2007). The first is the "all-pairs" approach in which

[*13] The parsing method that generates possible interpretations in each linguistic level in a step-by-step manner is called the generative parsing method.

[*14] More detailed explanation are given later in this section

every possible arc is considered in the construction of the optimal parse. The MSTParser (Maximum Spanning Tree parser) (McDonald et al., 2005), which searches the optimum tree from the dependency graph that encompasses all possible dependency trees for one WPP[*15] sequence for a sentence, is a typical example of the all-pairs approach. The second is the "stepwise" approach or "history-based" approach (Black et al., 1992), where the optimal parse is built stepwise depending on the previous decisions in parsing process. The Yamada-Matsumoto parser (Yamada and Matsumoto, 2003) and the MaltParser (Nivre and Scholz, 2004) are typical examples of the stepwise approach. These two approaches adopt the discriminative learning method.

All-pairs parsers can learn the features of the global sentence structure and excel in long sentence analysis. On the other hand, stepwise parsers can learn richer local features compared with the all-pairs parser and excels in shorter sentence analysis. The result of the CoNLL-X shared task shows almost the same sentence analysis accuracies for these different types of dependency parsers (McDonald and Nivre, 2007).

The multi-agent method obtains a better output by utilizing or combining the multiple outputs from the different types of agents. This idea is applicable to sentence analysis for improving the parsing accuracy (Inui and Inui, 2000; Zeman and Žabokrtský, 2005). Sagae and Lavie (2006) proposed a new parser ensemble method for dependency parsing where outputs from some dependency parsers are decomposed into their constituents and the best well-formed dependency tree is searched from the set of decomposed constituents. This method is examined using a single dependency parser with some different set-ups (Sagae and Tsujii, 2007).

As shown in Section 1.2, information from a dependency relation is widely utilized mainly as a preference source. In contrast, the phrase structure information is not widely utilized in dependency parsers. As described in Section 1.1.1, phrase structure and dependency structure are two major data structures for representing different aspects of the syntactic structure of a sentence and are expected to be used for compensating each other. However, it is not clear how and for what purpose the phrase structure should be used in dependency analysis. To clarify this matter, some problems in current dependency analysis methods are discussed below.

The first problem is related to the size of the possible dependency tree space to search. Two popular parsers, i.e., the MSTParser and the MaltParser, accept a sequence of words with POS tags as their input. The disambiguation of POS ambiguity is left for the task for some tagger. This poses a problem because the disambiguation errors in the tagging process cannot be solved by improving the ability of a dependency parser (Yamada and Matsumoto, 2003). On the other hand, a CDG parser generates inherent dependency trees for the first step by performing possible role value assignments, and then a set of constraints are applied to these role values to eliminate ungrammatical assignments. This approach causes poor parsing efficiency due to the size of possible interpretation space. Optimization methods such as the enhanced pruning method based on modifier and modifiee features and the role assignment restriction based on

[*15] WPP is a pair of a word and a part of speech (POS). The word "time" has WPPs such as "time/n" and "time/v." A compound word can be one WPP such as "flying saucer/n" which corresponds to two input words (or positions).

grammar and corpus information are introduced for this problem (Harper et al., 1999; Harper et al., 2000). However, a problem persists for the all-pair parsing approach because the introduction of POS ambiguity causes a magnification of the search space. In addition to this space problem, the introduction of POS ambiguity poses another crucial problem to the MSTParser. One of the parsing algorithms adopted in MSTParser for applying non-projective dependency analysis, i.e., Chu-Liu Edmonds algorithm, has the assumption that a well-formed dependency tree is a spanning tree of the dependency graph. This algorithm is not applicable to the dependency graphs containing multiple nodes for one word. This kind of graph is called "single-node graph" in this thesis[*16]

The second issue is about a description power of the single dependency model. Since the output of a sentence analysis system is prescribed by its preference knowledge and constraint knowledge, the potential ability of the sentence analysis system is prescribed by the description abilities for these two kinds of knowledge. As far as the preference knowledge model for the dependency structures is concerned, Eisner (1996c) proposed and examined four probabilistic models, i.e., bigram lexical affinities (model A), selectional preferences (model B), recursive generation or edge factored model (model C) and realistic selectional preferences (model D). The majority of dependency parsers based on the single dependency model adopt the edge factored model. However, the selectional preference and the realistic selectional preference models outperformed the edge factored model and the integrated use of tree-local information (preference on dependency relation), and string-local information (preference on word sequence) results in better parsing accuracy (Eisner, 1996c). This suggests that the dependency analyzer with the edge factored preference model can achieve more accuracy by introducing a more precise preference model with word sequence preference. On the other hand, the constraint description ability for the single dependency model is not sufficient in some cases. For example, the MSTParser can handle non-projective and projective parsing by switching two parsing algorithms (McDonald et al., 2005), i.e., the Chu-Liu-Edmonds maximum spanning tree algorithm (Chu and Liu, 1965; Edmonds, 1967) and the Eisner's algorithm (Eisner, 1996b). This implies that the well-formedness constraint for the dependency tree is bound to the algorithms. It is difficult to give the system a more detailed constraint for prescribing the well-formed non-projective dependency trees. The enhancement of the descriptive power of the constraint knowledge is one solution to this problem.

1.4 Integrated Use of Phrase Structure and Dependency Structure

This thesis investigates the idea of the integrated use of the phrase and dependency structures. This integration requires mapping between these two structures of a sentence. This is because sentence analyzers cannot combine any linguistic information without correspondence

[*16] The dependency graph that has nodes representing multiple roles for each of the input words is called "multiple-node graph" in this thesis.

between the two structures. The following shows some traditional approaches for constructing this mapping.

(1) Conversion from/to phrase structure to/from dependency structure

Collins (1999) presented a method for converting a phrase structure tree to a dependency tree. The corresponding dependency tree is uniquely generated from a headed phrase structure tree, as shown in Fig.1.4. The head of a phrase is determined by using heuristic rules based on nonterminal symbols, POS information, etc. The dependency relation is unlabeled and comprises four elements, i.e., parent node label, head daughter label, non-head daughter label, and non-head daughter direction. The corresponding dependency tree is automatically obtained by defining the head of each phrase structure node. The generated dependency tree reflects the structure of the original phrase structure tree, which indicates the simplicity of the mapping. The dependency trees obtained from the phrase structure trees are mainly used for evaluating the accuracies of phrase structure parsers (Clark and Curran, 2004). The evaluation methods based on the dependency structure are considered to be more stable and reliable as compared to those directly based on the phrase structure, because the former methods are dependent only on the word information (system independent) and not on the phrase boundaries and phrase categories (system dependent), as described in 6.1.1.

Xia and Palmer (2000) presented the following three methods for converting dependency structures into phrase structures; the X-bar-theory-based method, Collins' method, and another heuristic method. The conversion is performed in order to build the dependency structure annotated corpora from the phrase structure annotated corpora.

These structure conversion methods basically provide a mapping between the phrase and dependency structures by adopting some heuristics along with the advantages afforded by converted structures, such as the evaluation of parsing systems and the construction of different types of corpora. These conversion methods are not intended for the integrated use of the phrase and dependency structures in the sentence analysis process.

Rambow and Joshi (1995) studied the relation among three grammar formalisms, namely, CFG, TSG (tree substitution grammar), and DG*[17], from the viewpoint of the main factors of grammar formalism, i.e., elementary structures and combining operators. Rambow showed that the process of lexicalizing CFG naturally led to a TAG (tree adjoining grammar), and the

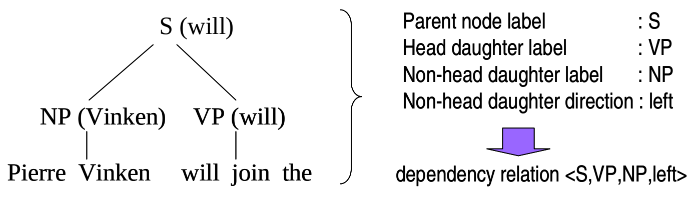


Fig.1.4 Collins' dependency tree

*[17] TAG (tree adjoining grammar) and MTT (meaning-text theory) are mentioned.

derivation trees generated in parallel with the phrase structure trees of the TAG analysis were the dependency trees that closely resemble those of MTT (meaning-text theory) (Mel'cuk, 1988; Wanner, 1994; Kahane, 2003). A derivation tree is constructed algorithmically by combining the lexical nodes corresponding to two phrase structure trees, t_1 and t_2, which are adjoined in the TAG analysis process. In some situations, derivation trees exhibit inconsistencies in the directions of dependencies with the MTT dependency trees as described by "... while tree t_1 is adjoined into t_2, but the lexemic element of t_2 depends on that of t_1. Thus, while adjunction corresponds to the establishment of a syntactic dependency relation, the direction of the relation cannot be determined from the direction of the adjunction alone." This method has an advantage in that it can automatically generate a dependency structure, without providing any additional information about the mapping between the phrase and dependency structures. However, this feature also leads to the generation of unnatural dependency structures, as described above.

(2) Partial Structure Mapping

The rewriting rule of CFG represents a part of the phrase structure, i.e., the partial tree. Seo and Simmons (1989) proposed a framework for mapping the phrase structure trees and dependency trees based on a set of rules. Each rule defines a headed CFG rewriting rule (partial phrase structure tree) and a mapping to the partial dependency tree. The nodes in a partial dependency tree are linked to the heads of constituents in the corresponding phrase structure rule. In this thesis, this mapping method is called the "partial structure mapping" method. Fig.1.5 shows the overall mapping framework based on the partial structure mapping method proposed in Seo and Simmons (1989). An extended CFG parser analyzes an input sentence and

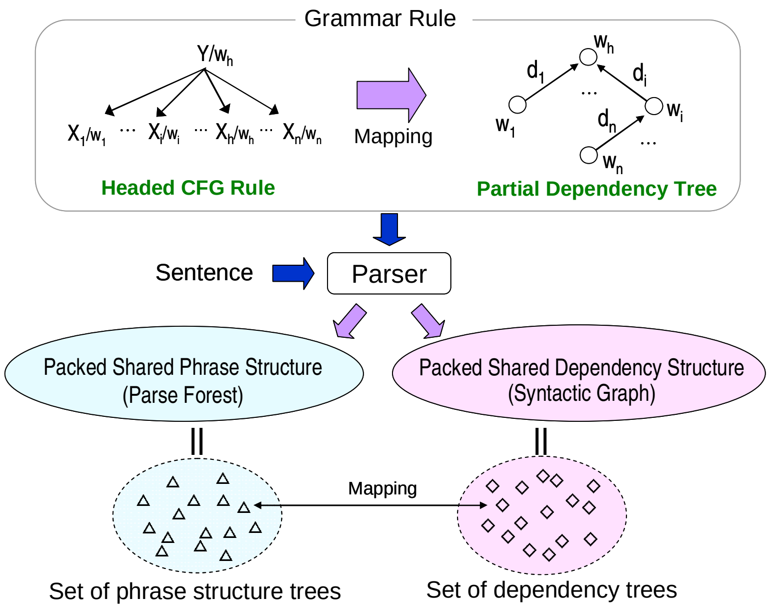

Grammar Rule
Y/w_h
X_1/w_1 ··· X_i/w_i ··· X_h/w_h ··· X_n/w_n
Mapping
w_h
d_1
d_i
w_1
d_n
w_i
w_n
Headed CFG Rule
Partial Dependency Tree
Sentence
Parser
Packed Shared Phrase Structure (Parse Forest)
Packed Shared Dependency Structure (Syntactic Graph)
Mapping
Set of phrase structure trees
Set of dependency trees

Fig.1.5 Framework based on partial structure mapping rules

generates two packed shared data structures, i.e., the headed parse forest[*18] encompassing all possible phrase structure trees and the "syntactic graph" encompassing all possible corresponding dependency trees. The later part of this thesis uses the term "phrase structure forest" instead of the parse forest to strike a clear contrast to the dependency forest described in Section 1.4.

In comparison with Abney's framework described in Section 1.1.2, the partial structure mapping rule defines not only the phrase head information but also the structural mapping between CFG and DG partial trees. The partial structure mapping rule is more flexible because it allows an arbitrary depth in the dependency structure corresponding to one CFG rule. The purpose of Seo and Simmons' research was to provide a compact packed shared data structure corresponding to the phrase structure forest of a sentence. Seo and Simmons did not discuss the equivalence between CFG and DG where the formal definition of DG as well as CFG was indispensable[*19].

Seo and Simmons defined the completeness and soundness of the syntactic graph with respect to the two mapping relations between the phrase structure forest and the syntactic graph as follows:

(Completeness) All phrase structure trees in the phrase structure forest can be mapped from the dependency trees in the syntactic graph.
$\forall PST(\textit{phrase structure tree})\ \exists DT(\textit{dependency tree})\ \textit{dependency tree corresponding to } PST \textit{ is } DT$

(Soundness) All phrase structure trees mapped from the dependency trees in the syntactic graph are in the phrase structure forest.
$\forall DT(\textit{dependency tree})\ \exists PST(\textit{phrase structure tree})\ \textit{dependency tree corresponding to } PT \textit{ is } DT$

Seo and Simmons (1989) proved the completeness but not the soundness of the syntactic graph. Hirakawa (2006b) showed that the soundness of the syntactic graph was not satisfied, i.e., the mappings between the phrase structure trees and the dependency trees were incomplete in the syntactic graph.

There appears to be no method for constructing the complete mapping between the phrase and dependency structures. PDG realizes the complete mapping between the phrase and dependency structures based on the partial structure mapping by introducing a new packed shared data structure called the "dependency forest" instead of the syntactic graph; furthermore, it realizes the integrated usage of the phrase and dependency structures at the syntax level. The details of the dependency forest are explained in Chapter 3.

[*18] The formal name of the "parse forest" is "packed shared parse forest" (Tomita, 1987).
[*19] Dependency grammar formalism based on the partial structure mapping rules is an interesting research topic that is beyond the scope of this thesis.

1.5 Contributions of This Thesis

This thesis proposes a new dependency analysis method through discussions on the design principles for multilevel NLA systems focusing on the treatment of preference and constraint knowledge. The proposed sentence analysis method (or framework) is called the "preference dependency grammar (PDG)." PDG is an all-pair multilevel dependency analysis method with the morphological and syntactic levels, and has the following features for the issues described in 1.3.

(a) Phrase structure analysis is utilized in the dependency structure analyzer
(b) POS ambiguities are handled in dependency structure analysis
(c) Detailed descriptions for preference and constraint knowledge for the dependency structure are available

The core technologies of PDG for enabling these features are a new data structure "dependency forest" and a new algorithm "graph branch algorithm," which are the main contributions of this thesis.

(1) Dependency Forest

The dependency forest is a new packed shared data structure for representing a set of dependency trees with their preference scores. The dependency forest consists of the dependency graph, constraint conditions, and preference information. The details of the dependency forest are described in Chapter 3. The following three are the main contributions related to the dependency forest.

(a) The method for obtaining the dependency forest for a sentence
Based on the partial structure mapping method briefly described in Section 1.4, the sentence analysis algorithm proposed in Chapter 3 generates the dependency forest, which has the complete and sound mapping to the corresponding phrase structure forest. The lack of soundness of the traditional approach for the partial structure mapping method, i.e., the syntactic graph, is also shown in Chapter 3. The complete mapping between the phrase structure forest and dependency forest provides the basis of the integrated use of these structures. The phrase structure grammar (CFG grammar) can function as a filter for the dependency structures for the input sentence, and the POS ambiguities retaining all possible WPP sequences are introduced into the dependency forests instead of adopting only one WPP sequence as an input to the dependency analyzer. Thus, the CFG filtering enabled by the dependency forest suppresses the explosion of dependency trees found in the all-pairs approach in the single dependency model.

(b) Proof of the completeness and soundness of the dependency forest
Chapter 3 gives the proof of the completeness and the soundness of the dependency forest

with respect to the phrase structure forest.

(c) Packed shared data structure with detailed preference and constraint knowledge description

The dependency forest provides higher descriptive ability compared to the existing dependency-graph-based packed shared data structures employed in major all-pairs dependency parsers.

[**Constraint Matrix**]

The dependency forest provides a precise definition of a set of dependency trees encompassed in the dependency graph by introducing the constraint matrix, which can express co-occurrence constraints between two arbitrary dependency relations in a dependency tree. The dependency forest can handle POS ambiguity, non-projective dependency trees, and the single valence occupation constraint*[20]. Traditional approaches utilizing the dependency graph as a packed shared data structure cannot handle these issues because they have no explicit means for expressing detailed constraints. As described in Chapter 4 in detail, the dependency graph searched by the Chu-Liu-Edmonds maximum spanning tree algorithm is restricted to a single node dependency graph and cannot encode POS ambiguity. The scored dependency graph searched by the DP based-algorithms such as Eisner (1996b) and Ozeki (1994) is restricted since they cannot handle non-projective dependency trees and cannot express the single valence occupation constraint between two dependency relations.

[**Preference Matrix**]

The edge factored model is widely used in the all-pairs approach for expressing the preferences of the dependency trees encompassed in a dependency graph. On the other hand, the preferences of the dependency trees in a dependency forest are defined by the preference matrix of the dependency forest. The preference matrix can express preferences for arbitrary two dependency relations (called the binary preference model of PDG) as well as the edge factored model (called the unary preference model of PDG)*[21]. The unary preference model of PDG can treat the word or WPP bigram preference as well as the dependency co-occurrence preference. The unary preference model of PDG enables the integrated use of tree-local information (preference on dependency relation) and string-local information (preference on word sequence) described in Section 1.3.

(2) Graph Branch Algorithm

This thesis proposes a new optimum search algorithm called the “graph branch algorithm” based on the branch and bound principle (Land and Doig, 1960; Ibaraki, 1978). The graph

*[20] This is a kind of co-occurrence constraint with respect to the valences of a predicate. The details are described in Section 4.1.4.

*[21] The details of the unary and binary preference models are described in Chapter 4.

branch algorithm can search the optimum well-formed dependency tree in a dependency forest. The DP-based search algorithms such as Eisner (1996b) and Ozeki (1994) as well as the maximum spanning tree algorithms cannot be applied to the dependency forest search due to its high description ability.

(3) New Evaluation Measures

In addition to the widely adopted evaluation measure for evaluating the comprehensive analysis ability, this thesis proposes two new evaluation measures for dependency-based NLA systems in Chapter 6. The possibly correct sentence ratio measures the hypothesis generation ability and the arc disambiguation precision ratio measures the disambiguation ability of dependency-based NLA systems. This thesis reports an experimental result for checking these measures using the PDG prototype system.

1.6 Chapter Summaries

The main contents of this thesis are divided into three parts. The first part, Chapter 2, discusses sentence analysis models for integrating multilevel preference and constraint knowledge and describes the overall framework of PDG. The second part, Chapters 3 to 5, describes the detailed data structures and algorithms employed in PDG. The last part, Chapter 6, reports some evaluation measures and the experimental results obtained using the experimental PDG system. The remaining chapters of this thesis are summarized as follows.

Chapter 2 discusses sentence analysis models for integrating multilevel linguistic knowledge and shows the PDG design. This chapter explains basic sentence analysis model consisting of a sentence interpretation space, three kinds of linguistic knowledge (generation, constraint, and preference knowledge) and an optimum interpretation extraction mean. After discussing the properties of the basic sentence analysis model, the multilevel sentence analysis model is introduced and investigated for clarifying the design principles toward the integrated use of phrase structure and dependency structure in a multilevel sentence analysis system. Based on this design investigation, this chapter explains the overall architecture of the PDG system as well as its processing flow.

Chapter 3 describes the details of the packed shared data structures of PDG that were introduced in Chapter 2, particularly the two data structures at the syntax level, i.e., the phrase structure forest and the dependency forest. This chapter describes the problems in traditional packed shared dependency structures and explains the details of a new data structure called the “dependency forest,” which has a complete and sound mapping to the corresponding phrase structure forest. This feature is indispensable for the data structure used in the multilevel sentence analysis model described in Chapter 2. This chapter describes the details of the PDG grammar formalism, parsing algorithm, and the algorithm for generating the phrase structure and dependency forests, and provides proof of the completeness and soundness of the dependency forest. This chapter also provides an experiment for analyzing prototypical ambiguous sentences and discusses the mapping relations between the phrase structure tree(s) and the dependency tree(s) as well as the treatment of non-projective dependency structures in PDG.

Chapter 4 proposes a new algorithm known as the "graph branch algorithm" that computes the optimum dependency tree(s) from a dependency forest with preference scores. As is true in the dependency forest, a dependency graph with preference scores on its arcs is widely used for packed shared data structures for representing a set of scored dependency trees. This chapter formalizes the optimum tree search problem on a scored dependency graph as a search problem with preferences as well as constraints, and shows that traditional methods such as the spanning tree search method and the dynamic programming method are not applicable to dependency forests. The graph branch algorithm enables the optimum solution search for a dependency forest. This algorithm is based on the branch and bound principle and inherently has an exponential order of computational complexity. An experiment using the prototype PDG system shows no serious combinatorial explosions for ordinary sentences and exhibits a very good performance for the pruning strategy described in this chapter. Finally, Chapter 4 describes an extension of a dependency forest with only scored arcs (called the unary model) to one with arc co-occurrence scores (called the binary model) and shows the graph branch algorithm for the binary model.

Chapter 5 describes a scoring process that computes the preference scores for the dependency forest of a sentence based on a various kind of preference knowledge. PDG utilizes corpus statistics of some partial linguistic structures such as word/POS frequency, word/POS bigram frequency and word/POS dependency frequency. Such statistical information that is obtained from each linguistic level is computed and integrated into the preference scores in the preference matrix of the dependency forest for a sentence. The optimum dependency tree(s) are obtained from this dependency forest by the graph branch algorithm described in Chapter 4. Chapter 5 explains the principle and basis of score integration and shows the formulas for computing the preference matrix for a sentence.

Chapter 6 discusses and proposes three evaluation measures for dependency structures and reports the experimental results obtained using the PDG prototype system. In addition to the widely adopted evaluation measure for evaluating the comprehensive analysis ability of a dependency-based NLA system, this chapter proposes two new measures for evaluating the hypothesis generation ability and disambiguation ability of NLA systems. An experiment for checking these measures is conducted. Then, experiments for evaluating some aspects of the PDG system performance with respect to the preference knowledge are conducted to demonstrate the effect of the integration of multiple preference knowledge.

Chapter 7 presents some possible directions for future research.

Chapter 8 summarizes and concludes this thesis.

Chapter 2

Sentence Analysis Model and the PDG design

2.1 Multilevel Sentence Analysis System

2.1.1 Basic Sentence Analysis Model

In general, an NLA system computes structures for a sentence by generating a set of its possible interpretations (application of interpretation generation knowledge), rejecting impossible interpretations (application of constraint knowledge), and obtaining the preference order of the possible interpretations (application of preference knowledge). Fig.2.1 presents this sentence analysis model[*1]. A set of interpretations of a sentence exists in the interpretation space prescribed by the interpretation description scheme. Each interpretation is either correct (◎), plausible (○), or implausible ($\times$) with respect to the real-world situation.

(1) Interpretation Description Scheme and Interpretation Space

A formal description of linguistic interpretation requires a proper representational scheme based on some appropriate data structure. The interpretation space defines a set of structural data for expressing the interpretation of sentences. For example, spaces defining phrase structure trees, dependency structure trees, semantic graphs, or logical formula are widely used as interpretation spaces. An interpretation description scheme defines the well-formedness of the structural data as data type. Well-formedness as an interpretation of a sentence is defined by the constraint knowledge.

(2) Generation Knowledge

The generation knowledge[*2] generates a set of candidate interpretations in the interpretation space (i.e., expressed in the interpretation description scheme) from the input data. Examples of interpretation generation include processing such as assigning POSs to words by consulting

[*1] Constraint knowledge can be defined as a type of preference knowledge that does not provide any possibilities. However, the application of constraint knowledge implies pruning in the computation, which is in clear contrast to the application of the preference knowledge.

[*2] "Interpretation generation knowledge" is simply called "generation knowledge" in this thesis.

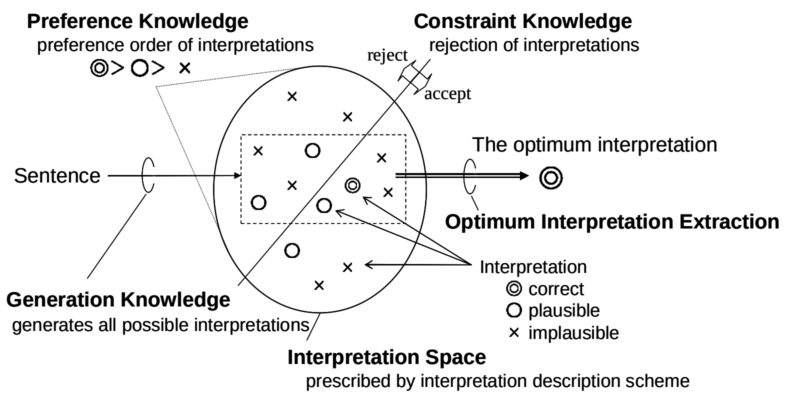

Preference Knowledge
preference order of interpretations
◎> O> ×
Constraint Knowledge
rejection of interpretations
reject
accept
Sentence
The optimum interpretation
Optimum Interpretation Extraction
Interpretation
◎ correct
O plausible
× implausible
Generation Knowledge
generates all possible interpretations
Interpretation Space
prescribed by interpretation description scheme

Fig.2.1 Natural language analysis system model

dictionary and generating possible phrase structure trees by appling CFG rules. Generation knowledge is a kind of constraint knowledge in the sense of the term, because it functions to extract the possible interpretations for a sentence from the whole interpretation space.

(3) Constraint Knowledge

Constraint knowledge defines a set of well-formed interpretations for a sentence and filters out the impossible interpretations in the candidate interpretations generated by the generation knowledge. The constraint knowledge in conjunction with the generation knowledge (or simply the constraint knowledge in the wider sense) defines the sentence coverage, i.e., a set of acceptable sentences, of the NLA system. Therefore, this knowledge corresponds to a grammar in linguistics from the Chomskyan viewpoint (Chomsky, 1957). Many computational grammar frameworks have been proposed and studied, in which a variety of linguistic knowledge has been incorporated. Grammar frameworks are based on interpretation description schemes that prescribe interpretation spaces such as phrase structure, dependency structure, semantic graph structure and logical formula.

(4) Preference knowledge

Preference knowledge provides the ordering of the interpretations in the interpretation space. Many researches on preference knowledge, such as preference semantics (Wilks, 1975), have been conducted in linguistics. In general, two approaches are followed for implementing preference knowledge in NLA systems, i.e., the heuristic approach and the corpus-based approach. In the heuristic approach, a human grammarian extracts and encodes the preference rules based on his/her linguistic insight to an NLA system and refines them through system development. The corpus-based approach attempts to extract the optimum preference knowledge from tagged or plain corpora by applying a learning technique to obtain statistical rules and/or parameters. The corpus-based approach is intensively studied in various application areas because the heuristic approach requires tremendous efforts, and occasionally, grasping the complexity in heuristic rule debugging is beyond the human ability.

Preference knowledge has been widely adopted for NLA systems through the use of the corpus based method adapted from speech technology. As statistical methods extend their application scope from the N-gram model (word sequence) to the context free grammar, dependency grammars, etc., more NLA systems can benefit from the statistical power obtained from large-scale corpora as described in Sections 1.2 and 1.3.

(5) Optimum Interpretation Extraction

The output of the NLA system is the optimum interpretation extracted from among the remaining interpretations according to the preference order. The optimum extraction is to search the interpretation space for the best interpretation that satisfies the well-formed constraints, i.e., a kind of combinatorial optimization problem. This kind of problem has a lot of variations from an easy one (requiring polynomial order computational complexity) to hard one (requiring exponential order computational complexity) depending on the characteristics of the target data structure, constraints, and preferences.

Various types of linguistic preference and constraint knowledge usable in sentence analysis lie in each linguistic layer. Fig.2.2 shows some examples of the preference and constraint knowledge at each linguistic analysis level[*3]. Constraint knowledge is divided into two categories. The lower part shows the basic language-independent constraints [*4] and the upper part shows the more detailed and language-dependent constraints. A detailed explanation of the preferences and constraints is provided in the latter part of this thesis.

(6) Linguistic Knowledge and System Examples

Before providing a more detailed explanation of the basic sentence analysis model or system, two examples are shown. Fig.2.3 corresponds to the sentence analysis model of the probabilistic CFG (PCFG) (Jelinek et al., 1992). The generation knowledge, preference knowledge, and interpretation space are the CFG rules, probabilities of the CFG rules, and phrase structures prescribed by the grammar rules, respectively. PCFG has no constraint knowledge. The optimum interpretation is computed by using the algorithm similar to the Viterbi algorithm. Fig.2.4 shows

Knowledge Type	Multi-level knowledge				
	Word	WPP Sequence	Phrase Structure	Dependency/Functio nal Structure	Semantic/Logical Strucutre
	String	Part of speech (V,N,ADJ)	Constituent (NP,VP)	Syntactic Function (Subject,Object)	Semantic Function (Agent,Goal)
Preference Knowledge	Word probability	N-gram probability	Phase structure probability	Dependency structure probabilty Word distance	Semantic dependency probability Case frame/valency
Constraint Knowledge		WPP adjacency	Constituent sequence	Attribute agreement	World knowledge
	Inflection	Simple WPP role	No overlapping Coverage constraint	Projectivity Coverage Tree form	Simple semantic role Single valence occupation

Fig.2.2 Preference knowledge and constraint knowledge for each linguistic layer

[*3] Not shown in Fig.2.2, contextual processing such as an anaphora resolution and so forth requires constraint knowledge (Walker et al., 1994; Mori et al., 2000) and preference knowledge (Seki et al., 2002).

[*4] This kind of constraint is sometimes called an axiom as shown in Robinson's axiom in Section 1.3.

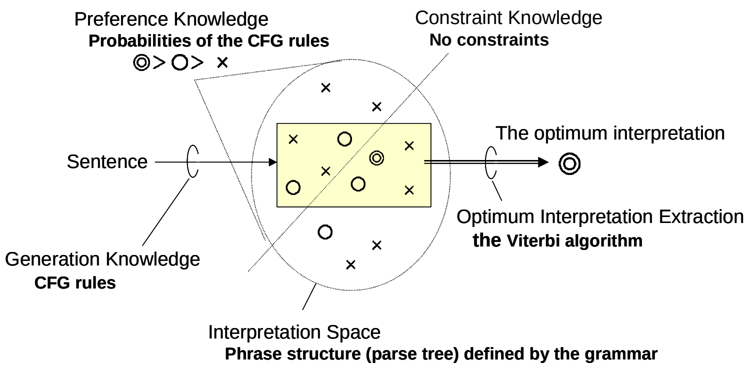

Preference Knowledge
Probabilities of the CFG rules
◎> ○> ×
Constraint Knowledge
No constraints
The optimum interpretation
◎
Sentence
Optimum Interpretation Extraction
the Viterbi algorithm
Generation Knowledge
CFG rules
Interpretation Space
Phrase structure (parse tree) defined by the grammar

Fig.2.3 Sentence analysis model of the PCFG

the sentence analysis model of the original CDG (Maruyama, 1990). CDG adopts the eliminative parsing method in which the parsing proceeds by filtering out the incorrect interpretation from all possible interpretations of a sentence by applying the unary and binary constraints. The original CDG has no preference knowledge[*5].

Needless to say, in order to formally use preference and constraint knowledge, they must be described on top of some formal schema or data structure. However, the constraint and preference knowledge working for the data (or the interpretation) in the interpretation space is independent of the description schema or data structure for the constraint and preference knowledge. For example, introducing the semantic knowledge as the means for restricting the interpretations in the syntactic interpretation space is a very popular technique. CDG shown in Fig.2.4 is a dependency grammar framework with unary or binary constraints. These constraints are used for incorporating morphological and semantic information (Maruyama, 1990). DCG (Pereira and Warren, 1980) and BUP (Matsumoto et al., 1983) have developed a mechanism to extend the CFG framework to incorporate arbitrary extra-conditions using Prolog codes that, for example,

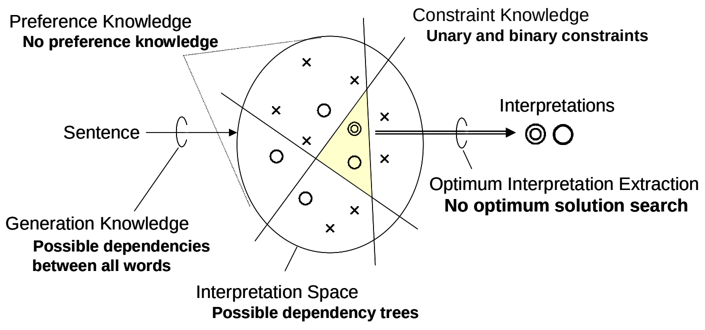

Preference Knowledge
No preference knowledge
Constraint Knowledge
Unary and binary constraints
Interpretations
◎ ○
Sentence
Optimum Interpretation Extraction
No optimum solution search
Generation Knowledge
Possible dependencies between all words
Interpretation Space
Possible dependency trees

Fig.2.4 Sentence analysis model of the original CDG

[*5] CDG extensions such as an introduction of graded constraints (Heineck et al., 1998) and probabilistic model (Wang and Harper, 2004) are proposed to treat preference knowledge within the CDG framework.

can be used for introducing semantic constraints (Muresan and Rambow, 2007). Some of the recent morphological taggers utilize syntactic data structures such as supertags (phrase structure data) (Bangalore and Joshi, 1999; Clark and Curran, 2004) and superARG (dependency data) (Wang and Harper, 2002; Wang and Harper, 2004). As shown in Section 1.2, dependency structure information and semantic information are utilized in phrase structure analysis.

In general, data structures referred from the different linguistic layer processing are not interpretations of a sentence but partial structures or features in the sentence.

(7) Optimum Solution Search Algorithm

As briefly explained in Section 1.3, there are two approaches for the optimum tree extraction, i.e., the history-based approach and the all-pairs approach*[6]. The history-based approach (Black et al., 1992) assumes that the probability of the parsed structure is determined by the parsing process, i.e., each tree-building procedure uses a probability model p(AlB) to weight any action A based on the available context, or history, B. The all-pairs method obtains the optimum parsed structure from among a set of possible parsed structures based on the probability (or preference score) defined on the parts of the parsed structures. This is undertaken in three steps, i.e., the generation of possible candidates, generation of preference scores, and search for the parsed structure with the highest preference score. The process of calculating the preference scores and setting them to some data structure is called “scoring” in this thesis.

Generally speaking, the history-based method realizes higher speed efficiency because it functions deterministically; occasionally, however, it suffers from the local minimum problem because the decisions during parsing are made based on local information, which may eventually lead to failure in capturing the correct global structure. On the other hand, the all-pairs method requires more computational resources but can handle global structure preferences and assures the optimality of the obtained structure. McDonald and Nivre (2007) reported that the accuracies of the Malt parser (history-based method) and MSTParser (all-pairs method) were almost identical irrespective of the methodological difference between them. Researches on the extension, improvement and integration of these dependency parsers has been conducted (Charniak and Johnson, 2005; Xiaodong and Chen, 2007; Huang and Chiang, 2007; Hall, 2007). This thesis focuses on the all-pairs full-decoding dependency analysis method and thereby on optimum search algorithms for the packed shared dependency structures. The applicability and performance of an optimum search algorithm is closely related to the characteristics of the target data structure, constraints and preferences.

2.1.2 Multilevel Sentence Analysis Model

From the viewpoint of the multilevel knowledge integration, sentence analysis frameworks are classified as either a single-level model or a multilevel model. The single-level model has one interpretation space and is merely the basic sentence model described in the previous section.

*[6] The “all-pairs approach” is not restricted to dependency parsing in this thesis.

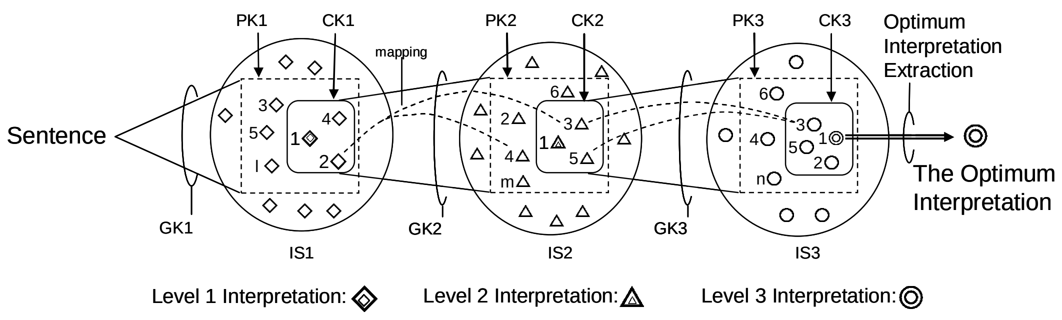


Fig.2.5 Multilevel sentence analysis model

The multilevel model has more than one interpretation spaces and possibly multiple description schemes. Fig.2.5 shows the basic constructions of the multilevel models. The multilevel model is basically a cascaded connection of some basic sentence models. However, this does not imply a sequence of cascaded processing modules. It shows the construction of linguistic knowledge and data structures. Each interpretation space represents the interpretations of a sentence in some layer of linguistic theory, such as morphology, syntax and semantics. The multilevel model assumes a layer structure among its levels, i.e., linguistic data structures. The input sentence side is referred to as the lower level and the output side, the upper level. The data structure of an intermediate level is considered to be an intermediate data structure bridging the interpretations from the lower adjacent level to the upper adjacent level. For example, a WPP sequence for a sentence (morphological analysis result) is the intermediate data structure bridging the interpretation from a character sequence (input sentence) to a phrase structure tree (syntactic analysis result).

Each level can have its generation, preference, and constraint knowledge. The generation knowledge generates possible interpretations from a set of its lower-level interpretations. Every interpretation of a sentence in some interpretation space should have a mapping called the "interpretation mapping" (represented by the dotted line labeled "mapping") to its counterpart in the lower adjacent level of the interpretation space; however, the inverse is not necessarily true. For example, there can be a morphological interpretation of a sentence having no corresponding syntactic interpretations; however, there cannot be a syntactic interpretation of a sentence having no corresponding morphological interpretations. One interpretation has 0 to $M(M{\geq}1)$ interpretation mappings toward the upper level and 1 to $N(N{\geq}2)$ interpretation mappings toward the lower level, reflecting the existence of the ambiguities in natural languages. Thus, the multilevel model should satisfy the following two conditions related to interpretation mapping.

Definition 2.1.1 [Multilevel model mapping condition]

(a) Every interpretation in an intermediate level has at least one mapping to an interpretation in its lower level

(b) The interpretation in one level has a mapping to at least one mapping to an interpretation in its upper level iff there is no pruning by some constraint

Conditions (a) and (b) are respectively called the soundness condition and completeness condition for the multilevel model mapping condition. The difference with the mapping condition for the syntactic graph lies in the completeness condition.

An optimum solution search is performed at the uppermost level to obtain the final output of the sentence in Fig.2.5. The interpretation mapping can be used for searching the optimum interpretation based on the upper-level decision and not on the current-level decision by tracing back the mapping from the optimum interpretation at the upper level to the corresponding interpretation at the current level. For example, the tagger based on the optimum semantic analysis result can be constructed naturally.

A sentence analysis system based on the multilevel model is called the "multilevel system" in this thesis. Multilevel systems can refer to interpretations in the intermediate levels and have richer linguistic knowledge descriptions compared to single-level systems. However, they also have considerably more design complexities on account of having two degrees in knowledge integration design, i.e., knowledge integration in one level and knowledge integration in multilevel construction.

Defining the data structure is one of the most important issues in NLA system design. There are two major approaches for data structure implementation, i.e., the enumeration approach (or k-best approach) and the packing approach. The enumeration method maintains the possible interpretations as a set of independent data. The packing method utilizes packed-shared data structures for expressing a set of interpretations efficiently to avoid the combinatorial explosion problem. In general, the set of interpretations of a sentence is defined by three components, i.e., a packed shared data structure, an interpretation extraction schema, and a set of well-formedness constraints on the structures. Occasionally, some of the well-formedness constraints are embedded in the interpretation extraction scheme.

The enumeration method is superior to the packing method in descriptive power or freedom because it has no restriction for expressing a set of interpretations. For example, though simple WPP trellis, which is widely used as a packed data structure for expressing a set of possible WPP sequences of a sentence, can express word bigram constraint efficiently, it cannot encode a word trigram or more constraint[*7]. In contrast, the enumeration method simply lists a set of possible WPP sequences. From the viewpoint of computational resource, the enumeration method easily becomes intractable due to the combinatorial explosion of the possible interpretations. In many implementations, the k-best pruning method is adopted for avoiding this problem. The combinatorial explosion is suppressed by the k-best threshold in intermediate level; however, this may lead to overpruning. k-best pruning requires the application of preference knowledge for that level of interpretation space.

The packing method avoids the pruning of interpretations to the maximum extent possible

[*7] The word trigram or more constraint can be expressed by a set of independent constraints in conjunction with the WPP trellis.

by suppressing the combinatorial explosion possibly into a polynomial order complexity. Since the representable sets of interpretations are prescribed by the constrcution of a packed-shared data structure, the constraint representation schema and the interpretation extraction schema (enumeration algorithm or optimum solution search algorithm) are key design issues. In contrast to the k-best method, the packing method does not require intermediate pruning as well as the intermediate application of preference knowledge.

Thus, the antinomy between resource (computational and space complexity) and accuracy (pruning and knowledge description ability) lies between the enumeration and packing methods. Various researches including the integration of these two methods have been conducted, as described in the next section. There are no definite and concrete criteria for comparing these two methods. Determining which of the methods is appropriate for a given problem seems to be a design issue. This thesis focuses on the multilevel system based on the packing method. Each analysis level encompasses all possible sentence interpretations in its interpretation space in the form of each packed shared data structure. This model is referred to as the "multilevel packed shared data connection (MPDC)" model in this thesis.

Finallly, some relation of the multilevel issues to the linguistic theory is described. The multilevel model explained in this section can be seen as a model based on MTT (meaning-text theory) (Mel'cuk, 1988; Wanner, 1994; Kahane, 2003). MTT proposes a multilevel language model wherein the mappings between meanings and texts are established through multilevel interpretation data structures. The mappings between interpretations (or data structures) in adjacent levels of interpretation spaces assures the overall mappings. Basically, MTT is a bidirectional linguistic theory covering sentence analysis and sentence generation. However, MTT is developed and presented strictly in the synthesis direction and has thus far been discussed insufficiently with regard to the analysis direction. As Kahane (2003) described, "If we want to present a real procedure of analysis or synthesis, it is much more complicated because we have to take into account the question of multiple choices between rules (and, consequently, problems of memorization, choices, backtracking and parallelism)." The treatment of multiple choices, i.e., the ambiguities in sentence analysis, is not focused on; consequently, the treatment of the preference knowledge seems to be beyond the scope of the MTT framework so far. The problem of multiple choices is not crucial, in some sense, for sentence generation because multiple choices simply generate different texts representing the same meaning. On the other hand, it is crucial for sentence analysis because it generates different (i.e., incorrect) meanings from one text expression. Multiple choices induce the computational problems of memorization, choices, backtracking, and parallelism along with the combinatorial explosion of sentence interpretations. The multilevel model is a type of MTT-based framework that is capable of managing the multiple choices and preference knowledge.

2.1.3 Conventional Multilevel Syntactic Analysis Systems

As described in the previous section, the output of an NLA system is an interpretation in a certain interpretation space (for example, the phrase structure tree and the dependency tree). One of the interpretation spaces of the multilevel system is selected as its output level, and the well-formedness conditions and preference measure are defined on the interpretations in the space. Theoretically, the output level need not be the uppermost level. The output data structure defines the linguistic layer of the NLA system. For example, even if a tagger utilizes phrase structure information or semantic information, it is a morphological analyzer. This section overviews conventional NLA systems or technologies from the viewpoint of the multilevel model to discuss design principles for multilevel systems.

Many conventional syntactic analysis systems adopt a two-level construction with the data structures in morphological and syntactic layers, i.e., WPP sequence and syntactic structure. Some adopt the 1-best method for phrase structure analysis (Collins, 1999; Charniak, 2000; Bikel, 2004) and dependency structure analysis (Hirakawa, 2001; McDonald et al., 2005; Yamada and Matsumoto, 2003; Nivre and Scholz, 2004). In this construction, the disambiguation of POS ambiguity is left as the task for the adopted tagger and issues a problem because the disambiguation errors in the tagging process cannot be solved by improving the ability of a dependency parser. One applicable solution to this problem is adopting the k-best system construction.

Parsing iteration (or pipeline parsing) proposes a sentence analysis architecture with multiple analysis modules connected in the pipeline (Charniak, 2000; Hollingshead and Roark, 2007). The earlier stage analyzer generates k-best solutions efficiently by utilizing simpler preference knowledge and the later stage module selects the best result based on more sophisticated preference knowledge, which requires more computational resources. Charniak (2000) applies a grammar in a simplified manner in the first stage and then applies the same grammar fully in the later stage. Charniak and Johnson (2005) use the generative parsing model for the first stage to obtain the k-best candidates and then reranks the candidates based on the maximum entropy model to select the optimum solution.

Researches on multilevel systems with a combination of shallow parsing and deep parsing have been conducted. The shallow parser identifies the partial or superficial structures of a sentence based on the local information observed in a sentence. It need not generate the overall structure of a sentence. In contrast, the deep parser analyzes the deep construction of a sentence, such as syntactic relations and semantic relations, and generates the overall structure of a sentence. One typical shallow parser in this construction is the supertagger. A supertag represents some structural information in a higher level interpretation space such as a partial phrase structure tree. Supertagging, or the selection of a supertag for every word in a sentence, is almost equivalent to parsing (almost parsing) because a supertag sequence almost defines the syntactic structure of a sentence (Bangalore and Joshi, 1999)[*8]. A supertagger is used as a shallow parser for improving the parsing speed without the deterioration of parsing accuracy of deep parsers [*9] such as the CCG parser (Clark and Curran, 2004; Djordjevic et al., 2007), HPSG parser (Ninomiya et al., 2006; Ninomiya et al., 2007), and CDG parser (Wang and Harper, 2002; Wang and Harper, 2004). This suggests the design principle that it is important to have accurate k-best implementations in the lower levels of multilevel systems.

Trellis (or lattice) is widely accepted as a packed shared data structure for representing the morphological interpretations of a sentence in multilevel systems. This data structure represents the possible adjacency relation between WPPs. Constraints on WPP adjacency is one of the important kinds of constraint knowledge in Japanese. The use of this morphological constraint knowledge in the syntactic parsing stage significantly improves the efficiency of the parsing process (Shirai et al., 2000).

2.2 Proposal for a Dependency Analysis System Utilizing the Phrase Structure

2.2.1 The Integrated Use of Linguistic Knowledge in a Multilevel Sentence Analysis System

There are two types of knowledge integrations, i.e., the different-type knowledge integration (how to treat the constraint and preference knowledge) and the multilevel knowledge integration (how to treat different levels of linguistic knowledge). These integrations pose a problem in satisfying two conflicting requirements, i.e., the suppression of the combinatorial explosion and the suppression of the overpruning of the possible interpretations of a sentence. In each level of a natural language, the number of computationally possible interpretations of a sentence generally increases exponentially with its length. This causes a serious problem with regard to the time and space in the computation of the sentence analysis. The pruning of possible interpretations by applying constraint knowledge is an effective method to avoid the combinatorial explosion. However, the overpruning of the possible interpretations may degrade the system accuracy. Therefore, NLA systems must have a proper mechanism to integrate the preference and constraint knowledge.

Multilevel knowledge integration also poses the same problem. Pruning is more effective at the morphological level than at higher levels such as the syntactic and semantic levels since an interpretation at the morphological level corresponds to multiple higher level interpretations. However, pruning of the lower level interpretations based on lower level linguistic knowledge may fail to provide the correct interpretation due to the lack of upper level linguistic information. Therefore, it is important for NLA systems to have a proper mechanism to integrate the multilevel

[*8] The well-formedness check and generation of the sentence interpretation remain to be undertaken in the uppermost level.

[*9] Parsing accuracy occasionally improves by the combined use of shallow and deep information.

linguistic knowledge.

(1) Integration of multilevel constraint knowledge

As described in Section 2.1.2, constraint knowledge can be applied in either the intermediate level or last level of a multilevel system construction. The application in an intermediate level corresponds to the pruning of interpretations, which is propagated naturally to the upper levels due to the multilevel model mapping condition. Prunings in the lower levels are very effective for efficiency improvement. Therefore, the application of constraint knowledge should be undertaken in the lower level to the maximum extent possible. On the other hand, the final level defines the output interpretation. This implies that the constraints in this level are well-formedness conditions that cannot be fully described in the lower level structure.

(2) Integration of multilevel preference knowledge

The preference knowledge in various linguistic layers is applicable to interpretations in one level interpretation space. The application of preference knowledge to the intermediate level simply defines the preferrential order of interpretations in that level and, unlike the constraint knowledge, has no direct influence on the preference orders of the interpretations in the other spaces[*10]. The application of preference knowledge in the intermediate level is necessary for the k-best approach to select a set of interpretations. This is a use of preference knowledge for pruning, i.e., constraint application.

The application of preference knowledge in the uppermost level defines the output of the NLA system. Ninomiya et al. (2007) compared two different use cases of preference knowledge in an NLA system, which consists of a supertagger and a HPSG parser. The first case utilizes the supertagger preference (word trigram and POS 5-gram model) to select k-best morphological interpretations and the best deep interpretation based on the HPSG stochastic preference model. In the second case, both the supertagger and HPSG preference models are integrated to select the best HPSG parse. The latter showed considerably superior accuracy compared to the former. Wang and Harper (2004) compared two cases for combining the SuperARV tagger and CDG parser, i.e., combining them by the k-best method (loosely coupled system) and applying two preferences simultaneously (tightly coupled system) and reported that the tightly coupled system outperformed the loosely coupled system. Charniak and Johnson (2005) utilize the discriminative maximum entropy model for the reranking of the pipeline parser (Charniak, 2000) and obtained the improvement in the parsing accuracy. The fact that this discriminative maximum entropy model includes various features in multiple linguistic layers suggests that the integrated use of various levels of preference knowledge is a key to accuracy improvement. These research results show the importance of preference knowledge integration in the uppermost level of a multilevel system.

[*10] There can be a system construction in which the optimum interpretation is searched in some lower level to obtain the higher level interpretation by tracing the interpretation mapping. In this case, some application of preference knowledge is required because the lower interpretation may have multiple counterparts in the upper level spaces. For example, one syntactic structure can have many possible semantic interpretations (Harada and Mizuno, 2001).

2.2.2 PDG Design

This section describes a new multilevel NLA method, called PDG, utilizing the phrase structure and dependency structure levels. PDG employs a three level architecture with two intermediate levels (morphological structure and phrase structure) and the uppermost level (dependency structure). The dependency structure is selected as the output of PDG because it has an affinity with the semantic structure, which lies within the scope of future research, as described in Chapter 7. Based on the previous discussions on multilevel systems, the following three issues are settled for PDG design principles.

(a) Avoiding overpruning as well as suppressing combinatorial explosion as much as possible
(b) Adopting effective pruning by applying possible constraints in the lower level
(c) Enabling the optimum search in the uppermost level to utilize various levels of preference knowledge

PDG adopts the MPDC model to achieve (a). This requires packed shared data structures for morphological structure, phrase structure, and dependency structure, which satisfy the multilevel model mapping condition (Definition 2.1.1). To fulfill this requirement, this thesis proposes a new method for obtaining a packed shared dependency data structure called the dependency forest, which satisfies the mapping condition against the phrase structure forest. Based on (b), the phrase structure level is utilized as a filter for the dependency level. This unique construction is an answer to the search space problem caused by introducing POS ambiguities to dependency analysis, as described in Section 1.3. In the three level architecture of PDG, the phrase structure filter suppresses the explosion of dependency trees and enables all-pairs dependency parsing for all POS ambiguities[*11]. Following principle (c), PDG adopts the preference knowledge description scheme called the preference matrix in the dependency structure level. The preference matrix is a more powerful descriptive scheme compared to the edge factored model, which is widely used for the single dependency model parsers. A new optimum tree search algorithm called the graph branch algorithm is proposed to realize the optimum tree search in the dependency forest, which is not achieved by the conventional graph search algorithms, as described in Chapter 4.

Finally, the PDG system is defined as an all-pairs dependency parsing system with the following features:

(a) Consisting of three level spaces (data structures) for WPP sequence, phrase structure tree, and dependency tree
(b) Utilizing three packed shared data structures, i.e., WPP trellis, phrase structure forest, and dependency forest

[*11] The descriptive power of the partial mapping model, i.e., mapping between the CFG rule structure and partial dependency structure, is one important issue for the appropriateness of the use of its CFG filtering. There can be a more powerful model with mapping between the arbitrary partial phrase structure tree and partial dependency tree structure. This issue lies beyond the scope of this thesis.

(c) Utilizing the graph branch algorithm for searching the optimum interpretation from a dependency forest

2.2.3 The Data Structure/Processing Model of PDG

Fig.2.6 shows the PDG analysis model. PDG has two basic linguistic layers, i.e., morphology and syntax. The syntax layer is further divided into two levels. In total, PDG has three levels of interpretation space, description scheme, and packed shared data structure. Fig.2.7 presents a brief explanation of the data structures and examples of the preference knowledge, the constraint knowledge, the packed shared data structure and the sentence interpretation at each level.

Morphological interpretations for a sentence are represented by sequences (or strings) of WPP nodes, which represent the adjacency relations between words. The WPP trellis is used as a packed shared data structure for representing a set of sequences of WPP nodes. The nodes in the PDG data structure can possess arbitrary linguistic attributes such as number, gender, and tense (not shown in the figure). A sentence interpretaion in the morphological level is a sequence of the WPP nodes in the line from “start” to “end” in the Figure. These two special nodes are sometimes not explicitly shown in this thesis.

The syntax level of PDG contains two types of data structures, i.e., phrase structure and dependency structure. A phrase structure tree represents the sub-categorization (or adjacency) relations of phrases. A set of phrase structure trees is represented by a phrase structure forest. Syntactic preference knowledge (e.g., phrase frequency) and constraint knowledge (e.g., number agreements) can be described on top of the phrase structure[*12]. The dependency structure is another data structure in the syntax level of PDG. A dependency tree consists of WPP nodes

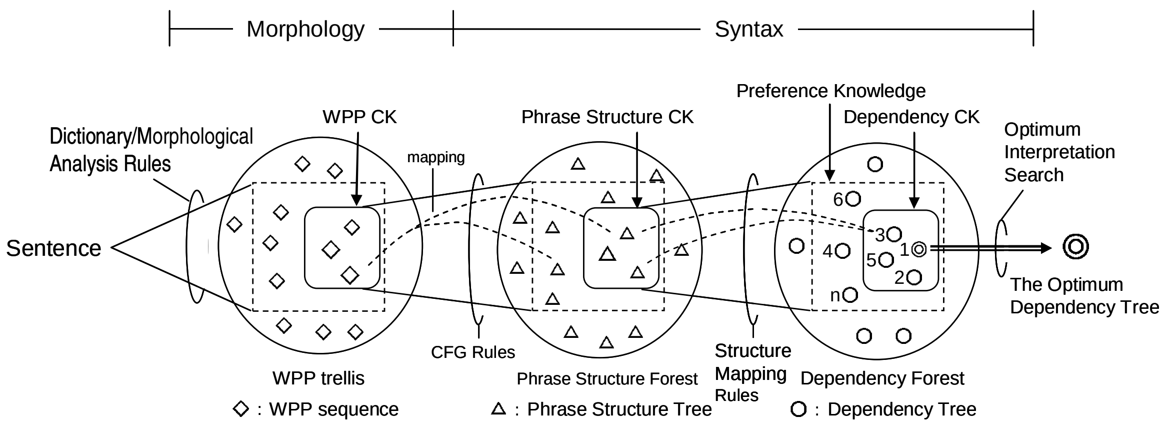


Fig.2.6 PDG implementation model

[*12] Constraints such as the number agreements can be described as constraints at another level or can be described in more than one level in parallel. This is a design issue in actual grammar development. In general, the number agreement constraint should be applied to the phrase structure level based on the design principle (b).

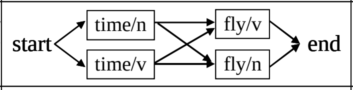
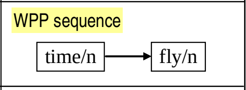
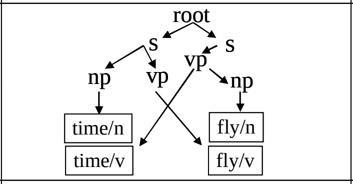
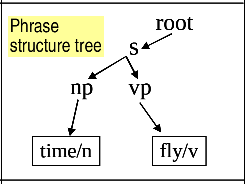
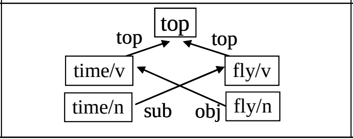
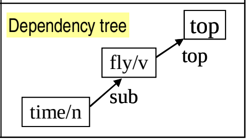
Packed Shared Structure	Node	Ex. of PK	Example	One Interpretation
	Relation	Ex. of CK		
WPP trellis	WPP	WPP frequency		
	adjacency	Adjacency constraint		
Phrase Structure Forest	Phrase Category	Phrase probability		
	Sub-categorization	Number agreements		
Dependency Forest	WPP	Dependency probability		
	Syntactic Dependency	Projectivity constraint		

Fig.2.7 Packed shared data structures in PDG

and arcs labeled with syntactic (or functional) dependency relations such as subject and object. A set of dependency trees representing the syntactic interpretations of a sentence is represented by a dependency forest. The dependency forest is a packed shared data structure that utilizes a dependency graph with a framework for describing the preference and constraint information for the arcs in the graph[*13]. The dependency probability and the projectivity constraint representable by the dependency representation are examples of the preference and constraint knowledge, respectively.

Fig.2.8 shows the relations in the multilevel data structures of PDG for the example sentence "Time flies." Each packed shared data structure corresponds to a set of interpretations in each

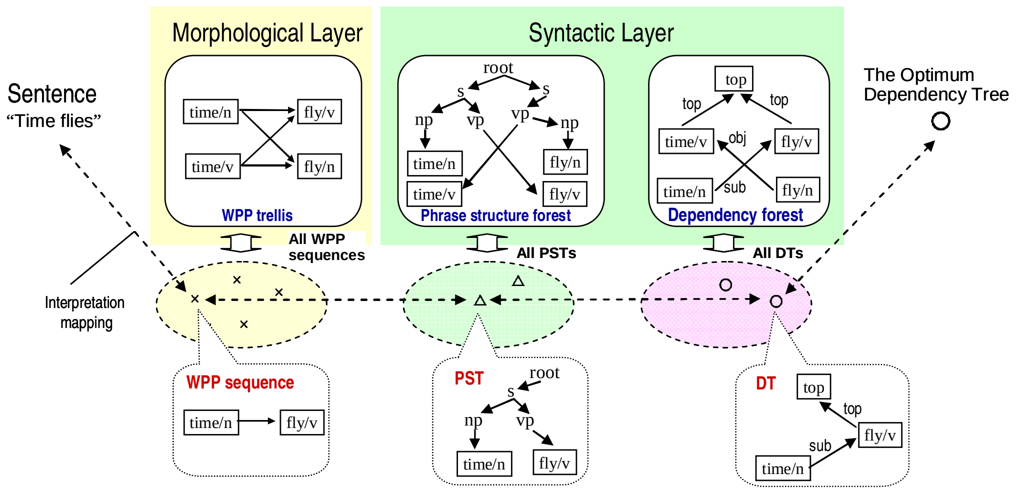


Fig.2.8 Relation between the data structures of the PDG implementation model

*13 As described in Chapter 4, these are represented by the constraint matrix and the preference matrix not shown in Fig.2.7.

interpretation space for a sentence. The WPP trellis encompasses four WPP sequences, i.e., "time/n+fly/v", "time/n+fly/n", "time/v+fly/v" and "time/v+fly/n." The phrase structure level has two phrase structure trees. One of them corresponds to the declarative interpretation of the sentence with mapping to "time/n+fly/v" and the other corresponds to the imperative interpretation of the sentence with mapping to "time/v+fly/n." The remaining two WPP sequences (the morphological interpretations) have no interpretation mappings to the phrase structure level in this example. The optimum interpretation of a sentence has a mapping to the input sentence through a series of interpretation mappings in multiple levels.

2.2.4 Scoring and Optimum Solution Search in PDG

In the MPDC model, the optimum well-formed interpretation can be basically defined in each interpretation space. However, it is not necessary to obtain or define the optimum well-formed interpretation of every interpretation space. The scoring and optimum solution (or interpretation) search methods for the WPP trellis and phrase structure forest are not described in this thesis because PDG is a framework for obtaining the optimum dependency tree for a sentence. The Viterbi algorithm is widely used for searching the optimum sequence in trellises with preference scores. A similar algorithm adopted in PCFG is a popular method for obtaining the optimum phrase structure tree from a phrase structure forest (Jelinek et al., 1992).

Fig.2.9 explains the scoring and optimum solution search for a dependency tree. The WPP sequence, phrase structure and dependency preference scores imply the preference scores computable based on the WPP trellis, phrase structure forest, and dependency forest, respectively. Examples of the reference knowledge of each data structure are shown in Fig.2.7. Such kinds of preference knowledge are integrated into a data structure called preference matrix defined in

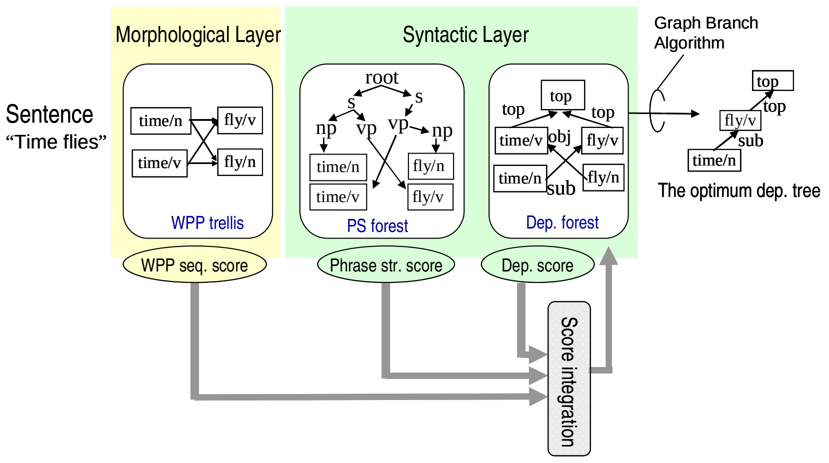


Fig.2.9 Scoring and optimum solution search

the dependency forest by the score integration module[*14]. The preference matrix can represent two kind of preference scores, i.e. unary preference score and binary preference score. Unary score represents the plausibility of one dependency relation and the binary score represents the plausibility of the co-occurrence between two dependency relations. Preference scores obtained from each level are converted and integrated into these preference scores. Two versions of the dependency forest, i.e., the unary and binary models are proposed and implemented in this thesis. The details of the score integration is described in 5. The optimum tree is searched from the unary or binary dependency forest using an algorithm called the "graph branch algorithm," which is described in detail in Chapter 4.

2.2.5 Processing Flow of the Experimental PDG System

Fig.2.10 shows the overall processing flow of the PDG experimental system. The morphological and syntactic parsing components are connected through data structures encompassing all ambiguities at each level. The morphological analysis module inputs a sentence and generates the WPP trellis by consulting the dictionary. This module is constructed by using standard technologies. The syntactic analysis module based on the chart parsing algorithm applies the PDG grammar rules to generate the PDG chart. PDG grammar rule consists of a CFG-based grammar rule (partial phrase structure) and partial dependency structure. The mapping between the phrase structure and dependency forests is essentially defined in the grammar rules. The

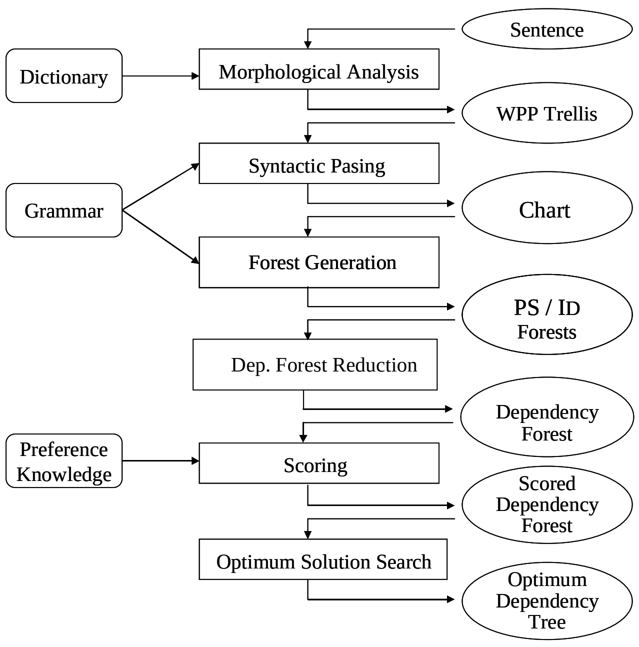


Fig.2.10 Analysis flow of the PDG experimental system

[*14] Not all preference knowledge kinds listed in Fig.2.7 are implemented in the PDG prototype system. Details are shown in Chapter 5

forest generation module extracts the phrase structure forest and the dependency forest called the "initial dependency forest" defined in Chapter 3 from the chart generated by the syntactic parser. The dependency forest reduction module generates the dependency forest from the initial dependency forest. The details of the syntactic analysis and dependency forest generation are described in Chapter 3.

The preference scores are integrated by the scoring module and are attached to the dependency forest. The dependency forest with the preference score is sometimes called the "scored dependency forest" explicitly. The morphological level preference knowledge (the WPP unigram and bigram frequencies) and the dependency level preference knowledge (the unary and binary arc frequencies) are utilized; however, the phrase structure oriented preference scores are not utilized in the current implementation of the PDG prototype system.

The optimum solution search module computes the most preferable well-formed interpretation of the sentence based on the preference scores generated by the scoring module based on the graph branch algorithm proposed in this thesis. The details of the optimum solution search algorithm are described in Chapter 4.

Currently, an experimental version of the PDG system has been implemented in Prolog aimed at the feasibility study of the PDG framework. The preference knowledge of this prototype system is extracted automatically from an English corpus by using the existing sentence analysis system (Amano et al., 1989) and the basic PDG grammar with around 1000 CFG rules is developed as described in Chapter 6. This thesis describes the details of the PDG and experiments using the experimental PDG system.

Chapter 3

Packed Shared Data Structures

3.1 Prerequisites for Packed Shared Data Structures

The following are prerequisites for the data structure at each level of the multilevel packed shared data structure connection model.

(a) no combinatorial explosion
(b) a set of proportionate interpretations
(c) satisfy the multilevel model mappping condition

(a) is a very important issue with regard to constructing practical NLA systems. In general, the enumerative treatment of interpretations leads to a lack of time and space or it degrades the analytical capability due to overpruning. (b) implies that the packed shared data structure at each level encompasses all possible solutions correctly, i.e., it assures there is no pruning of existing interpretations and no generation of nonexistent interpretations originating from the packed shared data structures. Provided this requirement is assured, it is beneficial for an NLA system to be capable of introducing possible pruning (application of constraint knowledge) in the early stage of sentence analysis considering the system performance. (c) is a prerequisite for the multilevel system described in Section 2.1.2.

3.2 Traditional Methods for the Packed Shared Data Structures

3.2.1 The WPP Trellis

PDG utilizes WPP trellis as the basis for the morphological analysis level. The WPP trellis is a packed shared data structure encompassing all WPP sequences for a sentence. Fig.3.1 shows an example of a WPP trellis for the sentence "Time flies like an arrow." Each node is labeled with the WPPs of each word in a sentence and has a variety of features such as word input position*1, lexical information, and morphological features. Arcs between the WPP nodes represent possible

*1 The word position is represented by zero origin basis.

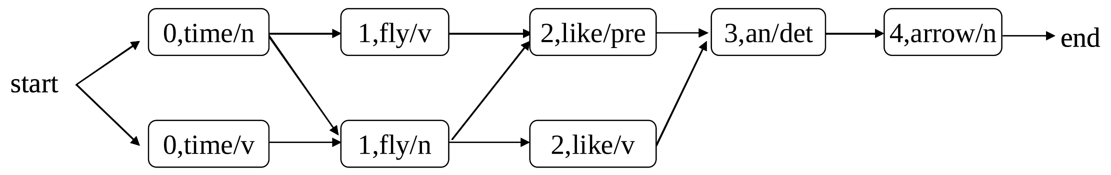


Fig.3.1 WPP trellis for "Time flies like an arrow"

adjacency relations. A WPP node sequence obtained by tracing from the top to the bottom of the trellis through the arcs corresponds to one morphological interpretation of the sentence. For example, the WPP node sequence "time/n fly/v like/pre an/det arrow/n" in Fig.3.1 is one interpretation of "Time flies like an arrow." Compound words occupy multiple input positions according to their word lengths. In general, a WPP sequence has 0 to N corresponding phrase structure trees and is considered to be an intension of the counterpart phrase structure trees.

3.2.2 The Packed Shared Phrase Structure Forest

The packed shared phrase structure forest, or simply phrase structure forest, is a well-known packed shared data structure for encompassing all phrase structure trees (Tomita, 1987). Fig.3.2 shows the packed shared phrase structure forest for the example sentence. A sub-tree headed with a non-terminal symbol that has multiple in-coming arcs is shared by its upper trees. A box containing the same nonterminal symbols, i.e., "s" or "vp," shows the packed sub-trees that have the same phrase boundaries (sentence span).

The WPP trellis and the packed shared phrase structure forest satisfy the interpretation mapping condition of the multilevel packed shared data structure connection model because each

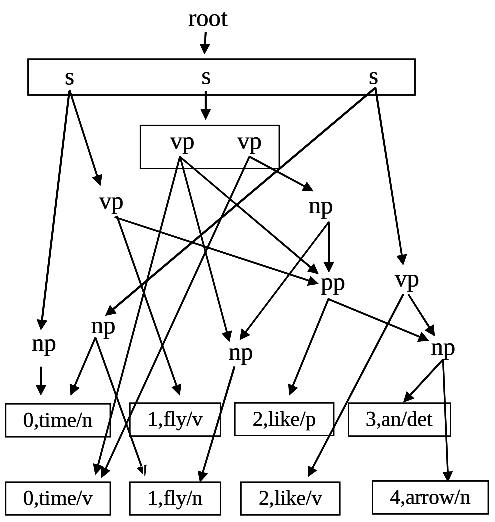


Fig.3.2 Phrase structure forest for "Time flies like an arrow"

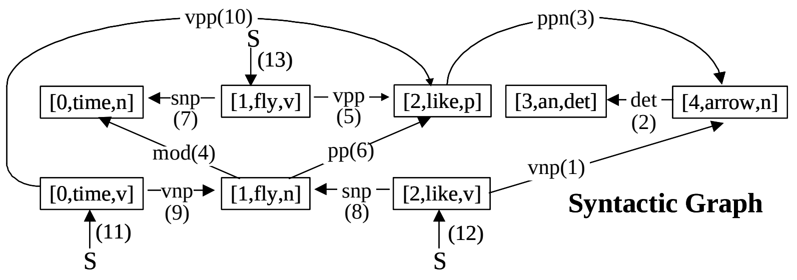
phrase structure tree in the packed shared phrase structure forest corresponds to a WPP sequence in the WPP trellis.

3.2.3 The Syntactic Graph

Seo and Simmons (1989) proposed the “syntactic graph”, which encompasses all dependency trees corresponding to phrase structure trees in the phrase structure forest for a sentence (Rim et al., 1990). The syntactic graph is a promising candidate for a packed shared data structure in PDG but it cannot be adopted as it is because it has a problem in satisfying the prerequisite (c) in Section 3.1 for the multilevel packed shared data structure connection model.

The syntactic graph is a directed graph, which consists of nodes representing WPPs and labeled arcs representing the syntactic relations between nodes. The syntactic graph defines a set of dependency trees (interpretations) for a sentence in combination with the “exclusion matrix”, which represents exclusive co-occurrence relations between arcs. The syntactic graph is a set of Triples containing arc name and two nodes (containing WPP, surface position etc.). Fig.3.3 shows the syntactic graph and the exclusion matrix for a sentence “Time flies like an arrow.” The numbers in arcs are arc-IDs. Multiple arcs targeting one node represent modification ambiguities. S corresponds to the starting symbol.

The exclusion matrix is a matrix whose rows and columns are a set of arcs in the syntactic graph that prescribes the co-occurrence relation between arcs. When (i,j) position in the exclusion matrix is set to 1, i-th arc and j-th arcs must not co-occur in any dependency tree (interpretation)



	1	2	3	4	5	6	7	8	9	10	11	12	13
1			1		1	1	1		1	1	1	1	
2													
3	1			1				1					1
4			1		1	1	1		1	1	1	1	
5	1			1		1		1	1	1	1		1
6	1			1	1		1	1		1		1	1
7	1			1		1		1	1	1	1		1
8			1		1	1	1		1	1	1	1	
9	1			1	1		1	1				1	1
10	1			1	1	1	1	1				1	1
11	1			1	1		1	1				1	1
12	1			1		1		1	1	1	1		1
13			1		1	1	1		1	1	1	1	

Exclusion Matrix

- 1 to 13 are arcs in the dependency graph
- “1” in a cell means exclusion of arc occurrence in dependency trees

  ex. EM(5,6)=“1”

  → No dependency tree can contains both arc5 (vpp fly like) and arc6 (pp fly like)

Fig.3.3 Syntactic graph and exclusion matrix for the example sentence

obtained from the syntactic graph. The syntactic graph and the exclusion matrix are generated from a kind of packed shared phrase structure forest. PDG adopts the same data structure and it is called a headed phrase structure forest. The detail of the headed phrase structure forest is described in Section 3.3. In the rest of this thesis, phrase structure forest means headed phrase structure forest. The traditional phrase structure forest (packed shared phrase structure forest) is called the headless phrase structure forest.

Seo and Simmons (1989) discussed the completeness and the soundness of the correspondence between the phrase structure forest and the syntactic graph. The completeness is satisfied if each phrase structure tree in the phrase structure forest has its counterpart(s) in the syntactic graph. The soundness is satisfied if each dependency tree in the syntactic graph has its counterpart(s) in the phrase structure forest. The completeness of the syntactic graph is shown in (Seo and Simmons, 1989) but the soundness is not assured. All exclusion matrix cells are initially set to 1 (this means no two triples co-occur). Then the cells for all the triple pairs in the dependency tree generated from phrase structure trees are set to 0. Since the exclusion matrix prescribes the co-occurrence relations for all dependency trees in the dependency graph[*2], the allowance of a co-occurrence of two triples (set 1 to the cell for two triples) is safe if and only if the restriction of these two triples is not necessary for all other interpretations (dependency trees). Appendix A shows an example in which the syntactic graph cannot satisfy the soundness condition.

3.3 Packed Shared Data Structures in PDG

PDG adopts the phrase structure forest and the dependency forest for the packed shared data structures for phrase structure and dependency structure representations, respectively.

3.3.1 Phrase Structure Forest

The phrase structure forest is a kind of packed shared parse forest and consists of edges corresponding to rewriting rules in CFG. The sub-trees, which satisfy the following conditions, are packed and shared.

Sub-trees have

(a) the same nonterminal symbol (category)
(b) the same coverage (phrase boundary)
(c) the same phrase head[*3] (head constituent)

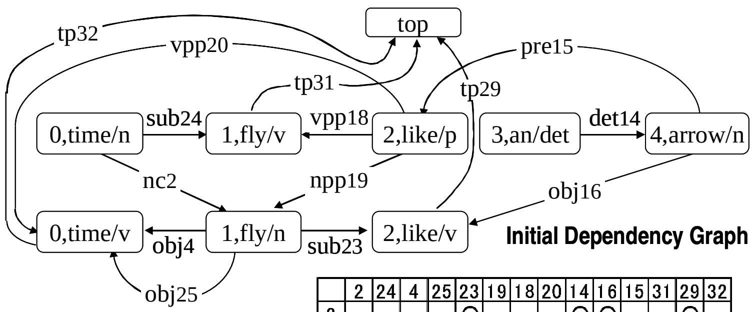
Conditions (a) and (b) constitute the headless phrase structure forest (Schiehlen, 1996). The phrase structure trees in the headed phrase structure forest have mapping to the phrase structure trees in the headless phrase structure forest. An example of the edges and the phrase structure forest in PDG is shown in Section 3.4 along with the parsing algorithm.

[*2] The constraint in the exclusion matrix is global in a sense.
[*3] Phrase head is a WPP in PDG.

3.3.2 Dependency Forest

The dependency forest (DF) consists of a "dependency graph" (DG) and a "constraint matrix" (CM, C-Matrix) expressed as DF=<DG,CM>[*4]. Fig.3.4 shows a dependency graph for the example sentence "Time flies like an arrow." The dependency graph consists of nodes and directed arcs. A node represents a WPP[*5] and an arc shows the dependency relation between nodes[*6]. An arc has its ID. The dependency graph has one special node called a top node, which is a root of all dependency trees in the dependency graph[*7]. In practice, the dependency graph is represented by a set of "dependency pieces". A dependency piece consists of one arc and its dependant (or modifier) node and governor (or modificand) node. Since dependency piece and arc have one to one correspondence, dependency piece is referred to as arc in this thesis. The number of arcs in the dependency graph is called a "size of the dependency forest". Dependency tree is a subset of dependency graph that forms a tree. Dependency trees represent interpretations of sentences or phrases at dependency relation level.

CM is a matrix whose rows and columns are a set of arcs in DG that prescribes the co-



Meaning of Arc Name

sub	: subject
obj	: object
npp	: noun-preposition
vpp	: verb-preposition
pre	: preposition
nc	: noun compound
det	: determiner
tp	: top

Initial Constraint Matrix

	2	24	4	25	23	19	18	20	14	16	15	31	29	32
2	–				○				○	○			○	
24		–					○		○		○	○		
4			–					○	○		○			○
25				–		○			○		○			○
23	○				–				○	○			○	
19				○		–			○		○			○
18		○					–		○		○	○		
20			○					–	○		○			○
14	○	○	○	○	○	○	○	○	–	○	○	○	○	○
16	○				○				○	–			○	
15		○	○	○		○	○	○	○		–	○		○
31		○					○		○		○	–		
29	○				○				○	○			–	
32			○	○		○		○	○		○			–

Fig.3.4 Initial dependency forest for the example sentence

[*4] The difinition of the dependency forest is extended to include the preference matrix PM in Chapter 4.

[*5] Node contains various information in the lexicon and surface position number.

[*6] The direction of dependency arc obeys the convention of the Japanese kakari-uke grammar (dependency grammar). The dependant node of an arc is the node located at the source of the arc. This is contrary to the convention in the syntactic graph, but not substantially different.

[*7] In this thesis, root of the dependency tree is called top node to distinguish it from the root of a phrase structure tree.

occurrence relation between arcs. Only when CM(i,j) is ○, arc_i and arc_j are co-occurable in one dependency tree. The co-occurrence relation is symmetric and CM is a symmetric matrix.

3.3.3 Well-formed Dependency Tree

Definition 3.3.1 [Well-formed dependency tree]
"Well-formed dependency tree" is a dependency tree DT in the dependency forest that satisfies the following conditions called the "well-formed dependency tree constraint".

[Well-formed dependency tree constraint]

(a) Every input word has a corresponding node in DT. (coverage constraint)
(b) No two nodes in DT occupy the same input position. (single role constraint)
(c) Each arc pair in DT has a co-occurrence relation in CM. (arc co-occurrence constraint)

(a) and (b) are collectively referred to as the "covering constraint". A dependency tree satisfying the covering constraint is called the "well covered dependency tree". A dependency tree satisfying (c) is called the "well co-occurred dependency tree". A set of well-formed dependency trees is the set of possible interpretations for an input sentence. The dependency forest in Fig.3.4 has four well-formed dependency trees. In PDG, a set of one WPP node is considered to be a special case of dependency tree with no arcs, which satisfies the well-formed dependency tree constraint.

3.3.4 Initial Dependency Forest and Reduced Dependency Forest

There can be more than one different-sized dependency forest encompassing the equivalent set of dependency trees with respect to the degree of arc sharing. PDG treats the "initial dependency forest" and the "reduced dependency forest" that is obtained from the initial dependency forest. The initial dependency forest consists of the "initial dependency graph" and the "initial C-matrix". The reduced dependency forest is simply called dependency forest in this thesis. The dependency graph of the initial dependency forest in Fig.3.4 is different from the syntactic graph in Fig.3.3 in terms of the number of arcs between "fly/n" and "time/v."

3.4 Generation of the Phrase Structure Forest and the Initial Dependency Forest

PDG generates the dependency forest from an input sentence through four processes, i.e., the morphological analysis, the syntactic analysis, the phrase structure/dependency forest generation and the dependency forest reduction.

WORD	Time	flies	like	an	arrow
WPP	0,time/n	1,fly/v	2,like/pre	3,an/det	4,arrow/n
	0,time/v	1,fly/n	2,like/v		4,arrow/v
			2,like/adj		
			2,like/n		

Fig.3.5 Dictionary lookup result for the example sentence

3.4.1 Morphological Analysis

As described in Section 2.2.3, morphological interpretations of a sentence are a set of WPP node sequences covering whole sentence, represented by WPP trellis. The upper adjacent interpretation space is represented by phrase structure forest. It is obvious that the interpretation mapping can be assured for these two data structures.

Morphological analysis is a well-established technology for major languages. PDG utilizes existing technologies for each language. This thesis gives simply illustrative explanation of the morphological analysis from the viewpoint described in Chapter 2.

Possible WPPs for a word are obtained by consulting a PDG dictionary. Fig.3.5 shows possible WPPs for each word in the example sentence "Time flies like an arrow." If no constraint exists for the adjacency relation between words, 32 ($2 \times 2 \times 4 \times 1 \times 2$) WPP sequences are obtained from the input sentence. Assuming the constraints that the adjacent sequences verb+verb "v v," adjective+determiner "adj det," determiner+verb "det v" are inhibited, a graph shown in Fig.3.6 is obtained by putting the available arcs between the WPPs. WPP nodes which have no possible path from the start or to the end position, for example "like/adj" and "arrow/v," can

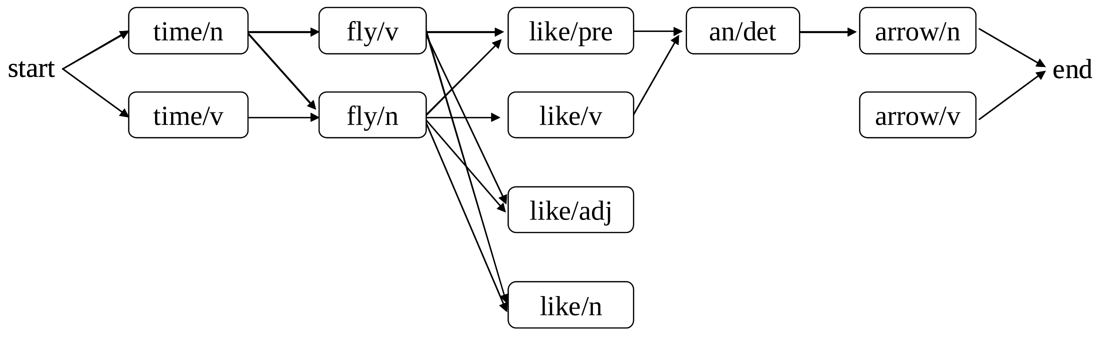


Fig.3.6 WPP Adjacency relation for the example sentence

be removed from the graph because it does not effect the total morphological interpretations of the sentence. This reduction of WPP nodes produces the WPP trellis shown in Fig.3.1. This trellis encompasses six WPP sequences, i.e., six morphological interpretations.

3.4.2 Grammar Rule

Grammar rules in PDG are extended CFG rules, which define the possible phrase structures and mapping from the phrase structures to the corresponding dependency structures. Grammar rules are written in the following format.

y/Y $\rightarrow$ x$_1$/X_1,...,x$_n$/X_n : [arc($arcname_1$,X_i,X_j),...,arc($arcname_{n-1}$,X_k,X_l)] ($0< i,j,k,l \leq n$)
ex. vp/V $\rightarrow$ v/V,np/NP,pp/PP : [arc(obj,NP,V),arc(vpp,PP,V)]

A grammar rule consists of two parts separated by ":", the rewriting rule part and the dependency structure part. The left side of the rewriting rule "y/Y" and constituent "x$_i$/X_i" mean "syntactic category/variable." Y is a head constituent called a "phrase head" and is the same as one of the variables "$X_1 \ldots X_n$" in the "rule body". The dependency structure part is a set of arcs in the form "arc(arcname,variable1,variable2)"[*8]. A variable is bound to a WPP node, which is a phrase head of a constituent in the rewriting rule. In the example above, dependency structure where dependants NP and PP are connected to the governor phrase head V by means of the obj arc and the vpp arc, respectively. The dependency structure part constitutes a partial dependency tree, which satisfies the following "partial dependency structure conditions".

Definition 3.4.1 [Partial dependency structure condition]

(a) Partial dependency structure constitutes a tree structure whose top node is a phrase head of the head constituent Y. Phrase heads of non-head constituents are the dependants of the phrase heads of the other constituents.
(b) The phrase heads of the constituents in the rule body have one to one correspondence with the variables in the partial dependency structure.

Fig.3.7 shows the grammar rules and lexicons for analyzing an example sentence "Time flies like an arrow." Rule (R0) whose rule head and rule body are "root" (predefined special symbol) and "s"(starting symbol) as rule body is a special rule for creating a "root edge" of the phrase structure forest and a "top node" [top]-x of the dependency forest[*9].

3.4.3 The Structure of Edge

The syntactic analysis of PDG is implemented by extending the bottom-up chart-parsing algorithm to generate a dependency structure. Ordinal chart parser utilizes edges composed

[*8] The dependency structure is a set of arcs but represented by list format using [ ]. In this thesis, sets are sometimes represented by [ ] in program codes.
[*9] This rule is introduced for convenience in the treatment of data structures.

```
root/[root]-x → s/S     : [arc(top,S,[top]-x)]          (R0)
s/VP  → np/NP,vp/VP     : [arc(sub,NP,VP)]              (R1)
s/VP  → vp/VP           : []                            (R2)
np/N  → det/DET,n/N     : [arc(det,DET,N)]              (R3)
np/N  → n/N             : []                            (R4)
np/N2 → n/N1,n/N2       : [arc(nc,N1,N2)]               (R5)
np/NP → np/NP,pp/PP     : [arc(npp,PP,NP)]              (R6)
vp/V  → v/V             : []                            (R7)
vp/V  → v/V,pp/PP       : [arc(vpp,PP,V)]               (R8)
vp/V  → v/V,np/NP       : [arc(obj,NP,V)]               (R9)
vp/V  → v/V,np/NP,pp/PP : [arc(obj,NP,V),arc(vpp,PP,V)] (R10)
pp/P  → pre/P,np/NP     : [arc(pre,NP,P)]               (R11)

word(n,[time]). word(n,[flies]). word(pre,[like]). word(det,[an]).
word(v,[time]). word(v,[flies]). word(v,[like]).   word(n,[arrow]).
```

Fig.3.7 Grammar and lexicon for the example sentence

of five elements <FP,TP,C,FCS,RCS> , i.e., the from-position (FP), the to-position (TP), the category (C), the found constituent sequence (FCS) and the remaining constituent sequence (RCS). The head of the grammar rule corresponds to the category. The body of the grammar rule corresponds to both the found constituents and the remaining constituents and is partitioned by the dot (・) which shows the boundary of FCS and RCS as shown in the following edge written in diagrammatic form.

<0,1, s → np ・ vp pp>

This edge is generated from the grammar rule "s → np vp pp" and has elements FP=0,TP=1, C=s, FCS=[np] and RCS=[vp,pp]. The result of the dictionary look-up for an input word is an inactive edge whose category is the POS of the word and whose found constituent sequence is a word list as follows:

<0,1, n → [time]・>

The parsing algorithm of PDG has two extensions, i.e., the treatment for the dependency structure part in a grammar rule and the construction of the packed shared data structure. The edge for PDG parsing has two additional elements, i.e., the phrase head (PH) and the dependency structure (DS) as follows:

Standard edge : <0,1, s → np ・ vp pp>
PDG edge : <0,1, s/PH → np/n1 ・ vp/PH pp/PP : DS>

As described in 3.4.2, PH and DS represent a phrase head (node) and dependency structure (a set of arcs), respectively. n1 shows a node (WPP), which is a head of np phrase. PDG utilizes another data structure called the "packed edge", which is obtained by packing inactive edges into one. The packed edge has the list of FCS and the list of DS instead of the FCS and the DS in PDG edge. The PDG edge with FSC and DS is called "single edge" in contrast to packed edge. The packed edge is equivalent to a set of single edges. The following shows the relation between single edge and packed edge diagrammatically.

Single edge : <0,5, s/n2 → np/n1 vp/n2 pp/n3 ・: DS1>
<0,5, s/n2 → np/n1 vp/n2 ・: DS2>
Packed edge : <0,5, s/n2 → [[np/n1 vp/n2 pp/n3], [np/n1 vp/n2]] ・: [DS1,DS2]>

n1 to n3 are nodes (WPPs) and n2 is a phrase head. [np/n1 vp/n2 pp/n3] and [np/n1 vp/n2] are constituent sequences with their phrase head (nodes). DS1 and DS2 are dependency structures (partial dependency trees). For convenience, a packed edge is represented in the form "E," "<E ...>" or "edge E" and a single edge is represented in "e," "<e ...>" or "edge e." "edge" is used for representing "packed edge" or "single edge" when it is not ambiguous. Inactive edges are represented by adding "*" at the top of edge symbol. Edge *E and *e are an inactive packed edge and an inactive single edge, respectively.

The syntactic parsing of PDG described below utilizes packed edges. Fig.3.8 shows the formal constitution of a packed edge. A packed edge consists of eight elements. FCSL and DSL are lists (or sequences) with the same length. The pair (FCS_i,DS_i) obtained by extracting the i-th elements of FCSL and DSL is called "CSDS pair." CSDS pair corresponds to the single edge described above.

The edges E1 to *E3 in Fig.3.8 shows a growth of the edge generated form a grammar rule for noun phrase. The edge *E3 is the inactive edge showing the interpretation that the input words "an arrow" constitute a noun phrase and its dependency structure is {arc(det-14,[an]-det-3,[arrow]-n-4)}. [arrow]-n-4 is a node for word [arrow], pos "n" and word position 4. The edge *E4 is an edge with more than one interpretation. Each two elements in FCSL and DSL have

Structure of edge: <ID,FP,TP,C,PH,FCSL,RCS,DSL>
① ID:Edge ID, ②FP(From Position): Start position, ③TP(To Position): End position, ④C(Category):Head category of the grammar rule, ⑤PH(Phrase Head):Head node of the head constituent, ⑥FCSL: Found Constituent Sequence List(List of arc ID sequence), ⑦RCS:Remaining Constituent Sequence, ⑧DSL(Dependency Structure List): List of partial dependency structures

Partial dependency structure: $[\mathrm{ARC}_1,..,\mathrm{ARC}_n]$ (a set of arcs($n \geqq 0$))
Arc structure: arc(REL-AID,DEP-DP,GOV-GP)
①REL(Relation):Dependency relation name, ②AID:Arc ID, ③DEP(Dependant):Dependant node, ④DP(Dependant Position) : Position of DEP, ⑤GOV(Governor):Governor node, ⑥GP(Governor Position):Position of GOV

Examples of edges:
Rule: np/N → det/DET, n/N : [arc(det,DET,N)]
```
   Edge  E1:<45,3,3,np,$2,[],[det/$1,n/$2],[[arc(det,$1,$2)]]>
   Edge  E2:<70,3,4,np,$2,[[153]],[n/$2], [[arc(det,[an]-det-3,$2)]]>
   Edge *E3:<160,3,5,np, [arrow]-n-4, [[153,156]],[],[[arc(det-14,[an]-det-3,[arrow]-n-4)]]>
```
Rule: vp/V→v/V,np/NP :[arc(obj,NP,V)]
vp/V→v/V,np/NP,pp/PP :[arc(obj,NP,V),arc(vpp,PP,V)]
```
   Edge *E4:<170,0,5,vp,[time]-v-0,[[103,169],[103,119,165]],[],
            [[arc(obj-25,[flies]-n-1,[time]-v-0)],
             [arc(obj-4,[flies]-n-1,[time]-v-0),arc(vpp-20,[like]-pre-2,[time]-v-0)]]>
```
Lexicon: word(n,[arrow])
```
   Edge @E5:<156,4,5,n,[arrow]-n-4,[[lex([arrow]-n)]],[],[[[arrow]-n-4]]>
```

Fig.3.8 Structure of edge

correspondence and constitute two CSDS pairs, i.e., ([103,169] {obj-25})[*10] and ([103,119,165] {obj-4,vpp-20}). The edge @E5 is an example of an edge generated from dictionary look-up operation, called a "lexical edge". The data structure of a lexical edge is a set of one node corresponding to the consulted word. The lexical edge is explicitly represented by adding "@."

3.4.4 Parsing Algorithm

Fig.3.9 shows the parsing algorithm of PDG. Basically, this algorithm is a standard bottom-up chart-parsing algorithm using agenda (Winograd, 1983). This algorithm inputs words from left to right one by one and adds lexical edges generated from the input words to the agenda (Fig.3.9 (a),(b)) , and combines the edges in the agenda to inactive edges in the chart or in the grammar rules until the agenda becomes empty (Fig.3.9 (e),(f)). Packed edges are generated by checking

```
Sent:Input Word List, Grammar: A Set of Grammar Rules
Chart := {}; Agenda := {}; /* Initialize */
for(Pstn:=0; Pstn < length(Sent); Pstn++) {      /* (a) */
 Agenda := get_lex_edges(Pstn,Sent);             /* (b) */
 while(Agenda != {}){       /* Repeat edge combination */
   A_Edge := pop(Agenda);  /* Take one edge from Agenda */
   /* Take out an edge mergable with A_Edge from Chart  */
   C_Edge := mergable_edge(A_Edge);              /* (c) */
   if( C_Edge != new ) {       /* Mergable edge exists? */
     MrgdEdge = merge_csds(A_Edge,C_Edge);       /* (d) */
     /* If changed by merge, update C_Edge to MrgdEdge  */
     if(MrgdEdge != C_Edge) { update_chart(MrgdEdge); } }
   else if( C_Edge == new ) {
     push(A_Edge,Chart); /* Register A_Edge to the Chart*/
     /* Register new edge from grammar to Chart/Agenda  */
     add_new_edge_by_searching_grammar(A_Edge);  /* (e) */
  /* Register new edge from active edge to Chart/Agenda */
     add_new_edge_by_existing_edge(A_Edge); /* (f) */} } }

add_new_edge_by_searching_grammar(Edge) {
 Edge = <ID, FP, TP, C, PH, FCSL, RCS, DSL>;
 foreach Rule in Grammar {
   /* Get elements in a grammar rule (Body length m)    */
   Rule = <Y/Yv, [X1/X1v,X2/X2v,..,Xm/Xmv], RDS>;
   if( X1 == C ) {                  /* same category? */
     /* Variable Binding: Bind X1v to phrase head PH (g)*/
     RDS := bind_var(X1v,PH,RDS);
     NewYv := bind_var(X1v,PH,Yv);
     NewEdge:=<new_id(),FP,TP,Y,NewYv,[[ID]],
               [X2/X2v,..,Xm/Xmv],[RDS]>;
     if( NewEdge is enactive edge? ) { push(NewEdge,Agenda); }
     else { push(NewEdge,Chart); } } } }

add_new_edge_by_existing_edge(Edge) {
 Edge = <ID, FP, TP, C, PH, FCSL, RCS, DSL>;
 foreach Edge_C in Chart {
   Edge_C = <IDc, FPc, TPc, Cc, PHc, FCSLc, RCSc, DSLc>;
   RCSc = [X1/X1v, X2/X2v,..,Xm/Xmv];  /* Get elements  */
   /* Combination Conditions: ①Edge_C is active edge   */
   /* ②First category X1 in RCSc equals category C     */
   /* ③Edge_C is adjacent to Edge                      */
   if( RCSc != [] && X1 == C && FP == TPc ) {
     /* === Generate NewEdge from Edge_C and Edge ===   */
     NewFCSL := add_id_to_FCSL(ID,FCSLc);
     NewDSL := bind_var(X1v,PH,DSLc); /* Var. Binding (h)*/
     NewDSL := add_arcid(NewDSL);   /* Put new arc ID (i)  */
     NewEdge:=<new_id(),FPc,TP,Cc,PHc,NewFCSL,
               [X2/X2v,..,Xm/Xmv],NewDSL>;
     if( NewEdge is enactive edge? ) { push(NewEdge,Agenda); }
     else { push(NewEdge,Chart); } } } }

mergable_edge(A_Edge) {
 foreach C_Edge in Chart {
  if(A_Edge and C_Edge have the same "phrase coverage(FP,TP)",
    "category(C)", "phrase head(PH)","RCS")       /* (j)  */
   { return(C_Edge); }}
 return(new); }

--------------- Funcions & notations ----------------
D = S     : Unification of D and S
X ∪ Y     : Union of X and Y. length(L): length of list L.
new_id()  : Unique ID generator.
push(E,X): Push element E to set X
pop(X)    : Take one element from set X and returns it.
add_arcid(DSL): Put new ID to the fixed arc(s) in DSL
  ex. When DSL=[[arc(subj,[i]-n-0,[love]-v-1),
                 arc(obj,$3,  [love] -v-1)]]
  add_arcid(DSL)→[[arc(subj-23,[i]-n-0,[love]-v-1),
                   arc(obj,$3,[love]-v-1)]]
add_id_to_FCSL(ID,FCSL): Add arc ID to each element in FCSL.
  ex. When ID=29, FCSL=[[23,12],[11]],
  add_id_to_FCSL(ID,FCSL) → [[12,23,29],[11,29]]
bind_var(Var,Val,DS): Replace Var in DS with Val
  ex. When Var=$1,Val=a(x),DS=[b($1),c($2)],
  bind_var(Var,Val,DS) → [b(a(x)),c($2)]
update_chart(E): Update E to edge with the same ID in the Chart
get_lex_edges(FP,S): Generate a lexical edges by consulting a
    dictionary at position FP in word list S.
   ex. When FP=1,S=[I,love,you], returns
   {<10,2,3,v,[love]-v-2,[[lex([love]-v)]],[],[[]]>,
    <11,2,3,n,[love]-n-2,[[lex([love]-n)]],[],[[]]>}
merge_csds(E1,E2)  : Add CSDS pair in edge E1 to E2.
 ex.E1=<8,3,5,s,$1,[[17]],[vp/$1],[[arc(obj-3,[i]-n-2,$1)]]>,
    E2=<9,3,5,s,$1,[[13]],[vp/$1],[[arc(subj-5,[i]-n-2,$1)]]>
  merge_csds(E1,E2)→<9,3,5,s,$1,[[13],[17]],[vp/$1],
      [[arc(subj-5,[i]-n-2,$1)],[arc(obj-3,[i]-n-2,$1)]]>
 If CSDS pair in E1 is in E2, then it is not added.
 E2=<7,3,5,s,$1,[[13]],[vp/$1],[[arc(obj-5,[i]-n-2,$1)]]>
 then,merge_csds(E1,E2)→E2
```

Fig.3.9 PDG bottom-up chart parsing algorithm

[*10] obj-25 is the abbreviation of arc(obj-25,[flies]-n-1,[time]-v-0)

each edge in the agenda is mergable or not ((c),(j)) and then merging it to the existing edge if possible. The detailed explanation of the algorithm is omitted. The following part explains the construction of the data structure, which is peculiar to the PDG parsing.

The PDG parser creates dependency structures in parallel with the generation of edges. This is done by binding variables in the dependency structures in an edge. Variable binding is performed by *bind_var*, which binds the phrase head (node) of the inactive edge to the variable in the first constituent of the remaining constituent sequence of the edge, when a new edge is created from a grammar rule (Fig.3.9 (g)) or an active edge in the agenda (Fig.3.9 (h)). If this binding generates an arc whose dependant and governor are bound, *add_arcid* generates a unique arc-ID and it is attached to the arc (Fig.3.9 (i)). The arc with a bound dependant and governor is called "fixed arc". The edge *E3 in Fig.3.8 is generated by binding the variable $2 in E2 to the node [arrow]-n-4.

Edges are associated through edge-IDs. The lower edge can be traced from the upper edge. The edge *E3 (edge#160) in Fig.3.8 is an edge generated from the grammar rule "np $\rightarrow$ det n." The edges in the constituent sequence [153,156] in edge#160 , i.e., edge#153 and edge#156, have the phrase category "det" and "n," respectively. The edge#153 and edge#156 are called reachable from the edge#160. This "reachable" relation is associative. Edges with more than one CSDS pair like *E4 in Fig.3.8 are generated by *merge_csds*(Fig.3.9 (d)). Since only inactive edges are merged, no active edge has more than one CSDS pair in this algorithm. If the whole sentence is parsed successfully, the chart has one inactive edge with phrase head [top]-x covering whole sentence. This edge is called "root edge" and described as *E_{root}.

3.4.5 Generation of Phrase Structure Forest and Initial Dependency Forest

When parsing is finished, the chart has active and inactive edges. The phrase structure forest for an inactive edge *E, hpf(*E), is defined as a set of edges reachable from the edge *E. The phrase structure forest PF is defined as hpf(*E_{root}). PF is a subset of the inactive edges in the chart because there exist inactive edges unreachable from *E_{root}. The initial dependency graph IDG is a set of arcs in the dependency structures of the edges in PF. Fig.3.10 shows the algorithm to compute PF, IDG and the initial C-Matrix ICM from *E_{root}. Fig.3.11 shows PF for the example sentence computed by the algorithm using the grammar in Fig.3.7. All RCSL of the edges in PF are [ ] and are not shown in Fig.3.11. The number of edges in the phrase structure forest is called a size of the phrase structure forest. The size of the headed phrase structure forest is more than or equal to that of the headless phrase structure forest because the edge merge condition of the headed phrase structure forest (Fig.3.9 (j)) is more strict compared with that of the headless phrase structure forest.

The algorithm in Fig.3.10 traverses the chart by using three mutually recursive functions, *try_edge*, *try_FCSL* and *try_CS* which compute PF, IDG and ICM for their arguments, i.e., the packed edge, the constituent sequence list and the constituent, respectively. *try_edge* calls

```
/* Phrase Structure Forest PF, Dependency Graph DG,
/* Co-occurrence Matrix CM                                    */
PF  := {}; DG := {}; CM := {};                 /* Initialize */
VE  := {};           /* Visited Edge: Already tried edges */
TER := {}; /* Memory for edge ID and arcs governed by it*/
try_edge(E_root);       /* Compute PF,DG,CM from E_root  (a) */
exit;                           /* Results are PF,DG,CM   */

% Compute PF,DG,CM from E. Returns arcs governed by E
try_edge(E) {
 E = <ID,FP,TP,C,PH,FCSL,RCS,DSL>;          /* Get elements */
 if(ID∈VE) { return(get(ID,TER)); }/* Done before?  (b) */
 else if( Is E lexical edge? ) {                   /* (c) */
   push(E,PF); put(ID,{},TER); return({}); }
 A_FCSL :=try_FCSL(FCSL,DSL); /* Process CSDS pairs  (d) */
 push(E,PF);                         /* Add E to PF   (e) */
 A_Edge := arcs(DSL) ∪A_FCSL; /* Arcs governed by E  (f) */
 put(ID,A_Edge,TER); /* Register computation result  (g) */
 return(A_Edge); }

% Compute PF,DG,CM from FCSL. Returns arcs governed by FCSL
try_FCSL(FCSL,DSL) {
 A_FCSL := {}; /* Initialize */
 foreach (CS,DS) in (FCSL,DSL){ /* Get CSDS pair(CS,DS) */
   A_CS := try_CS(CS);               /* Process one CS(h) */
   set_CM(DS,DS);                /* CM processing (1)  */
   set_CM(DS,A_CS);              /* CM processing (2)  */
   A_FCSL := A_FCSL ∪ A_CS; }/* Computes arcs(FCSL) (i) */
 return(A_FCSL); }       /* Returns arcs governed by FCSL*/

% Compute PF,DG,CM from CS. Returns arcs governed by CS
try_CS(CS) {
 A_CS := {}; /* Initialize */
 foreach E in CS {
   A_Edge := try_Edge(E);          /* Compute edge E (j) */
   set_CM(A_Edge,A_CS);            /* CM processing  (3) */
   A_CS := A_CS ∪ A_Edge; }    /* Compute arcs(CS)  (k) */
 return(A_CS); }          /* Returns arcs governed by CS */

--- Functions & notations ---
put(I,E,X): Add element E to set X with index I
get(I,X)   : Returns the element with index I in X
set_CM(A1,A2) : Set ○ to CM(A1,A2) and CM(A2,A1) where A1,A2
  are arc list/set. ex. When A1={a1,a2},A2={a3,a4}, a1～a4 has
   IDs id1～id4, set_CM(A1,A2) sets ○ to CM(id1,id3),
 CM(id1,id4),CM(id2,id3),CM(id2,id4),CM(id3,id1),
 CM(id4,id1), CM(id3,id2),CM(id4,id2) (=Add to CM)
arcs(X)    : Returns arcs in data structure X.
```

Fig.3.10 Algorithm for computing phrase strucuture forest and initial dependency forest

try_FCSL (Fig.3.10 (d)), *try_FCSL* calls *try_CS* (Fig.3.10 (h)) and *try_CS* calls *try_edge* (Fig.3.10 (j)). The arc in the arc sets returned from *try_edge*(E),*try_FCSL*($FCSL$) and *try_CS*(CS) are called the arc governed by E, $FCSL$ and CS, respectively.

The algorithm starts from the Fig.3.10 (a) by calling *try_edge*(*E_{root}). *try_edge* judges the argument is already computed or not at (b). If it has already been computed, *try_edge* simply returns the set of arcs recorded in TER. The registration of a set of arcs is performed in (g) when new result is obtained. At (c) and (e), new edges are added to PF. As shown at (f), the arcs governed by the edge E are the union of the DSL in E and the arcs governed by FCSL.

try_FCSL processes a set of CSDS pairs and *try_CS* processes one CSDS pair in it. As shown at (i), the set of arcs governed by FCSL is the union of the arcs governed by the CSs in the FCSL. As shown at (k), the set of arcs governed by CS is the union of the arcs governed by the packed edges in the CS.

Fig.3.12 shows the execution process for the E#170 in Fig.3.11. (c#) shows a function call and (r#) shows the return value, i.e., the set of arcs, of the function. (c1) to (c4) correspond to the function calls (j), (d) and (h) in Fig.3.10, respectively. (c4) returns {} at (r4) because the E#103 is a lexical edge. The function call (c4) returns result (r4). Then the second CSDS pair ([103,119,165],{obj-4,vpp-20}) is processed by the function call (c5). The second time execution for "*try_edge*(E#103)" occurs at (c6). This time, the execution result stored in TER at Fig.3.10 (b) is searched and returned. Finally the set of arcs at (r1) is obtained.

The generation of CM is performed based on the following C-Matrix setting conditions which

```
Time flies like an  arrow
0   1    2    3  4      5
<186,0,5,root,[top]-x-top,[[181],[176],[174]],  [[arc(top-32,[time]-v-0,[top]-x-top)],
                                                  [arc(top-31,[flies]-v-1,[top]-x-top)],
                                                  [arc(top-29,[like]-v-2,[top]-x-top)]]>
<181,0,5,s,  [time]-v-0,  [[170]],              [[]]>
<176,0,5,s,  [flies]-v-1, [[105,168]],          [[arc(sub-24,[time]-n-0,[flies]-v-1)]]>
<174,0,5,s,  [like]-v-2,  [[121,166]],          [[arc(sub-23,[flies]-n-1,[like]-v-2)]]>
<170,0,5,vp, [time]-v-0,  [[103,169],[103,119,165]],
                                                 [[arc(obj-25,[flies]-n-1,[time]-v-0)],
                                                  [arc(obj-4,[flies]-n-1,[time]-v-0),
                                                   arc(vpp-20,[like]-pre-2,[time]-v-0)]]>
<169,1,5,np, [flies]-n-1, [[119,165]],          [[arc(npp-19,[like]-pre-2,[flies]-n-1)]]>
<168,1,5,vp, [flies]-v-1, [[117,165]],          [[arc(vpp-18,[like]-pre-2,[flies]-v-1)]]>
<166,2,5,vp, [like]-v-2,  [[140,160]],          [[arc(obj-16,[arrow]-n-4,[like]-v-2)]]>
<165,2,5,pp, [like]-pre-2,[[138,160]],          [[arc(pre-15,[arrow]-n-4,[like]-pre-2)]]>
<160,3,5,np, [arrow]-n-4, [[153,156]],          [[arc(det-14,[an]-det-3,[arrow]-n-4)]]>
<156,4,5,n,  [arrow]-n-4, [[lex([arrow]-n)]],   [[[arrow]-n-4]]>
<153,3,4,det,[an]-det-3,  [[lex([an]-det)]],    [[[an]-det-3]]>
<140,2,3,v,  [like]-v-2,  [[lex([like]-v)]],    [[[like]-v-2]]>
<138,2,3,pre,[like]-pre-2,[[lex([like]-pre)]],  [[[like]-pre-2]]>
<121,0,2,np, [flies]-n-1, [[101,115]],          [[arc(nc-2,[time]-n-0,[flies]-n-1)]]>
<119,1,2,np, [flies]-n-1, [[115]],              [[]]>
<117,1,2,v,  [flies]-v-1, [[lex([flies]-v)]],   [[[flies]-v-1]]>
<115,1,2,n,  [flies]-n-1, [[lex([flies]-n)]],   [[[flies]-n-1]]>
<105,0,1,np, [time]-n-0,  [[101]],              [[]]>
<103,0,1,v,  [time]-v-0,  [[lex([time]-v)]],    [[[time]-v-0]]>
<101,0,1,n,  [time]-n-0,  [[lex([time]-n)]],    [[[time]-n-0]]>
```

Fig.3.11 Phrase structure forest for "time flies like an arrow"

work to allow co-occurrence between all arcs in the set of edges constituting a phrase structure tree in the phrase structure forest.

Definition 3.4.2 [C-Matrix setting conditions]
The "C-Matrix setting condition" is either of the following three conditions

(CM1) The arcs in the same DS co-occur with one another.
(CM2) Given a CSDS pair (CS,DS), the arcs in DS co-occur with the arcs governed by CS.
(CM3) The arcs governed by one CS co-occur with one another.

(CM1) to (CM3) correspond to the CM processing (1) to (3) in Fig.3.10. In processing E#170, set_CM({obj-4,vpp-20},{obj-4,vpp-20}) (set_CM is defined in Fig.3.10) is executed at the CM processing (1) because the arcs in the second CSDS pair ([103,119,165],{obj-4,vpp-20}) satisfy the co-occurrence setting condition (CM1). At the CM processing (2), A_CS has been set to the set of arcs shown at (r5) in Fig.3.12 and set_CM({obj-4,vpp-20},{pre-15,det-14}) is executed due to (CM2). In processing try_CS([103,119,165]), the CM processing (3) set CM from among the arcs governed by E#103,E#119 and E#165 due to (CM3). The outputs PF and IDF are shown in Fig.3.11 and Fig.3.4, respectively. E#181, E#176 and E#174 have the same category "s" and the same coverage (from 0 to 5), but they are not shared because their phrase heads are different.

```
try_edge(E#170)                                      (c1)
  try_FCSL([[103,169],[103,119,165]],
           [[obj-25],[obj-4,vpp-20]])                (c2)
    try_CS([103,169])                                (c3)
      try_edge(E#103) ⇒ { }                          (c4) ⇒ (r4)
              :
    ⇒ { npp-19,pre-15,det-14 }                       (r3)
    try_CS([103,119,165])                            (c5)
      try_edge(E#103) ⇒ { }                          (c6) ) ⇒ (r6)
              :
      try_edge(E#119) ⇒ { }
              :
      try_edge(E#165) ⇒ { pre-15,det-14 }
    ⇒ { pre-15,det-14 }                              (r5)
  ⇒ { pre-15,det-14,obj-4,vpp-20,npp-19,obj-25 }     (r2)
⇒ { pre-15,det-14,obj-4,vpp-20,npp-19,obj-25 }       (r1)
```

Fig.3.12 Example of algorithm execution

3.5 Generation of the Reduced Dependency Forest

The two arcs obj4 and obj25 in IDF in Fig.3.4 have the same structure except for their arc-IDs. IDF may contain arcs of this kind called "equivalent arcs". Equivalent arcs are sometimes generated from one grammar rule and sometimes generated from different grammar rules. For example, obj4 and obj25 are generated from the constituent sequence "vp np" in (R9) and (R10) in Fig.3.7. In fact, (R9) and (R10) differ in terms of the existence of the prepositional phrase, but the interpretation of the "obj" relations in (R9) and (R10) are considered to be the same. Actually, if (R9) and (R10) are merged into one grammar rule "vp/V $\rightarrow$ v/V,np/NP, $\{$pp/$PP\}$: [arc(obj,NP,V),arc(vpp,PP,V)]" by introducing the description schema "{}" for optional elements. This grammar rule does not generate the equivalent arcs for obj relation.

Now, some definitions are given for treating equivalent arcs. The "generalized arc" is an arc with arc-ID '?'. Arcs with number IDs are called IDed arcs. The generalized arc for an IDed arc is obtained by simply replacing the arc-ID in the IDed arc with '?'. A dependency tree consisting of generalized arcs is called a "generalized dependency tree". A dependency tree consisting of IDed arcs is called an "IDed dependency tree". The generalized arc for an IDed arc X is written as ?X. The generalized tree for an IDed tree DT is written as ?DT. Two dependency trees that have the same generalized tree are called equivalent. The reduced dependency forest is obtained by reducing the initial dependency forest. The reduction of the dependency forest is an operation in that more than one equivalent arc is merged into one arc without increasing the number of the generalized dependency trees in the dependency forests. The reduced dependency has smaller size compared with the original dependency forest before the merge operation.

3.5.1 Merge Operation of Equivalent Arcs

The merge operation for the equivalent arcs X,Y (written in equiv(X,Y)) is defined as follows:

Definition 3.5.1 [Arc merge operation]

(1) Compute a new dependency graph DG' by removing Y from DG. (DG'=DG$-\{Y\}$)
(2) Generate a new C-Matrix CM' from CM by applying $set\_CM(X, I)$ for arc I(I∈DG,I≠X, I≠Y,CM(Y,I)=○)

The merge operation generates a new dependency forest <DG',CM'>. Fig.3.13 shows an example of merge operation diagrammatically. In the following sections, changes of various values are discussed, for example, the numbers of generalized dependency trees in the dependency forest before and after merge operation. To make this distinction, the expression "wrt <DG,CM>" or "wrt DF" (wrt: with respect to) is used. For example, the set of arcs that co-occurs with an arc A is defined as co(A). Then "co(A) wrt <DG,CM>" and "co(A) wrt <DG',CM'>" represent the set of arcs before and after merge operation. "co(A) wrt <DG,CM> = co(A) wrt <DG',CM'>" means the set of arcs is not changed by the merge operation. In order to make the description simple, "wrt <DG,CM>" is not shown in default.

3.5.2 Merge Condition for Equivalent Arcs

From the definition, the condition of the dependency forest reduction is to preserve the soundness, i.e., no new generalized dependency tree (interpretation) increase by the merge operation.

[**Merge Condition for Equivalent Arcs**]
When the dependency forest DF' is generated from DF by merging arc Y to X in the dependency graph of DF, the condition for the forest reduction is "a set of generalized dependency trees in DF = a set of generalized dependency trees in DF' ."

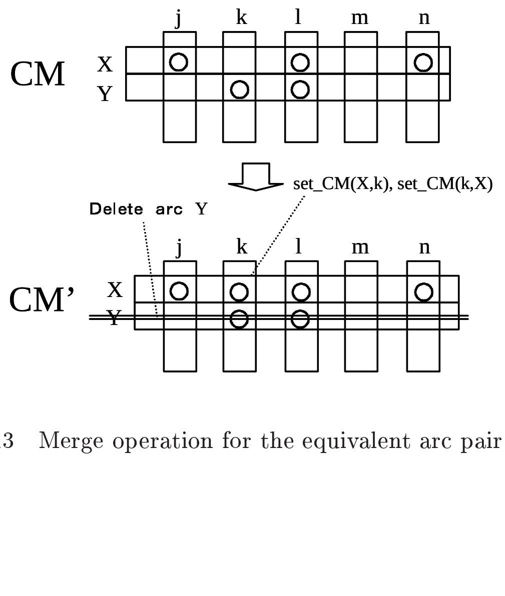


Fig.3.13 Merge operation for the equivalent arc pair (X,Y)

This condition is verifiable by searching a new generalized dependency tree in DF'. The condition for the existence of a generalized dependency tree not in DF but in DF' is called the "increase condition for generalized dependency trees" (ICG) in this thesis. The merge condition is represented as "ICG is not satisfied."

A dependency forest is a set of IDed arcs and prescribes the set of IDed dependency trees and the set of generalized dependency trees. The condition for the existence of an IDed dependency tree not in DF but in DF' (this kind of tree is called a "new" dependency tree) is called the "increase condition for IDed dependency trees" (ICI). Obviously, if no IDed dependency tree increases by the merge operation, no generalized dependency tree increases. Moreover, even if there exist new IDed dependency trees in DF', no generalized dependency trees increase if the generalized dependency trees of the new IDed dependency trees are equivalent to those in DF. This means ICI is a necessary condition for ICG. In the following, ICI is discussed first, then ICG is verified to obtain a detailed merge condition for equivalent arcs.

3.5.3 Increase Condition for IDed Dependency Trees

Increase of a new IDed dependency tree is caused by allowing a new co-occurrence relation between arcs caused by a merge of equivalent arcs. The allowance of the co-occurrence relation in CM, i.e., the change from CM(U,V)$\neq \bigcirc$ to CM(U,V)$= \bigcirc$ for arcs U, V, is called the "allowance of the arc pair (U,V)". The following lemma is established.

Lemma 3.5.1 (The allowance of arc pair and the increase of dependency trees) If a new well-formed dependency tree increases by the allowance of the arc pair (U,V), the dependency tree includes U and V.

Proof: This lemma is obvious because well-formed dependency trees in DG' which do not include both U and V exist in DG. $\square$

Here, uniq and diff are the sets of arcs defined for the equivalent arcs X and Y as follows:

$\text{uniq(X,Y)}=\{I \mid \text{CM}(X,I)=\bigcirc, \text{CM}(Y,I)\neq \bigcirc, I\in \text{DG}\}$
$\text{diff(X,Y)}=\{(I,J) \mid I\in\text{uniq(X,Y)}, J\in\text{uniq(Y,X)}\}$

For the arcs X, Y in Fig.3.13, uniq(X,Y)={j,n},uniq(Y,X)={k}and diff(X,Y)={(j,k),(n,k)}. The following lemma is established.

Lemma 3.5.2 (Arcs in a new well-formed dependency tree) In the case that a well-formed dependency tree is generated by the merge of the equivalent arcs X,Y, the new tree includes at least two arcs A,B such that (A,B) $\in$ diff(X,Y).

Proof: Let DF and DF' be the dependency forests before and after the merge of X and Y. Assuming that a new dependency tree DT$_x$ is obtained by the allowance of the arc pair (X,B$_i$) caused by the merge of X and Y, X and B$_i$ are included in DT$_x$ according to the lemma 3.5.1.

Here, let R=DT$_x$−{X,B$_i$}. CM(X,U)= ○ wrt DF',CM(B$_i$,U)=○ wrt DF' for U∈R because DT$_x$ is a well-formed dependency tree.

Assuming that there is no arc U such that CM(Y,U)≠ ○ wrt DF (i.e., CM(Y,U)= ○ wrt DF,U∈R), DT$_y$={Y,B$_i$}+R is a well-formed dependency tree in DF. DT$_x$ is not a new generalized dependency tree because DT$_x$ and DT$_y$ differ only in the equivalent arc X and Y, i.e., DT$_x$ and DT$_y$ are equivalent. Therefore, DT$_x$ must include at least one U$_i$ such that CM(Y,U$_i$)≠ ○ wrt DF is a new generalized dependency tree. This lemma is established because (B$_i$,U$_i$)∈diff(X,Y).

□

The following theorem is derived from lemma 3.5.2,

Theorem 3.5.1 (The increase condition for IDed dependency tree)
Let arc pair (A,B)∈diff(X,Y) for equivalent arcs X,Y in DG of the dependency forest <DG,CM>. The increase of IDed dependency trees occurs if and only if <DG',CM'> obtained by the merge of Y to X have IDed dependency tree NDT which includes {X,A,B}.

Proof: This theorem is proved by showing a new well-formed IDed dependency tree includes {X,A,B} and a well-formed IDed dependency tree which includes {X,A,B} is a new well-formed IDed dependency tree. Assuming that NDT is a new IDed dependency tree existing in <DG',CM'>, there exists at least one arc pair (A$_i$,B$_i$)∈diff(X,Y),A$_i$∈NDT,B$_i$∈NDT. On the other hand, X∈NDT is true due to lemma 3.5.1. Therefore, a new well-formed IDed dependency tree includes {X,A,B}. Moreover, no IDed well-formed dependency trees exist in <DG,CM> because (A,B)∈diff(X,Y). Therefore, an IDed well-formed dependency tree which includes {X,A,B} is a new well-formed IDed dependency tree.

□

Some functions and notations are introduced for the discussion on the increase condition of dependency trees.

same_position(U,V) : The positions of dependant nodes of U and V are the same.
dts(S) wrt $<DG,CM>$: a set of IDed well-formed dependency trees which consist of arcs in arc set $S \subset DG$ and satisfy the arc co-occurrence constraint
co(U) wrt $<DG,CM>$: a set of arcs which co-occur with arc U including U, i.e., $\{X \mid X=U$ or CM(X,U)=○, $X \in DG\}$
dts_with_arcs(A_1,A_2,…,A_n) wrt $<DG,CM>$: a set of the well-formed dependency trees in $<DG,CM>$ which include arcs A_1,A_2,…,A_n, i.e., dts(co(A_1)∪⋯∪co(A_n)) wrt $<DG,CM>$

ICI with respect to the arc pair (A,B) ∈ diff(X,Y), equiv(X,Y) can be checked by searching the existence of a well-formed dependency tree including {X,A,B} in <DG',CM'> according to theorem 3.5.1. To make this search process more efficient, the following three cases with respect to arc X, A and B are considered.

(RC1) Any of same_position(A,B), same_position(X,A) or same_position(X,B) is true
(RC2) CM(A,B)$\neq$ ○ is true
(RC3) Except for (RC1) and (RC2)

In cases (RC1) and (RC2), no well-formed dependency trees which include {X,A,B} exist in <DG',CM'> because of the existence of the single role constraint and the co-occurrence constraint, respectively. In the case of (RC3), the existence of a well-formed dependency tree which includes {X,A,B}, i.e., dts_with_arcs(X,A,B) wrt <DG',CM'>={}, should be checked for ICI.

3.5.4 Increase Condition for Generalized Dependency Trees

As described above, ICI is a necessary condition for ICG. Therefore, ICG is defined as follows:

[**The increase condition for generalized dependency tree**]
Let DF' be a dependency forest generated from the dependency forest DF by merging arc Y to X where X and Y are equivalent arcs and let DT_{new} be the set of IDed dependency trees which are in DF' but not in DF. There exists at least one IDed dependency tree DT $\in$ DT_{new} such that the generalized dependency tree ?DT is not included in DF.

The merge condition for equivalent arcs is the negation of ICG for the arcs.

3.5.5 Dependency Forest Reduction Algorithm

Fig.3.14 shows the dependency forest reduction algorithm for <DG,CM> based on the merge condition for equivalent arcs (i.e., the condition for the forest reduction) described in the previous section. In this algorithm, CM is represented as a set of co-occurable arc pairs. Arc X and Y are co-occurable if <X,Y>$\in$CM.

Fig.3.14 (a) picks up a pair of the equivalent arcs X,Y in DG, checks if a new generalized dependency tree is generated in the dependency forest when arc pair (A,B) is allowed from (b) to (h) in Fig.3.14. If all arc pairs in $diff$(X,Y) do not generate any new generalized dependency trees, the forest reduction is performed at (i).

The availability of the allowance of (A,B) is determined by checking ICG after checking ICI. At Fig.3.14 (b), conditions (RC1) and (RC2) in Section 3.5.3 are checked. If either of the conditions is satisfied, the processing proceeds to the next arc pair in $diff$(X,Y) because the allowance of (A,B) generates no new IDed dependency trees. If not, the processing proceeds to the check of ICI. At (c), <DG',CM'> is generated by merging Y to X. Since the existence check of a new IDed tree is basically performed by tree search for <DG',CM'>, the reduction of search space improves the efficiency. Based on theorem 3.5.1, the search space is reduced from DG' to DG_XAB(=$co(X)$∩$co(A)$∩$co(B)$) at (d). Then *search_dt* at (e) searches a new IDed

```
DG_init: Initial Dependency Graph, CM_init: Initial CM, wn: Number of Input Words
/* Initialization */
DG:=DG_init; CM:=CM_init; DG':={}; CM':={};
while(Get one unprocessed arc pair X,Y) {           /* (a) */
  Result := mergable;
  foreach (A,B) in diff(X,Y) {                      /* Process arc pairs for merging Y with X */
    /* (b) Check the increase condition for IDed dependency trees: (RC1),(RC2) */
    if( same_position(X,A)||same_position(A,B)||same_position(X,B)||¬(<A,B>∈CM)) { next; }
    CM':= CM∪add_cm(X,Y,DG,CM); DG':=DG-{Y};        /* (c) Generate DG',CM' */
    DG_XAB := (co(X)∩co(A)∩co(B) wrt DG',CM');      /* (d) Set of arcs co-occurring with X,A,B */
    while((DT:=search_dt(DG_XAB,CM',0,{}))!=false) { /* (e) Search DTs for DG_XAB (RC3) */
      if(new_generalized_dt(DT,CM,DG) == true)       /* (f) Check increase condition fro generalized
                                                            dependency trees */
        { Result:=unmergable; break; } }            /* (g) */
    if(Result == unmergable) { break; } }            /* (h) */
  if(Result==mergable) { CM:=CM'; DG:=DG';} }       /* (i) Merge X and Y (forest reduction)  */
% Search dependency trees for DG,CM
search_dt(DG,CM,P,SA) {
 if(P ≧ wn) { return({}); }                          /* (j) Has selected arcs in all position */
 foreach Arc in arcs_at(DG,P) {                      /* (k) Select one arc at position P */
  if(inconsistent(Arc,SA,CM) == true) { next; }      /* (l) Check co-occurrence condition */
  Result := search_dt(DG,CM,P+1,SA∪{Arc});           /* (m) Try the next position */
  if( Result == false) { next; }                     /* (n) */
  else {return({Arc}∪Result);} }                     /* (o) */
 return(false); }                                    /* (p) */

% Check a new IDed dependency tree DT if it is new as a generalized tree in CM,DG
new_generalized_dt(DT,CM,DG) {
  DG_X := add_equiv_arcs(DT,CM,DG);                  /* (q) Add equivalent arcs of arcs in DT */
  if(search_dt(DG_X,CM,0,{})==false) {return(true);} /* (r) Search DT for DG_X,CM */
  else { return(false); } }

% ==== functions ====
add_cm(X,Y,DG,CM) : Returns {<X,I>|I∈uniq(Y,X);}∪{<I,X>|I∈uniq(Y,X)} wrt DG,CM

inconsistent(A,S,CM) : Given arc A, set of arcs S, constraint matrix CM, returns false if ∃X X∈S,
      ¬(<A,X>∈CM), otherwise returns true.

arcs_at(DG,P) : A set of arcs with position P in DG

add_equiv_arcs(DT,CM,DG) : Add equivalent arcs of each arc in dependency tree DT
  ex. When DT=[1,5], the equivalent arcs of 1 and 5 are [2,3] and [] (in CM,DG) respectively
      add_equiv_arcs([1,5],CM,DG) → [1,2,3,5]
```

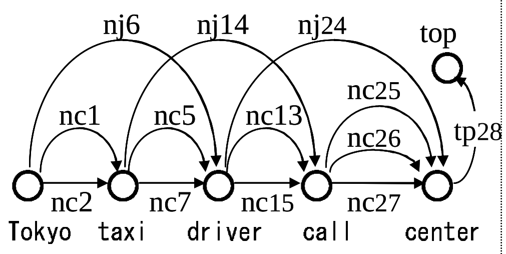
Fig.3.14 Algorithm for reduction of dependency forest

dependency tree for DG_XAB. If no dependency tree is obtained, allowance of (A,B) satisfies the merge condition for equivalent arcs. If a dependency tree DT is obtained, DT is an IDed dependency tree, which includes {X,A,B}. *new_generalized_dt*(DT,CM,DG) at (f) checks if the generalized dependency tree ?DT is new or not new by searching ?DT for <DG,CM>. The detailed explanation of *new_generalized_dt* is omitted here, but it realizes the search for the generalized dependency tree by limiting the arc set DG_X so that it has only the equivalent arcs of the arcs in DT by *add_equiv_arcs* at (q). When ?DT is a new generalized dependency tree, the merge between X and Y is not available. The processing for arcs X and Y is terminated by (g) and (h), and proceeds to the check of the next equivalent arc pair. If ?DT is not new, the merge of X and Y, i.e., the forest reduction, is performed at (i). Furthermore, when ?DT is proved to be not new at (f), the search of other dependency trees for DG_XAB is performed at (e). *search_dt* searches a dependency tree which satisfies the co-occurrence condition in depth first manner with respect to input position P. *search_dt* selects one arc from the arc set *arcs_at*(DG,P) that is a set of arcs with position P. *search_dt* searches all possible dependency trees by selecting another arc at (k) when there are no solutions for arcs from P+1 to the end position.

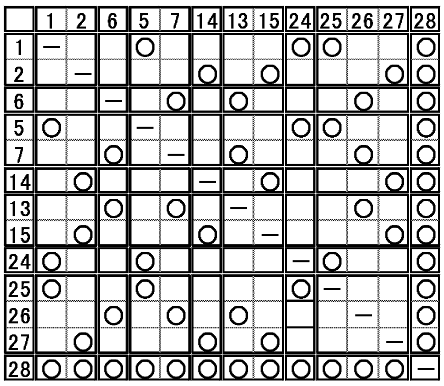
3.5.6 Execution Example of the Dependency Forest Reduction Algorithm

This section explains the execution process of the algorithm in Fig.3.14 for the example sentence "Tokyo taxi driver call center" in appendix A. The reduced dependency forest for this example has equivalent arcs. Fig.3.15 (a) shows the initial dependency forest for the example sentence. It has four sets of equivalent arcs, (1, 2),(5,7),(13,15),(25,26,27) which are surrounded with double lines.

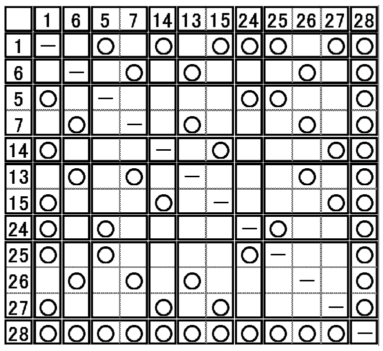
The forest reduction is performed along with the algorithm in Fig.3.14. The first equivalent arc pair (1,2) is selected to set X=1,Y=2. $diff(X,Y)$ is computed as {(5,14),(5,15),(5,27),...} by combining the elements in $uniq(X,Y)$={5,24,25} and $uniq(Y,X)$={14,15,27}. The first arc pair



(a) Initial dependency forest

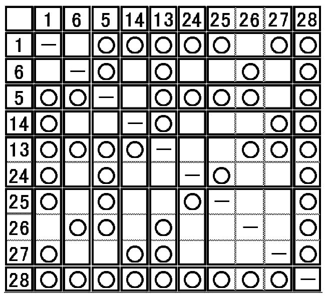


	1	2	6	5	7	14	13	15	24	25	26	27	28
1	—			○					○	○			○
2		—				○		○				○	○
6			—		○		○				○		○
5	○			—					○	○			○
7			○		—		○				○		○
14		○				—		○				○	○
13			○		○		—				○		○
15		○				○		—				○	○
24	○			○					—	○			○
25	○			○					○	—			○
26			○		○		○				—		○
27		○				○		○				—	○
28	○	○	○	○	○	○	○	○	○	○	○	○	—



	1	6	5	7	14	13	15	24	25	26	27	28
1	—		○		○		○	○	○		○	○
6		—		○		○				○		○
5	○		—					○	○			○
7		○		—		○				○		○
14	○				—		○				○	○
13		○		○		—				○		○
15	○				○		—				○	○
24	○		○					—	○			○
25	○		○					○	—			○
26		○		○		○				—		○
27	○				○		○				—	○
28	○	○	○	○	○	○	○	○	○	○	○	—

(b) Arc 2 merged to arc 1



	1	6	5	14	13	24	25	26	27	28
1	—		○	○	○	○	○		○	○
6		—	○		○			○		○
5	○	○	—		○	○	○	○		○
14	○			—	○				○	○
13	○	○	○	○	—			○	○	○
24	○		○			—	○			○
25	○		○			○	—			○
26		○	○		○			—		○
27	○			○	○				—	○
28	○	○	○	○	○	○	○	○	○	—

(c) Final reduced CM

Fig.3.15 Initial dependency forest and its reduction

(5,14) is skipped by the condition check for (RC1) at Fig.3.14 (b) because *same_position*(5,14) is true. The second arc pair (5,15) is tried. (5,15) does not match the conditions at (b), CM' and DG' are generated at (c). CM'=CM+{<1,14>,<1,15>,<1,27>} as shown in Fig.3.15 (b). Then *DG_XAB* is computed at (d). X =1,A =5 and B =15 result in *DG_XAB* =co(1)∩co(5)∩co(15) wrt <DG',CM'>={1,28}. The allowance of (5,15) generates no new dependency trees because the search of the dependency tree for *DG_XAB* by *search_dt* at (e) fails. The processing proceeds to the check of the next arc pair (5,27). In a similar way, all arc pairs in $diff$(1,2) are assured to generate no new dependency trees, and then DG' and CM' is set to DG and CM at (i), respectively.

Fig.3.15 (c) shows the reduced dependency forest finally obtained by the algorithm. It has three equivalent arcs 25, 16 and 27. The processing of the algorithm for this dependency forest is described. Let X =25,Y =26. Then, $uniq(X,Y)$ ={1,24},$uniq(Y,X)$ ={6,13},$diff(X,Y)$ ={(1,6), (1,13),(24,6),(24,23)}. The arc pair (1,6) is skipped by the condition check for (RC1). The arc pair (1,13) is not skipped by the condition check for (RC1) and (RC2). Then, *DG_XAB* is computed at Fig.3.14 (d). X = 25,A = 1 and B = 13 result in *DG_XAB* =co(25)∩co(1)∩co(13) wrt <DG',CM'> ={25,1,13,5,28}. A new IDed dependency tree {25,1,13,5,28} is obtained by *search_dt* at (e) for *DG_XAB*[*11]. Then, *new_generalized_dt* at (f) is called and *add_equiv_arcs* at (q) computes a set of arcs *DG_X* where the equivalent arcs of the arcs in DT are added. In the case of Fig.3.15 (c), arc 25 has equivalent arcs 26 and 27. Addition of these arcs results in *DG_X* ={25,26,27,1,13,5,28}. *search_dt* at Fig.3.14 (r) tries to get a dependency tree for *DG_X* but it fails because all equivalent arcs 25, 26 and 27 have inconsistent arcs in *DG_X*, i.e., <25,13>,<26,1> and <27,5> are not in CM. As a result, *new_generalized_dt* at (f) becomes true, that is, the increase of the generalized dependency tree occurs. Therefore, the merge of X=25 and Y=26 is not performed. The dependency forest in Fig.3.15 (c) includes the dependency trees in Fig.A.1 (a) to (c) and retains the soundness.

The above algorithm does not assure generation of one reduced dependency forest. The output dependency forest may vary by the application order of the merge operations for the equivalent arcs. There exist different dependency forests containing the same three generalized dependency forest for the above example. The algorithm in Fig.3.14 does not assure that it generates the minimum dependency forest. In fact, there exists a dependency forest smaller than the Fig.3.15 (c). Moreover, there is room for improving the computational amount in the above algorithm. The construction of the smallest reduced dependency forest and the improvement of the performance of the algorithm are future tasks[*12].

[*11] This tree corresponds to the tree in Fig.26 (d).

[*12] PDG allows arbitrary mapping between the constituent sequences and the partial dependency trees defined in grammar rules. Therefore, any dependency structure can be assigned for any constituent sequence provided that they satisfy the partial dependency structure condition. This feature suggests that not only the optimization techniques in the general algorithm but also the techniques based on the structural analysis of the grammar rules are effective.

3.6 Proof of the Completeness and Soundness of the Dependency Forest

3.6.1 Proof of the Completeness and Soundness of the Initial Dependency Forest

The phrase structure forest PF and the dependency forest DF=<DG,CM> is assumed in the following proof. Before showing the proof, some relations between the phrase structure forest and the dependency forest generated from the algorithms explained in Section 3.4.4, and some lemmas required for the proof are described.

[**Packed Edge and Single Edge**]
The phrase structure forest is a set of packed edges. As described in section 3.4.3, a packed edge is equivalent to a set of single edges. In this proof, packed edges are treated as a set of single edges. The following packed edge is shown in Fig.3.8.

Packed edge : <ID,FP,TP,C,PH,FCSL,RCS,DSL>
where FCSL=[CS_1,. . .,CS_n],DSL=[DS_1,. . .,DS_n]

is equal to the following set of single edges.

*e_1 : <ID-1,FP,TP,C,PH,(CS_1 DS_1),RCS>
:
*e_n : <ID-n,FP,TP,C,PH,(CS_n DS_n),RCS>

For example, *E4 in Fig.3.8 is a set of arcs *e_1,*e_2[*13]

Single edge *e_1: <170-1,0,5,vp, [time]-v-0, [103,169], [], {arc(obj-25,[flies]-n-1,[time]-v-0)}>
Single edge *e_2: <170-2,0,5,vp, [time]-v-0, [103,119,165],[],
{arc(obj-4,[flies]-n-1,[time]-v-0),arc(vpp-20,[like]-pre-2,[time]-v-0)}>

Every single edge is identified in the phrase structure forest by the packed edge-ID and the position in the CSDS pair of the packed edge. For example *e_1 is identified by 170-1. The lexical edge is treated as a set consisting of a single lexical edge. @E5 in Fig.3.8 is equal to the set @e_3.

Single edge @e_3: <156-1,4,5,n,[arrow]-n-4,[lex([arrow]-n)],{[[arrow]-n-4]}>

Various elements included in a packed edge and a single edge have correspondences with one another. The following shows the definitions of terms and relations.

[*13] The partial dependency tree is represented in {} because it is a set of arcs.

Edge and its elements

cs(X) : The constituent sequence of the single edge X.
ex. cs(*e_1) =[103,169] where 103 and 169 are packed edge-IDs.

ds(X) : The dependency structure DS of the single edge X or the node of the single lexical edge X. ex. ds(*e_1)={arc(obj-25,[flies]-n-1,[time]-v-0)}, ds(@e_3)={[arrow]-n-4}

Arcs in the dependency forest and edge : The arcs in a single edge X mean a∈ds(X). An arc in a packed edge Y means a∈ds(X),X∈Y. An arc in the phrase structure forest means a∈ds(X),X∈Y,Y∈PF.

Relations between arcs and nodes

gov(X) : The governor node of the arc X.
ex. gov(arc(obj-25,[flies]-n-1,[time]-v-0)) = [time]-v-0

dep(X) : The dependant node of the arc X.
ex. dep(arc(obj-25,[flies]-n-1,[time]-v-0) = [flies]-n-1

top_node(X) : The top node of the dependency tree X (The node which is not a dependant of any arcs in X).

Relations between arcs X, Y in the dependency tree DT

sib(X,Y) : gov(X)=gov(Y). X and Y are called the sibling arcs.

$X \xrightarrow[\mathrm{DT}]{1} Y$: dep(X)=gov(Y). X is a parent of Y and Y is a child of X. This relation is called parent relation.

$X \xrightarrow[\mathrm{DT}]{+} Y$: There is a parent relation chain from X to Y. X is an ancestor arc of Y and Y is a descendant arc of X.

$X \xrightarrow[\mathrm{DT}]{*} Y$: $X = Y$ or $X \xrightarrow[\mathrm{DT}]{+} Y$

[Edges and Phrase Structure Trees in the Phrase Structure Forest]

The phrase structure forest PF is a directed acyclic graph consisting of packed edges where the root is the root edge *E_{root} and the leaves are lexical edges. The "path in PF" is defined as follows:

Definition 3.6.1 [Path in the phrase structure forest]

A path in the phrase structure forest is a sequence consisting of packed edges and single edges obtained by tracing a packed edge and a single edge one after another by selecting one single edge from a packed edge (a set of single edges) and selecting one packed edge from the constituent sequence (a sequence of packed edges) of a single edge.

Now, let *E_0,*E_1,*$E_2 \cdots$ in the phrase structure forest as follows:

E_0 ={e_1,*e_2}, *E_1 ={*e_3}, *E_2 ={*e_4,*e_5}, *E_3 ={*e_6,*e_7} $\ldots$
cs(*e_1)=[*E_1,*E_2], cs(*e_2)=[*E_3], cs(*e_3)=[*E_4,*E_5] $\ldots$

The following are examples of paths.

$$[{}^*\mathrm{E}_0, {}^*\mathrm{e}_1, {}^*\mathrm{E}_2, {}^*\mathrm{e}_5],\ [{}^*\mathrm{E}_0, {}^*\mathrm{e}_1, {}^*\mathrm{E}_2],\ [{}^*\mathrm{e}_1, {}^*\mathrm{E}_1, {}^*\mathrm{e}_3, {}^*\mathrm{E}_5],\ [{}^*\mathrm{e}_1, {}^*\mathrm{E}_1, {}^*\mathrm{e}_3]$$

The following shows the definitions of terms and relations used in the latter part.

Terms and relations related to the phrase structure forest

$X \xrightarrow[\mathrm{PF}]{+} Y$: There is a path $[X, \ldots, Y]$ from a packed or single edge X to a packed or single edge Y. X is an ancestor of Y.

$X \xrightarrow[\mathrm{PF}]{*} Y$: $X = Y$ or $X \xrightarrow[\mathrm{PF}]{+} Y$ is true for single or packed edges X and Y. Y is called "reachable" from X.

$X \swarrow_{[\mathrm{PF}]} \searrow Y$: $X \neq Y, \neg(X \xrightarrow[\mathrm{PF}]{*} Y), \neg(Y \xrightarrow[\mathrm{PF}]{*} X)$ are true for single or packed edges X and Y, and there exists at least one single or packed edge Z in the phrase structure forest PF such that $Z \xrightarrow[\mathrm{PF}]{*} X$ and $Z \xrightarrow[\mathrm{PF}]{*} Y$.

Arcs governed by an edge : Arc X is governed by packed edge *E if $X \in \mathrm{ds}({}^*\mathrm{e}), {}^*\mathrm{E} \xrightarrow[\mathrm{PF}]{*} {}^*\mathrm{e}$ is true.

From the definition of the phrase structure forest, there exists a path from the root packed edge ${}^*\mathrm{E}_{root}$ to every single or packed edge in the phrase structure forest. Using the definition above, the C-Matrix setting conditions in section 3.4.5 are defined as follows:

Definition 3.6.2 [The C-Matrix setting conditions]

Arcs X, Y are co-occurable if any of the following conditions is satisfied.

(C1) There exists a single edge *e such that $X, Y \in \mathrm{ds}({}^*\mathrm{e}), {}^*\mathrm{e} \in {}^*\mathrm{E}, {}^*\mathrm{E} \in \mathrm{PF}$

(C2) There exist ${}^*\mathrm{e}_x$ and ${}^*\mathrm{e}_y$ such that $X \in \mathrm{ds}({}^*\mathrm{e}_x), Y \in \mathrm{ds}({}^*\mathrm{e}_y), {}^*\mathrm{e}_x \xrightarrow[\mathrm{PF}]{+} {}^*\mathrm{e}_y$ or ${}^*\mathrm{e}_y \xrightarrow[\mathrm{PF}]{+} {}^*\mathrm{e}_x$.

(C3) There exist ${}^*\mathrm{e}_x$ and ${}^*\mathrm{e}_y$ such that $X \in \mathrm{ds}({}^*\mathrm{e}_x), Y \in \mathrm{ds}({}^*\mathrm{e}_y), {}^*\mathrm{e}_x \swarrow_{[\mathrm{PF}]} \searrow {}^*\mathrm{e}_y$.

A phrase structure tree is defined as follows:

Definition 3.6.3 [Phrase Structure Tree]

A phrase structure tree for a packed edge *E is a set of single arcs obtained by a recursive procedure get_tree(*E) defined in Fig.3.16

```
# Compute a phrase structure tree PT for a packed edge RPE
get_tree(RPE){
  SE := select(RPE); /* Select one single edge in RPE */
  PT := {SE}; /* Add selected single edge to phrase structure(ps) tree PT */
  if( lexical_edge(SE) ) { return(PT); } /* Lexical edge? */
  /* Compute ps trees for packed edges in cs(SE) and add them to PE */
  foreach(PE in cs(SE)) { PT := PT ∪ get_tree(PE); }
  return(PT); }
```

Fig.3.16 Algorithm for obtaining phrase structure tree

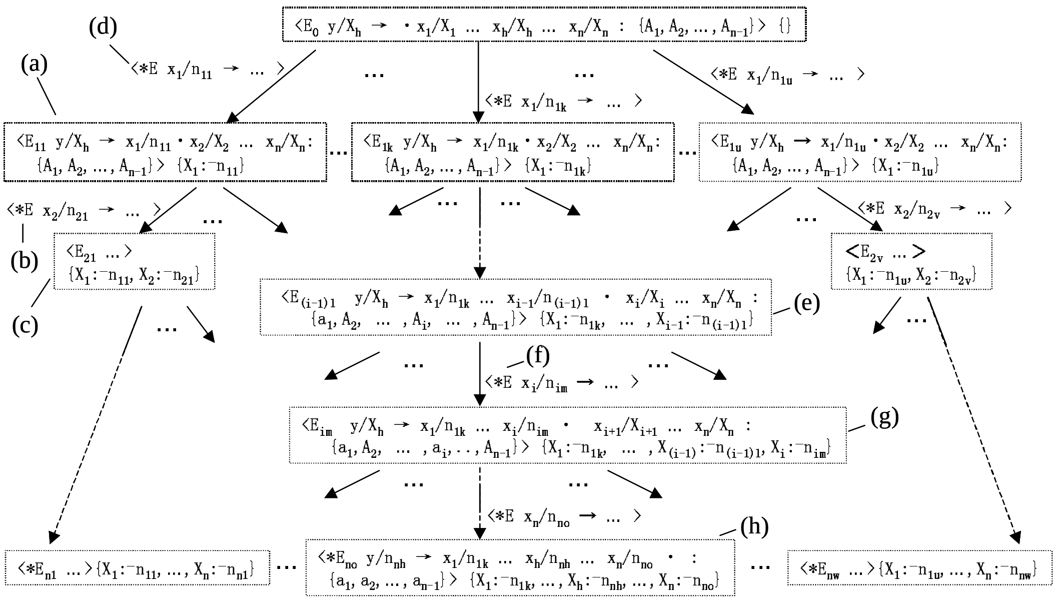
select(RPE) in the figure selects one arbitrary arc included in a packed edge RPE. lexical_edge(SE) is true if SE is a lexical edge. A phrase structure tree covers the words from the from-position to the to-position of the edge *E.

Definition 3.6.4 [All phrase structure trees]
ps_trees(*E) is a set of all phrase structure trees for *E.

[Relations between Edges and Arcs/Partial Dependency Trees]

Although the parsing algorithm is constructed using packed edges as basic data structures, the packing of edges is performed only when inactive edges are generated (Fig.3.9 (c),(d)). Therefore, every active packed edge has one single edge[*14] and one active packed edge corresponds to one single active edge and vice versa. In the following discussion, the word "edge" is used for representing a packed edge.

Parsing proceeds by generating new edges by combining an inactive edge to an active edge. Using a diagrammatic expression as described in section 3.4.3, a combination of two arcs generates a new edge by moving ' • ' in the active edge to the right neighbor position and binding the variable for the constituent at ' • ' to the phrase head (node) of the inactive edge to combine. Fig.3.17 is a tree called "edge combination tree," which represents the generation process of inactive edges from a grammar rule by edge combinations. The inactive edges located at the leaf of the edge



<E0 y/Xh → • x1/X1 ... xh/Xh ... xn/Xn : {A1, A2, ..., An-1}> {}
(d)
(a)
<*E x1/n11 → ... >
<*E x1/n1k → ... >
<*E x1/n1u → ... >
<E11 y/Xh → x1/n11 • x2/X2 ... xn/Xn: {A1, A2, ..., An-1}> {X1:⁻n11}
<E1k y/Xh → x1/n1k • x2/X2 ... xn/Xn: {A1, A2, ..., An-1}> {X1:⁻n1k}
<E1u y/Xh → x1/n1u • x2/X2 ... xn/Xn: {A1, A2, ..., An-1}> {X1:⁻n1u}
<*E x2/n21 → ... >
<*E x2/n2v → ... >
(b)
<E21 ...> {X1:⁻n11, X2:⁻n21}
(c)
<E2v ...> {X1:⁻n1u, X2:⁻n2v}
<E(i-1)1 y/Xh → x1/n1k ... xi-1/n(i-1)1 • xi/Xi ... xn/Xn : {a1, A2, ... , Ai, ... , An-1}> {X1:⁻n1k, ... , Xi-1:⁻n(i-1)1}
(e)
(f)
<*E xi/nim → ... >
<Eim y/Xh → x1/n1k ... xi/nim • xi+1/Xi+1 ... xn/Xn : {a1, A2, ... , ai, . . , An-1}> {X1:⁻n1k, ... , X(i-1):⁻n(i-1)1, Xi:⁻nim}
(g)
<*E xn/nno → ... >
(h)
<*En1 ...>{X1:⁻n11, ..., Xn:⁻nn1}
<*Eno y/nnh → x1/n1k ... xh/nnh ... xn/nno • : {a1, a2, ..., an-1}> {X1:⁻n1k, ..., Xh:⁻nnh, ..., Xn:⁻nno}
<*Enw ...>{X1:⁻n1u, ..., Xn:⁻nnw}

Fig.3.17 Edge combination tree

[*14] The parsing algorithm utilizes packed edges because another algorithm, which shares active edges, can be constructed.

combination tree are generated from a grammar rule located at the root of the tree[*15] through the active edges in between. The grammar rule is as follows:

$$\mathrm{y}/X_h \rightarrow \mathrm{x}_1/X_1 \cdots \mathrm{x}_h/X_h \cdots \mathrm{x}_n/X_n : \{A_1, A_2, \ldots, A_{n-1}\}$$

A_i is an arc in the form of arc(a_i,X_k,X_l) where a_i is an arcname, $1 \leq k \leq n$, $1 \leq l \leq n$, $k \neq l$. A set of arcs $\{A_1, \ldots, A_{n-1}\}$ satisfies the partial dependency structure condition in section 3.4.2.

The edges in the edge combination tree are expressed in diagrammatic form using '・', neglecting the from and to positions. The arrows between edges represent the edge combinations where the edge at the source of the arrow is combined with the inactive edge attached to the arrow to generate the resulting edge at the target position of the arrow. For example, E_{11} (Fig.3.17 (a)) is combined with <*E $\mathrm{x}_2/\mathrm{n}_{21} \rightarrow \ldots$ > ((b)) to generate E_{21} ((c)). Since '・' moves to the right neighbor position, the depth of the tree, i.e., the number of arcs from E_0 to each inactive edge, is equal to n, i.e., the number of elements in the rule body in the grammar rule.

The phrase head (node) is bound to a variable in the active edge during edge combination. The variable bindings are shown at the right of the edge by { }. For example, the combination of E_0 and the edge (d) whose phrase head is n_{11} generates E_{11}, and then it is combined with the edge (b) whose phrase head is n_{21} to produce E_{21}. As a result, the variable bindings of E_{21} are $\{X_1{:=}\mathrm{n}_{11}, X_2{:=}\mathrm{n}_{21}\}$[*16]. An arc whose governor and dependant are bound is called "fixed arc" and has a new unique arc-ID generated by add_arcid at Fig.3.9 (i). Fixed arcs are represented by a small letter such as a_1 in Fig.3.17 (e). One variable binding may generate zero or more than zero fixed arcs. The generated fixed arc has one unique variable binding corresponding to one edge combination. This edge combination generates one unique edge. This unique edge is called the edge which generated the fixed arc or simply "source edge" of the fixed arc, and is referred to as src_Edge(a) where 'a' is a fixed arc. For example, in the edge combination between (e) and (f) in Fig.3.17, provided that the binding of the node n_{im} (let it [like]-pre-3) to the variable X_i generates the fixed arc a_i (let it arc(pre-28,[like]-pre-3,[time]-v-0)) from the unfixed arc A_i (arc(pre,X_i,[time]-v-0)), the edge which generated the fixed arc a_i, i.e., src_Edge(a_i) is E_{im} in Fig.3.17 (g).

Every inactive edge (leaf of the combination tree) (ex. Fig.3.17 (h)) has only fixed arcs because all variables including phrase head variable in the edge are bound due to the partial dependency structure condition. Inactive edge represents a result of a sequence of variable bindings caused by the edge bindings from the root to the leaf of the edge combination tree. The following shows the definitions of terms and relations related to the edge combination tree.

[*15] The grammar rule is written in edge form in the edge combination tree. This edge is not generated in the real parsing process but is introduced for convenience of explanation.

[*16] Scope of a variable is within edge.

fixed arc : Arc whose governor and dependant nodes are fixed by the variable bindings caused by edge combinations.

src_Edge(a) (source edge) : The active or inactive edge which generated a fixed arc a. Mapping from a fixed arc to its corresponding edge is one to one, whereas the reverse is 1 to 0 - many.

$X \xrightarrow[\text{CT}]{*} Y$ (origin) : Edge X is located on a route from the root node to the edge Y or $X = Y$. X is called an origin of Y.

origin relation : Edges X, Y in an edge combination tree CT are said to be in origin relation if $X \xrightarrow[\text{CT}]{*} Y$ or $Y \xrightarrow[\text{CT}]{*} X$ is true.

edge(a,DT) (corresponding edge) : The corresponding edge for a fixed arc a and a well-formed dependency tree DT is a single edge e which satisfies the following condition (defined in lemma 3.6.4).

$DT \supseteq \text{ds}(e), \text{a} \in \text{ds}(e)$

According to the structure of the edge combination tree described above, two fixed arcs a_i and a_j have the following relations.

Lemma 3.6.1 (Relation between arcs in one partial dependency tree) Let a_i and a_j be fixed arcs $\text{a}_i,\text{a}_j \in \text{ds}(e)$ where e is a single edge. Their source edges src_Edge(a_i) and src_Edge(a_j) are in origin relation.

A fixed arc in edge e in the edge combination tree is in the edges that have e as their origins. For example, fixed arc a_i generated at (g) is contained in (h).

Lemma 3.6.2 (Relation between an arc and the edge which generated the arc) Suppose that fixed arcs a_i,a_j satisfy src_Edge(a_i)$\xrightarrow[\text{CT}]{*}$src_Edge($\text{a}_j$), $\text{a}_j \in \text{ds}(\text{*e})$ implies $\text{a}_i \in \text{ds}(\text{*e})$ for every single arc *e in PF (*e∈*E,*E∈PF).

Since a unique arc-ID is assigned to each arc, all inactive single edges in the inactive packed edges in an arc combination tree, i.e., the leaves of the tree $\text{*E}_{n1} \cdots \text{*E}_{no} \ldots \text{*E}_{nw}$ in Fig.3.17, have different partial dependency trees. Therefore, $\text{ds}(\text{*e}_i) \neq \text{ds}(\text{*e}_j)$ for arbitrary arcs $\text{*e}_i,\text{*e}_j$ ($\text{*e}_i \neq \text{*e}_j$) in PF. A single inactive edge and a partial dependency tree have one-to-one mapping. In the parsing process, inactive edges which satisfy the conditions shown in Fig.3.9 (j) are merged into one and this merged edge becomes an element of the phrase structure forest. The one-to-one mapping relation between a single edge and a partial dependency tree is assured in the phrase structure forest because this merge operation does not change the dependency structures in the single edges.

Lemma 3.6.3 (Constraints with respect to the arcs in the edges included in a path) Suppose arcs $\text{a}_i \in \text{ds}(\text{e}_i),\text{a}_j \in \text{ds}(\text{e}_j)$. If $\text{e}_i \xrightarrow[\text{PF}]{+} \text{e}_j$, then $\text{dep}(\text{a}_i) \neq \text{dep}(\text{a}_j)$ is true. Inversely, if $\text{dep}(\text{a}_i) = \text{dep}(\text{a}_j)$, then $\neg(\text{e}_i \xrightarrow[\text{PF}]{+} \text{e}_j)$ is true.

Proof: This lemma is established because data structures in the single edges are trees whose top nodes are phrase heads according to the partial dependency structure condition in 3.4.2.

$\square$

Lemma 3.6.4 (Existence of corresponding edge) Suppose an arc a_i in a well-formed dependency tree DT ($\mathrm{a}_i \in$DT). There exists one and only one packed edge E$\in$PF and single edge e$\in$E which satisfy the following condition.

DT $\supseteq$ ds(e),$\mathrm{a}_i \in$ds(e)

Proof: Let number of nodes in DT n (Number of arcs is $n-1$). Divide arc set DT into the following two arc sets IN_ARCS,OUT_ARCS with respect to a_i.

IN_ARCS = $\{\ a_j \mid \mathrm{src\_Edge}(a_i) \xrightarrow[\mathrm{CT}]{*} \mathrm{src\_Edge}(a_j) \text{ or } \mathrm{src\_Edge}(a_j) \xrightarrow[\mathrm{CT}]{*} \mathrm{src\_Edge}(a_i)\}$
OUT_ARCS = DT$-$IN_ARCS

Let SRC_EDGES be a set of source edges corresponding to the arcs in IN_ARCS.

SRC_EDGES = $\{E \mid E = \mathrm{src\_Edge}(a),\ a \in \mathrm{IN\_ARCS}\}$

First, the following statement is to be established.

$$\text{“Arbitrary edges in SRC\_EDGES are in origin relation''} \qquad (A)$$

Suppose edges U,$V \in$SRC_EDGES. U and V are in origin relation by definition and in one of the following three cases.

(a) One arc is an origin of E_i and E_i is an origin of the other arc. $U \xrightarrow[\mathrm{CT}]{*} \mathrm{E}_i, \mathrm{E}_i \xrightarrow[\mathrm{CT}]{*} V$

(b) Both arcs are origins of E_i. $U \xrightarrow[\mathrm{CT}]{*} \mathrm{E}_i, V \xrightarrow[\mathrm{CT}]{*} \mathrm{E}_i$

(c) E_i is an origin of both arcs. $\mathrm{E}_i \xrightarrow[\mathrm{CT}]{*} U, \mathrm{E}_i \xrightarrow[\mathrm{CT}]{*} V$

In cases (a) and (b), the statement (A) obviously holds according to the structure of the edge combination tree. The following shows that a contradiction is derived from the assumption that U and V are not in origin relation in case (c).

Suppose that U and V are not in origin relation. There exist arcs a_u, $\mathrm{a}_v \in$DT such that U=src_Edge(a_u), V=src_Edge(a_v) according to the premise. Since all arcs in DT are in the packed shared forest, there exist inactive edges that have U and V as their origin, respectively. Let $^*\mathrm{E}_u$ and $^*\mathrm{E}_v$ be origins of U and V, respectively. Let e_u in $^*\mathrm{E}_u$ and $^*\mathrm{e}_v$ in $^*\mathrm{E}_v$ be edges such that $\mathrm{a}_u \in {}^*\mathrm{e}_u$, $\mathrm{a}_v \in {}^*\mathrm{e}_v$. From Lemma 3.6.2, both $^*\mathrm{e}_u$ and $^*\mathrm{e}_v$ contain a_i. Arcs a_u and a_v in the well-formed dependency tree DT satisfy either of the C-Matrix setting conditions (C1) to (C3). a_u and a_v do not satisfy (C1) because the contradiction for the assumption that U and V are not in origin relation is deduced from (C1), i.e., the existence of e such that a_u,$\mathrm{a}_v \in$ds(e) according to lemma 3.6.1. (C2) is that $^*\mathrm{e}_u \xrightarrow[\mathrm{PF}]{+} {}^*\mathrm{e}_v$ is true (The reverse case is shown in the same way). Nodes included in arcs in ds($^*\mathrm{e}_u$) are phrase heads of the constituents in cs($^*\mathrm{e}_u$) according to the partial dependency structure condition. This implies that either dep(a_i) or gov(a_i) of $\mathrm{a}_i \in$ds($^*\mathrm{e}_u$) is a node which locates outside of the coverage of $^*\mathrm{e}_v$. On the other hand, both dep(a_i) and gov(a_i)

must be in the coverage of $*e_v$ because $a_i \in ds(*e_v)$ is true. From this contradiction, a_u and a_v do not satisfy (C2). (C3), i.e., $*e_u \swarrow_{[PF]} \searrow *e_v$, is not satisfied by a_u and a_v because the coverage of $*e_u$ and $*e_v$ have to be overlapped due to the premise that a_i is in both $*e_u$ and $*e_v$. From the above, the supposition that U and V are not in origin relation contradicts the C-Matrix setting conditions between a_u and a_v. Therefore statement (A) is true.

Now, let E_{last} be the last edge connected from E_i, i.e., the edge which satisfies the following conditions.

$E_i \xrightarrow[CT]{*} E_{last}$

E_{last} is the only edge E_j such that $E_{last} \xrightarrow[CT]{*} E_j$ ($E_j \in$SRC_EDGES)

Fig.3.18 shows the relation between IN_ARCS and SRC_EDGES diagrammatically. E_{start} corresponds to a grammar rule. The grammar rule is as follows:

$$y/X_h \rightarrow x_1/X_1 \cdots x_z/X_z \ :\{A_1,\ldots,A_{z-1}\}$$

SRC_EDGES constitutes a route on the edge combination tree CT with root E_{start}, containing $E_1,\ldots,E_{last}$. The source edge E_i for a_i exists somewhere on this route. There exists at least one arc a_{last} which is generated by E_{last} in IN_ARCS.

E_{last} is either an active edge or an inactive edge. Fig.3.18 shows a case where E_{last} is an active edge. E_{last} is proved to be an inactive edge as follows:

Suppose that E_{last} is an active edge. As shown in Fig.3.19, E_{last} (Fig.3.19 (a)) has at least one remaining constituent x_{u+1} (Fig.3.19 (b). Variable for the constituent is not shown). Let s1 and t1 be a from-position and a to-position of E_{last}, respectively.

From the premise $a_{last} \in$DT, there exists at least one inactive edge $*E_x$ (Fig.3.27 (c)) which has E_{last}, the source edge of a_{last}, as its origin in the phrase structure forest PF. As shown in the figure, $*E_x$ has the from-position equal to the from-position s1 of E_{last} and the to-position greater than the to-position t1 of E_{last}.

Consider the node n_{t1+1} at the position t1+1 (Fig.3.19 (d))[*17]. DT has one arc

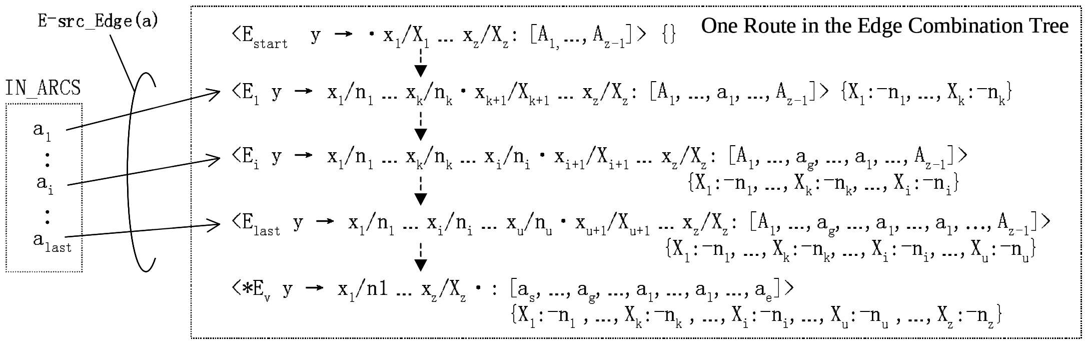

E-src_Edge(a)
IN_ARCS
a1
ai
alast
One Route in the Edge Combination Tree
<Estart y → • x1/X1 ... xz/Xz: [A1, ..., Az-1]> {}
<E1 y → x1/n1 ... xk/nk • xk+1/Xk+1 ... xz/Xz: [A1, ..., a1, ..., Az-1]> {X1:¬n1, ..., Xk:¬nk}
<Ei y → x1/n1 ... xk/nk ... xi/ni • xi+1/Xi+1 ... xz/Xz: [A1, ..., ag, ..., a1, ..., Az-1]> {X1:¬n1, ..., Xk:¬nk, ..., Xi:¬ni}
<Elast y → x1/n1 ... xi/ni ... xu/nu • xu+1/Xu+1 ... xz/Xz: [A1, ..., ag, ..., a1, ..., a1, ..., Az-1]> {X1:¬n1, ..., Xk:¬nk, ..., Xi:¬ni, ..., Xu:¬nu}
<*Ev y → x1/n1 ... xz/Xz • : [as, ..., ag, ..., a1, ..., a1, ..., ae]> {X1:¬n1 , ..., Xk:¬nk , ..., Xi:¬ni, ..., Xu:¬nu , ..., Xz:¬nz}

Fig.3.18 Edges corresponding to arcs in DT

[*17] One n_{t1+1} exists due to the well-formedness condition of DT

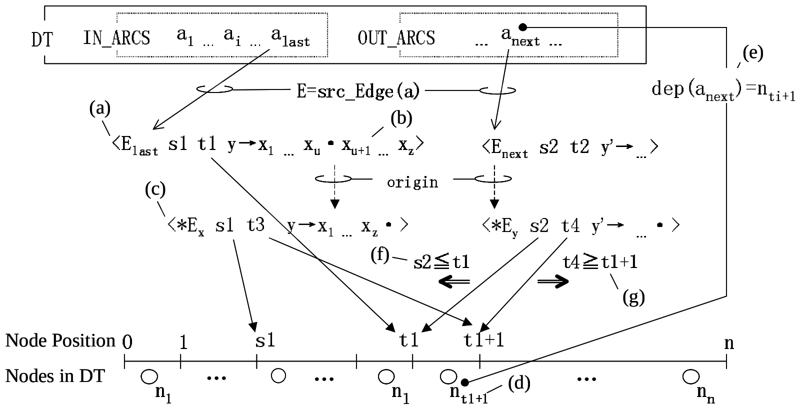


Fig.3.19 Existence of corresponding edge

a_{next}($dep(a_{next})=n_{t1+1}$ (Fig.3.19 (e)). a_{next} is an element of OUT_ARCS due to the definition of a_{last}. $a_{next}\in$DT implies that at least one inactive edge $*E_y$ whose origin is E_{next}, the source edge of a_{next}, exists in the phrase structure forest PF. Since E_{next} has a_{next} within its coverage, E_{next} has the from-position s1 less than or equal to t1 (Fig.3.19 (f)) and the to-position t2 is greater than or equal to t1+1 (Fig.3.19 (g)). Therefore, the from-position of $*E_y$ is less than or equal to t1. From the above, $*E_x$ which has its origin E_{last} overlaps $*E_y$ which has its origin E_{next} at the t1 position.

Now, no co-occurrence setting conditions for a_{last} and a_{next} hold as follows:

Arcs a_{last} and a_{next} do not satisfy (C3) because $*E_x$ and $*E_y$ overlap as explained above. (C1) is not satisfied from the premise $*E_x \neq *E_y$. Consider (C2) meaning that $*E_x \xrightarrow[PF]{+} *E_y$ is true (the reverse case is shown in the same way). Let a_m be the arc in $*E_x$, whose dependant node is n_{t1+1}. $a_m \neq a_{next}$[*18]. According to lemma 3.6.3, if $*E_x \xrightarrow[PF]{+} *E_y$ is not true, then (C2) is not satisfied. From the above, a_{last} and a_{next} satisfy no co-occurrence setting conditions. This contradicts the premise that DT is a well-formed dependency tree. Therefore, E_{last} is not an active edge.

Let E_{last} be an inactive edge $*E_{last}$.

(a) $*E_{last}$, the source edge of a_{last}, is an inactive edge (a leaf of the edge combination tree) then no other packed edges contain a_{last}. Moreover, there exist only one $*e_{last}$ such that $a_{last}\in ds(*e_{last}), *e_{last}\in *E_{last}$.

(b) $*E_{last}\in$PF is true due to $a_{last}\in$DT.

(c) According to the premise $a_{last}\in$DT and lemma 3.6.2, DT$\supseteq ds(*e_{last})$ is true.

Lemma 3.6.2 is true due to (a) to (c).

□

[*18] If $a_m = a_{next}$, src_Edge(a_{next}) and E_i are in origin relation. This contradicts the premise $a_{next}\in$OUT_ARCS.

[Relation between Connected Arcs and their Corresponding Edges]

Arcs a_i,a_j($a_i \xrightarrow[DT]{1} a_j$ or sib(a_i,a_j)) are called "connected arcs." The following two lemmas are established with respect to connected arcs in a well-formed dependency tree DT.

Lemma 3.6.5 (Connected arcs and their corresponding edges) Suppose $*e_i$=edge(a_i,DT), $*e_j$=edge(a_j,DT) for connected arcs a_i,a_j in DT. At least one of (a), (b), (c) is true.

(a) $*e_i = *e_j$
(b) $*e_i \xrightarrow[PF]{+} *e_j$
(c) $*e_j \xrightarrow[PF]{+} *e_i$

Lemma 3.6.5 means that if two arcs in DT are connected, one of their corresponding edges is reachable from another edge in PF.

Proof: $*e_i$ and $*e_j$ satisfy at least one of the C-Matrix setting conditions (r1) to (r3) because $*e_i$ and $*e_j$ co-occurs in DT.

(r1) $*e_i = *e_j$
(r2) $*e_i \xrightarrow[PF]{+} *e_j$ or $*e_j \xrightarrow[PF]{+} *e_i$
(r3) $*e_i \swarrow {\scriptstyle [PF]} \searrow *e_j$

Let n be a node shared by the connected arcs a_i and a_j. Both $*e_i$ and $*e_j$ covers n. Therefore, (r3) is not satisfied by $*e_i$ and $*e_j$. $*e_i$ and $*e_j$ has to satisfy (r1) or (r2). □

Lemma 3.6.6 (Ancestor-descendant arcs and their corresponding arcs) Suppose $*e_i$=edge(a_i,DT), $*e_j$=edge(a_j,DT) for a_i,a_j($a_i \xrightarrow[DT]{+} a_j$).

$$*e_i \xrightarrow[PF]{*} *e_j$$

Proof: In the case that dep(a_i)=gov(a_j), one of (a), (b) or (c) in lemma 3.6.5 is true. From the node positioning relation prescribed by the partial dependency structure condition in section 3.4.2, (c) is not satisfied by a_i and a_j because $\neg(*e_j \xrightarrow[PF]{+} *e_i)$ is true for $*e_j$ and $*e_i$. Therefore, parent-child arcs satisfy either (a) or (b). Lemma 3.6.6 is established for $a_i \xrightarrow[DT]{*} a_j$ due to the associativity of the relation $\xrightarrow[PF]{*}$.

□

[Top single edge top_edge(DT)]

Definition 3.6.5 [Top single edge]

A "top single edge" for DT top_edge(DT) is the single edge which locates in the topmost position in PF among the edges which correspond to the arc just under the top node of DT. That is, top_edge(DT) is edge a_i satisfying the following conditions.

top_node(DT)=gov(a$_i$)
edge(a$_i$)$\xrightarrow[\text{PF}]{*}$edge(a$_j$) for all a$_j$ such that top_node(DT)=gov(a$_j$)

If DT is a tree consisting of one node, top_node(DT) is the single lexical edge corresponding to the node.

Lemma 3.6.7 (Relation between top_edge(DT) and edge(a$_j$,DT)) *e$_t$$\xrightarrow[\text{PF}]{*}$*e$_j$ is true for *e$_t$=top_edge(DT) and *e$_j$=edge(a$_j$,DT) (a$_j$∈DT).

Proof: If a$_j$ is an arc just under the top node of DT, i.e., gov(a$_j$)=top_node(DT), *e$_t$$\xrightarrow[\text{PF}]{*}$*e$_j$, is true according to lemma 3.6.5 and the definition of top_edge. If not, a$_j$ is a descendant of one of the arcs just under the top node of DT. *e$_t$$\xrightarrow[\text{PF}]{*}$*e$_j$ is true according to lemma 3.6.6.

□

[**Division of Well-formed Dependency Tree**]

The "division of a well-formed dependency tree DT" means the creation of a set of partial dependency trees DT$_1$,...,DT$_m$ by removing a set of arcs in ds(top_edge(DT)) from DT, where m is a number of nodes in ds(top_edge(DT)). Nodes isolated from all other nodes by this operation are dependency trees that consist of one node. For example, suppose that ds(top_edge(DT))={a$_s$,a$_t$,a$_u$,a$_w$} in Fig.3.20, DT is divided into partial dependency trees DT$_s$,DT$_t$,DT$_u$,DT$_v$,DT$_w$ whose top nodes are n$_s$,n$_t$,n$_u$,n$_v$, n$_w$, respectively. Since nodes n$_s$, n$_w$ are isolated from other nodes, DT$_s$ and DT$_w$ are dependency trees consisting of single node, i.e.,{n$_s$} and {n$_w$}, respectively. The phrase heads of the packed edges in cs(e) of a single edge e have one-to-one correspondence with the nodes in the partial dependency tree df(e) due to the partial dependency structure condition in section 3.4.2. Therefore, there exists one packed

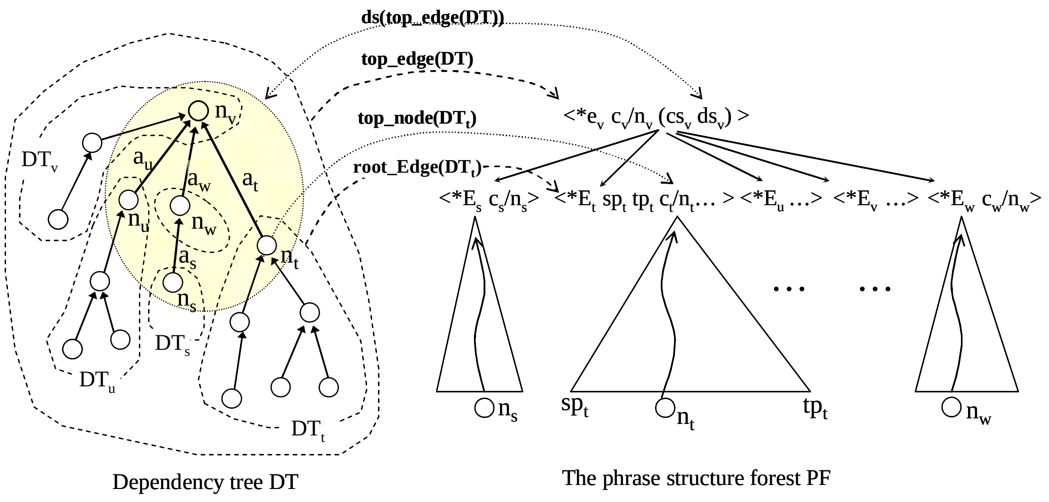


Fig.3.20 Division of well-formed DT

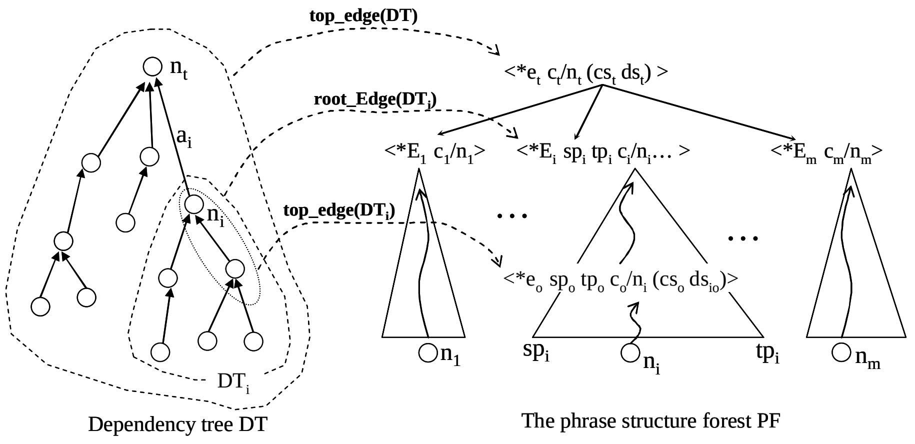

top_edge(DT)
<*e_t c_t/n_t (cs_t ds_t) >
root_Edge(DT_i)
<*E_1 c_1/n_1>
<*E_i sp_i tp_i c_i/n_i... >
<*E_m c_m/n_m>
top_edge(DT_i)
<*e_o sp_o tp_o c_o/n_i (cs_o ds_io)>
n_t
a_i
n_i
DT_i
n_1
sp_i
n_i
tp_i
n_m
Dependency tree DT
The phrase structure forest PF

Fig.3.21 Well-formed DT$_i$ obtained from division of DT

edge *E$_i$ ($1\leq i\leq m$) whose phrase head is a top node n$_i$ of DT$_i$($1\leq i\leq m$). *E$_i$ is called the "root packed edge" for DT$_i$ and is referred to as root_Edge(DT$_i$).

Definition 3.6.6 [Root packed edge]

Suppose DT$_i$ and its top node n$_i$ ($1\leq i\leq m$) is obtained by the division of DT. A root_Edge(DT$_i$) is a packed edge which is an element of cs(top_edge(DT)) and whose phrase head is top_node(DT$_i$).

For example, in Fig.3.20, *e$_v$ is top_edge(DT) and has the constituent sequence *E$_s$,*E$_t$,*E$_u$, *E$_v$,*E$_w$ whose phrase heads are n$_s$,n$_t$,n$_u$,n$_v$,n$_w$, respectively. Then, root_Edge(DT$_t$) is *E$_t$. The following two lemmas are explained with reference to Fig.3.21.

Lemma 3.6.8 (Relation between root packed edge and top single edge) Suppose partial dependency trees DT$_i$ obtained from the division of the well-formed dependency tree DT. $*E_i\xrightarrow[PF]{+}*e_o$ is true for the root packed edge *E$_i$=root_Edge(DT$_i$) and the top single edge *e$_o$ =top_edge(DT$_i$).

Proof: Let *e$_t$ be top_edge(DT) and n$_i$ be the top node of DT$_i$ (Fig.3.21). Consider the case where DT is a tree consisting of a single node, i.e., DTi ={n$_i$}. *e$_o$ is the single lexical edge corresponding to n$_i$. $*E_i\xrightarrow[PF]{+}*e_o$ is true because the phrase head of *E$_i$ is n$_i$. Consider another case where DT is a tree consisting of arcs. $*e_t\xrightarrow[PF]{*}*e_o$ is true according to lemma 3.6.6. Therefore, $X\xrightarrow[PF]{*}*e_o$ is true for some packed edge X in cs(*e$_t$). X=*E$_i$ is true because *E$_i\in$cs(*e$_t$) from definition and the phrase heads of *E$_i$ and *e$_o$ are the same.

□

Lemma 3.6.9 (Well-formedness of partial trees obtained by division of DT) Suppose partial dependency trees DT$_i$ and its root packed edge *E$_i$ with the from-position sp$_i$ and to-position tp_i obtained from the division of the well-formed dependency tree DT. DT_i is a well-formed dependency tree, which covers from sp_i to tp_i.

Proof: DT_i is a well-formed dependency tree if it satisfies the co-occurrence constraint and the well covering constraint. Obviously DT_i satisfies the co-occurrence constraint because DT satisfies the co-occurrence constraint. The following part shows the well covering constraint.

In the case that DT is a tree consisting of a single node, it satisfies the well covering constraint from the definition. Consider the case where DT is a tree consisting of arcs. Let n_i and n_j be the top node of DT and one of any other nodes in DT ($n_j \neq n_i$). There exists arc a_j such that $n_j = dep(a_j)$ in DT. Moreover, there exists $a_k \in DT_i$ such that $gov(a_k)=n_i, a_k \xrightarrow[DT]{*} a_j$ for $a_j \in DT_i$. Let $*e_j$ and $*e_k$ be $*e_j=edge(a_j,DT)$, $*e_k=edge(a_k,DT)$. $*e_k \xrightarrow[PF]{*} *e_j$ is true due to lemma 3.6.6 because a_k is equal to a_j or a_k is an ancestor of a_j. Now, let $*e_t$ be the top single node of DT, $*e_t \xrightarrow[PF]{*} *e_k$ is true according to lemma 3.6.7. $*e_k$ is reachable from one of the packed edges in $cs(*e_t)$. $*E_i \xrightarrow[PF]{*} *e_k$ is true because the phrase head of $*e_k$ is n_i.

From the above, $*E_i \xrightarrow[PF]{*} *e_k \xrightarrow[PF]{*} *e_j$ is true and n_j is in the coverage of $*E_i$, that is, all nodes in DT_i are in the coverage from sp_i to tp_i. Furthermore, the nodes in $DT_k (k \neq i)$ are not in the coverage from sp_i to tp_i. Since DT satisfies the well covering constraint, all nodes in DT_i occupy whole positions from sp_i to tp_i.

□

[Proof of the Completeness and Soundness of the Dependency Forest]

A corresponding dependency tree dependency_tree(PT) for a phrase structure tree PT = $\{*e_1, \ldots, *e_m\}$ is defined as follows:

Definition 3.6.7 [Dependency tree for a phrase structure tree PT]

dependency_tree(PT) = $ds(*e_1) \uplus \cdots \uplus ds(*e_m)$

The operator $\uplus$ is similar to the union operator $\cup$, which is introduced to manage the union of dependency structures which may be either a set of arcs or a set of a node. $\uplus$ removes nodes from the union of dependency structures if it has at least one arc. The following are examples of $\uplus$ where n_i and a_i represent node and arc, respectively.

$$\{n_1\} \uplus \{a_1,a_2\} = \{a_1,a_2\}$$
$$\{a_1\} \uplus \{a_2,a_3\} = \{a_1,a_2,a_3\}$$
$$\{n_1\} \uplus \{\} = \{n_1\}$$

dependency_tree(PT) is a tree because it is constructed by combining each partial dependency tree $ds(*e_i)$.

Theorem 3.6.1 (The Completeness of the dependency forest)

Let PT be a phrase structure tree in the phrase structure forest PF. DT=dependency_tree(PT) is a well-formed dependency tree in the dependency forest DF.

Proof: From the definition of the dependency forest, DT is included in DG. Nodes contained in DT and PT have one-to-one relation according to the partial dependency structure condition. Since PT covers whole sentence, DT is a well covered dependency tree. According to the C-Matrix setting conditions, every two arcs in DT satisfy the co-occurrence constraint. Therefore, dependency_tree(PT) is a well covering and well co-occurred dependency tree in DF

□

Theorem 3.6.2 (The soundness of the dependency forest)
Let DT be a well-formed dependency tree in the dependency forest DT. There exists a phrase structure tree PT in the phrase structure forest PF such that DT=dependency_tree(PT).

Proof: The existence of a phrase structure tree PT which satisfies PT∈ps_trees(*E$_{root}$) and dependency_tree(PT)=DT is shown below.

Let n be a number of input words. The following is an algorithm, called the phrase structure tree generation algorithm, which generates a phrase structure tree from a packed edge *E$_r$ with from-position sp$_r$ and to-position tp$_r$ (1≤sp$_r$<tp$_r$≤n) and a well-formed dependency tree DT, which covers from sp$_r$ to tp$_r$. The proof that the phrase structure tree generation algorithm generates a phrase structure tree, which satisfies the above conditions, is shown below using mathematical induction for the number of arcs in the dependency tree.

```
[Phrase Structure Tree Generation Algorithm]
In the case that DT is a set of arcs:

    A-Step1(Identification of the Top Single Edge) : Let *e_t be the top single edge
      top_edge(DT).
    A-Step2 (Identification of a Path) : Identify a path from *E_r to *e_t. Let PATH be a set
      of single edges in the path except for *e_t.
    A-Step3 (Division of DT) : Divide DT by removing edges in ds(top_edge(DT)) to get a set
      of partial dependency trees DT_i(1≤i≤m) and root packed edges *E_i=root_Edge(DT_i).
    A-Step4 (Computation of Partial Phrase Structure Trees) : Apply the phrase structure
      tree generation algorithm to each DT_i and *E_i(1≤i≤m) and compute each PT_i.
    A-Step5 (Construction of Phrase Structure Tree) : Returns PT=PATH ∪ {*e_t} ∪ PT_1
      ∪···∪PT_m as a phrase structure tree for DT,*E_r.

In the case that DT is a set of a node (DT={n}):

    N-Step1 (Identification of Lexical Edge) : Identify the lexical edge @e_lex which generated
      node n.
    N-Step2 (Identification of a Path) : Identify a path from *E_r to e_lex and returns a set of
      single edges in the path as a phrase structure tree for DT,*E_r.
```

When DT is s a set of arcs, phrase structure tree PT is constructed through A-Step1 to A-Step5. Fig.3.22 shows the behavior of the algorithm diagrammatically. A-Step1 computes the

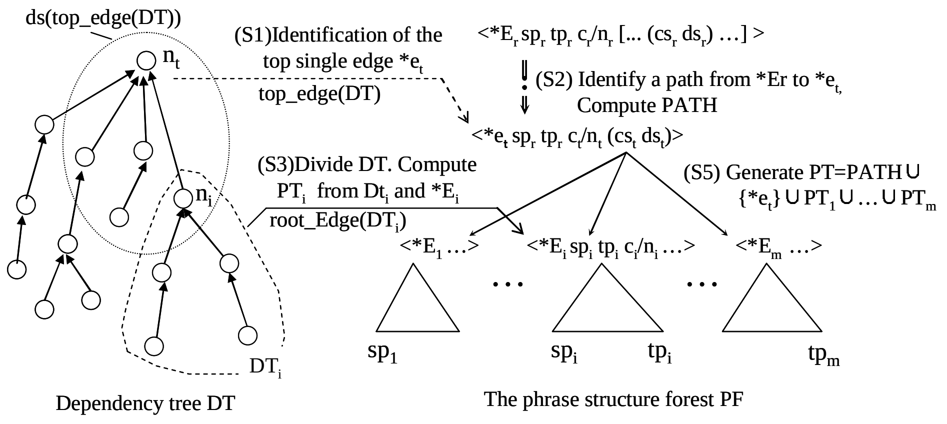

ds(top_edge(DT))
n_t
(S1)Identification of the top single edge *e_t
top_edge(DT)
<*E_r sp_r tp_r c_r/n_r [... (cs_r ds_r) ...] >
(S2) Identify a path from *Er to *e_t, Compute PATH
<*e_t sp_r tp_r c_t/n_t (cs_t ds_t)>
(S3)Divide DT. Compute PT_i from Dt_i and *E_i
root_Edge(DT_i)
(S5) Generate PT=PATH ∪ {*e_t} ∪ PT_1 ∪ ... ∪ PT_m
n_i
<*E_1 ...>
<*E_i sp_i tp_i c_i/n_i ...>
<*E_m ...>
sp_1
sp_i
tp_i
tp_m
DT_i
Dependency tree DT
The phrase structure forest PF

Fig.3.22 Generation of PT from DT and $^*\mathrm{E}_r$

top single edge $^*\mathrm{e}_t$(=top_edge(DT)) (Fig.3.22 (S1)). A-Step2 identifies a path from $^*\mathrm{E}_r$ to $^*\mathrm{e}_t$ and computes PATH, a set of single edges. The existence of this path ($^*\mathrm{E}_r \xrightarrow[\mathrm{PF}]{+} {}^*\mathrm{e}_t$) is assured as follows: In the case that $^*\mathrm{E}_r$ is $^*\mathrm{E}_{root}$, it is obvious. In the case that $^*\mathrm{E}_r$ is obtained by the division of a dependency graph ($^*\mathrm{E}_i$ in A-Step4), it is assured by lemma 3.6.8. The dependency structure parts of the single edges in PATH are {}. This is obvious because $^*\mathrm{E}_r$ and $^*\mathrm{e}_t$ have the same coverage since all nodes in DT are in the coverage of $^*\mathrm{e}_t$.

$$dependency\_tree(PATH) = \{\} \qquad (A)$$

A-Step3 performs the division of DT and generates DT_i,$^*\mathrm{E}_i(1 \leq i \leq m)$ as shown in Fig.3.22 (S3). According to lemma 3.6.9, DT_i is a well-formed dependency tree covering the coverage of $^*\mathrm{E}_i$ and the phrase structure tree generation algorithm is applicable recursively at A-Step4. A-Step5 computes the phrase structure tree PT (Fig.3.22 (S5)). From the definition of the phrase structure tree, it is obvious that PT is a phrase structure tree if each PT_i is phrase structure tree.

When DT is a set of a node (DT={n}), N-Step1 and N-Step2 computes a phrase structure tree PT. The existence of a path from $^*\mathrm{E}_r$ to e_{lex} is assured for the same reason described in the explanation of A-Step1.

The phrase structure tree PT generated by the phrase structure tree generation algorithm satisfying DT=dependency_tree(PT) is shown as follows:

In the case that DT={n_r}, the algorithm generates PT at N-Step2. PT is a phrase structure tree containing one node n_r. From the definition of dependency_tree, dependency_tree(PT)={n_r} is true. In the case that DT is a dependency tree which consists of a set of arcs, the phrase structure tree generation algorithm, the definition of dependency_tree and (A) make the following equation.

dependency_tree(PT)
 = dependency_tree(PATH ∪ {$^*\mathrm{e}_t$} ∪ PT_1 ∪$\cdots$∪ PT_m)
 = dependency_tree(PATH)⊎dependency_tree({$^*\mathrm{e}_t$}) ⊎dependency_tree(PT_1)⊎$\cdots$⊎ depen-

dency_tree(PT_m)

$= \text{dependency\_tree}(\{{*}e_t\}) \uplus \text{dependency\_tree}(PT_1) \uplus \cdots \uplus \text{dependency\_tree}(PT_m)$

Assume that PT_i corresponding to DT_i,$*E_i$ in A-Step4 satisfies the following.

$\text{dependency\_tree}(PT_i)=DT_i \ (1 \leq i \leq m)$

Now, PT generated at A-Step5 generates DT as shown below.

$$\begin{aligned}&\text{dependency\_tree}(PT)\\&= ds_t \uplus DT_1 \uplus \cdots \uplus DT_m\\&= DT\end{aligned}$$

□

3.6.2 Correspondence between Phrase Structure Forest and Dependency Forest

Section 3.6.1 showed that the initial dependency forest satisfies the completeness and soundness with respect to the phrase structure forest. Since the reduced dependency forest has the same set of generalized dependency trees as the initial dependency forest, the soundness and the completeness between the (reduced) dependency forest and the phrase structure forest are assured. The correspondences between the phrase structure trees (phrase structure trees) in the phrase structure forest and the dependency trees in the dependency forest are not necessarily simple one to one relations. One phrase structure tree may correspond to more than one dependency tree, whereas more than one phrase structure tree may correspond to one dependency tree. Considering the variety (one meaning can be expressed by more than one expression) and the ambiguity (one expression expresses more than one meaning) encoded in natural languages, these multiple-correspondences may be natural. The correspondences between phrase structure trees and dependency trees are discussed in the next section by referring to the experiments for sentence analysis using a PDG prototype system.

3.7 Experiment for Analysis of Example Sentences

One of the design targets of PDG is the suppression of the combinatorial explosion caused by a variety of ambiguities using the packed shared data structures. This section describes the experiment for analyzing typical ambiguous sentences using PDG grammar rules, which contain various kinds of ambiguities. This section also discusses the relation between the phrase structure forest and the dependency forest, and the generation of non-projective dependency trees based on real analysis examples. The performance of the algorithm is also one of the important factors from a practical point of view. The algorithms for parsing, generation of phrase structure forest and initial dependency forest and dependency forest reduction described in this thesis are implemented for verifying the PDG's analysis. The practical implementation

```
=========== s/Sentence ===========
 (R1) s/VP  → np/NP,vp/VP          : [arc(sub,NP,VP)]             % Declarative sentence
 (R2) s/VP  → vp/VP                : []                           % Imperative sentence
========= np/Noun Phrase ========
 (R3) np/N  → n/N                  : []                           % Single noun
 (R4) np/N2 → n/N1,n/N2            : [arc(nc,N1,N2)]              % Compound noun
 (R5) np/N  → det/DET,n/N          : [arc(det,DET,N)]             % Determiner
 (R6) np/NP → np/NP,pp/PP          : [arc(npp,PP,NP)]             % Prepositional phrase attachment
 (R7) np/N  → ving/V,n/N           : [arc(adjs,V,N)]              % Adjectival usage (subject)
 (R8) np/N  → ving/V,n/N           : [arc(adjo,V,N)]              % Adjectival usage (object)
 (R9) np/V  → ving/V,np/NP         : [arc(obj,NP,V)]              % gerund phrase
(R10) np/V  → ving/V,np/NP,pp/PP   : [arc(obj,NP,V),arc(vpp,PP,V)] % gerund phrase with PP
(R11) np/NP → np/NP0,and/AND,np/NP : [arc(and,NP0,NP),arc(cnj,AND,NP0)] % Coordination (and)
(R12) np/NP → np/NP0,or/OR,np/NP   : [arc(or,NP0,NP),arc(cnj,OR,NP0)]   % Coordination (or)
========= vp/Verb phrase ========
(R13) vp/V  → v/V                  : []                           % Intransitive verb
(R14) vp/V  → v/V,np/NP            : [arc(obj,NP,V)]              % Transitive verb
(R15) vp/V  → be/BE,ving/V,np/NP   : [arc(obj,NP,V),arc(prg,BE,V)] % Progressive
(R16) vp/BE → be/BE,np/NP          : [arc(dsc,NP,BE)]             % Copular
(R17) vp/VP → vp/VP,pp/PP          : [arc(vpp,PP,VP)]             % PP-attachment
(R18) vp/VP → adv/ADV,vp/VP        : [arc(adv,ADV,VP)]            % Adverb modification
(R19) vp/V  → v/V,np/NP,adv/ADV,relc/RELP                         % Non-projective pattern
                                   : [arc(obj,NP,V),arc(adv,ADV,V),arc(rel,RELP,NP)]
======== pp/Prepositional phrase ========
(R20) pp/P  → pre/P,np/NP          : [arc(pre,NP,P)]
```

Fig.3.23 Grammar for the example sentences

and its evaluation are subjects for future work. The following experiment utilizes a prototype PDG system implemented in Prolog.

3.7.1 PDG Grammar Rules for Example Sentences

Fig.3.23 shows PDG grammar rules used for analyzing the example sentences containing various kinds of prototypical ambiguities. The POSs in the grammar rules are determiner(det), n(noun), be-verb(be), present particle of verb (ving), verb (v), preposition (pre), adverb (adv) and relational phrase (relp). The grammar rules are not for linguistic analysis but for experiment with respect to the PDG framework and algorithms. The grammar rules include the following kinds of ambiguities.

PP-attachment : Two kinds of attachment ambiguities in R6(noun attachment) and R10,R17(verb attachment)

Coordination scope : R11(and) and R12(or) are coordination noun phrase rules

Be-verb interpretation : Two structural ambiguities caused by be-verb interpretations, i.e., R15(present progressive) and R16(copula)

Interpretation of the present progressive form of verb : the following three ambiguities of the present progressive form of verb are described

(a) adjective usage where modified noun occupies the subject role (R7)
(b) adjective usage where modified noun occupies the object role (R8)
(c) gerund usage (R9,R10)

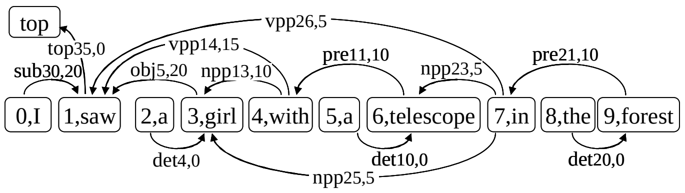
R8 and R9 are similar since both rules prescribe the relation between noun and verb as object. However they have different interpretations in phrase head and generate the different structures of dependency trees. The grammar contains declarative form (R1) and imperative sentence form (R2) to produce structural ambiguities in combination with POS ambiguities of words for sentences like "Time flies like an arrow." (R19) is a rule for generating a non-projective dependency structure.

3.7.2 Analysis of Prototypical Ambiguous Sentences

The example for the ambiguous sentence with POS ambiguities has already been shown in the previous sections in detail. The examples in the following sections show prototypical syntactic ambiguities, i.e., PP-attachment ambiguity, coordination scope ambiguity and ambiguities in structural interpretations.

(1) PP-attachment Ambiguity

Fig.3.24 shows the dependency forest for the sentence "I saw a girl with a telescope in the forest," which has PP-attachment ambiguities. Each arc in the dependency graph has an arc name attached by arc-ID and preference score[*19]. The table in Fig.3.24 shows POS and position information of each node. This sentence has no POS ambiguities but has PP-attachment ambiguities for preposition "with" (two ambiguities: *npp*13,*vpp*14) and "in" (three ambiguities: *npp*23,*npp*25,*vpp*26). CM in Fig.3.24 inhibits some of the combinations of these ambiguous arcs. *npp*13 and *vpp*14 (or *npp*23, *npp*25 and *vpp*26) have no co-occurrence relation because they have the same position (the single role constraint). The co-occurrence between *vpp*14 and *npp*25 is also inhibited. If this co-occurrence constraint does not exist, the dependency forest has six interpretations caused by two PP-attachment ambiguities ($2 * 3 = 6$). CM(14,25)$\neq$ ○, which



0,I	: [i]-n-0
1,saw	: [saw]-v-1
2,a	: [a]-det-2
3,girl	: [girl]-n-3
4,with	: [with]-pre-4
5,a	: [a]-det-5
6,telescope	: [telescope]-n-6
7,in	: [in]-pre-7
8,the	: [the]-det-8
9,forest	: [forest]-n-9
top	: [top]-x-top

	30	4	5	13	14	10	11	25	23	26	20	21	35
30	–	○	○	○	○	○	○	○	○	○	○	○	○
4	○	–	○	○	○	○	○	○	○	○	○	○	○
5	○	○	–	○	○	○	○	○	○	○	○	○	○
13	○	○	○	–		○	○	○	○	○	○	○	○
14	○	○	○		–	○	○		○	○	○	○	○
10	○	○	○	○	○	–	○	○	○	○	○	○	○
11	○	○	○	○	○	○	–	○	○	○	○	○	○
25	○	○	○	○		○	○	–			○	○	○
23	○	○	○	○	○	○	○		–		○	○	○
26	○	○	○	○	○	○	○			–	○	○	○
20	○	○	○	○	○	○	○	○	○	○	–	○	○
21	○	○	○	○	○	○	○	○	○	○	○	–	○
35	○	○	○	○	○	○	○	○	○	○	○	○	–

Fig.3.24 DF for the example sentence including PP-attachments

[*19] Preference scores show the degree of preference of arcs (Hirakawa, 2006a). The preference score is not used in this chapter.

corresponds to the projectivity constraint, excludes a non-projective dependency tree from a set of well-formed dependency trees in the dependency forest. This dependency forest has five well-formed dependency trees, which are possible interpretations for the example sentence.

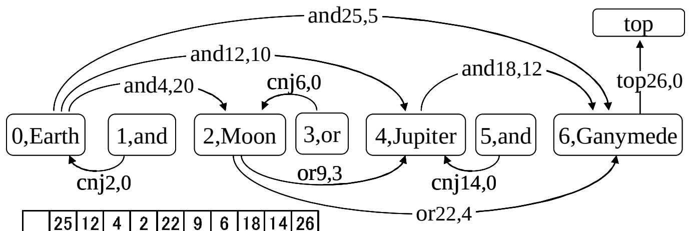
The sizes of the phrase structure forest, the initial dependency forest and the reduced dependency forest for this example are 25, 18 and 13, respectively. The phrase structure forest contains five phrase structure trees[*20] corresponding to the five interpretations of the sentence. The initial dependency forest and the reduced dependency forest have five IDed dependency trees, which correspond to the five generalized dependency trees. *obj*5,*npp*13, *vpp*14 and *pre*11 have 2, 1, 1 and 1 equivalent arcs in the initial dependency forest. For example, *obj*5 and its equivalent arcs are generated from the edges, shown below in diagrammatic form, originated in the grammar rule (R14).

<1, 4, vp/([saw]-v-1) → v(ID:109) np(ID:126) ・, {arc(obj-5,[girl]-n-3,[saw]-v-1)}>
<1, 7, vp/([saw]-v-1) → v(ID:109) np(ID:163) ・, {arc(obj-15,[girl]-n-3,[saw]-v-1)}>
<1,10, vp/([saw]-v-1) → v(ID:109) np(ID:203) ・, {arc(obj-28,[girl]-n-3,[saw]-v-1)}>

The first edge has the coverage from 1 to 4 ("saw a girl"), phrase head [saw]-v-1, constituent sequence v(ID:109)[*21] np(ID:126) and the obj arc with arc-ID 5. The above equivalent arcs are generated from the combination with edges np(ID:126), np(ID:163) and np(ID:203) that correspond to noun phrases with different coverage. The reduced dependency forest has no equivalent arcs because all equivalent arcs in the initial dependency forest are merged.

(2) Coordination Scope Ambiguity

Fig.3.25 shows the dependency forest for the sentence "Earth and Moon or Jupiter and Ganymede," which has coordination scope ambiguities. "Earth" and "Moon" have two and



	25	12	4	2	22	9	6	18	14	26
25	–			○	○	○	○	○	○	○
12		–		○		○	○	○	○	○
4			–	○	○	○	○	○	○	○
2	○	○	○	–	○	○	○	○	○	○
22	○		○	○	–		○	○	○	○
9	○	○	○	○		–	○	○	○	○
6	○	○	○	○	○	○	–	○	○	○
18	○	○	○	○	○	○	○	–	○	○
14	○	○	○	○	○	○	○	○	–	○
26	○	○	○	○	○	○	○	○	○	–

0,Earth	: [Earth]-n-0
1,and	: [and]-and-1
2,Moon	: [Moon]-n-2
3,or	: [or]-or-3
4,Jupiter	: [Jupiter]-n-4
5,and	: [and]-and-5
6,Ganymede	: [Ganymede]-n-6
top	: [top]-x-top

Fig.3.25 DF for the example sentence including conjunctions

[*20] In judging the equivalence of the phrase structure trees, phase heads (head nodes) are taken into consideration.

[*21] The packed edge whose category is v and edge-ID is 109.

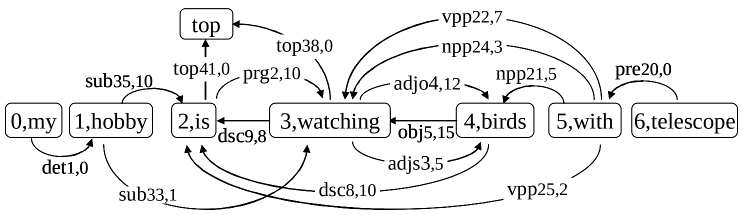
three outgoing arcs, respectively, corresponding to coordination scope ambiguities. CM(22,12)≠ ○, which corresponds to the projectivity constraint, excludes a non-projective dependency tree from a set of well-formed dependency trees in the dependency forest. This dependency forest has five well-formed dependency trees, which are possible interpretations for the example sentence.

The sizes of the phrase structure forest, the initial dependency forest and the reduced dependency forest for this example are 18, 17 and 10, respectively. The phrase structure forest contains five phrase structure trees corresponding to the five interpretations of the sentence. The initial dependency forest and the reduced dependency forest have five IDed dependency trees, which correspond to the five generalized dependency trees. *or*22,*or*9,*cnj*6, *and*18 and *cnj*14 have 1, 1, 1, 2, and 2 equivalent arcs in the initial dependency forest, respectively. The reduced dependency forest has no equivalent arcs because all equivalent arcs in the initial dependency forest are merged. The coordination scope ambiguity is similar to the PP-attachment ambiguity in the previous section but is different from the PP-attachment ambiguity because it has the modification scope problem described below.

(3) Ambiguity in Structural Interpretation

Fig.3.26 shows the dependency forest for the sentence "My hobby is watching birds with a telescope," which has ambiguities such as the interpretation of be-verb (present progressive form or copula), the interpretation of "watching birds" (*adjs*3,*adjo*4, *obj*5), and PP-attachment (*npp*21,*vpp*22,*npp*24,*vpp*25). This sentence has ten interpretations.

The sizes of the phrase structure forest, the initial dependency forest and the reduced dependency forest for this example are 23, 24 and 16, respectively. The phrase structure forest contains eight phrase structure trees corresponding to ten interpretations of the sentence. The initial dependency forest and the reduced dependency forest have ten IDed dependency trees, which correspond to the ten generalized dependency trees. *dsc*9,*dsc*8,*obj*5, *npp*21 and *vpp*22 have



0,my	:	[my]-det-0
1,hobby	:	[hobby]-n-1
2,is	:	[is]-be-2
3,watching	:	[watching]-ving-3
4,birds	:	[birds]-n-4
5,with	:	[with]-pre-5
6,telescope	:	[telescope]-n-6
top	:	[top]-x-top

	1	35	33	2	4	3	9	8	5	21	24	26	22	20	41	38
1	–	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
35	○	–			○	○	○	○	○	○	○	○	○	○	○	
33	○		–	○					○	○			○	○		○
2	○		○	–					○	○			○	○		○
4	○	○			–			○		○		○		○	○	
3	○	○				–		○		○		○		○	○	
9	○	○					–		○	○	○	○	○	○	○	
8	○	○			○	○		–		○		○		○	○	
5	○	○	○	○			○		–	○	○	○	○	○	○	○
21	○	○	○	○	○	○	○	○	○	–				○	○	○
24	○	○					○		○		–			○	○	
26	○	○			○	○	○	○	○			–		○	○	
22	○	○	○	○			○		○				–	○	○	○
20	○	○	○	○	○	○	○	○	○	○	○	○	○	–	○	○
41	○	○			○	○	○	○	○	○	○	○	○	○	–	
38	○		○	○					○	○			○	○		–

Fig.3.26 DF for the example sentence including structural ambiguities

2, 2, 5, 2, and 2 equivalent arcs in the initial dependency forest, respectively. In contrast to the example in Section 3.7.2, the equivalent arcs are generated from more than one grammar rule. For example, the equivalent arcs of *obj*5 are generated from the edges, shown below, originated in the grammar rule (R9), (R10) and (R15)

(R9) ⇒ <3, 5, np/([watching]-ving-3) → ving(ID:121) np(ID:130) • ,
{arc(obj-5,[birds]-n-4,[watching]-ving-3)}>
(R10) ⇒ <3, 7, np/([watching]-ving-3) → ving(ID:121) np(ID:130) pp(ID:176) • ,
{arc(obj-6,[birds]-n-4,[watching]-ving-3),arc(vpp-22,[with]-pre-5,[watching]-ving-3)} >
(R15) ⇒ <2,5, vp/([watching]-ving-3) → be(ID:117) ving(ID:121) np(ID:130) • ,
{arc(prg-2,[is]-be-2,[watching]-ving-3),arc(obj-7,[birds]-n-4,[watching]-ving-3)} >

The reduced dependency forest has no equivalent arcs. In this example, the number of the phrase structure trees in the phrase structure forest is smaller than number of the generalized dependency trees in the dependency forest. One phrase structure tree corresponds to more than one dependency tree. The following section discusses the correspondence between phrase structure trees and dependency trees.

3.7.3 1 to N Correspondence from Phrase Structure Tree to Dependency Trees

The correspondence between a phrase structure tree and a dependency tree is assured in PDG, but sometimes one phrase structure tree has more than one corresponding dependency tree and more than one phrase structure tree has one corresponding dependency tree.

When one phrase structure has more than one interpretation, one phrase structure tree may correspond to more than one dependency tree. For example, when "watching birds" is assigned one phrasal structure where the verb in present particle form modifies the noun, two dependency structures "watching $\xrightarrow{\text{subj}}$ birds" and "watching $\xrightarrow{\text{obj}}$ birds" are assigned to the phrasal structure. This happens when there exists more than one grammar rule, which has the same rewriting rule but has different dependency structure parts. This is the case for (R7) and (R8) in Fig.3.23. (R7) and (R8) are arbitrary rules introduced for verifying the dependency forest. Two kinds of ambiguities in 1 to N mapping from phrase structure tree to dependency trees are considered, i.e., the ambiguities in syntactic relation and semantic relation. The former means the ambiguities in functional assignments (subject, object, etc.) for phrase structures. The functional structures and phrase structures have close relation and the difference in functional structures can be reflected by the difference of phrase structure*[22]. Therefore, it seems not to be usual for more than one functional structure to be assigned to one rewriting rule. In contrast, the assignment of more than one semantic structure seems to be quite a general phenomenon. However, introduction of

*[22] For example, the category in the rewriting rule is segmented into more detailed categories reflecting the difference of functional assignments. This segmentation works to remove ambiguous mapping from one phrase structure to more than one functional structure.

```
# Phrase Structure Tree [1]
s[0,7,is/be]:207
  +--np[0,2,hobby/n]:108
  |     +--det[0,1,my/det]:101
  |     +--n[1,2,hobby/n]:104
  +--vp[2,7,is/be]:182
        +--be[2,3,is/be]:117
        +--np[3,7,birds/n]:179
              +--np[3,5,birds/n]:132
              |     +--ving[3,4,watching/ving]:121
              |     +--n[4,5,birds/n]:128
              +--pp[5,7,with/pre]:176
                    +--pre[5,6,with/pre]:165
                    +--np[6,7,telescope/n]:170
                          +--n[6,7,telescope/n]:168
```

```
# Dependency Structure [1]
 [is/be,2]
   + <-(dsc-31)- [birds/n,4]
   |          + <-(adjs-3)- [watching/ving,3]
   |          + <-(npp-23)- [with/pre,5]
   |                     + <-(pre-20)- [telescope/n,6]
   + <-(sub-35)- [hobby/n,1]
              + <-(det-1)- [my/det,0]

 # Dependency Structure [2]
 [is/be,2]
   + <-(dsc-31)- [birds/n,4]
   |          + <-(adjo-4)- [watching/ving,3]
   |          + <-(npp-23)- [with/pre,5]
   |                     + <-(pre-20)- [telescope/n,6]
   + <-(sub-35)- [hobby/n,1]
              + <-(det-1)- [my/det,0]
```

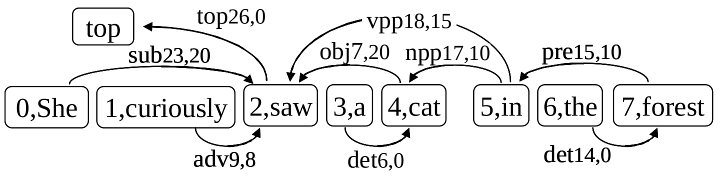
Fig.3.27 Mapping from one phrase structure tree to two dependency tree

1 to N mapping (from phrase structure to dependency structures) into grammar rules may cause problems in terms of system performance and grammar rule maintenance. In general, sentence analysis approaches, which treat syntactic analysis and semantic analysis independently, are widely proposed and utilized. The framework itself, for mapping one phrase structure tree to more than one dependency tree, is independent of the linguistic discussion here. Rules with mappings to semantic structures can be utilized properly with respect to the requirements from the design and development of the grammar.

The example in Section 3.7.2 (Fig.3.26) contains a phrase structure tree which corresponds to two dependency structures generated from (R7) and (R8). Fig.3.27 shows this phrase structure tree and dependency trees.

3.7.4 N to 1 Correspondence from Phrase Structure Trees to One Dependency Tree

Spurious ambiguity (Noro et al., 2002) is one of the examples of N to 1 mapping from phrase structures to dependency structure. The real ambiguity is an ambiguity where the difference in syntactic structures represents the difference in semantic interpretations. The spurious ambiguity is an ambiguity where the difference in syntactic structures does not represent the difference in semantic interpretations, or an ambiguity caused by linguistically illegal syntactic structure generated by incomplete grammar rules. The spurious ambiguity is an important issue in grammar development from corpora (Noro et al., 2005). Although it is not CFG, CCG (Combinatory Categorial Grammar) has a lot of spurious ambiguities due to the flexibility of rule application. The method for obtaining one normal form tree is proposed (Eisner, 1996a). This method assures that one phrase structure tree among the phrase structure trees in one semantic class is obtained based on the definition that phrase structure trees which have the same set of leaf CCG category have the same meaning. In PDG, phrase structure trees corresponding to the same generalized dependency tree (interpretation of a sentence) can be classified into one semantic class.



0,She	: [she]-n-0
1,curiously	: [curiously]-adv-1
2,saw	: [saw]-v-2
3,a	: [a]-det-3
4,cat	: [cat]-n-4
5,in	: [in]-pre-5
6,the	: [the]-det-6
7,forest	: [forest]-n-7
top	: [top]-x-top

	23	9	6	7	17	18	14	15	26
23	–	O	O	O	O	O	O	O	O
9	O	–	O	O	O	O	O	O	O
6	O	O	–	O	O	O	O	O	O
7	O	O	O	–	O	O	O	O	O
17	O	O	O	O	–		O	O	O
18	O	O	O	O		–	O	O	O
14	O	O	O	O	O	O	–	O	O
15	O	O	O	O	O	O	O	–	O
26	O	O	O	O	O	O	O	O	–

Fig.3.28 DF containing the mapping from N phrase structure trees to 1 dependency tree (spurious ambiguity)

Fig.3.28 shows the dependency forest for "She curiously saw a cat in the forest" using the example grammar which has a spurious ambiguity. There is only one co-occurrence constraint between *npp*17 and *vpp*18 that corresponds to the single role constraint. The dependency forest has two dependency trees, which has different governors for the part "in the forest." The phrase structure forest has three phrase structure trees. The initial dependency forest has three IDed dependency trees and two generalized dependency trees and the reduced dependency forest has two IDed dependency trees and two generalized dependency trees. The spurious ambiguities are generated from the difference of rule application order of (R17) and (R18) for attaching a modification phrase to a verb phrase. Fig.3.29 shows the phrase structure trees and the dependency tree.

Obviously, the method of identifying the semantic class based on dependency tree does not capture all semantic aspects in natural languages. For example, the subtle semantic difference (Eisner, 1996a)[*23] and the ambiguities in number/quantifier scope[*24] have to be considered in discussing the equivalent semantic class. The treatment of difference in semantic interpretation requires further study. Mel'cuk (1988) describes some linguistic structures where ordinary dependency structure fails to express the interpretations of a sentence. These structures are classified into two categories, the structures, which cannot be treated by phrase structure properly, and the others. The former includes the model theoretic interpretation of a sentence. The latter is observed when a dependency structure has a head word which has dependants located at the right-hand side and the left-hand side of the headword. In this case, the dependency structure has ambiguities in modification scope, i.e., the right-hand modifier modifies only the headword

[*23] For example, "softly knock twice" has two equivalent semantic interpretations softly(twice(knock)) and twice(softly(knock)), whereas "intentionally knock twice" has two different semantic interpretations intentionally(twice(knock)) and twice(intentionally(knock)).

[*24] The model theoretic ambiguities as observed in "Three men bought ten cups" cannot be distinguished by standard phrase structure and dependency structure representations.

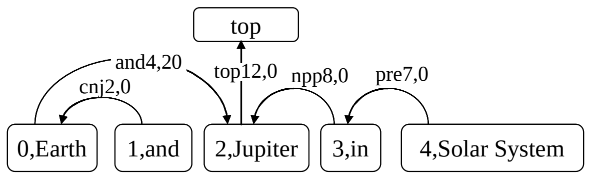
```
# Phrase Structure Tree [1]
s[0,8,saw/v]:192
  +--np[0,1,she/n]:103
  |     +--n[0,1,she/n]:101
  +--vp[1,8,saw/v]:176
        +--vp[1,5,saw/v]:146
        |     +--adv[1,2,curiously/adv]:109
        |     +--vp[2,5,saw/v]:142
        |           +--v[2,3,saw/v]:112
        |           +--np[3,5,cat/n]:133
        |                 +--det[3,4,a/det]:126
        |                 +--n[4,5,cat/n]:129
        +--pp[5,8,in/pre]:172
                    +--pre[5,6,in/pre]:153
                    +--np[6,8,forest/n]:163
                          +--det[6,7,the/det]:156
                          +--n[7,8,forest/n]:159
# Phrase Structure Tree [2]
s[0,8,saw/v]:192
  +--np[0,1,she/n]:103
  |     +--n[0,1,she/n]:101
  +--vp[1,8,saw/v]:176
        +--adv[1,2,curiously/adv]:109
        +--vp[2,8,saw/v]:175
              +--vp[2,5,saw/v]:142
              |     +--v[2,3,saw/v]:112
              |     +--np[3,5,cat/n]:133
              |           +--det[3,4,a/det]:126
              |           +--n[4,5,cat/n]:129
              +--pp[5,8,in/pre]:172
                    +--pre[5,6,in/pre]:153
                          +--np[6,8,forest/n]:163
                                +--det[6,7,the/det]:156
                                +--n[7,8,forest/n]:159
```

```
#### Dependency Tree
[saw/v,2]
  + <-(adv-9)- [curiously/adv,1]
  + <-(obj-7)- [cat/n,4]
  |          + <-(det-6)- [a/det,3]
  + <-(sub-23)- [she/n,0]
  + <-(vpp-18)- [in/pre,5]
             + <-(pre-15)- [forest/n,7]
                        + <-(det-14)- [the/det,6]
```

Fig.3.29 The example of mapping from N phrase structure trees to 1 dependency tree (spurious ambiguity)

or the phrase including the left-hand modifier. This problem is called the "modification scope problem" in this thesis.

Fig.3.30 shows the dependency forest for "Earth and Jupiter in the Solar System." This sentence has two interpretations, i.e., the prepositional phrase modifies only the headword "Jupiter" or the phrase "Earth and Jupiter." The phrase structure forest has two phrase structure trees corresponding to these two interpretations. On the other hand, the initial dependency forest has



0,Earth	: [Earth]-n-0
1,and	: [and]-and-1
4,Jupiter	: [Jupiter]-n-4
5,in	: [in]-pre-5
6,Solar System	: [solar_system]-n-6
top	: [top]-x-top

	4	2	8	7	12
4	–	○	○	○	○
2	○	–	○	○	○
8	○	○	–	○	○
7	○	○	○	–	○
12	○	○	○	○	–

Fig.3.30 DF containing the mapping from N phrase structure trees to 1 dependency tree (real ambiguity)

two IDed dependency trees and one generalized dependency tree and the reduced dependency forest has one IDed dependency tree corresponding to one generalized dependency tree. The two phrase structure trees and one dependency tree are shown in Fig.3.31.

Mel'cuk (1988) proposes introducing a concept called "grouping" into the dependency structure framework to solve the modification scope problem. Grouping is theoretically equivalent to phrase in the sense that it specifies the word coverage information. However, grouping information is not attached to every part of the structure but to some specific structures including the "conjoined structure" and "operator word" such as "not" and "only." The grammar framework for a machine translation system (Amano et al., 1989) incorporates a mechanism similar to the grouping*[25]. In this grammar development for the real-world application, the scope nodes are used only for conjoined structures*[26]. This experience suggests the limitation of the application scope of grouping proposed by Mel'cuk is reasonable. Moreover, the treatment of the modification scope ambiguity differs from language to language. According to Mel'cuk (1988), some modification scope ambiguities are distinguishable by lexical or syntactic marking in Russian. Japanese does not have the modification scope problem inherently because Japanese has the basic grammatical constraint that modifiers should be located at the left-hand side of their modificand. In the PDG

```
# Parse Tree [1]
np[0,5,jupitor/n]:142
  +--np[0,3,jupitor/n]:122
  |      +--np[0,1,earth/n]:103
  |      |      +----n[0,1,earth/n]:101
  |      +--and[1,2,and/and]:110
  |      +--np[2,3,jupitor/n]:115
  |             +----n[2,3,jupitor/n]:113
  +--pp[3,5,in/pre]:140
         +--pre[3,4,in/pre]:128
         +--np[4,5,solar_system/n]:133
                +--n[4,5,solar_system/n]:131
# Parse Tree [2]
np[0,5,jupitor/n]:142
  +--np[0,1,earth/n]:103
  |      +----n[0,1,earth/n]:101
  +--and[1,2,and/and]:110
  +--np[2,5,jupitor/n]:141
         +--np[2,3,jupitor/n]:115
         |      +--n[2,3,jupitor/n]:113
         +--pp[3,5,in/pre]:140
                +--pre[3,4,in/pre]:128
                +--np[4,5,solar_system/n]:133
                       +----n[4,5,solar_system/n]:131
```

```
#### Dependency Structure
[jupitor/n,2]
 + <-(and-4)-[earth/n,0]
 |              + <-(cnj-2)-[and/and,1]
 + <-(npp-8)-[in/pre,3]
                + <-(pre-7)-[solar_system/n,4]
```

Fig.3.31 The example of mapping from N phrase structure trees to 1 dependency tree (real ambiguity)

*[25] A special node called "scoping node" is introduced to specify the scope of a dependency modification as required.

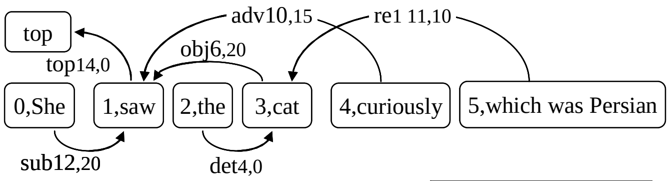
*[26] This is the case for an English-to-Japanese system. The requirement level may differ according to the language pairs. For example, translation between languages in the same family may not require a grouping mechanism because the modification scope ambiguities are avoided by bypassing, i.e., an ambiguous source language structure is mapped to the corresponding target language structure without disambiguation.

framework, equivalent arcs represent the difference of modification scope as shown in the previous section. Therefore, the modification scope problem may be avoided by introducing grouping into the treatment of equivalent arcs. This is a future task.

3.7.5 Generation of Non-projective Dependency Tree

The projectivity constraint[*27] is a basic constraint adopted by many dependency analysis systems and these parsers are called projective parsers. Projective parsers fail to analyze sentences with non-projective structures. Almost all sentences in many languages are projective, but some types of non-projective sentences exist (Mel'cuk, 1988). For example, "John saw a dog yesterday which was a Yorkshire Terrier." in English, "私は本を東京に買いに昨日行きました。"(I went to Tokyo to buy a book yesterday) in Japanese have projective dependency structures. McDonald et al. (2005) reported the non-projective parser outperformed the projective parser in overall accuracy for the analysis of Czech, which has a high degree of word order freedom compared with English.

As described in Section 3.4.2, the mapping between the constituent sequence (the body of grammar rule) and the partial dependency tree (the dependency structure of the grammar rule) is defined in the grammar rule in PDG. This framework in combination with the description ability of the C-Matrix enables a controlled non-projectivity instead of all-or-nothing non-projectivity. The controlled non-projectivity means that the non-projective structures are defined by some rules, which prescribe the well-formedness conditions. (R19) in Fig.3.23 is a grammar rule for a phrase pattern where an adverb is inserted before a relative clause, and produces a well-formed non-projective dependency structure. Fig.3.32 shows the dependency tree for "She saw the cat curiously which was Persian."[*28] obtained by the example PDG grammar. The dependency



0,She	: [she]-n-0
1,saw	: [saw]-v-1
2,the	: [the]-det-2
3,cat	: [cat]-n-3
4, curiously	: [curiously]-adv-4
5,which was Persian	: [which_was_percian]-relc-5
top	: [top]-x-top

	12	4	6	10	11	14
12	–	○	○	○	○	○
4	○	–	○	○	○	○
6	○	○	–	○	○	○
10	○	○	○	–	○	○
11	○	○	○	○	–	○
14	○	○	○	○	○	–

Fig.3.32 Example of non-projective dependency tree generation

[*27] The projectivity condition consists of two conditions, i.e., "no cross dependency exits" and "no dependency covers the top node." The second condition is unnecessary when a special top node is introduced at the top or end of a sentence.

[*28] This is an artificial example only for showing the rule applicability.

forest has one non-projective dependency tree.

3.8 Concluding Remarks for Chapter 3

This chapter described the multilevel packed shared data connection model that is the basic analysis model adopted by PDG and explained two packed shared data structures of PDG, i.e., the phrase structure forest and the dependency forest. The completeness and the soundness of the correspondence between the phrase structure forest and the dependency forest are assured. This means the sentence interpretations represented in packed shared phrase structure and the sentence interpretations represented in packed shared dependency structure have mappings. This thesis also described the experimental results for analyzing some typical ambiguous sentences using an example PDG grammar.

The current implementation of the PDG system focuses on the feasibility study of the PDG framework. The practical PDG system and its performance evaluation are future tasks. Extension of the PDG grammar formalism (such as the introduction of optional element specification and feature conditions) and improvement in performance by efficient codes and optimizing methods based on grammar analysis, should be studied in order to realize a practical system.

Chapter 4

Optimum Solution Search

PDG is a kind of framework for dependency analysis because the final output of PDG is one or more dependency trees. This chapter describes the optimum solution search algorithm for PDG and shows some experiments for estimating the behavior and computational complexity of the algorithm.

As described in (McDonald et al., 2005), various dependency analysis methods are proposed. Some methods utilize lexicalized phrase-structure parsers with the ability to output dependency information (Collins, 1999; Charniak, 2000) and some methods obtain dependency trees directly (Ozeki, 1994; Katoh and Ehara, 1989; Eisner, 1996b; Yamada and Matsumoto, 2003; Nivre and Scholz, 2004). In this thesis, parsers in the former category are called phrase-structure based dependency parsers and those in the latter category are called direct dependency parsers. PDG is in the former category because it utilizes a lexicalized phrase-structure parser to generate packed shared data structure (dependency forest) based on structure mapping information in grammar rules.

(Collins, 1999; Charniak, 2000) are basically lexicalized phrase structure parsers and work as dependency parsers by attaching a function for conversion from a phrase-structure to a dependency structure. The dependency tree for a sentence is generated from the headed phrase structure tree obtained by the phrase structure parser. For example, each nonterminal symbol and its child constituents in the phrase structure tree correspond to the dependency structure that has one governor node (the phrase head of the nonterminal symbol) and its dependant nodes (the phrase heads of the child constituents) in (Collins, 1999). On the other hand, PDG generates a dependency structure based on structure mapping information in grammar rules This mechanism enables generation of flexible dependency structures with dependency relation labels. For example, PDG can provide phrase structure rules which generate non-projective dependency structures which are not produced by (Collins, 1999; Charniak, 2000) and the majority of direct dependency parsers as described in Section 4.1.3. The phrase-structure based dependency parsers have a possibility to utilize the descriptive power of the phrase structure rules to prescribe the dependency structures.

Training corpora and statistical information are used for computing the most appropriate dependency tree in many parsers. As shown in Chapter 1, one class of parsers adopts a history-based approach (Black et al., 1992) in which each tree-building procedure uses a probability model p(AlB) to weight any action A based on the available context, or history, B. (Yamada and Matsumoto, 2003; Nivre and Scholz, 2004) can be regarded as history-based direct dependency parsers which choose the optimum decision during the parsing process based on information obtained from the training data. Another class of parsers generates various dependency graphs encompassing all possible dependency trees for a sentence[*1] and searches for the optimum tree based on preference scores[*2] attached to the dependency graph (Ozeki, 1994; Katoh and Ehara, 1989; Hirakawa, 2001; McDonald et al., 2005). This method is called the all-pairs based approach and a dependency graph with preference scores is called scored dependency graph in this thesis. In general, the history-based method seems to be more efficient than the all-pairs based method because it makes decisions before completing the full parse. However, the history-based method may fail to obtain the optimum solution because it does not utilize the full parse information. PDG is classified as a all-pairs based method since it searches for the optimum tree in a dependency forest with a scored dependency graph. A dependency forest with preference scores is sometimes explicitely called scored dependency forest.

Rest of this thesis focuses on all-pairs based methods and discusses some approaches to the optimum tree search for dependency graphs and proposes an optimum tree search algorithm for the dependency forest named the “graph branch algorithm.” PDG (and the graph branch algorithm) is an successor to the sentence analysis method based on semantic dependency graph (Hirakawa and Amano, 1989b; Hirakawa, 2001).

4.1 Optimum Dependency Tree Search Methods for Dependency Graphs

4.1.1 Basic Framework

Scored dependency graphs are widely used as packed shared data structures representing a set of dependency trees. Fig.4.1 shows the basic framework of the optimum dependency tree search in a dependency graph. In general, nodes in a dependency graph correspond to words in the sentence and the arcs show some kind of labeled or non-labeled dependency relations between nodes. Each arc has a preference score representing plausibility of the relation. The well-formed dependency tree constraint is a set of well-formed constraints which should be satisfied by all dependency trees representing sentence interpretations. A pair of a dependency graph and a well-formed dependency tree constraint defines a set of well-formed dependency trees. The score of a dependency tree is the sum total of arc scores[*3]. The optimum tree is a dependency tree

[*1] In fact, a set of possible dependency trees is represented by a dependency graph and a set of constraints as shown in Section 4.1.

[*2] Preference score represents the plausibility of the arc.

[*3] Dependency arc numbers in each well-formed dependency tree for a sentence are not necessarily be the same because some of them have compound word WPPs. The adjustment of the scores for compound WPP nodes are introduced in the scoring process described in Chapter 5

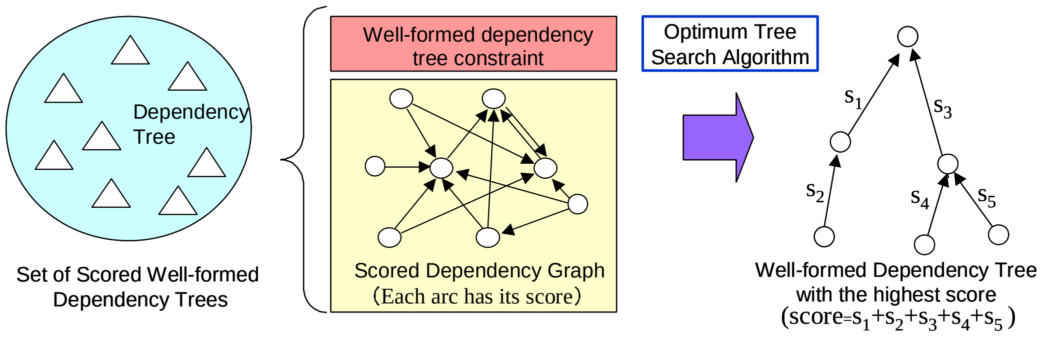

Dependency Tree
Set of Scored Well-formed Dependency Trees
Well-formed dependency tree constraint
Scored Dependency Graph (Each arc has its score)
Optimum Tree Search Algorithm
s1
s2
s3
s4
s5
Well-formed Dependency Tree with the highest score (score=s1+s2+s3+s4+s5)

Fig.4.1 Framework of optimum tree search in a scored dependency graph

with the highest score in the set of dependency trees defined by the dependency graph and the well-formed dependency tree constraint.

4.1.2 Dependency Graph

Dependency graphs are classified into some classes based on the types of nodes and arcs. This thesis assumes three types of nodes, i.e., word-type, WPP-type and concept-type[*4]. The types of dependency graphs are called a word dependency graph, a WPP dependency graph and a concept dependency graph, respectively, in this thesis. Dependency graphs are also classified into non-labeled and labeled graphs. There are some types of arc labels such as syntactic label (ex. "subject," "object") and semantic label (ex. "agent," "target"). Various types of dependency graphs are used in existing systems according to these classifications, such as non-label word dependency graph (Lee and Choi, 1997; Eisner, 1996b; McDonald et al., 2005), syntactic-label word dependency graph (Maruyama, 1990), semantic-label word dependency graph (Hirakawa, 2001), non-label WPP dependency graph (Ozeki, 1994; Katoh and Ehara, 1989), syntactic-label WPP dependency graph (Wang and Harper, 2004), semantic-label concept dependency graph (Harada and Mizuno, 2001) [*5].

4.1.3 Well-formedness Constraints for Dependency Tree

There can be a variety of well-formedness constraints for dependency trees from very basic and language-independent constraints to specific language-dependent constraints. This thesis focuses on the following four basic and language-independent constraints which may be embedded in data structure and/or the optimum tree search algorithm.

(C1) Coverage constraint: Every input word has a corresponding node in the tree
(C2) Single role constraint: No two nodes in a dependency tree occupy the same input position
(C3) Projectivity constraint: No arc crosses another arc

[*4] "concept" coresopnds to lexical concept defined in a system dictionary.
[*5] This data structure encompasses semantic dependency trees for one word-dependency tree.

(C4) Single valence occupation constraint: No two arcs in a tree occupy the same valence of a predicate

(C1) and (C2) are basic constraints adopted by almost all dependency parsers. (C1) and (C2) are collectively referred to as "covering constraint." (C3) is also adopted by the majority of dependency parsers which are called projective dependency parsers. A projective dependency parser fails to analyze non-projective sentences. Most sentences of a language are projective, but several types of non-projective sentences exist (Mel'cuk, 1988). The non-projective parsing model obtained improvement in overall accuracy compared with the projective model in an experiment on Czech, which has more flexible word order than English (McDonald et al., 2005). In this case, all possible non-projective dependency trees are candidates for the sentence structure because no projectivity constraint is applied in contrast to projective parsing model. This type of non-projectivity is called an uncontrolled non-projectivity in this thesis. As described below, PDG does not adopt (C3) directly. Therefore PDG can generate non-projective dependency trees for input sentences. (C4) is a basic constraint for valency but is not adopted by the majority of dependency parsers.

(C2)-(C4) can be described as a set of co-occurrence constraints between two arcs in a dependency graph. As described below, PDG adopts co-occurrence constraints between two arbitrary arcs in a dependency graph using constraint matrix (CM). Constraints represented by CM are called arc co-occurrence constraints.

(C5) Arc co-occurrence constraint: Each arc pair in a tree has a co-occurrence relation in CM

More precise constraints compared with (C2) - (C4) are representable by means of CM. For example, it can allow non-projectivity for only some special arcs. In PDG, the mapping between a sequence of constituents (the body of a CFG rule) and a set of arcs (a partial dependency tree) is defined in an extended CFG rule. As described below, this grammar framework allows generating non-projective structures defined by grammar rules. This type of non-projectivity is called a controlled non-projectivity in this thesis. The controlled non-projectivity can reduce the generation of illegal non-projective dependency trees compared with the uncontrolled non-projectivity. Treatment of non-projectivity as described in (Kahane et al., 1998; Nivre and Nilsson, 2005) is an important topic out of the scope of this thesis.

The optimum tree search in a scored dependency graph is a task of searching for a dependency tree with the highest score satisfying the well-formed dependency tree constraint. The algorithm for this task is closely related to the types of dependency graphs and/or well-formedness constraints. Graph search algorithms, such as the Chu-Liu-Edmonds maximum spanning tree algorithm (Chu and Liu, 1965; Edmonds, 1967), algorithms based on the dynamic programming(DP) principle (Ozeki, 1994; Eisner, 1996b) and the algorithm based on the branch and bound (B&B) method (Hirakawa, 2001), are used for optimum tree search in scored dependency graphs. The Chu-Liu-Edmonds algorithm is very fast ($O(n^2)$ for sentence length n), but it works correctly only on word dependency graphs. Maximum spanning tree algorithms cannot satisfy the single role constraint for WPP and concept dependency graphs. DP-based algorithms can satisfy (C1) - (C3) and run efficiently, but seems not to satisfy (C4). Hirakawa (2001) proposed a B&B-based algorithm working on word dependency graphs satisfying (C1) - (C4). This thesis extends this algorithm to work on WPP and concept dependency graphs. The next section explains the problems of the DP-based method in treating (C4).

4.1.4 Single Valence Occupation Constraint and Dynamic Programming

Ozeki proposed an algorithm for obtaining the optimum kakari-uke tree and its score from a set of all possible scored kakari-uke relations (Ozeki, 1986; Ozeki, 1994). This algorithm can be extended to treat general dependency relations (Katoh and Ehara, 1989). This algorithm is generalized into the minimum cost partitioning method (MCPM), which is a partitioning computation based on the recurrence equation given below (Ozeki and Zhang, 1999). MCPM is also a generalization of the probabilistic CKY algorithm and the Viterbi algorithm[*6].

Considering the phrase $(w_i, \cdots w_j; a_i, \cdots, a_j; A)$ partitioned into $(w_i, \cdots, w_k; a_i, \cdots, a_k; B)$ and $(w_{k+1}, \cdots, w_j; a_{k+1}, \cdots, a_j : C)$ where w_x, a_x, and A-C mean word, analog information (such as prosodic information), and features like phrase name, respectively. MCPM computes the optimum solution based on the following recurrence equation for total cost F.

$$F(i,j,A) = min[F(i,k,B) + F(k+1,j,C) + cost(w_i, \cdots, w_j, a_i, \cdots, a_j, k, A, B, C)]$$

$F(i,j,A)$ is the total cost of phrase A covering from the i-th to the j-th word in a given sentence. $cost(w_i, ...w_j, a_i, ..., a_j, k, A, B, C)$ is a cost function where k is a partitioning position. The minimum cost partition of the whole sentence is calculated very efficiently by the DP principle for this equation. The optimum partitioning obtained by this method constitutes a tree covering the whole sentence satisfying the single role and projectivity constraints. However, it is not assured that the single valence occupation constraint adopted in PDG for basic semantic level constraint is satisfied by MCPM.

Fig.4.2 shows a dependency graph for the Japanese phrase "Isha-mo wakaranai byouki-no kanjya" encompassing dependency trees corresponding to "a patient suffering from a disease that the doctor doesn't know," "a sick patient who does not know the doctor," and so on. The dependency graph has two kinds of ambiguities, i.e., semantic role ambiguity and attachment ambiguity. For example, *wakaranai*(*not_know*) has four outgoing arcs with different semantic roles (*agent* and *target*) and different attachments (*byouki*(*sickness*) and *kanjya*(*patient*)) in Fig.4.2. The single valence occupation constraint prevents *wakaranai*(*not_know*) from being connected with the same two semantic role arcs. OS_1 - OS_4 represent the optimum solutions for the phrases specified by their brackets computed based on MCPM. For example, OS_3 gives an optimum tree with a score of 22 (consisting of *agent*1 and *target*4) for the phrase "Isha-

[*6] Specifically, MTCM corresponds to probabilistic CKY and the Viterbi algorithm because it computes both the optimum tree score and its structure.

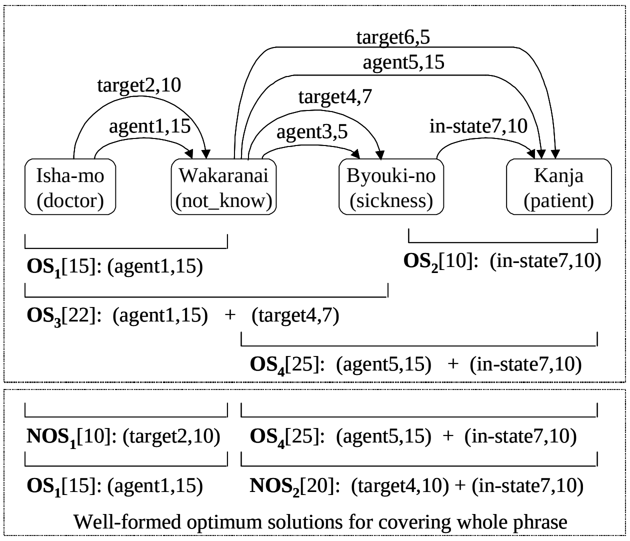


Fig.4.2 Optimum solution search satisfying the single valence occupation constraint

mo wakaranai byouki-no." The optimum solution for the whole phrase is either $OS_1 + OS_4$ or $OS_3 + OS_2$ due to MCPM. The former has the highest score $40(= 15 + 25)$ but does not satisfy the single valence occupation constraint because it has $agent1$ and $agent5$ simultaneously. The optimum solutions satisfying this constraint are $NOS_1 + OS_4$ and $OS_1 + NOS_2$ shown at the bottom of Fig.4.2. NOS_1 and NOS_2 are not optimum solutions for their word coverages. In this case, MCPM generates a non-optimum tree in $OS_3 + OS_2$ if it adopts the strategy of neglecting inconsistent trees. Otherwise, MCPM generates a high score but an ill-formed tree in $OS_1 + OS_4$. This shows that it is not assured that MCPM will obtain the optimum solution satisfying the single valence occupation constraint. On the contrary, it is assured that the graph branch algorithm will compute the optimum solution(s) satisfying any co-occurrence constraints in the constraint matrix including the single valence occupation constraint. It is an open problem whether an algorithm based on the DP framework exists which can handle the single valence occupation constraint and arbitrary arc co-occurrence constraints.

4.2 Semantic Dependency Graph and Dependency Forest

The semantic dependency graph, as shown in Section 4.2.1, is a semantic-label word dependency graph designed for Japanese sentence analysis (Hirakawa and Amano, 1989a). The optimum solution for a sentence is obtained by searching for the optimum tree in a semantic dependency graph with preference scores (Hirakawa, 2001).

The sentence analysis method based on the semantic dependency graph, the predecessor of PDG, is effective because it employs linguistic constraints as well as linguistic preferences. However, this method is lacking in terms of generality in that it cannot handle backward dependency and multiple WPP because it depends on some linguistic features peculiar to Japanese. PDG employs the dependency forest instead of the semantic dependency graph. Since the dependency forest has none of the language-dependent premises that the semantic dependency graph has, it is applicable to English and other languages. PDG has one more advantage in that it can generate non-projective dependency trees because the mapping from phrase structure to dependency structure is defined in grammar rules.

The optimum tree search algorithm for the semantic dependency graph is not applicable to the dependency forest. This thesis gives a brief explanation of the dependency forest and shows the graph branch algorithm for obtaining the optimum solution (tree) in the dependency forest.

4.2.1 Semantic Dependency Graph and its Drawbacks

Fig.4.3 shows a semantic dependency graph for "Watashi-mo Kare-ga Tukue-wo Katta Mise-ni Utta" (Hirakawa, 2001). The nodes in the graph correspond to the content words in the sentence and the arcs show possible semantic dependency relations between the nodes. Each arc has an arc ID and a preference score. Interpretations of a sentence are well-formed spanning trees that satisfy the projectivity constraint and the single valence occupation constraint. The bold arcs in the graph in Fig.4.3 show the optimum interpretation with a maximum score of 130.

The semantic dependency graph is designed based on the Japanese kakari-uke relation and assumes the following features of Japanese.

(a) A dependant always locates to the left of its governor (no backward dependency)
(b) POS ambiguities are quite minor compared with English[*7]

The semantic dependency graph and its optimum solution search algorithm adopt these as their premises. Therefore, this method is inherently inapplicable to languages like English that require

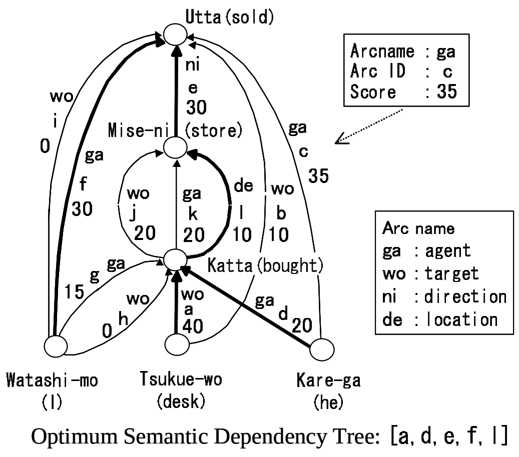
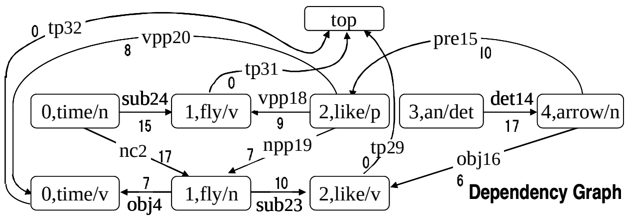


Fig.4.3 Example of semantic dependency graph and its optimum solution

[*7] Word boundary ambiguity corresponding to the compound word boundary ambiguity in English exists in Japanese. Treatment of this ambiguity is a practical problem for the semantic dependency graph even when applied to Japanese sentence analysis.



Meaning of Arc Name

sub	: subject
obj	: object
npp	: noun-preposition
vpp	: verb-preposition
pre	: preposition
nc	: noun compound
det	: determiner
tp	: top

Constraint Matrix

	2	24	4	23	19	18	20	14	16	15	31	29	32
2	–			○				○	○			○	
24		–				○		○		○	○		
4			–		○		○	○		○			○
23	○			–				○	○			○	
19			○		–			○		○			○
18		○				–		○		○	○		
20			○				–	○		○			○
14	○	○	○	○	○	○	○	–	○	○	○	○	○
16	○			○				○	–			○	
15		○	○		○	○	○	○		–	○		○
31		○				○		○		○	–		
29	○			○				○	○			–	
32			○		○		○	○		○			–

Fig.4.4 Scored DF for "Time flies like an arrow"

backward dependency and multiple POS analysis.

4.2.2 Dependency Forest

As described in Chapter 3, the dependency forest is a packed shared data structure encompassing all possible dependency trees for a sentence. A dependency forest consists of a dependency graph (DG) and a constraint matrix (CM). A dependency forest with a scored dependency graph is called a scored dependency forest. Fig.4.4 shows a scored dependency forest for the example sentence "Time flies like an arrow."

The dependency forest has correspondence with the phrase structure forest. This means that the dependency forest provides a means to treat all possible interpretations of a sentence in dependency structure representation. One sentence interpretation is represented by one well-formed dependency tree which satisfies the well-formed dependency tree constraint, i.e., the covering constraint and the arc co-occurrence constraint described in Section 4.1.3. The algorithm for the dependency forest has to treat the covering constraint.

Fig.4.5 shows four well-formed dependency trees for the dependency forest in Fig.4.4. Top nodes are omitted in the figure for simplicity. Tree (a) is the optimum dependency tree with the highest score of 51.

4.2.3 Relation between Semantic Dependency Graph and Dependency Forest

The dependency forest and the semantic dependency graph utilize the WPP dependency graph and the word graph, respectively. The word dependency graph can be seen as a special case of the WPP dependency graph. Therefore, the semantic dependency graph is a subset of the dependency graph of the dependency forest. On the other hand, well-formedness constraints introduced

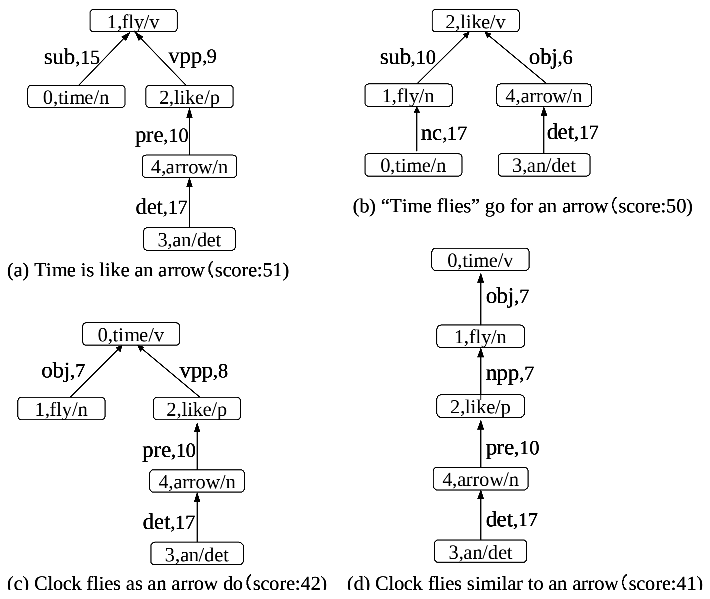


Fig.4.5 Well-formed dependency trees for the example sentence

to a semantic dependency graph, i.e., the projectivity and single valence occupation constraints, are a type of arc co-occurrence constraints representable by means of CM. Therefore, the dependency forest is a generalized and more powerful data structure covering the representative power of the semantic dependency graph.

4.3 Optimum Tree Search for Dependency Forest Based on Graph Branch Method

The graph branch method works on the branch and bound principle and searches for the optimum well-formed tree from a dependency graph by applying partial sub-problem expansions called graph branching. The algorithm in (Hirakawa, 2001) applies the graph branch method to the semantic dependency graph. Unfortunately, this algorithm is not directly applicable to the dependency forest search problem. The following shows a new algorithm for applying the graph branch method to the dependency forest.

4.3.1 Outline of Branch and Bound Method

The branch and bound method is a principle for solving computationally hard problems such as NP-complete problems. The basic strategy is that the original problem is decomposed into easier partial-problems (branching) and the original problem is solved by solving them. Pruning called a bound operation is applied if it turns out that the optimum solution to a partial-problem is inferior to the solution obtained from some other partial-problem (dominance test), or if it turns out that a partial-problem gives no optimum solutions to the original problem (maximum value test). The dominance test is not used in the graph branch method. Usually, the branch and bound algorithm is constructed to minimize the value of the solution. The graph branch algorithm in this thesis is constructed to maximize the score of the solution because the best solution is the maximum tree in the dependency forest.

The following features for the maximum bound value test with respect to the problem P and its partial-problem P_c must be satisfied in the branch and bound method.

(MC1) $g(P_c) \geq f(P)$ where $g(P_c)$ is the maximum value of P_c, and $f(P)$ is the maximum value of P.

(MC2) If $g(P_c) = l(P)$ where l gives a value of a feasible solution to P, then the feasible solution is a solution to P.

(MC3) If P_c has no feasible solutions then P has no solutions.

(MC4) If a feasible solution with an incumbent value z is obtained for some partial-problem, and if $g(P_c) \leq z$, then partial-problems branched from problem P have no better solutions than z.

These conditions are called model conditions in this thesis. In the case of MC2-MC4., partial-problem P_c can be terminated. Fig.4.6 shows a general branch and bound algorithm for obtaining one optimum solution (Ibaraki, 1978).

```
A : Set of active partial problems (not yet terminated nor expanded)
N : Set of generated partial problems
O : Set of optimum solutions
z : Incumbent value
l : l(P) gives value of feasible solution of a partial problem P
g : g(P) gives upper-bound value of a partial problem P
s : s(A) selects one partial-problem in A
G : Set of partial problems with no feasible solution or g(P)=f(P)
f : f(P) is the optimum solution of P
D : If $P_i$ D $P_j$, $P_i$ dominates $P_j$
S1(initial value setup): A:={$P_0$}, N:= {$P_0$}, z=-∞, O:={}
S2(search) : If A={} goto S9 else Pi:=s(A). Goto S3.
S3(incumbent value update) : If l(Pi)>z then z:=l(Pi), O:={x} (x is a feasible solution of Pi satisfying
        f(x)≧l(x)). Goto S4.
S4(G test) : If Pi∈G goto S8 else goto S6.
S5(upper bound test) : If g(Pi)≦z goto S8 else goto S6.
S6(dominance test) : If there exists Pk (≠Pi)∈N satisfying Pk D Pi goto S8 else goto S7.
S7(branching operation) : Generate child partial problem $Pi_1$, $Pi_2$,..$Pi_k$ of Pi. Set A:=A∪{$Pi_1$,$Pi_2$,..$Pi_k$}−
        {Pi}, N:=N ∪{$Pi_1$,$Pi_2$,..,$Pi_k$}. Goto S2.
S8(termination of Pi) : Set A:=A−{Pi}. Goto S2.
S9(stop) : Computation stop. If z=-∞ then $P_0$ has no feasible solutions else z is the optimum value f($P_0$)
        and x in O is the optimum solution to $P_0$.
```

Fig.4.6 Skeleton of branch and bound algorithm

4.3.2 Graph Branch Algorithm

Fig.4.6 shows a skeleton of the algorithm. In order to make it running code, each operation in the algorithm must be realized for the target problem. The graph branch algorithm applies the branch and bound method to the optimum tree search problem with the binary arc co-occurrence constraint by introducing the graph branch operation for the partial-problem expansion operation. Fig.4.7 shows the graph branch algorithm which has been extended from the original skeleton to search for all optimum trees for a dependency graph. The following sections explain how the components of the branch and bound method in Fig.4.6 are implemented in the graph branch algorithm.

(1) Partial-problem

Partial-problem P_i in the graph branch method is a problem searching for all the well-formed optimum trees in a dependency forest DF_i consisting of the dependency graph DG_i and constraint matrix CM_i. Partial-problem P_i consists of the following elements.

(a) Dependency graph DG_i

```
$P_0$ : Initial problem,   $P_i$ : Partial problem,   AP : Active partial problem list,
O  : Set of incumbent solutions,   z : Incumbent value
start: /* S1(initial value setup) */
    AP := {$P_0$};  z = -1;  O := {};  UB = get_ub($P_0$);   /* Upper bound of $P_0$ */
search_top: /* S2(search) */
    if(AP == {}) { goto exit; } else{ $P_i$ := select_problem(AP); }
    (FS,LB) := get_fs($P_i$); /* Compute the feasible solution and the lower bound for $P_i$ */
    if(FS == no_solution) { goto terminate_problem; }
    /* S3(incumbent value update): */
    if(LB > z) { z := LB;  O := {FS}; }  /*  If LB is better than z, update O and z  */
    /* S5(upper bound test): */
    if(UB < z) { goto terminate_problem; }
    IAPL := get_iapl($P_i$);   /* Compute inconsistent arc pair list IAPL. */
    if(LB < UB) { BACL := IAPL; goto branch; }  /* If LB < UB, execute graph branch */
    /* Lower bound equals to upper bound => optimum solution */
    elsif(LB == UB) {
        O := {FS} ∪ O; /* Add this FS as incumbent solution */
        /* S8(search for more optimum solutions) */
        if(IAPL != {}) { BACL := IAPL; goto branch; }  /* (a) existence of IAPL */
        BACL := arcs_with_alternatives(FS);            /* (b) existence of a rival arc */
        if(BACL != {}) { goto branch; } else { goto terminate_problem; } }
branch: /* S6(branching operation) */
    ChildProblemList := graph_branch($P_i$,BACL); /* Generate child problems */
    AP := AP ∪ ChildProblemList - {$P_i$}; goto search_top;
terminate_problem:   /* S7(termination of $P_i$) */
    AP := AP - {$P_i$}; goto search_top;
exit: /* S9(stop) */
    if(z == -1) { Problem $P_0$ has no solution } else { O is a set of the optimum solutions }
```

Fig.4.7 Graph Branch Algorithm

(b) Constraint matrix CM_i
(c) Feasible solution value LB_i (corresponding to $l(P)$ in Fig.4.6)
(d) Upper bound value UB_i (corresponding to $g(P)$ in Fig.4.6)
(e) Inconsistent arc pair list $IAPL_i$.

The constraint matrix is common to all partial-problems, so one CM is shared by all partial-problems. DG_i is represented by "rem[..]" which shows a set of arcs to be removed from the whole dependency graph DG_i, i.e., DG_i is obtained by removing rem[..] from DG_i. For example, "$rem[b,d]$" represents a partial dependency graph $[a,c,e]$ in the case $DG = [a,b,c,d,e]$. This reduces the memory space and the computation for a feasible solution as described below. $IAPL_i$ is a list of inconsistent arc pairs. An inconsistent arc pair is an arc pair which does not satisfy some co-occurrence constraint.

(2) Algorithm for Obtaining Feasible Solution and Lower Bound Value

In the graph branch method, a well-formed dependency tree in the dependency graph DG of the partial-problem P is assigned as the feasible solution FS (corresponding to x in Fig.4.6) of P [*8]. The score of the feasible solution FS is assigned as the lower bound value LB (corresponding to

G	:	Dependency graph of a partial problem,
n	:	Number of words in an input sentence
FS	:	Area for saving arc IDs of a feasible solution
BP	:	Area for saving the nearest backtrack points
S_i $(1 \leqq i \leqq n)$	:	Set of arcs having the dependant nodes with the same position
N(S)	:	Number of elements in arc set S
a(i,j)	:	j-th arc in arc set S_i
score(FS)	:	Sum total of scores of arcs in FS

step1(grouping and sorting arcs): Classify the arcs in graph G by their starting nodes, and generate the sets of arcs $S_1,S_2,\ldots,S_n$. Sort elements in each S_i with respect to their weights in descending order. Then, sort $S_1,S_2,\ldots,S_n$ with the maximum score of the arcs in the set in descending order. This is renamed $S_1,S_2,\ldots,S_n$。

step2(initialize): FS:=[],BP:=[],i:=1,j:=1,k:=1,l:=0

step3(termination check1): If i>n then terminate by returning the feasible solution FS and score(FS). If i≦n then goto step4.

step4(termination check2): If $N(S_i) \geqq j$ then goto step5 else set FS:=no_solution and terminate. (No feasible solution)

step5(constraint check): If $j>N(S_i)$ (no arcs in S_i satisfies the co-occurrence constraint), goto step6. Perform the co-occurrence constraint check between j-th element a(i,j) of S_i and each element $e_1,e_2,\ldots,e_{i-1}$ in FS in reverse order. If a(i,j) does not satisfy the co-occurrence constraint with element e_k $(1 \leqq k \leqq i-1)$, set l:=max(l,k), j:=j+1, goto step5. If all co-occurrence constraint checks are satisfied then goto step7.

step6(backtracking): Remove $e_l,e_{l+1},\ldots,e_{i-1}$ from S. Set j:= BP[l]+1, i:=l. Goto step4.

step7(next node): Add a(i,j) to the last of FS. Set BP[i]:=j, i:=i+1, j:=1. Goto step3.

Fig.4.8 Algorithm for obtaining FS and LB

[*8] A feasible solution may not be optimum but is a possible interpretation of a sentence. Therefore, it can be used as an approximate output when the search process is aborted.

$l(P)$ in Fig.4.6). The function for computing these values get_fs is called a feasible solution/lower bound value function. Fig.4.8 shows the algorithm of get_fs. Basically, get_fs searches for one feasible solution in higher-score-first and depth-first manner. When an arc which violate co-occurrence constraint against one of the selected arcs is found, get_fs backtracks at *step*5 to the nearest choice point which resolves the contradiction. This assures that the obtained solution satisfies the co-occurrence constraint. Furthermore, if get_fs finds no solution, then the problem P has no solution. Since get_fs selects one arc for each position in a sentence, the obtained arcs satisfies the well-covered constraint.

Arc groups S_1 to S_n are sorted according to their scores in *step*1. This operation is introduced to obtain a better (higher score) feasible solution, since the better feasible solution lead to a higher incumbent value which bounds more partial-problems.

(3) Algorithm for Obtaining Upper Bound Value

Given a set of arcs A which is a subset of a dependency graph DG, if the set of dependent nodes of arcs in A satisfies the covering constraint described above, the arc set A is called the well-covered arc set. The “maximum well-covered arc set” is defined as a well-covered arc set with the highest score. In general, the maximum well-covered arc set does not satisfy the single role constraint and does not form a tree. In the graph branch method, the score of the maximum well-covered arc set of a dependency graph G is assigned as the upper bound value UB (corresponding to $g(P)$ in Fig.4.6) of the partial-problem P. Upper bound function get_ub calculates UB by scanning the arc lists sorted by the surface position of the dependent nodes of the arcs.

The above settings satisfy the model conditions. In these settings, P and get_ub corresponds to P_c and $g(P_c)$, respectively. (MC1) is satisfied because $get\_ub(P) \geq f(P)$ is true for $f(P)$ (the score of the optimum tree). (MC2) and (MC4) are satisfied because get_ub is the score of the maximum well-covered arc set. (MC3) is satisfied since $get\_ub(P)$ always has its solution. Therefore, partial-problem P is prunable if the incumbent value z satisfies $z \geq g(P)$[*9].

(4) Branch Operation

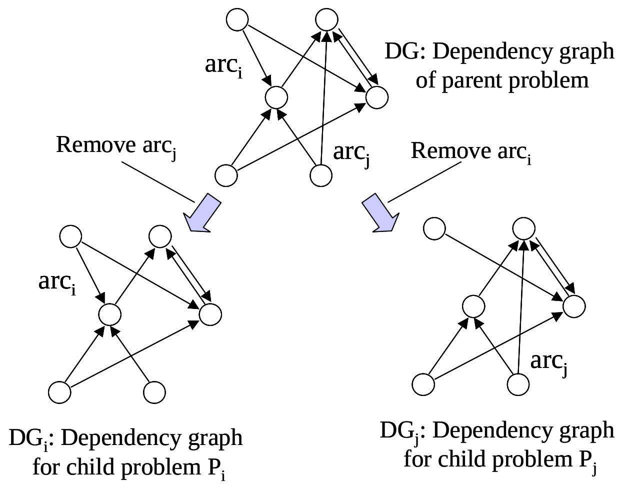
Fig.4.9 shows a branch operation in the graph branch method called a graph branch operation. Child partial-problems of P are constructed as follows:

(a) Search for an inconsistent arc pair (arc_i, arc_j) in the maximum well-covered arc set for the dependency graph of P.

(b) Create child partial-problems P_i, P_j which have new dependency graphs $DG_i = DG - \{arc_j\}$ and $DG_j = DG - \{arc_i\}$, respectively.

Since a solution to P cannot have both arc_i and arc_j simultaneously due to the co-occurrence constraint, the optimum solution of P is obtained from either/both P_i or/and P_j. The child partial-problem is easier than the parent partial-problem because the size of the dependency

[*9] In the case of obtaining all optimum solutions ,the terminate condition should be changed to $z > g(P)$.



arc$_i$
DG: Dependency graph of parent problem
Remove arc$_j$
arc$_j$
Remove arc$_i$
arc$_i$
arc$_j$
DG$_i$: Dependency graph for child problem P$_i$
DG$_j$: Dependency graph for child problem P$_j$

Fig.4.9 Graph Branching

graph of the child partial-problem is less than that of its parent.

In Fig.4.7, *get_iapl* computes the list of inconsistent arc pairs $IAPL$(Inconsistent Arc Pair List) for the maximum well-covered arc set of P_i. Then the graph branch function *graph_branch* selects one inconsistent arc pair (arc_i, arc_j) from $IAPL$ for branch operation. The selection criteria for (arc_i, arc_j) affects the efficiency of the algorithm. *graph_branch* selects the inconsistent arc pair containing the highest score arc in $BACL$(Branch Arc Candidates List). *graph_branch* calculates the upper bound value for a child partial-problem by *get_ub* and sets it to the child partial-problem. Simultaneously, *graph_branch* executes bound operation by immediately pruning the child partial-problem whose upper bound value is less than the incumbent value z.

(5) Selection of Partial-problem from Active Partial-problems

select_problem in Fig.4.8 corresponds to the search $s(A)$ in Fig.4.6. The best bound search is employed for *select_problem*, i.e., it selects the partial-problem which has the maximum bound value among the active partial-problems. It is known that the number of partial-problems decomposed during computation is minimized by this strategy in the case that no dominance tests are applied (Ibaraki, 1978).

(6) Computing All Optimum Solutions

In order to obtain all optimum solutions, partial-problems whose upper bound values are equal to the score of the optimum solution(s) are expanded at $S8(SearchMoreOptimumSolutions)$. In the case that at least one inconsistent arc pair remains in a partial-problem (i.e., $IAPL \neq \{\}$), graph branch is performed based on the inconsistent arc pair. Otherwise, the obtained optimum solution FS is checked if one of the arcs in FS has an equal rival arc by *arcs_with_alternatives* function in Figure 4.6. The equal rival arc of arc A is an arc whose position and score are equal to those of arc A. If an equal rival arc of an arc in FS exists, a new partial-problem is generated by removing the arc in FS. $S8$ assures that no partial-problem has an upper bound value greater than or equal to the score of the optimum solutions when the computation stopped.

(7) Correctness of the Graph Branch Algorithm

All Dependency trees are generated by the feasible solution and lower bound value function *get_fs*. *get_fs* does not violate the covering constraint(the single role constraint and the coverage constraint) because it selects one arc for one input position at the *step*7 in Fig.4.8. It also assures the co-occurrence constraint by checking the CM value for every two arcs in a tree at *step*5. Therefore, output dependency trees of the graph branch algorithm satisfy the well-formed dependency tree constraint.

4.4 Example of Optimum Tree Search

This section presents an example showing the behavior of the graph branch algorithm using the dependency forest in Fig.4.4.

4.4.1 Feasible Solution/Lower Bound Value Function

The following section shows the behavior of feasible solution/lower bound value function *get_fs* for the example sentence.

*step*1(*grouping and sorting of arcs*) in Fig.4.8 is performed once at the beginning for the initial dependency forest. The result of *step*1 is shown in Fig.4.10. *Pos* and *MaxScore* mean the position of the arc in the sentence and the maximum arc score at that position respectively. Arcs with no rival arc have *MaxScore* ∞ and are located at the top of the arc group list. Arc groups with start positions 3,0,4,1 and 2 are assigned to S_1,S_2,S_3,S_4 and S_5, respectively.

```
S1 : Pos = 3 , MaxScore = ∞ :
   a(1,1) arc(det-14:17,[an]-det-3,[arrow]-n-4)
S2 : Pos = 0 , MaxScore = 17 :
   a(2,1) arc(nc-2:17,[time]-n-0,[flies]-n-1)
   a(2,2) arc(sub-24:15,[time]-n-0,[flies]-v-1)
   a(2,3) arc(tp-32:0,[time]-v-0,[top]-x-top)
S3 : Pos = 4 , MaxScore = 10 :
   a(3,1) arc(pre-15:10,[arrow]-n-4,[like]-pre-2)
   a(3,2) arc(obj-16:6,[arrow]-n-4,[like]-v-2)
S4 : Pos = 1 , MaxScore = 10 :
   a(4,1) arc(sub-23:10,[flies]-n-1,[like]-v-2)
   a(4,2) arc(obj-4:7,[flies]-n-1)/1:5,[time]-v-0)
   a(4,3) arc(tp-31:0,[flies]-v-1,[top]-x-top)
S5 : Pos = 2 , MaxScore = 9 :
   a(5,1) arc(vpp-18:9,[like]-pre-2,[flies]-v-1)
   a(5,2) arc(vpp-20:8,[like]-pre-2,[time]-v-0)
   a(5,3) arc(npp-19:7,[like]-pre-2,[flies]-n-1)
   a(5,4) arc(tp-29:0,[like]-v-2,[top]-x-top)
```

Fig.4.10 Grouped and Sorted Arcs

$step2(initialize)$ initializes variables. After $step3$ and $step4$ are executed, $step5$ checks that $a(i,j) = a(1,1) = 14(= det14)$ can be registered to FS. In this case, no violation of the co-occurrence constraint occurs, and then $step7$ registers $a(1,1)$ to FS [*10], then backtrack point $BP[1]$ at the position $i(= 1)$ is set to $j(= 1)$.

$FS = [a(1,1)](= [14])$, $BP = [1, -, -, -, -]$, $i = 2$, $j = 1$, $k = 1$, $l = 0$

Next, $step3$-5 try the first arc $a(2,1)(= nc2)$ in S_2. Since $CM(a(1,1), a(2,1)) = CM(14,2) = \bigcirc$ in Fig.4.4, $a(2,1)$ and $a(1,1)$ satisfy the co-occurrence constraint and then $a(2,1)$ is registered to FS.

$FS = [a(1,1), a(2,1)](= [14, 2])$, $BP = [1, 1, -, -, -]$, $i = 3$, $j = 1$, $k = 1$, $l = 0$

$a(3,1)(= pre15)$ is skipped because $CM(a(3,1), a(2,1)) = CM(15,2) \neq \bigcirc$. Then $a(3,2)$ is tried.

$FS = [a(1,1), a(2,1), a(3,2)](= [14, 2, 16])$
$BP = [1, 1, 2, -, -]$, $i = 4$, $j = 1$, $k = 1$, $l = 1$

In a similar manner, $a(4,1)(= sub23)$ and $a(5,4)(= rt29)$ are added to FS, then the termination condition at $step3$ is satisfied.

$FS = [a(1,1), a(2,1), a(3,2), a(4,1), a(5,4)](= [14, 2, 16, 23, 29])$,
$BP = [1, 1, 2, 1, 4]$, $i = 6$, $j = 1$, $k = 4$, $l = 4$,

The FS here is a feasible solution and the sum total of arc scores, i.e., $17 + 17 + 6 + 10 + 0 = 50$ is the score of the feasible solution.

No backtracking occurred in this example. Backtracking occurs when all arcs in S_i are found to be inconsistent with either of the arcs in FS at that point. In this case, $step6(backtracking)$ backtracks to the l position. l is assured to be the rightmost position, where some element in S_i is inconsistent with the selected arc in FS. This mechanism is introduced to optimize backtracking.

4.4.2 Example of Graph Branch Algorithm

The search process of the branch and bound method can be shown as a search diagram constructing a partial-problem tree representing the parent-child relation between the partial-problems. Fig.4.11 is a search diagram for the example dependency forest showing the search process of the graph branch method.

In this figure, box P_i is a partial-problem with its dependency graph rem, upper bound value UB, feasible solution and lower bound value LB and inconsistent arc pair list $IAPL$. Suffix i of P_i indicates the generation order of partial-problems. Updating of global variable z (incumbent value) and O (set of incumbent solutions) is shown under the box. The value of the left-hand side of the arrow is updated to that of right-hand side of the arrow during the partial-problem processing. Details of the behavior of the algorithm in Fig.4.7 are described below.

In $S1(initialize)$, z, O and AP are set to -1, $\{\}$ and $\{P_0\}$, respectively. The dependency graph

[*10] In fact, the arc ID 14 is registered to FS. The $a(i,j)$ form is used here for clarity.

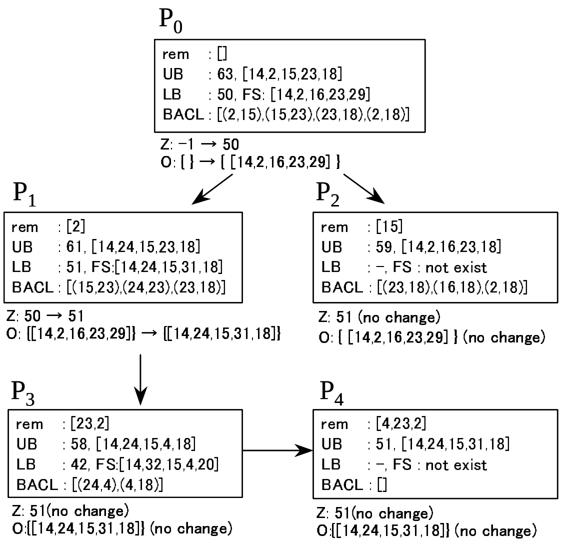

P0
rem : []
UB : 63, [14,2,15,23,18]
LB : 50, FS: [14,2,16,23,29]
BACL : [(2,15),(15,23),(23,18),(2,18)]
Z: −1 → 50
O: { } → { [14,2,16,23,29] }
P1
rem : [2]
UB : 61, [14,24,15,23,18]
LB : 51, FS:[14,24,15,31,18]
BACL : [(15,23),(24,23),(23,18)]
Z: 50 → 51
O: {[14,2,16,23,29]} → {[14,24,15,31,18]}
P2
rem : [15]
UB : 59, [14,2,16,23,18]
LB : −, FS : not exist
BACL : [(23,18),(16,18),(2,18)]
Z: 51 (no change)
O: { [14,2,16,23,29] } (no change)
P3
rem : [23,2]
UB : 58, [14,24,15,4,18]
LB : 42, FS:[14,32,15,4,20]
BACL : [(24,4),(4,18)]
Z: 51(no change)
O:{[14,24,15,31,18]} (no change)
P4
rem : [4,23,2]
UB : 51, [14,24,15,31,18]
LB : −, FS : not exist
BACL : []
Z: 51(no change)
O:{[14,24,15,31,18]} (no change)

Fig.4.11 Search diagram for the example sentence

of P_0 is that of the example dependency forest. This is represented by $rem = []$. *get_ub* sets the upper bound value (=63) of P_0 to UB. In practice, this is calculated by obtaining the maximum well-covered arc set of P_0. In $S2(search)$, *select_problem* selects P_0 and $get\_fs(P_0)$ is executed. The feasible solution FS and its score LB are calculated based on the algorithm in Fig.4.8 to set $FS = [14, 2, 16, 23, 29]$, $LB = 50$ (P_0 in the search diagram). $S3(incumbent\ value\ update)$ updates z and O to new values. Then, $get\_iapl(P_0)$ computes the inconsistent arc pair list $[(2, 15), (15, 23), (23, 18), (2, 18)]$ from the maximum well-covered arc set $[14, 2, 15, 23, 18]$ and set it to $IAPL$. $S5(maximum\ value\ test)$ compares the upper bound value UB and the feasible solution value LB. In this case, $LB < UB$ holds, so $BACL$ is assigned the value of $IAPL$. The next step $S6(branch\ operation)$ executes the *graph_branch* function. *graph_branch* selects the arc pair with the highest arc score and performs the graph branch operation with the selected arc pair. The following is a $BACL$ shown with the arc names and arc scores.

$$[(nc2[17], pre15[10]), (pre15[10], sub23[10]), (sub23[10], vpp18[9]), (nc2[17], vpp18[9])]$$

Scores are shown in [ ]. The arc pair containing the highest arc score is $(2, 15)$ and $(2, 18)$ containing $nc2[17]$. Here, $(2, 15)$ is selected and partial-problems $P_1(rem[2])$ and $P_2(rem[15])$ are generated. P_0 is removed from AP and the new two partial-problems are added to AP resulting in $AP = \{P_1, P_2\}$. Then, based on the best bound search strategy, $S2(search)$ is tried again. *select_problem* selects P_1 because the upper bound value of P_1 (=61) is greater than that of P_2 (=59). Since the upper bound of P_1 (=61) is greater than the feasible solution score (=51), *get_iapl* is executed and sets $BACL$ to the value shown in P_1 in Fig.4.11. The graph branch function *graph_branch* gets two candidates for child partial-problems corresponding to $rem[24, 2]$ and $rem[23, 2]$ because the inconsistent arc pair $(24, 23)$ is selected as the source of the graph branch operation (arc 24 has the highest score of 15). The former candidate for

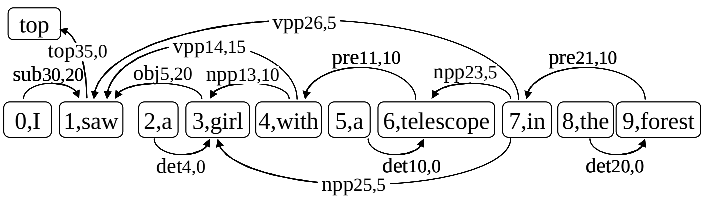
$rem[24,2]$ is pruned immediately, because its upper bound value (=46) is smaller than the incumbent value (=51) (termination by the upper bound test). Therefore, *graph_branch* returns $\{P_3(rem[23,2])\}$. The upper bound value UB of P_3 is 58 which is less than that of its parent problem P_1. The processing for P_1 is completed and P_1 is removed from AP. *select_problem* selects P_2 by comparing the upper bound values of P_2 and P_3 in AP. Partial-problem P_2 is terminated because it has no feasible solution (FS = *no_solution*). Then, the next partial-problem P_3 is processed. P_3 has a feasible solution with a score of 41. Updating of the incumbent value does not occur because the obtained score is lower than the existing incumbent value. The next partial-problem P_4 has no feasible solution, so all processing is terminated at $S8$(*stop*). At this time, the values of O and z are the optimum solution(=$\{[14,24,15,31,18]\}$) and its score (=51) respectively. This solution corresponds to the dependency tree (a) in Fig.4.5.

4.4.3 Prototypical Ambiguous Sentences

In addition to the previous example for homophone ambiguities, this section shows two examples of prototypical ambiguous sentences.

(1) PP-attachment Ambiguity

Fig.4.12 shows a dependency forest for "I saw a girl with a telescope in the forest." There are no homophones in the forest but two prepositional phrases with attachment ambiguities. The preposition "with" has two possible dependencies (*npp*14,*vpp*16) and "in" has three (*vpp*27,*npp*26,*npp*29). The combination number of these arcs is $2 * 3 = 6$, but there exists five well-formed dependency trees due to the existence of the co-occurrence constraint between arcs 16 and 29 ($CM(16,29) \neq \bigcirc$) corresponding to the projectivity constraint. The scores of these arcs are assumed to be calculated based on the preposition, the governor and dependant nodes of the preposition. *vpp*16 has a higher score compared with *npp*14 because "telescope" is a tool for seeing something. On the other hand, *vpp*27,*npp*26 and *npp*29 have the same scores. The search



0,I	: [i]-n-0
1,saw	: [saw]-v-1
2,a	: [a]-det-2
3,girl	: [girl]-n-3
4,with	: [with]-pre-4
5,a	: [a]-det-5
6,telescope	: [telescope]-n-6
7,in	: [in]-pre-7
8,the	: [the]-det-8
9,forest	: [forest]-n-9
top	: [top]-x-top

	30	4	5	13	14	10	11	25	23	26	20	21	35
30	–	○	○	○	○	○	○	○	○	○	○	○	○
4	○	–	○	○	○	○	○	○	○	○	○	○	○
5	○	○	–	○	○	○	○	○	○	○	○	○	○
13	○	○	○	–		○	○	○	○	○	○	○	○
14	○	○	○		–	○	○		○	○	○	○	○
10	○	○	○	○	○	–	○	○	○	○	○	○	○
11	○	○	○	○	○	○	–	○	○	○	○	○	○
25	○	○	○	○		○	○	–			○	○	○
23	○	○	○	○	○	○	○		–		○	○	○
26	○	○	○	○	○	○	○			–	○	○	○
20	○	○	○	○	○	○	○	○	○	○	–	○	○
21	○	○	○	○	○	○	○	○	○	○	○	–	○
35	○	○	○	○	○	○	○	○	○	○	○	○	–

Fig.4.12 DF for the example sentence including PP attachments

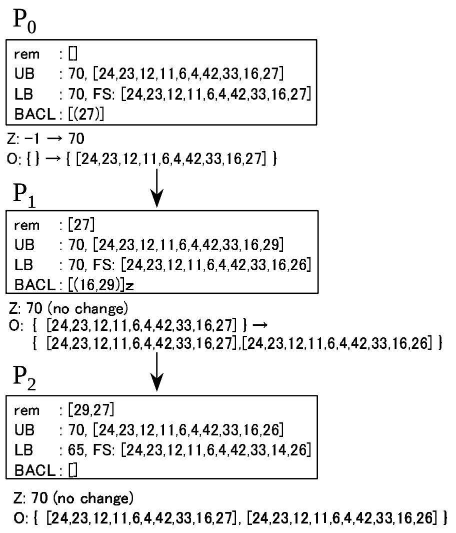
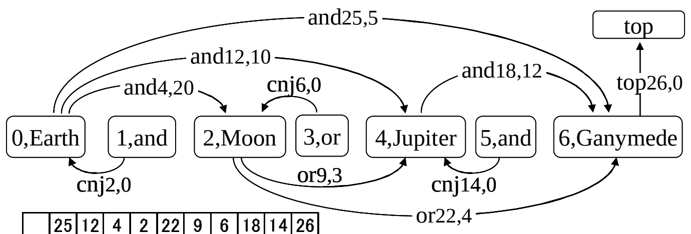

P0
rem : []
UB : 70, [24,23,12,11,6,4,42,33,16,27]
LB : 70, FS: [24,23,12,11,6,4,42,33,16,27]
BACL : [(27)]
Z: -1 → 70
O: { } → { [24,23,12,11,6,4,42,33,16,27] }
P1
rem : [27]
UB : 70, [24,23,12,11,6,4,42,33,16,29]
LB : 70, FS: [24,23,12,11,6,4,42,33,16,26]
BACL : [(16,29)]z
Z: 70 (no change)
O: { [24,23,12,11,6,4,42,33,16,27] } →
{ [24,23,12,11,6,4,42,33,16,27],[24,23,12,11,6,4,42,33,16,26] }
P2
rem : [29,27]
UB : 70, [24,23,12,11,6,4,42,33,16,26]
LB : 65, FS: [24,23,12,11,6,4,42,33,14,26]
BACL : []
Z: 70 (no change)
O: { [24,23,12,11,6,4,42,33,16,27], [24,23,12,11,6,4,42,33,16,26] }

Fig.4.13 Search diagram for the example sentence including PP attachments

diagram for this example is shown in Fig.4.13. P_0 generates the optimum solution ($UB = LB$) with a score of 70. $S8$(*search more optimum solution*) in Fig.4.7 is executed. P_0 has no graph branch candidates in the inconsistent arc pair list ($IAPL == \{\}$). $arcs\_with\_alternatives(FS)$ selects arc $vpp27$ as a candidate of graph branching because it has rival arcs with the same score ($npp26$,$npp29$). Then P_1 is generated to obtain the second optimum solution including $npp26$. Next P_2 with $rem[26, 27]$ is generated and a feasible solution to P_2 is calculated. This solution is not added to the incumbent solution list because it has a lower score (65) than the obtained optimum solutions. This example has two optimum solutions.

(2) Coordination Scope Ambiguity

Fig.4.14 shows a dependency forest for "Earth and Moon or Jupitor and Gamymede." Corresponding to the combination of the scopes of the three coordinations, "Earth" and "Moon" have three and two outgoing arcs, respectively. Since there exists a co-occurrence constraint



	25	12	4	2	22	9	6	18	14	26
25	—			○	○	○	○	○	○	○
12		—		○		○	○	○	○	○
4			—	○	○	○	○	○	○	○
2	○	○	○	—	○	○	○	○	○	○
22	○		○	○	—		○	○	○	○
9	○	○	○	○		—	○	○	○	○
6	○	○	○	○	○	○	—	○	○	○
18	○	○	○	○	○	○	○	—	○	○
14	○	○	○	○	○	○	○	○	—	○
26	○	○	○	○	○	○	○	○	○	—

0,Earth	: [Earth]-n-0
1,and	: [and]-and-1
2,Moon	: [Moon]-n-2
3,or	: [or]-or-3
4,Jupiter	: [Jupiter]-n-4
5,and	: [and]-and-5
6,Ganymede	: [Ganymede]-n-6
top	: [top]-x-top

Fig.4.14 DF for the example sentence including coordinates

(projectivity constraint) between *and*12 and *or*22, the dependency forest has five well-formed dependency trees. Arc scored are assigned assuming preference knowledge like "Planet names tend to co-occur" and "The name of a planet and its secondary planet tend to co-occur."

The search diagram for this example is shown in Fig.4.15. The feasible solution to the initial problem P_0 happens to be the optimum solution. No branch operation is performed because $IAPL$ of P_0 is [] and all arcs in the optimum solution have no rival arcs.

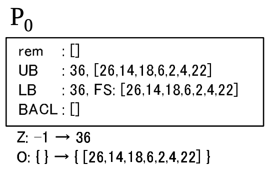
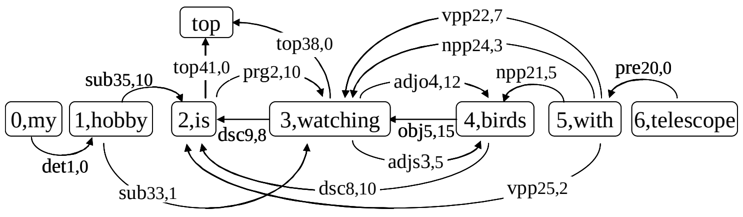

P0
rem : []
UB : 36, [26,14,18,6,2,4,22]
LB : 36, FS: [26,14,18,6,2,4,22]
BACL : []
Z: −1 → 36
O: { } → { [26,14,18,6,2,4,22] }

Fig.4.15 Search diagram for the example sentence including coordinates

(3) Structural Ambiguity

Fig.4.16 shows a dependency forest for "My hobby is watching birds with a telescope." This example has no homophone ambiguities but has ambiguities in the structural interpretation of the word "be" (copula vs. progressive form) and "watching birds"(*sub*5,*obj*6,*adj*4) as well as PP-attachment ambiguities(*vpp*24,*npp*27,*npp*23). This dependency forest encompasses eight well-formed dependency trees. Fig.4.17 is a search diagram for this example. P_0 generates a feasible solution $[22, 1, 6, 33, 44, 38, 24]$ corresponding to "My hobby = watching birds using a telescope." Since the score of this feasible solution (40) is lower than the upper bound value (54), P_0 is branched to P_1 and P_2. P_1 generates a feasible solution $[22, 1, 6, 33, 44, 38, 23]$ but the incumbent value and the optimum solution list are not updated because the feasible solution score(38) is lower than the current incumbent value(40). The succeeding computation process generates no better solutions and terminates by guaranteeing that the solution with a score of



0,my	:	[my]-det-0
1,hobby	:	[hobby]-n-1
2,is	:	[is]-be-2
3,watching	:	[watching]-ving-3
4,birds	:	[birds]-n-4
5,with	:	[with]-pre-5
6,telescope	:	[telescope]-n-6
top	:	[top]-x-top

	1	35	33	2	4	3	9	8	5	21	24	26	22	20	41	38
1	–	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
35	○	–			○	○	○	○	○	○	○	○	○	○	○	
33	○		–	○					○	○			○	○		○
2	○		○	–					○	○			○	○		○
4	○	○			–			○		○		○		○	○	
3	○	○				–		○		○		○		○	○	
9	○	○					–		○	○	○	○	○	○	○	
8	○	○			○	○		–		○		○		○	○	
5	○	○	○	○			○		–	○	○	○	○	○	○	○
21	○	○	○	○	○	○	○	○	○	–				○	○	○
24	○	○					○		○		–			○	○	
26	○	○			○	○	○	○	○			–		○	○	
22	○	○	○	○			○		○				–	○	○	○
20	○	○	○	○	○	○	○	○	○	○	○	○	○	–	○	○
41	○	○			○	○	○	○	○	○	○	○	○	○	–	
38	○		○	○					○	○			○	○		–

Fig.4.16 DF for the example sentence including structural ambiguities

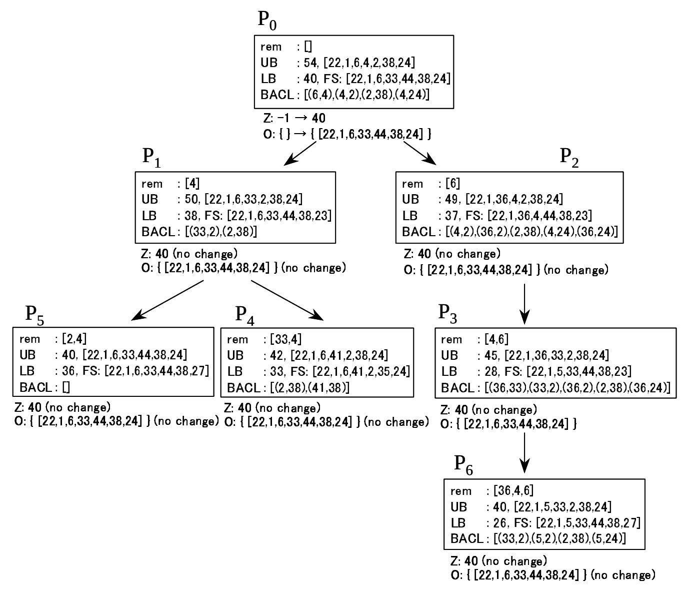

P0
rem : []
UB : 54, [22,1,6,4,2,38,24]
LB : 40, FS: [22,1,6,33,44,38,24]
BACL : [(6,4),(4,2),(2,38),(4,24)]
Z: -1 → 40
O: { } → { [22,1,6,33,44,38,24] }
P1
rem : [4]
UB : 50, [22,1,6,33,2,38,24]
LB : 38, FS: [22,1,6,33,44,38,23]
BACL : [(33,2),(2,38)]
Z: 40 (no change)
O: { [22,1,6,33,44,38,24] } (no change)
P2
rem : [6]
UB : 49, [22,1,36,4,2,38,24]
LB : 37, FS: [22,1,36,4,44,38,23]
BACL : [(4,2),(36,2),(2,38),(4,24),(36,24)]
Z: 40 (no change)
O: { [22,1,6,33,44,38,24] } (no change)
P5
rem : [2,4]
UB : 40, [22,1,6,33,44,38,24]
LB : 36, FS: [22,1,6,33,44,38,27]
BACL : []
Z: 40 (no change)
O: { [22,1,6,33,44,38,24] } (no change)
P4
rem : [33,4]
UB : 42, [22,1,6,41,2,38,24]
LB : 33, FS: [22,1,6,41,2,35,24]
BACL : [(2,38),(41,38)]
Z: 40 (no change)
O: { [22,1,6,33,44,38,24] } (no change)
P3
rem : [4,6]
UB : 45, [22,1,36,33,2,38,24]
LB : 28, FS: [22,1,5,33,44,38,23]
BACL : [(36,33),(33,2),(36,2),(2,38),(36,24)]
Z: 40 (no change)
O: { [22,1,6,33,44,38,24] }
P6
rem : [36,4,6]
UB : 40, [22,1,5,33,2,38,24]
LB : 26, FS: [22,1,5,33,44,38,27]
BACL : [(33,2),(5,2),(2,38),(5,24)]
Z: 40 (no change)
O: { [22,1,6,33,44,38,24] } (no change)

Fig.4.17 Search diagram for the example sentence including structural ambiguities

40 is the optimum solution.

4.5 Experiment for Graph Branch Algorithm

This section describes some experimental results showing the computational complexity of the graph branch algorithm.

4.5.1 Environment and Performance Metric of the Experiment

An English text corpus, PDG grammar and preference knowledge are prepared. Preference knowledge source in this experiment is the WPP frequencies (node frequencies) and the dependency relation frequencies (arc frequencies) in the corpus. Preference score is calculated from these statistical data and attached to the arcs in the dependency graphs.

Experiment data of 125,320 sentences extracted from English technical documents is divided into open data (8605 sentences) and closed data (116,715 sentences). The closed data is used for producing WPP and dependency frequencies. An existing sentence analysis system (called the oracle system) is used as a generator of these frequencies. The oracle system is a real-world rule-based system with a long development history (Amano et al., 1989; Hirakawa et al., 2000).

PDG grammar called a basic grammar is prepared. The basic grammar consists of basic grammar rules which cover sentence variations such as noun/verb/adjective/adverbial/ prepositional phrases, simple/complex/compound sentences, relative/subordinate clauses and Onions' 5 sentence patterns[*11]. The basic grammar does not accept insertion, omission, inversion and idiomatic structures (ex. not only .. but also ..). More detailed information on the environment

[*11] S+V,S+V+C,S+V+O,S+V+O+O and S+V+O+C patterns

of this experiment is described in Section 6.1.4.

The expanded problem number, a principal computational complexity factor of the B&B method, is adopted for performance metric. The following three metrics are used in this experiment.

(a) Expanded Problem Number in Total (EPN-T): The number of the expanded problems which are generated in the entire search process.
(b) Expanded Problem Number for the First Optimum Solution (EPN-F): EPN-F is the number of the expanded problems when the first optimum solution is obtained.
(c) Expanded Problem Number for the Last Optimum Solution (EPN-L): EPN-L is the number of the expanded problems when the last optimum solution is obtained. At this point, all optimum solutions are obtained.

Optimum solution number (OSN) for a problem, i.e., the number of optimum dependency trees in a given dependency forest, gives the lower bound value for all these metrics because one problem generates at most one solution. The minimum value of OSN is 1 because every dependency forest has at least one dependency tree. As the search process proceeds, the algorithm finds the first optimum solution, then the last optimum solution, and finally terminates the process by confirming no better solution is left. Therefore, the three metrics have the relation EPN-F $\leq$ EPN-L $\leq$ EPN-T. Average values for these are described as Ave:EPN-F, Ave:EPN-L and Ave:EPN-T. Average values for the optimum solution number is described as Ave:OSN.

4.5.2 Experimental Results

An evaluation experiment for the open data is performed using a prototype PDG system implemented in Prolog. The test sentences containing more than 22 words are neglected due to the limitation of Prolog system resources in the parsing process. 4334 sentences out of 6882 test sentences are parsable. Without applying special treatment such as construction of the whole phrase structure tree from partial phrase structure trees, unparsable sentences (2584 sentences) are simply neglected in this experiment.

All optimum trees are computed by the graph branch algorithm described in Section 4.3.2. Fig.4.18 shows averages of EPN-T, EPN-L, EPN-F and OSN with respect to sentence length. Overall averages of EPN-T, EPN-L, EPN-F and OSN for the test sentences are 3.0, 1.67, 1.43 and 1.15. The result shows that the average number of problems required is relatively small. The CFG filtering by the phrase structure level reduces the search space for the dependency level and the feasible solution search function based on the greedy algorithm in Fig.4.8 seems to give a good feasible solution for a given problem and suppresses the number of expanded problems. The gap between Ave:EPN-T and Ave:EPN-L (3.0-1.67=1.33) is much greater than the gap between Ave:EPN-L and Ave:OSN(1.67-1.15=0.52). This means that the major part of the computation is performed only for checking if the obtained feasible solutions are optimum or not.

Hirakawa (2001) reported the experiment for the B&B-based optimum search algorithm im-

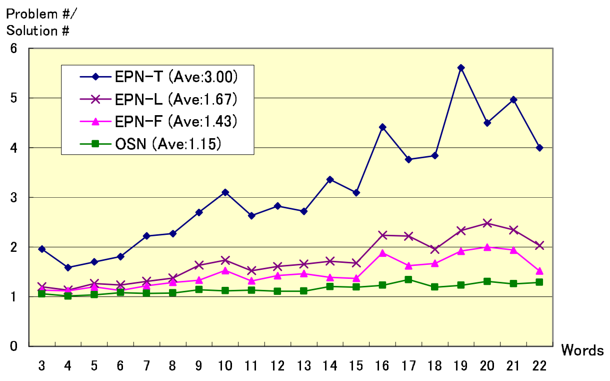


Fig.4.18 Total problem number, problem number for the first optimum solution and optimum solution number

plemented in C language using the semantic dependency graphs obtained from 100 sentences randomly selected from Japanese technical documents. Compared with the experiment reported in this thesis, the previous experiment was performed in different conditions and settings with regard to, for example, the target language(English vs. Japanese), the target dependency graph(syntactic-label WPP dependency graph vs. semantic-label word dependency graph), the scoring method(statistics-based vs. rule-based) and the search target(all optimum solution search vs. one optimum solution search). However, the two experiments have the same basic structure, i.e., the optimum tree search for scored dependency graphs with arc constraints based on the B&B principle. The B&B-based algorithms of the two experiments have very similar components of the branch and bound method and the main factor of the computational complexity is the number of the expanded problems. The previous experiment shows that overall averages of EPN-T, EPN-F are 2.91, 1.33[*12]. These result values are very similar to those in the new experiment. The tendency for the optimum solution to be obtained in the early stage of the search process was observed in the previous experiment just as it is in this experiment. Hirakawa (2001) introduced two improvements of the algorithm, i.e., the introduction of an improved upper bound function g'(P) and the optimized feasible solution search. As a result, the Ave:EPN-T is reduced from 2.91 to 1.82 and the Ave:EPN-F is increased from 1.33 to 1.35. The average CPU time is reduced from 305.8ms to 162.1ms (on engineering work station AS-4260). In the new experiment, the g'(P) is introduced to the graph branch algorithm and has obtained the reduction of the Ave:EPN-T from 3.00 to 2.68 and the reduction of the Ave:EPN-F from 1.43 to 1.36. g'(P) is also effective to some extent in this experiment.

The tendency for the optimum solution to be obtained in the early stage of the search process suggests that limiting the number of problems to expand is an effective pruning strategy. Fig. 4.19 shows the ratios of the sentences obtaining the whole problem expansion, the first optimum solution and the last optimum solution to whole sentences with respect to the expanded problem numbers. This kind of ratio is called an achievement ratio (AR) in this thesis. From Fig. 4.19,

[*12] OSN and EPN-L was not measured because the algorithm searches for only one optimum solution.

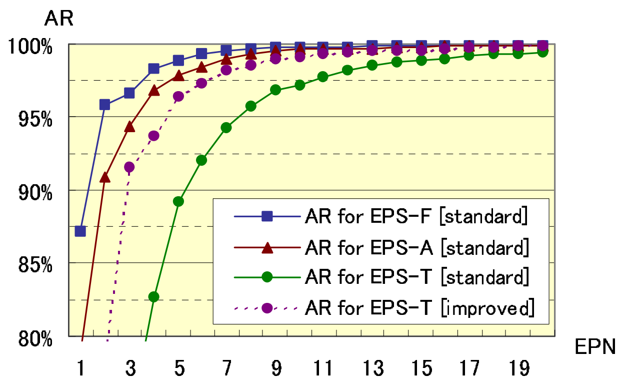


Fig.4.19 Achievement ratios for full expansion, first optimum solution expansion and last optimum solution expansion cases with respect to max problem expansion number

the ARs for EPN-T, EPN-L, EPN-F (plotted in solid lines) are 97.1%,99.6%,99.8%, respectively at the expanded solution number 10. The dotted line shows the AR for EPN-T of the improved algorithm using g'(P). The use of g'(P) increases the AR for EPN-T from 97.1% to 99.1% at the expanded solution number 10. However, the effect of g'(P) is quite small for EPN-F and EPN-L. ARs for EPN-F and EPN-L in using g'(P) is almost the same as those shown in Fig. 4.19. This result shows that the pruning strategy based on the expanded problem number is effective and g'(P) works for the reduction of the problems generated in the posterior part of the search processes.

Behavior of the search process should be affected by the scoring strategy (resources of preference knowledge and their application methods) and the structure of dependency graphs defined by grammar rules. The search process should be analyzed in greater detail along with scoring strategies and dependency graph structures. The performance of the algorithm described in (Hirakawa, 2001) is sufficient for real-world applications. The practical code implementation of the graph branch algorithm and its performance evaluation with an extended grammar are subjects for future work.

4.6 Extension to the Binary Preference Model

All optimum solution search methods for scored dependency graphs including PDG described in 4.1 treat preference scores attached to the arcs in a dependency graph. The arc scores are independent of each other or constant for all possible dependency trees. This type of dependency graph framework is called a “unary preference model” (or unary model) in this thesis. This section describes the extension of PDG to the “binary preference model” (or binary model) which can treat the preference knowledge represented by two arcs, called binary arc preference.

4.6.1 Extension of the Dependency Forest

This section gives the extension of the dependency forest and the definition of the optimum dependency tree in the binary preference model.

(1) Preference matrix

Binary arc preferences are represented by a new data structure called "preference matrix" (PM). Fig.4.20 shows an example of the dependency forest of the binary preference model.

Preference score between arc$_i$ and arc$_j$ is represented by the score (number) in the cell PM(i,j). PM(i,i) and PM(i,j) ($i \neq j$) represent the "unary arc score" and the "binary arc score," respectively. The unary preference score of arc$_i$ is the arc score of the unary model. The preference score can be a negative value that represents the negative preference, Score 0, represented by empty cell, represents the neutral preference.

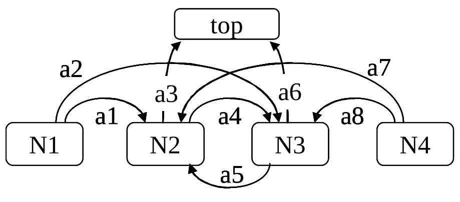
Fig.4.20 shows an example of the dependency forest <DG,CM,PM> of the binary model. The constraint matrix of the binary preference model is the same as that of the unary preference model, but is called "constraint matrix" (CM) in the binary model in order to make clear contrast with the preference matrix. The numbers in the diagonal cells in PM are unary arc scores and the other numbers are binary arc scores. This dependency forest has two well-formed dependency trees, i.e., {1,3,5,7} and {2,4,6,8}.

(2) The optimum dependency tree of the binary model

The score of a dependency tree in the unary model is defined as the sum total of the scores of the arcs in the tree. The score of the dependency tree DT in the binary model is defined as follows:

$$\text{score}(DT) = \sum_{a_i, a_j \in \text{DT}, i \leq j} \text{PM}(a_i, a_j) \tag{4.1}$$

This score is expressed by the sum total of the arc scores in DT as follows:



DG

		N1		N2		N3		N4	
		1	2	3	4	5	6	7	8
N1	1	–		○		○		○	
	2		–		○		○		○
N2	3	○		–		○		○	
	4		○		–		○		○
N3	5	○		○		–		○	
	6		○		○		–		○
N4	7	○		○		○		–	
	8		○		○		○		–

CM

		N1		N2		N3		N4	
		1	2	3	4	5	6	7	8
N1	1	10						10	5
	2		15		5			5	20
N2	3								
	4		5		15				
N3	5					20		-5	
	6								
N4	7	10	5			-5		10	
	8	5	20						10

PM

Fig.4.20 Example of the dependency forest of binary model

$$\text{score}(DT) = \sum_{a_i \in \text{DT}} \text{arc\_score}(a_i, \text{DT}) \tag{4.2}$$

$$\text{arc\_score}(a_i, \text{DT}) = \text{PM}(a_i, a_i) + \frac{1}{2} \sum_{a_j \in \text{DT}, a_j \neq a_i} \text{PM}(a_i, a_j) \tag{4.3}$$

The score definition of the binary model is a generalization of that of the unary model. The score of the dependency tree {1,3,5,7} in Fig.4.20 is computed as follows:

$$\begin{aligned}
\text{score}(\{\text{a1}, \text{a3}, \text{a5}, \text{a7}\}) &= \sum_{a_i, a_j \in \{\text{a1,a3,a5,a7}\}, i \leq j} \text{PM}(a_i, a_j) \\
&= \text{PM}(1,1) + \text{PM}(3,3) + \text{PM}(5,5) + \text{PM}(7,7) + \\
&\quad \text{PM}(1,3) + \text{PM}(1,5) + \text{PM}(1,7) + \text{PM}(3,5) + \text{PM}(3,7) + \text{PM}(5,7) \\
&= 10 + 0 + 10 + 20 + 0 + 0 + 5 + 0 + 0 - 5 = 40
\end{aligned}$$

The score of each arc is as follows:

$$\begin{aligned}
\text{arc\_score}(a1, \{\text{a1}, \text{a3}, \text{a5}, \text{a7}\}) &= \text{PM}(1,1) + \frac{1}{2}(\text{PM}(1,3) + \text{PM}(1,5) + \text{PM}(1,7)) = 15 \\
\text{arc\_score}(a3, \{\text{a1}, \text{a3}, \text{a5}, \text{a7}\}) &= \text{PM}(3,3) + \frac{1}{2}(\text{PM}(1,3) + \text{PM}(3,5) + \text{PM}(3,7)) = 0 \\
\text{arc\_score}(a5, \{\text{a1}, \text{a3}, \text{a5}, \text{a7}\}) &= \text{PM}(5,5) + \frac{1}{2}(\text{PM}(1,5) + \text{PM}(3,5) + \text{PM}(5,7)) = 17.5 \\
\text{arc\_score}(a7, \{\text{a1}, \text{a3}, \text{a5}, \text{a7}\}) &= \text{PM}(7,7) + \frac{1}{2}(\text{PM}(1,7) + \text{PM}(3,7) + \text{PM}(5,7)) = 7.5
\end{aligned}$$

The sum total of the arc scores, i.e., the score of the tree, is 40.

4.6.2 Extension of the Graph Branch Algorithm

The basic skeleton of the graph branch algorithm for the binary model is the same as that of the unary model described in Section 4.3.1. This section describes the binary model version of each component of the graph branch algorithm described in Section 4.3.2.

(1) Partial-problem

PM is added to the partial-problem of the unary model for the binary model. PM is shared with all partial problems because it is common to all partial-problems.

(2) Algorithm for Obtaining Feasible Solution and Lower Bound Value

The algorithm for obtaining a feasible solution and lower bound value for the binary model is basically equal to that of the unary model. The difference is the calculation of the arc score. The unary model simply calculates the sum total of unary arc scores of the feasible solution. The binary model calculates the arc score according to formula (4.3) described above.

In order to obtain a better (higher score) feasible solution, the sorting of arc groups as shown in *step*1 of Fig.4.8 is conducted by using the upper bound scores of the arcs obtained by the formula (4.5) described below.

(3) Algorithm for Obtaining Upper Bound Value

Given the dependency forest <DG,CM,PM> for an input sentence with word length n, a partial problem P has its dependency graph DG' which is a subset of DG. The upper bound value G of P is defined with respect to the dependency forest <DG',CM,PM> as follows.

$$\mathrm{G} = \sum_{i=0}^{\mathrm{n}-1} \max_{A \in \mathrm{arcs\_at}(i,\mathrm{DG}')} \mathrm{ubs\_arc}(A) \tag{4.4}$$

$$\mathrm{ubs\_arc}(A) = \sum_{j=0}^{\mathrm{n}-1} \mathrm{ub\_arc\_score(A,j)} \tag{4.5}$$

$$\mathrm{ub\_arc\_score}(A,j) = \begin{cases} \mathrm{PM}(A,A) & (\mathrm{position}(A) = j) \\ \displaystyle\max_{X \in \mathrm{arcs\_at}(j,\mathrm{DG}'),\mathrm{CM}(A,X)=\bigcirc} \frac{\mathrm{PM}(A,X)}{2} & (\mathrm{position}(A) \neq j) \end{cases} \tag{4.6}$$

Formula (4.4) means that the upper bound value G is calculated by summing the maximum score of ubs_arc at each position of the input sentence. ubs_arc(A) is the upper bound of arc A which is the sum of the unary arc score of A, i.e., PM(A,A) and the maximum binary arc scores between A and the arcs of each position as defined in formula (4.6). The set of arcs selected in formula (4.4) for each input position is called the "maximum well-covered binary arc set" and does not necessarily constitute a tree and is not necessarily consistent with the arcs selected in formula (4.6).

The following shows the example of the upper bound computation of the dependency forest in Fig.4.20. The input position of N1, N2, N3 and N4 are 0, 1, 2 and 3, respectively. At first, an example of the computation of ub_arc_score for arc a1 and a2 is shown as follows:

$$\begin{aligned}
&\mathrm{ub\_arc\_score}(a1,0) = \mathrm{PM}(1,1) = 10 \\
&\mathrm{ub\_arc\_score}(a1,1) = 0.5 \times \max(\mathrm{PM}(1,3),\mathrm{PM}(1,4)) = 0.5 \times \max(0,0) = 0 \\
&\mathrm{ub\_arc\_score}(a1,2) = 0.5 \times \max(\mathrm{PM}(1,5),\mathrm{PM}(1,6)) = 0.5 \times \max(0,0) = 0 \\
&\mathrm{ub\_arc\_score}(a1,4) = 0.5 \times \max(\mathrm{PM}(1,7),\mathrm{PM}(1,8)) = 0.5 \times \max(10,5) = 5 \\
&\mathrm{ub\_arc\_score}(a2,0) = \mathrm{PM}(2,2) = 15 \\
&\mathrm{ub\_arc\_score}(a2,1) = 0.5 \times \max(\mathrm{PM}(2,3),\mathrm{PM}(2,4)) = 0.5 \times \max(0,5) = 2.5 \\
&\mathrm{ub\_arc\_score}(a2,2) = 0.5 \times \max(\mathrm{PM}(2,5),\mathrm{PM}(2,6)) = 0.5 \times \max(0,0) = 0 \\
&\mathrm{ub\_arc\_score}(a2,4) = 0.5 \times \max(\mathrm{PM}(2,7),\mathrm{PM}(2,8)) = 0.5 \times \max(5,20) = 10
\end{aligned}$$

The ubs_arc is the sum total of ub_arc_score values, i.e., ubs_arc(a1) = 15 and ubs_arc(a2) = 27.5. Arc a2 is selected as a member of the maximum well-covered binary arc set for DG and PM, because the arc which has the maximum ubs_arc scores at each position is selected as the upper bound arc at the position as described above. Similarly, the upper bound arcs are selected and the maximum well-covered binary arc set is computed as {a2, a4, a5, a8} that has the upper bound score 85 (27.5+17.5+20+20).

(4) Branch Operation

The branch operation is basically equivalent to that of the unary model as described in 4.3.2. An inconsistent arc pair (arc_i, arc_j), i.e., CM(i,j)$\neq \bigcirc$, is searched from the maximum well-covered binary arc set for graph branch operation. If no inconsistent arc pair is found, the maximum well-covered binary arc set is one of the optimum solutions for the partial problem. In the case of the algorithm for searching for all optimum solutions, branch operation continues until all partial problems have proved to have no optimum solutions as described in (6) below.

(5) Selection of Partial-problem from Active Partial-problems

This process is the same as that of the unary model, i.e., the best bound search is employed.

(6) Computing All Optimum Solutions

When a new optimum solution for a partial-problem is obtained, the optimum solution is recorded in the incumbent solution list and further branch operation is performed until the upper bounds of the partial-problems become less than the incumbent value. Arcs to be removed from the dependency graph of the current partial-problem, i.e., the candidates for branch operation, are computed by picking up the rival arcs of the arcs in the obtained optimum solution. The rival arc of arc A of the binary model is an arc whose position is equal to that of arc A and the upper bound score is equal to or more than that of arc A. When all partial problems are proved to have upper bound scores less than the incumbent value, the search process terminates.

4.7 Concluding Remarks for Chapter 4

This section has described the graph branch algorithm for obtaining the optimum solution for a dependency forest used in the preference dependency grammar. In addition to the basic model, i.e., the unary model, the graph branch algorithm for the binary model is introduced for treating the arc co-occurrence preference.

The well-formedness dependency tree constraints are represented by the constraint matrix of the dependency forest, which has flexible and precise description ability so that controlled non-projectivity is available in PDG framework. The graph branch algorithm assures the search for the optimum trees with arbitrary arc co-occurrence constraints, including the single valence occupation constraint which has not been treated in DP-based algorithms so far. The dependency forest has wider applicability compared with the semantic dependency graph because it can handle whole morphological ambiguity caused by homonyms and word boundary divisions. The experimental result shows the averages of EPN-T, EPN-L and EPN-F for English test sentences are 3.0, 1.67 and 1.43, respectively. This suggests the graph branch algorithm for PDG would show a performance comparable to the algorithm for the semantic dependency graph applied in real-world applications.

Chapter 5

Scoring

Various kinds of preference knowledge exists at various analysis levels as shown in Fig.2.2. The scoring process gives preference scores to the packed shared data structures prescribed by the generation and constraint knowledge in PDG. The scoring process determines the output, i.e., the accuracy of the NLA system under the given generation and constraint knowledge. The resources of preference knowledge, the integration method and the target application domain (sentences) should be mutually related with respect to the performance of the scoring process of PDG-based systems. Investigating the way to construct the best scoring method will require a great deal of research. The purpose of this chapter is to show the basis of the scoring framework with respect to the dependency forest and the optimum solution described in the previous chapters, and to show the first step of the scoring method of PDG which integrates some kind of multilevel preference knowledge enabling the experiment of the PDG framework.

5.1 Preference Knowledge and Score Integration

5.1.1 Principle of Score Integration

In designing the scoring method of PDG, the following issues are taken into consideration.

(a) Corpus oriented data are used as the resources of the preference knowledge.

(b) Different kinds of preference knowledge obtained from different corpora can be the resources of the preference knowledge. (knowledge resource robustness)

(c) Utilize learning methods to optimize the scoring parameters.

(a) seems to be the only and the best way to get a large amount and coverage of preference knowledge, because the hand coding of preference scores is intractable. The combination of human insight (inductive ability) and the computational power of computers will be a good approach for large-scale knowledge development. This requires the combination of the rule-based (human-based) and the statistics-based (computer-based) methods (Su et al., 1996; Riezler et al., 2002). (b) is a requirement for the large-scale knowledge development. It seems to be very difficult to prepare the entire spectrum of preference knowledge data for one very large corpus, if it requires human processing. Robustness for the preference knowledge resources is one of the desirable features for NLA systems. (c) is necessary for obtaining the optimum NLA system. Recently, a great deal of research on learning methods for natural language sentence analysis has been done for both generative (Eisner, 1996b; Collins, 1999; Charniak, 2000) and discriminative models (Riezler et al., 2002; Miyao and Tsujii, 2002; Clark and Curran, 2003; Clark and Curran, 2004; Taskar et al., 2004; McDonald et al., 2005). These research results should be considered and incorporated in the scoring process of PDG. As described above, the target of this research is to show the first framework for integrating multilevel preference knowledge. Introduction of the learning techniques is an important subject for future work.

5.1.2 Basis of Score Integration

In order to integrate the preference knowledge, it should be converted into some numeric values, i.e., preference scores. The descriptive power of the preference knowledge is prescribed by the descriptive ability of the interpretation description scheme and the optimum solution search method. For example, WPP bigram preference score can be represented as arc scores in the WPP trellis and the optimum WPP sequence is computed by the Viterbi algorithm*1. WPP dependency preference score can be represented as arc scores in the dependency graph and the optimum dependency tree is computed by some algorithms as described in the previous chapter. PDG adopts the dependency forest as the basis of the preference score integration. All preference scores obtained from preference knowledge at each analysis level are integrated into the preference scores in the dependency forest by use of the interpretation mappings among the WPP trellis, the phrase structure forest and and the dependency forest. The descriptive power of the preference knowledge is prescribed by the descriptive ability of the dependency forest and the graph branch algorithm in PDG.

The dependency graph of the dependency forest can register the "unary node scores" and the "unary arc scores". The unary node scores can be represented by the unary arc scores because each arc has one dependency node and the top node has a constant score. As described in the previous chapter, the preference matrix PM registers the "binary arc scores" as well as the unary arc scores for a dependency graph. PM can represent the "binary node scores" because they can be represented by the corresponding binary arc scores. These four scores, i.e., the unary node score, the unary arc score, the binary node score and the binary arc score, constitute the basis for the scoring for all kinds of preference knowledge and are integrated into the scores of PM. Of course, PM has limitation in its representation ability, for example, it cannot express higher order preference knowledges based on more than three elements, such as sequences with more than three nodes and co-occurrences of three or more arcs (dependencies). PM can represent the preference of the phrase structure rules with less than four constituents because they have less than three arcs. Phrase structure preference for CFG in the Chomsky normal form can be

*1 WPP trigram preference score cannot be treated in this method.

handled by unary model because each PDG rule has only one dependency arc. As described in Chapter 1, majority of current dependency analysis systems adopts the edge factored model which corresponds to the unary model of PDG. Higher order preference scores should be available by introducing higher order preference matrix in exchange for higher computational expenses. This is beyond the scope of this thesis.

The value of the PM is defined by two major functions, i.e., unary_score and binary_score functions, as follows:

$$\mathrm{PM}(a_i, a_j) = \begin{cases} \tau\cdot\text{unary\_score}(a_i) & (i = j) \\ (1-\tau)\cdot\text{binary\_score}(a_i, a_j) & (i \neq j) \end{cases}$$

τ ($0 \leq \tau \leq 1$) is a parameter called the "unary/binary score distribution ratio" or simply "UB ratio" that is used for adjusting the balance between the unary score and the binary score. The unary and binary scores are described in the succeeding sections.

The preference knowledge about the relation concerning more than three nodes or three arcs, for example N-gram sequence where N $\geq$ 3 and phrase structure rule with more than or equal to four constituents, are outside the scope of the current PDG scoring processing.

5.2 Scoring Function and Scaling Coefficient

The majority of the preference knowledge obtained from corpora is represented as the frequencies of the linguistic elements or relations, such as word, WPP, WPP sequence, phrase structures and dependencies, in combination with various kinds of attributes of the elements. The frequency data should be converted into preference scores, which are the basis of the integration operation. These conversions are performed by heuristic functions called "scoring functions." Scoring functions apply frequency normalization because PDG assumes that various kinds of corpora are used as the resources of the preference knowledge as described above. "logave" is the basic form of the scoring functions for an element E.

$$\text{basic\_score}(E) = \text{logave}(X, AddX, AveX) = \text{BaseScore}\cdot\frac{\log((X+1) + AddX)}{\log(AveX + 1)}$$

where X is the frequency of the element E, $AddX$ is an extra frequency for E called "frequency compensation term" or "frequency compensation," and $AveX$ is the average frequency of the data type to which E belongs. BaseScore is a standard score assigned to the average frequencies and is set to 1000 currently. For example, if the word 'theorem' has 99 frequency and the average word frequency is 9 in a corpus, the basic score of 'theorem' is 2000 with no frequency compensation as follows:

$$\text{basic\_score}(\mathit{theorem}) = \text{logave}(99, 0, 9) = 1000 \cdot \frac{\log(99+1)}{\log(9+1)} = 2000$$

The frequency in logave is biased by 1 so that zero frequency element generates the zero score. In the case that X is equal to $AveX$, the basic score is BaseScore(=1000) with no frequency compensation. This is introduced to normalize the frequency. "log" function is applied for leveling the frequencies. There is no theoretical reason for this leveling function but the scoring method without this leveling function leads to a poor result according to the result of the preliminary experiments[*2]. The frequency compensation is used for, for example, adjusting the frequency of compounds. The details are described below.

PDG introduces another type of functions called "scaling functions," which generate the "scaling coefficients" for an element E. When scaling function 'f' is defined for E, the total score of an element E is the product of the scoring function and the scaling function of E as follows:

$$\mathrm{score}(E) = \mathrm{f}(E)\cdot\mathrm{basic\_score}(E)$$

Scaling functions are also heuristic functions for representing the distribution or importance of E in the scoring process. Examples are shown below.

There is no theoretical or experimental grounding for the correctness or the optimality of the above scoring function and the scaling function that are determined by some preliminary experiments.

5.3 Unary Score Formula

The "unary score" (UnaryScore) is a combination of the unary node score (UnaryNodeScore) and the unary arc score (UnaryArcScore) as follows:

$$\mathrm{UnaryScore} = \frac{\alpha\cdot\mathrm{UnaryNodeScore} + (1-\alpha)\cdot\mathrm{UnaryArcScore}}{2}$$

where α is the unary node/arc score distribution ratio (UNA ratio) satisfying $0 \leq \alpha \leq 1$.

5.3.1 Unary Node Score Formula

The current implementation of the unary node score formula contains only one preference score calculated from the WPP frequencies in a corpus. The basic formula is very simple but it has to be extended with a compensation term in order to treat compound words.

The unary node score for WPP node N is calculated by the following formula.

$$\begin{aligned}\mathrm{unary\_node\_score}(N) &= \mathrm{logave}(\mathrm{freq}(N), \mathrm{un\_comp}(N), \mathrm{AveWPPF}) \\ &= \mathrm{BaseScore}\cdot\frac{\log(\mathrm{freq}(N) + 1 + \mathrm{un\_comp}(N))}{\log(\mathrm{AveWPPF} + 1)}\end{aligned}$$

where AveWPPF is the average WPP frequency in the corpus[*3]. The un_comp(N) (unary node compensation) is the frequency compensation term for compound words defined as follows:

[*2] Introduction of statistical distribution model to the basic scoring function may lead to better results.

[*3] This is the average occurrence number for the WPPs found in the corpus

$$\text{un\_comp}(N) = \begin{cases} \text{element\_freq}(N) + \text{AveWPPF}\cdot CWC \cdot 2^{\text{wrdlen}(N)-1} & (N \text{ is compound word}) \\ 0 & (\text{Otherwise}) \end{cases}$$

where element_freq(N) is the sum total of the frequencies of the words in N. CWC (compound word coefficient) is a parameter for adjusting the preference of compound words against non-compound words. wrdlen(N) is a word number of N.

In general, compound words have very small frequency compared with their constituent words but should have higher preferences. The first term, i.e., element_freq(N), assures that the compound word has higher frequency and the second term gives extra frequency for the compound that has more than two constituents. The current setting of CWC is 3.

5.3.2 Unary Arc Score Formula

The basic resources of the unary arc scores are the dependency frequencies, i.e., the frequencies of the dependency pieces, in a corpus in the current implementation. The unary arc score of arc A is calculated by the following formula.

$$\text{unary\_arc\_score}(A) = \text{basic\_arc\_score}(A)\times\text{distance\_ratio}(A)\times\text{POS\_ratio}(A)$$

basic_arc_score gives the basic unary arc score for A. distance_ratio and POS_ratio are scaling coefficients for the distance between the dependant node and the governor node and the compensation based on the type of the arc, respectively.

(1) basic_arc_score(A)

basic_arc_score calculates the basic arc score for arc A based on the dependency piece frequencies in a corpus. This frequency is called the asis arc frequency[*4]. In addition to this standard arc frequency, three additional frequencies are used for the resources of the preference score.

(a) Asis arc frequency(ASIS_AF) : Frequency of the dependency piece
(b) Generalized arc frequency (GEND_AF) : Frequency of the generalized dependency piece
(c) Asis PP frequency(ASIS_PF) : Frequency of the PP-attachment frequency
(d) Generalized PP frequency(GEND_PF) : Frequency of the generalized PP-attachment frequency

(a) is a basic arc frequency, i.e., the frequency of a dependency piece. A dependency piece consists of three elements, i.e., the dependant node, the governor node and the arc, and is a more complicated data structure than a simple node. This causes the data sparseness problem. In order to manage this problem, a backoff method based on (b) is introduced. The abstract dependency piece is obtained by generalizing the POSs of the dependant node and the governor node in the dependency piece.[*5] The generalization of the POS is done by simply taking the

[*4] Expression "arc frequency" is used instead of "dependency piece frequency"

[*5] Various kinds of semantic abstraction and word sense disambiguation methods (Hearst and Schutze, 1993; Resnik, 1993; Resnik, 1995a; Resnik, 1995b; Hirakawa et al., 1996; McCarthy, 1997; Seki et al., 1997;

first character of the POS as follows:

Dependency piece : [time/nx] $\xrightarrow{\text{subj}}$ [fly/vt]
Generalized dependency piece: [time/n] $\xrightarrow{\text{subj}}$ [fly/v]

(c) and (d) are introduced for incorporating PP-attachment preference which cannot be represented correctly by (a) and (b). The following shows an example of PP-attachment for "see girl with telescope."

(e1) [see/vt] $\xleftarrow{\text{vpp}}$ [with/pre] $\xleftarrow{\text{pre}}$ [telescope/n]
(e2) [girl/n] $\xleftarrow{\text{npp}}$ [with/pre] $\xleftarrow{\text{pre}}$ [telescope/n]

The arc frequencies of the arcs in (e1) and (e2) cannot capture the competition between the PP-attachments "see with telescope" and "girl with telescope" because "[see/vt] $\xleftarrow{\text{vpp}}$ [with/pre]" and "[girl/n] $\xleftarrow{\text{npp}}$ [with/pre]" are independent of "[with/pre] $\xleftarrow{\text{pre}}$ [telescope/n]." This problem is solved by reducing the preposition node into an arc label as follows:

(e3) [see/vt] $\xleftarrow{\text{vpp\_with}}$ [telescope/n]
(e4) [girl/n] $\xleftarrow{\text{npp\_with}}$ [telescope/n]

This method is not adopted in the current implementation of the PDG system because the output of the data structure is the same as that of the existing NLA system for evaluation as described in the section below. Instead, (c) and (d) are introduced to solve this problem.

The asis PP frequency(ASIS_PF) in (c) corresponds to the frequency of the relation such as (e3) or (e4). Generalized PP frequency (GEND_PF) is the frequency of the generalized relation introduced to manage the data sparseness problem. The generalized relation has the generalized POSs and arc name as shown in the following example, the generalized relation corresponding to (e3).

(e5) [see/v] $\xleftarrow{\text{with}}$ [telescope/n]

The basic arc score is defined as follows:

$$\text{basic\_arc\_score}(A) = \mu\cdot\text{asis\_arc\_score}(A) + (1-\mu)\cdot\text{generalized\_arc\_score}(A)$$

asis_arc_score and generalized_arc_score represent the arc scores computed from the asis arc frequencies and the generalized arc frequencies, respectively. μ is called the asis/generalized arc distribution ratio which is defined later in this section. First, asis_arc_score and generalized_arc_score are described.

[asis_arc_score(A)]

$$\begin{aligned}\text{asis\_arc\_score}(A) &= \text{logave}(\text{asis\_max\_freq}(A), 0, \text{AveASIS\_AF})\\ &= \text{BaseScore}\cdot\frac{\log(\text{asis\_max\_freq}(A)+1)}{\log(\text{AveASIS\_AF}+1)}\end{aligned}$$

Kimura and Hirakawa, 2000) are also candidates for the backoff method in future study.

$$\text{asis\_max\_freq}(A) = \begin{cases} \max\{\text{freq}(\text{Rel}, \text{I}, \text{J}) | \text{I} \subseteq \text{DN}, \text{J} \subseteq \text{GN}\} + \text{PPC}\cdot\text{asis\_pf}(A) \\ \qquad\qquad\qquad (\text{DN and/or GN of } A \text{ are/is compound word(s)}) \\ \text{asis\_af}(A) + \text{PPC}\cdot\text{asis\_pf}(A) \\ \qquad\qquad\qquad (\text{Otherwise}) \end{cases}$$

where Rel,DN and GN is the relation, dependency node and the governor node of arc A. freq(Rel,DN,GN) is the frequency of the dependency piece. asis_af and asis_pf are the frequencies of (a) and (c) above, respectively. PPC(Prepositional Phrase Coefficient) is the coefficient for adjusting the effect of the PP-attachment frequency described above, which is currently set to 5. AveASIS_AF is the average of the asis arc frequency in the corpus.

[generalized_arc_score(A)]

The generalized maximum frequency of arc A is defined as follows:

$$\begin{aligned} \text{generalized\_arc\_score}(A) &= \text{logave}(\text{gend\_max\_freq}(A), 0, \text{AveGEND\_AF}) \\ &= \text{BaseScore}\cdot\frac{\log(\text{gend\_max\_freq}(A) + 1)}{\log(\text{AveGEND\_AF} + 1)} \end{aligned}$$

$$\text{gend\_max\_freq}(A) = \begin{cases} \max\{\text{freq}(\text{GEND\_Rel}, \text{I}, \text{J}) | \text{I} \subseteq \text{GEND\_DN}, \text{J} \subseteq \text{GEND\_GN}\} + \text{PPC}\cdot\text{gend\_pf}(A) \\ \qquad (\text{GEND\_DN and/or GEND\_GN of } A \text{ are/is compound word(s)}) \\ \text{gend\_af}(A) + \text{PPC}\cdot\text{gend\_pf}(A) \quad (\text{Otherwise}) \end{cases}$$

where GEND_Rel,GEND_DN and GEND_GN is the generalized relation, dependency node and the governor node of arc A. freq(GEND_Rel,GEND_DN,GEND_GN) is the frequency of the generalized dependency piece. gend_af and gend_pf are the frequencies of (b) and (d) above, respectively. AveGEND_AF is the average of the generalized arc frequency in the corpus.

[Asis/generalized arc distribution ratio μ]

In the current implementation, the asis/generalized arc distribution ratio is defined such that the influence of the asis frequency and the influence of the generalized frequency become the same in total as follows:

$$\mu = \frac{\text{ASIS\_KIND\_NUM}}{\text{ASIS\_KIND\_NUM} + \text{GEND\_KIND\_NUM}}$$

where ASIS_KIND_NUM and GEND_KIND_NUM are the cardinal numbers of the set of asis arcs and the set of generalized arcs in the corpus, respectively.

(2) Scaling coefficient: distance_ratio(A)

The distance ratio compensates the arc score based on the distance between the dependant node and the governor node of an arc. Collins (1996) reported that 95.6% and 99.0% of the words have dependant words within word distance 5 and 10, respectively. Distance parameters are utilized by many NLA systems (Eisner, 1996b; Collins, 1999; McDonald et al., 2005).

Let the same(X,Y) mean that arc X and Y have the same relation, the same dependant node and the same governor node and let distance(A) mean the distance between the dependant node and the governor node of arc A. The following is the distance ratio formula in the current implementation of the PDG system.

$$\text{distance\_ratio}(A) = 1 + \text{K}\cdot \log(\text{distance\_degree}(A))$$

where K is a parameter for adjusting the degree of the distance ratio. The current setting of K is $\text{K} = \frac{0.5}{\log(10)}$.[*6] Basically, distance_degree is a ratio of the arc frequency with the distance D against the average distance of the arc A.

$$\text{distance\_degree}(A) = \text{logave}(\text{df}(A), \text{dfc}(A), \text{average\_df}(A))$$

where df(A) is the frequency of the arc X such that same(X,A), distance(X)=distance(A).

dfc is the distance frequency compensation defined as follows:

$$\text{dfc}(A) = \begin{cases} 1 & (\text{df}(A) = 0) \\ 0 & (\text{Otherwise}) \end{cases}$$

average_df(A) is the average frequency of the arc with respect to the node distance defined as follows:

$$\text{average\_df}(A) = \frac{1}{|\text{DS}|} \sum_{D \in \text{DS}} \text{freq}(A, D)$$

where $\text{DS} = \{D_i | D_i = \text{distance}(A_i), \text{same}(A_i, A), A_i \in \text{CorpusArcs}\}$. freq(A,D) is the frequency of the arc X such that same(X,A),distance(X)=D.

(3) Scaling coefficient: POS_ratio(A)

The arc type ratio POS_ratio(A) compensates the arc score based on the type of the arc. The arc type is mainly characterized by the POS of the dependant node of the arc.

$$\text{POS\_ratio}(A) = \begin{cases} 0.01 & (\text{the POS of the dependant of } A = \text{det}) \\ 0.2 & (\text{the dependant of} A = [\text{be}] - \text{v and the relation of } A = \text{subj}) \\ 1 & (\text{Otherwise}) \end{cases}$$

The first compensation is introduced to reduce the influence of the score of the determiner because determiners have high frequencies or scores but are relatively unimportant for determining the overall structures. The second compensation is introduced to adjust the score balance of the two usages of be-verb,i.e., copula and present progressive interpretation as contained in the following sentence.

My hobby is watching birds.

These two compensations are encoded by hand through a simple observation of the analysis results. The current implementation of POS_ratio(A) seems to be poor. Various compensations based on the arc types should be introduced widely and their parameters should be optimized by an appropriate learning method in future.

[*6] If the distance_degree(A) is 10, the distance_ratio is 1.5. This means the arc score is multiplied by 1.5.

5.4 Binary Score Formula

The binary score (BinaryScore) is a combination of the binary node score (BinaryNodeScore) and the binary arc score (BinaryArcScore) as follows:

$$\text{BinaryScore} = \frac{\beta\cdot\text{BinaryNodeScore} + (1-\beta)\cdot\text{BinaryArcScore}}{2}$$

where β is the binary node/arc score distribution ratio (BNA ratio) $0\leq\beta\leq1$.

5.4.1 Binary Node Score Formula

The current implementation of the binary node score formula contains a preference score based on the WPP bigram frequencies in a corpus. Generalized WPP bigram frequencies are introduced as in the case of unary node score formula. The binary node score for an WPP node $N1$ and $N2$ is calculated by the following formula:

$$\text{binary\_node\_score}(N1, N2) = \text{basic\_binary\_node\_score}(N1, N2)$$

No scaling function is applied in the current implementation. basic_binary_node_score calculates the basic binary node score for node N based on the following bigram frequencies in a corpus.

(a) GWPP_BGM frequency(WPP_BF) : Frequency of the generalized WPP bigram
ex. [time/nx] [fly/vt] (WPP bigram) $\rightarrow$ [time/n] [fly/v] (GWPP bigram)

(b) POS_BGM frequency(POS_BF) : Frequency of the POS bigram
ex. [time/nx] [fly/vt] (WPP bigram) $\rightarrow$ [nx] [vt] (POS bigram)

The basic binary node score is defined as follows:

$$\begin{aligned}&\text{basic\_binary\_node\_score}(N1, N2)\\&= \chi\cdot\text{GWPP\_BGM\_score}(N1, N2) + (1-\chi)\cdot\text{POS\_BGM\_score}(N1, N2)\end{aligned}$$

GWPP_BGM_score and POS_BGM_score represent the node scores computed from the GWPP bigram frequencies and the POS bigram frequencies, respectively. χ is called the bigram score distribution ratio (BGM ratio) which is defined later in this section. First, GWPP_BGM_score and POS_BGM_score are described.

[GWPP_BGM_score(N1,N2)]

The GWPP_BGM_score($N1$,$N2$) is defined as follows:

$$\begin{aligned}&\text{GWPP\_BGM\_score}(N1, N2)\\&= \text{logave}(\text{GWPPBGM\_freq}(N1, N2), \text{GWPPBGM\_comp}(N1, N2), \text{AveGWPP\_BF})\\&= \text{BaseScore}\cdot\frac{\log(\text{GWPPBGM\_freq}(N1, N2) + 1 + \text{GWPPBGM\_comp}(N1, N2))}{\log(\text{AveGWPP\_BF} + 1)}\end{aligned}$$

AveGWPP_BF is the average of the generalized WPP bigram frequency in the corpus. GWPPBGM_freq (generalized WPP bigram compensation) is the frequency compensation term for compound words defined as follows:

$$\text{GWPPBGM\_comp}(N1, N2) = \text{CBC}\cdot\text{AveGWPP\_BF}\cdot(\text{wrdnum}(N1) + \text{wrdnum}(N2) - 2)$$

where CBC(Compound Bigram Coefficient) is the coefficient for adjusting the degree of the compound word frequency compensation, which is currently set to 3.

[**POS_BGM_score(N1,N2)**]

The POS_BGM_score($N1$,$N2$) is defined as follows:

$$\begin{aligned}&\text{POS\_BGM\_score}(N1, N2)\\ &= \text{logave}(\text{POSPBGM\_freq}(N1, N2), \text{POSBGM\_comp}(N1, N2), \text{AvePOS\_BF})\\ &= \text{BaseScore}\cdot\frac{\log(\text{POSBGM\_freq}(N1, N2) + 1 + \text{POSPBGM\_comp}(N1, N2))}{\log(\text{AvePOS\_BF} + 1)}\end{aligned}$$

AvePOS_BF is the average of the POS bigram frequency in the corpus. POSBGM_freq (POS bigram compensation) is the frequency compensation term for compound words defined as follows:

$$\text{POSBGM\_comp}(N1, N2) = \text{CBC}\cdot\text{AvePOS\_BF}\cdot(\text{wrdnum}(N1) + \text{wrdnum}(N2) - 2)$$

where CBC is the coefficient for adjusting the degree of the compound word frequency compensation, which is currently set to 3.

[**Bigram score distribution ratio** χ]

In the current implementation, the bigram distribution ratio is defined such that the influence of the generalized WPP bigram frequency and the influence of the POS bigram frequency become the same in total as follows:

$$\chi = \frac{\text{GWPP\_BGM\_KIND\_NUM}}{\text{GWPP\_BGM\_KIND\_NUM} + \text{POS\_BGM\_KIND\_NUM}}$$

where GWPP_BGM_KIND_NUM and POS_BGM_KIND_NUM are the cardinal numbers of the set of generalized WPP bigrams and the set of POS bigrams in the corpus, respectively.

5.4.2 Binary Arc Score Formula

The binary arc score provides the detailed preference knowledge with wider context, which may compete with the unary arc preference. For example, “eat gasoline” has low preference because of the semantic preference that “gasoline” cannot be eaten. However, this is not true in the sentence “This car eats gasoline.” The preference score for “eat gasoline” should be changed with respect to the subject of “eat.” This kind of preference knowledge is represented by the binary arc co-occurrence preference score. In addition to the generalized dependency piece introduced in Section 5.3.2, word dependency piece is introduced to get more abstract arc frequency in consideration of lower frequencies of binary arcs compared to those of unary arcs. The word dependency piece is obtained by omitting the POS and dependency relation in a dependency piece as follows:

Dependency piece : [time/nx] $\xrightarrow{\text{subj}}$ [fly/vt]
Word dependency piece : [time] $\rightarrow$ [fly]

There can be various definitions for binary arc co-occurence. For example, the "the two arcs co-occuring within a sentence" is one of the possible definitions. Since the purpose of the arc co-occurence score is to measure the plausibility of a sentence interpretation, it should reflect some grammatical or semantic relation as much as possible. From this consideration, two connected arcs are counted as co-occured arcs in current implementation. There are two types of connection relations, i.e., parent relation and sibling relation. Arcs in parent relation have a common shared node which is a dependant node of one arc and is a governor node of another arc. Arcs in sibling relation have a common shared governor node. Sentence "This car eats gasoline" shows an example of sibling arcs connected via the node corresponding to "eat" as follows;

Sibling arcs : [car/n] $\xrightarrow{\text{subj}}$ [eat/vti], [gasoline/n] $\xrightarrow{\text{obj}}$ [eat/vti] [*7]

The generalized dependency arcs (pieces) and the word dependency arcs (pieces) for these co-occured arecs are as follows;

Generalized sibling arcs : [car/n] $\xrightarrow{\text{subj}}$ [eat/v], [gasoline/n] $\xrightarrow{\text{obj}}$ [eat/v]
Word dependency sibling arcs : [car] $\rightarrow$ [eat], [gasoline] $\rightarrow$ [eat]

The binary arc score for two arcs $A1$ and $A2$ is calculated by the following formula:

$$\text{binary\_arc\_score}(A1, A2) = \text{basic\_binary\_arc\_score}(A1, A2)$$

No scaling function is applied in the current implementation. basic_binary_arc_score calculates the basic binary arc score for arc A1 and A2 based on the following arc co-occurence frequencies in a corpus.

(a) CGA frequency(CGAF) : Frequency of the connected generalized arcs
    CGAF(A1,A2) is sum-total of the P_CGAF(A1,A2) and S_CGAF(A1,A2).[*8]
    P_CGAF : Frequency of the connected generalized arcs in parent relation
    S_CGAF : Frequency of the connected generalized arcs in sibling relation
(b) CWA frequency(CWAF) : Frequency of the connected word arcs
    CWAF(A1,A2) is sum-total of the P_CWAF(A1,A2) and S_CWAF(A1,A2).
    P_CWAF : Frequency of the connected word arcs in parent relation
    S_CGAF : Frequency of the connected word arcs in sibling relation

[*7] POS "vti" specifies the verb which can either be intransitive or transitive.
[*8] P_CGAF(A1,A2) and S_CGAF(A1,A2) are 0 if the two arcs are not in parent relation and in sibling relation, respectively.

If A1 and A2 are in neither parent relation nor sibling relation, CGAF(A1,A2) and CWAF(A1,A2) have 0 value according to the definition.

The basic binary arc score is defined as follows:

$$\text{basic\_binary\_arc\_score}(A1, A2) = \psi\cdot\text{CGA\_score}(A1, A2) + (1-\psi)\cdot\text{CWA\_score}(A1, A2)$$

CGA_score and CWA_score represent the binary arc scores computed from the CGA frequencies and the CWA frequencies, respectively. ψ is called the connected arc score distribution ratio (CAS ratio) which is defined later in this section. First, CGA_score and CWA_score are described.

[**CGA_score(A1,A2)**]

The CGA_score($A1$,$A2$) is defined as follows:

$$\begin{aligned}\text{CGA\_score}(A1, A2) &= \text{logave}(\text{CGAF}(A1, A2), \text{CGA\_comp}(A1, A2), \text{Ave\_CGAF}) \\ &= \text{BaseScore}\cdot\frac{\log(\text{CGAF}(A1, A2) + 1 + \text{CGA\_comp}(A1, A2))}{\log(\text{Ave\_CGAF} + 1)}\end{aligned}$$

Ave_CGAF is the average of the connected generalized arc frequency in the corpus. CGA_comp (connected generalized arc compensation) is the frequency compensation term for compound words defined as follows:

$$\text{CGA\_comp}(A1, A2) = \text{CCAC}\cdot\text{Ave\_CGAF}\cdot(\text{wrdnum}(A1) + \text{wrdnum}(A2) - 4)$$

where wrdnum(A) is the total number of words of the dependant node and the govoner node in arc A. CCAC(Compound Connected Arc Coefficient) is the coefficient for adjusting the degree of the connected frequency compensation, which is currently set to 3.

[**CWA_score(A1,A2)**]

The CWA_score($A1$,$A2$) is defined as follows:

$$\begin{aligned}\text{CWA\_score}(A1, A2) &= \text{logave}(\text{CWAF}(A1, A2), \text{CWA\_comp}(A1, A2), \text{Ave\_CWAF}) \\ &= \text{BaseScore}\cdot\frac{\log(\text{CWAF}(A1, A2) + 1 + \text{CWA\_comp}(A1, A2))}{\log(\text{Ave\_CWAF} + 1)}\end{aligned}$$

Ave_CWAF is the average of the connected word arc frequency in the corpus. CWA_comp (connected word arc compensation) is the frequency compensation term for compound words defined as follows:

$$\text{CWA\_comp}(A1, A2) = \text{CCAC}\cdot\text{Ave\_CWAF}\cdot(\text{wrdnum}(A1) + \text{wrdnum}(A2) - 4)$$

where wrdnum(A) is the total number of words of the dependant node and the govoner node in arc A. CCAC(Compound Connected Arc Coefficient) is the coefficient for adjusting the degree of the connected frequency compensation, which is currently set to 3.

[**Connected arc score distribution ratio** ψ]

In the current implementation, the connected arc score distribution ratio is defined such that the influence of the connected generalized arc frequency and the influence of the connected word arc frequency become the same in total as follows:

$$\psi = \frac{\mathrm{CGA\_KIND\_NUM}}{\mathrm{CGA\_KIND\_NUM} + \mathrm{CWA\_KIND\_NUM}}$$

where CGA_KIND_NUM and CWA_KIND_NUM are the cardinal numbers of the set of connected generalized arcs and the set of connected word arcs in the corpus, respectively.

Chapter 6

Evaluation

PDG is a new framework using multiple kinds of packed shared data structures to utilize multilevel preference and constraint knowledge. Traditional evaluation methods are not enough for evaluating some system abilities which are targeted by PDG. This chapter first discuss how to evaluate the performance of PDG-based systems, then shows the experiment for investigating the possibilities of the PDG framework.

6.1 Evaluation Measures for Dependency-graph-based Systems

6.1.1 Traditional Evaluation Measures and Points in the PDG Evaluation

Various methods are proposed for evaluating natural language analysis systems (Carroll et al., 1998). The method proposed by GEIG computes recall and precision ratio based on phrase boundaries obtained from phrase structure trees (Grishman et al., 1992). This method has a merit in that it can be applicable to various parsing systems, but has a problem in that sometimes it produces evaluation results format variance to human intuition. Sampson proposed an evaluation method reflecting the grammar category information in phrase structure trees and claims that the evaluation results of this method are closer to human ones compared with the boundary-based method (Sampson, 2000). However, Sampson's method has lower applicability since it requires compatibility in grammar categories of the parsing systems in order to compare. In general, every parsing system has its own phrase structure tree and grammar category system depending on its analysis grammar. From this point of view, a new evaluation schema called relational schema is adopted in several evaluation methods (Lin, 1998; Srinivas, 2000; Briscoe et al., 2002).

The relational schema measures the accuracy of syntactic or logical dependency relations between words obtained from phrase structure trees. The extraction of word relation from phrase structure tree is not straightforward and no standard method based on relational schema has been established so far.

As shown above, PDG has both phrase structure trees and dependency trees in the sentence analysis process. Therefore, both phrase-structure-tree-based and dependency-structure-based approaches are applicable to PDG-based systems in principle. Considering the trend toward relational schema in sentence evaluation frameworks and the fact that PDG's final output is dependency structures, a dependency-structure-based approach is adopted for evaluating PDG-based systems.

Ratio of correct dependencies in output dependency structures / trees is used for evaluating dependency analysis systems (Ozeki, 1998; Kudo and Matsumoto, 2005; Harper et al., 1999). This thesis adopts this kind of measures named "arc precision ratio" (APR) and "word dependency precision ratio" (WDPR) as comprehensive evaluation measures for total analysis ability of PDG-based systems. In addition to the comprehensive evaluation measures, this thesis proposes two different kinds of measures, i.e., "possibly correct sentence ratio" (PCSR) for evaluating the system ability to generate the correct hypothesis for the input sentence, and "arc disambiguation precision ratio" (ADPR) for evaluating the system's disambiguation ability.

This thesis focuses on the dependency structure as evaluation target because of the size of preparable data amount of the correct analysis results and preference knowledge as described above. However, evaluation methods are applicable to all dependency structures of the single dependency model. In addition to the comprehensive evaluation measure, this thesis proposes two more measures for evaluating hypothesis generation ability and disambiguation ability since the enhancement of these abilities in natural language analysis systems is a principal target of PDG.

6.1.2 Comprehensive Analysis Ability

This section describes the arc precision ratio (APR) and the arc disambiguation precision ratio (ADPR) which are adopted as comprehensive evaluation measures for PDG.

[**Arc Precision Ratio**]

APR shows the accuracy of output dependency trees as defined below.

$$APR = \frac{Number\ of\ correct\ arcs\ in\ ODT}{Number\ of\ all\ arcs\ in\ ODT}$$

ODT is a set of arcs in the optimum dependency trees for the test sentences. The comprehensive analysis ability of a system is measured by APR ranging from 0 to 1.

Fig.6.1 shows the correct dependency tree CDT and the optimum dependency trees ODT_1,ODT_2 for the example sentence "Time flies like an arrow"[*1]. The APR for this example is 0.6 since there are six correct arcs, i.e., oa1 - oa5 and oa9, exist in ten arcs contained in ODT_1 and ODT_2.

[*1] Scores are not shown in Fig.6.1. The output trees in Fig.6.1 are not the solutions for the DF shown in Fig.3.4

[Word Dependency Precision Ratio]

It is difficult to apply APR to various sentence analysis systems since it requires system-dependent information such as WPP and dependency relation. To avoid this problem, this thesis adopts another measure called “word dependency precision ratio” (WDPR) as a comprehensive evaluation measure for dependency trees with wide applicability. WDPR is the same as APR except that each output arc is judged correct if it has the same dependent and governor words as its corresponding correct arc. WDPR is obtained from the computation algorithm for obtaining APR by simply neglecting the difference of POS and dependency relation name in matching between a correct arc and an output arc. WDPR for the previous example is $8/10 = 0.8$ since two more output arcs, oa6 and oa10, are judged correct in addition to the correct arcs for APR.

6.1.3 Hypothesis Generation and Disambiguation Ability

[Possibly Correct Sentence Ratio]

PDG has the following three functionalities from the viewpoint of the treatment of hypotheses[*2].

(a) Generation of hypotheses for an input sentence (hypothesis generation)
(b) Rejection of hypotheses by constraint knowledge (hypothesis rejection)
(c) Extraction of optimum solutions based on the scoring by preference knowledge (hypothesis selection)

```
Correct Dependency Tree:
 CDT = { arc(sub,[time]-n-0,[flies]-v-1)},   /* ca1 */
         arc(top,[flies]-v-1),[])            /* ca2 */,
         arc(vpp,[like]-pre-2,[flies]-v-1)), /* ca3 */
         arc(pre,[arrow]-n-4,[like]-pre-2),  /* ca4 */
         arc(det,[an]-det-3,[arrow]-n-4) }   /* ca5 */

Output Dependency Trees:
 ODT1 ={ arc(sub,[time]-n-0,[flies]-v-1)},   /* oa1 */
         arc(top,[flies]-v-1),[]),           /* oa2 */
         arc(vpp,[like]-pre-2,[flies]-v-1)), /* oa3 */
         arc(pre,[arrow]-n-4,[like]-pre-2),  /* oa4 */
         arc(det,[an]-det-3,[arrow]-n-4) }   /* oa5 */

 ODT2= { arc(nc,[time]-n-0,[flies]-n-1),     /* oa6 */
         arc(sub,[flies]-n-1,[like]-v-2),    /* oa7 */
         arc(top,[like]-v-2),[]),            /* oa8 */
         arc(det,[an]-det-3,[arrow]-n-4),    /* oa9 */
         arc(obj,[arrow]-n-4,[like]-v-2)  }  /* oa10 */
```

Fig.6.1 CDT and ODTs for the example sentence

[*2] hypotheses here means possible interpretations inherently contained in a sentence

The hypothesis generation is successful if a sentence analysis system can generate a correct interpretation or correct hypothesis as a candidate internally. The hypothesis rejection is successful if a system rejects incorrect hypotheses generated by the hypothesis generation process. The hypothesis selection is successful if a system selects the correct hypothesis from possible hypotheses irrejectable by the constraint knowledge. In PDG, dependency forest is the result of process (a) and (b) in total. Therefore, comprehensive hypothesis generation ability can be measured by checking the existence of the correct dependency tree in the obtained dependency forest. A sentence whose dependency forest contains the correct answer is called a possibly correct sentence, and the number ratio of the possibly correct sentences to those of the whole sentences is called the “possibly correct sentence ratio” (PCSR). PCSR shows the comprehensive hypothesis generation ability of a PDG-based system. On the other hand, the hypothesis selection ability is basically measured by checking the correct arcs in the dependency forest. The next section describes a measure for the hypothesis selection ability.

[**Arc Disambiguation Precision Ratio**]

ADPR measures the disambiguation ability of a PDG-based system. ADPR should reflect the complexity of the disambiguation task. Choosing a correct answer from two candidates is

```
position(a)   : Start Position of Arc a
arc_num(a,S)  : Number of arc s in arc collection S
arc_num_at_position(p,S) : Number of arcs in S, that has the
                           arc start position p
/* step0: Initialize  */
MaxArcScore :=0; /* Max arc score (# of all arc candidates) */
ArcScore = 0;      /* Arc Score (# of arcs) */
/* step1: Get one arc in CDT and calculate score for it  */
foreach onearc in CDT {
{
 /* step2: Correct arc not in DG -> neglect it */
 if( onearc ∉ DG )  { next; }
 /* step3: sp (surface position of onearc)              */
 /*        Num_in_DG(possible arc candidate # at sp)     */
 sp := position(onearc);
 ArcNum_in_DG := arc_num_at_position(sp,DG);
 /* step4: Only one candidate -> neglect */
 if( ArcNum_in_DG == 1 ) { next; }
 /* step5: Compute total possible arc number            */
 MaxArcScore := MaxArcScore + ArcNum_in_DG;
 /* step6: Compute total obtained score (arc #)          */
 CorrectArcRatio :=
   arc_num(onearc,ODTArcs)/arc_num_at_position(sp,ODTArcs);
 OneArcScore := ArcNum_in_DG * CorrecArcRatio;
 ArcScore := ArcScore + OneArcScore;
}
/* step7: Compute Arc Disambiguation Precision Ratio */
ArcSelectionAbilityRatio := ArcScore/MaxArcScore;
```

Fig.6.2 Algorithm for computing ADPR

easier than from ten candidates. This feature is incorporated into ADPR by assigning a score proportional to the number of candidate arcs in the disambiguation task. If the generated dependency forest has no correct arcs, no preference knowledge exists on the basis of which a correct answer can be selected. Conversely, if the dependency forest has no incorrect arcs, a correct answer can be selected on the basis of any preference knowledge. These cases are out of the scope of evaluation of the disambiguation ability and should be omitted in evaluation.

Based on the above considerations, arc disambiguation precision ratio (sometimes called disambiguation precision) is defined as shown in Fig.6.2.

This algorithm inputs a correct dependency tree CDT, an output dependency tree ODT_1 to ODT_n (n is the number of the optimum trees), and a dependency graph DG. Here, the collection of the arcs in ODT_1 - ODT_n is described as $ODTArcs$. Step1 extracts one correct arc *onearc* from CDT. If *onearc* is not contained in DG, it is not a target of evaluation (step2). *onearc* is also neglected when it has no ambiguities (step4). If there is an ambiguity for *onearc*, step5 adds the number of arcs which have a start position *sp* and exist in DG to $MaxSrcScore$ as a score for current *onearc*. In step6, arc correction ratio *CorrectArcRatio* for current *onearc* is computed and the score for *onearc* is computed as the product of $MaxSrcScore$ and *CorrectArcRatio*. This score is added to total arc score $ArcScore$. Arc disambiguation precision ratio is calculated in step7 as the ratio of total arc score $ArcScore$ to the maximum arc score $MaxArcScore$. Arc disambiguation precision ratio varies from 0 to 1.

Fig.6.1 shows a correct dependency tree and an output dependency tree and Fig.6.3 shows DG for the example sentence.

In step1 of Fig.6.2, the first arc ca1 in Fig.6.1 is set to *onearc*. Step2 sets a start position of node "[time]-n-0," i.e., 0, to *sp*. Step3 does not neglect *onearc* since it is in DG. In step4, *onearc* is judged as a target of evaluation since $arc\_num\_at\_position(0, DG) = 3$. Step5 sets $MaxArcScore$ to 3. Step6 obtains $CorrectArcRatio = 1/2 = 0.5$, $OneArcScore = 3 * 0.5 = 1.5$; then $OneArcScore$ is computed as 1.5. The computation continues in a similar way. Correct arc ca5 is neglected since DG has only one arc pa11 on start position 3. The computation results

```
Dependency Graph:
 DG = { arc(nc,[time]-n-0,[flies]-n-1),      /* pa1 */
        arc(sub,[time]-n-0,[flies]-v-1),     /* pa2 */
        arc(top,[time]-v-0,[]),              /* pa3 */
        arc(obj,[flies]-n-1,[time]-v-0),     /* pa4 */
        arc(sub,[flies]-n-1,[like]-v-2),     /* pa5 */
        arc(top,[flies]-v-1,[]),             /* pa6 */
        arc(top,[like]-v-2,[]),              /* pa7 */
        arc(npp,[like]-pre-2,[flies]-n-1),   /* pa8 */
        arc(vpp,[like]-pre-2,[flies]-v-1),   /* pa9 */
        arc(vpp,[like]-pre-2,[time]-v-0),    /* pa10 */
        arc(det,[an]-det-3,[arrow]-n-4),     /* pa11 */
        arc(obj,[arrow]-n-4,[like]-v-2),     /* pa12 */
        arc(pre,[arrow]-n-4,[like]-pre-2) }  /* pa13 */
```

Fig.6.3 Dependency Graph for the example

are as follows;

Arc	sp	Arcnum	CrctArcRto	ArcScr
ca1	0	3	0.5	1.5
ca2	1	3	0.5	1.5
ca3	2	4	0.5	2.0
ca4	4	2	0.5	1.0
total		12		6.0

Arcnum, CrctArcRto, ArcScr correspond to *Arcnum_in_DG*, *CorrectArcRatio*, *OneArcScore*, respectively.

The final value of *MaxArcScore* is 12 (sum total of *Arcnum_in_DG*) and that of *ArcScore* is 6 (sum total of *OneArcScore*). Therefore, *ArcSelectionAbilityRatio*=6/12 =0.5. In this example, every *arc_accuracy* is 0.5. If *CorrectArcRatio* of ca3 is 1 then *ArcSelectionAbilityRatio* is 8/12= 0.67, whereas if *CorrectArcRatio* of ca4 is 1 then *ArcSelectionAbilityRatio* is 7/12 = 0.58. This shows that ADPR reflects the difficulty of the arc disambiguation task.

6.1.4 Environment of Experiment for Evaluation Measures

This section and the next section describe some experimental results showing the behaviors of the proposed evaluation measures with respect to some parameters such as sentence length, preference knowledge, grammar coverage and so on.

An English text corpus, correct dependency trees, PDG grammar and dictionary and preference knowledge are prepared for evaluating the proposed evaluation methods. Preference knowledge here is a WPP frequency in the corpus. Preference score PS(N) for node N (WPP) is defined as follows.

$$\mathrm{PS}(N) = \log(X)/\log(\mathrm{MF}) \quad (0 \leq \mathrm{PS}(X) \leq 1)$$

where X is the frequency of N in the corpus, MF is the maximum frequency of WPPs in the corpus. The optimum tree has the highest total preference scores among the well-formed dependency trees for a given sentence.

The text corpus consists of technical documents containing around 620,000 sentences (4,630,000 words[*3]) in total. In order to prepare a large amount of correct dependency trees and WPP frequency data, an existing sentence analysis system (called the oracle system) is used as the generator of those data. The oracle system (Amano et al., 1989) is a real world rule based system with a long development history, which is currently used for translating technical documents, web pages, mail texts and so on.

Data filtering is applied to the original text corpus since it contains many tables, indices characteristic of technical manuals and many ungrammatical sentences originated from typing and sentence extraction errors. The correct dependency trees are not obtainable for the sentences

[*3] counted by unix "wc" command

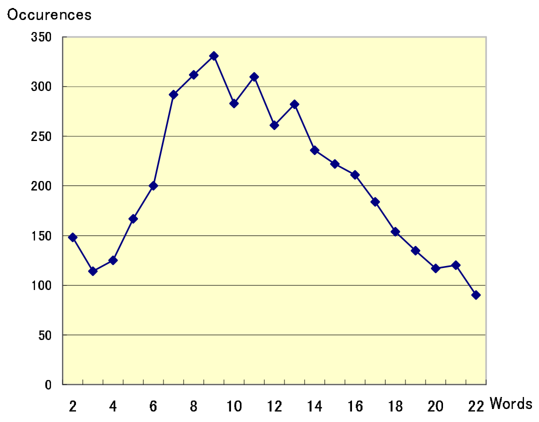


Fig.6.4 Distribution of sentence length for sentences in the open data

which are not parsable by the oracle system. The following sentences are removed from the original corpus.

(a) Unparsable sentences (around 71,000 sentences)
(b) Parsable sentences whose last character is not a period (around 204,000 sentences)
(c) Parsable and period-ending sentences whose first character is not a capital letter (around 220,000 sentences)

The remaining corpus has 125,320 sentences (1,844,758 words). The oracle system generates the correct dependency trees and WPPs for these sentences. This corpus is divided into open data (8,605 sentences, 126,684 words) and close data (116,715 sentences, 1718074 words). The open data is used for evaluation test set and the close data is used for preference knowledge resource, i.e., the source of WPP frequencies. The number of extracted WPPs is 1,869,000 (44,470 kinds of WPP)[*4]. Fig.6.4. shows the distribution of word length of sentences in the open data. In order to see a brief accuracy of the oracle system, 136 sentences are selected randomly but with similar distribution shown in Fig.6.4 from a set of sentences which are parsable using the basic grammar described below[*5]. The APR for this sentence set with respect to human analysis results is 97.2%. Therefore, the output of the oracle system is a good approximation of human correct data.

Two PDG grammars called a basic grammar (Grammar-B) and a mini grammar (Grammar-M) are prepared. The basic grammar consists of basic grammar rules which covers sentence variations such as noun/verb/adjective/adverbial/prepositional phrases, simple/complex/compound sentences, relative/subordinate clauses and Onions' 5 sentence patterns. The basic grammar does not accept insertion, omission, inversion and idiomatic structures (ex. not only .. but also ..). The basic grammar is superior to the mini grammar in the generation and constraint abil-

[*4] The number of WPPs here is not the same as that of words, since WPP is counted based on the result of morphological analyzer

[*5] Since unparsable sentences have no output, they are neglected in the succeeding evaluation experiments. Some extra method is required for obtaining partial phrase structure trees for unparsable sentences.

ities. The basic grammar has higher generation ability compared with the mini grammar since it accepts syntactic patterns with richer phrase variations (numeric/symbol expressions in noun phrase, double quote expression, optional expressions etc.), coordinations (noun phrase, adjective phrase, adverb phrase etc.), greater number of optional elements (prepositional phrases, adverbs etc.) and so forth. The basic grammar has richer and more precise constraints, such as additional number agreement[*6] as found in "these desks," sequence of tenses, sub-categorization frames of verbs, structural constraint based on morphological features and so on. The basic grammar consists of 907 CFG rules whereas the mini grammar consists of 377 CFG rules. These grammars produce the same type of dependency structures as the oracle system. The morphological analyzer is shared with the PDG system and the oracle system.

6.1.5 Evaluation Experiment for Evaluation Measures

An evaluation experiment for the open data and basic grammar is performed using a prototype PDG system implemented in Prolog. The test sentences containing more than 22 words are neglected due to the limitation of Prolog system resources. 4334 sentences out of 6882 test sentences are parsable by Grammar-B. The parse success ratio is 63%. Without applying special treatment such as construction of the whole phrase structure tree from partial phrase structure trees, unparsable sentences (2584 sentences) are simply neglected in this experiment.

[Comparison between APR and WDPR]

Fig.6.5 shows the comparison of APR and WDPR with/without the preference knowledge (PK). Results obtained without the preference knowledge are called baseline performance in this experiment. In total, APR with PK (AK) is 85.1%, APR without PK (baseline) (AB) is 77.8%, WDPR with PK (WK) is 87.9% and WDPR without PK (DB) is 81.8%. Arc precision and word dependency precision are equivalent measures and have the same fluctuation for sentence length.

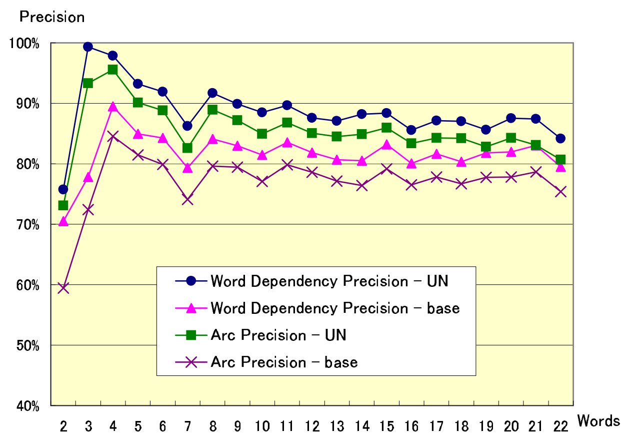


Fig.6.5 Comparison of WDPR and APR

[*6] Basic number agreement constraint is introduced into both basic grammar and mini grammar.

Only a few syntactic variations exists for very short sentences. This seems to cause low APR for sentence length 2 and 3. AK and AB have an average gap of 7.3%, whereas WK and WB have one of 6.1%. WDPR has a greater gap than APR. AK and WK have an average gap of 2.3%, whereas AB and WB have one of 4.0%. Fewer gaps is observed when the preference knowledge is utilized.

In this experiment, English documents are used. English is a structural language where word order has important roll in deciding functional relations between words. Therefore, word dependency may have high correlation with functional relations between words. On the other hand, for example in Japanese, word order is less important to decide functional relation between words. The behavior of APR and WDPR may be different in Japanese.

[Comparison of APR and ADPR with respect to preference knowledge]

Fig.6.6 shows the comparison of APR and ADPR with/without the preference knowledge (PK). In total, APR with PK (AK) is 85.1%, APR without PK (baseline) (AB) is 77.8%, ADPR with PK (DK) is 65.8% and ADPR without PK (DB) is 42.0%. Although the preference knowledge is simple, both measures show significant improvement by applying PK. For example, sentence "The integer constant for the sentence buffer." has two readings corresponding to "The integer *constant/n* for ..." (correct) and "The integer *constant/adj* for ..." (incorrect). In this case, the correct interpretation is selected as the optimum solution since WPP *constant/n* has larger frequency than that of *constant/adj*.

The fluctuations of AK and DK show overall mutual relation. But a few exceptions are seen; e.g. in word length 14 and 15 where AK decreases in spite of increase of DK. This is reasonable since ADPR measures a disambiguation ability while APR measures a comprehensive sentence analysis ability including the disambiguation ability as described above.

DB is almost constant with respect to sentence length. This means the difficulty of the disambiguation task (number of ambiguous arcs) does not show remarkable increase with respect to sentence length. In contrast, DK decreases as sentence length increases. This means the current strategy for applying preference knowledge provides less performance for longer sentences.

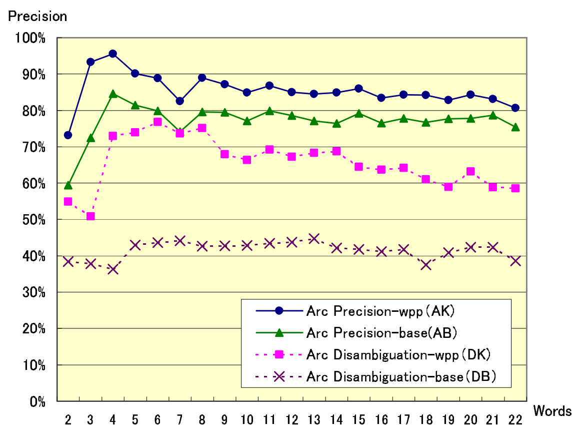


Fig.6.6 Comparison of APR and ADPR with respect to preference knowledge

[Comparison of APR and ADPR with respect to PCSR]

There are 3224 possibly correct sentences out of 4334 parsable sentences. The PCSR is 74.4%. To clarify the influence of PCSR, data for only the possibly correct sentences, i.e., PCSR is 100%, is estimated. Fig.6.7 shows the comparison of APR and ADPR obtained from the data with 100%-PCSR (C: Correct answer contained) and 74%-PCSR (A: All sentences). In total, APR for 100%-PCSR (AC) is 90.4%, APR for 74%-PCSR (AA) is 85.1%, ADPR for 100%-PCSR (DC) is 85.1% and ADPR for 74%-PCSR (DA) is 42.0%. Very large improvement of APR and ADPR is achieved by increasing the APR of the target sentence collection.

Comparing DC and DA, ADPR seems to be independent of PCSR and decreases as target sentence length increases. In contrast, APR seems to be dependent on PCSR, since AC (100%-PCSR) is almost constant for sentences with more than 6 words while AA (74%-PCSR) seems to decrease as sentence length increases. This suggests that PCSR has a relation with the decrease of APR against sentence length and the improvement in comprehensive hypothesis generation ability is effective for keeping high APR for longer sentences.

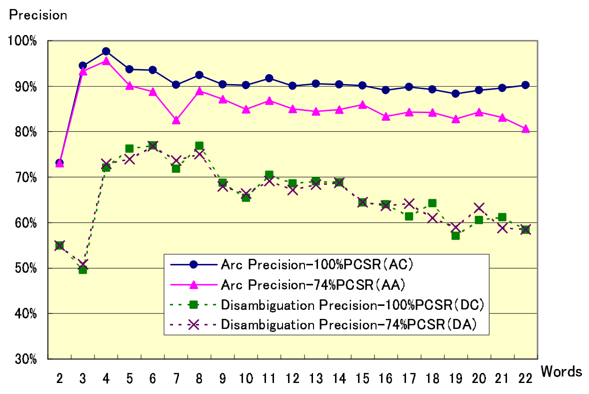


Fig.6.7 Comparison of APR and ADPR with respect to PCSR

[Comparison of APR and ADPR (Coverage of Grammar)]

The experiment using the basic grammar (Grammar-B) and the mini grammar (Grammar-M) shows that Grammar-B has 4334 parsable sentences (parse success ratio 63.0%) containing 3224 possibly correct sentences (PCSR 74.4%), and Grammar-M has 3139 parsable sentences (parse success ratio 45.6%) containing 2135 possibly correct sentences (PCSR 68.0%). Fig.6.8 shows the comparison of APR and ADPR obtained from Grammar-B and Grammar-M. In total, APR for Grammar-B (AB) is 85.1%, APR for Grammar-M (AM) is 83.4%, ADPR for Grammar-B (AB) is 65.8% and ADPR for Grammar-M is 68.9%.

DB and DM have almost the same values with some fluctuations for sentences with length 6 to 16. In contrast, AB always has slightly higher values compared with those of AM in the same sentence length range. This suggests the reasonable assumption that ADPR is basically independent of grammar, whereas ACR is dependent on grammar. If ADPR is independent of grammar, the decrease of ADPR against sentence length should mainly be caused by the current strategy for applying preference knowledge.

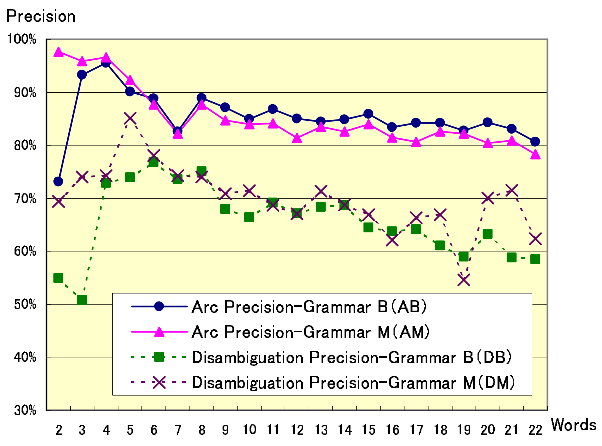


Fig.6.8 Comparison of APR and ADPR with respect to difference of grammar

6.2 System Evaluation with respect to Preference Knowledge

The evaluation of the prototype PDG system described above is done with respect to the combination of various kinds of preference knowledge described in Chapter 5. The current implementation incorporates four types of preference scores, i.e., the unary node score based on WPP frequency, the unary arc score based on arc (i.e., dependency piece) frequency, the binary node score based on WPP bigram frequency and the binary arc score based on arc co-occurrence frequency. These preference sources are expressed by UN (unary node), UA (unary arc), BN (binary node) and BA(binary arc), respectively. The combination of preference knowledge is represented by '+' operator. For example, PDG system using the UN and UA is written as 'UN+UA'. Empty preference knowledge, i.e., the baseline system performance, is represented by ϕ symbol. The combination of the measurement and preference knowledge is represented by "/." For example, the measurement "APR (arc precision ratio)" for the knowledge combination "UN+UA" is written as "APR/UN+UA." The test sentences and the corpus are the same as those described in Section 6.1.4.

The baseline evaluation is done with no preference knowledge. This means all well-formed dependency trees are the optimum solutions for the input sentence. In the following experiments, the number of solutions in baseline execution is limited up to 100. This is because the number of the optimum solutions grows exponentially as the sentence length increases and a whole set of dependency trees cannot be obtained due to computational resource limitation.

There are a lot of combinations of preference knowledge. The following section basically reports the baseline (ϕ) performance, single knowledge (UN, UA, BN and BA) performances and the three combined knowledge (UN+UA, UN+BN and UA+BN) performances which have shown good performance in the experiment.

6.2.1 Evaluation of Comprehensive Sentence Analysis Ability

Fig.6.9 shows the comparison of APR for whole sentences with respect to the combination of preference knowledge sources. In total, as shown in the figure, the average of APR/ϕ, APR/UN, APR/UA, APR/BN, APR/BA, APR/UN+UA, APR/UN+BN and APR/UA+BN are 77.4%, 84.6%, 87.4%, 83.6%, 80.4%, 88.3%, 84.3% and 88.3%, respectively. The PCSR for the test sentences is 74% as described above. APR for the whole sentences shows APR-74PCSR, i.e., APR for 74% PCSR sentences.

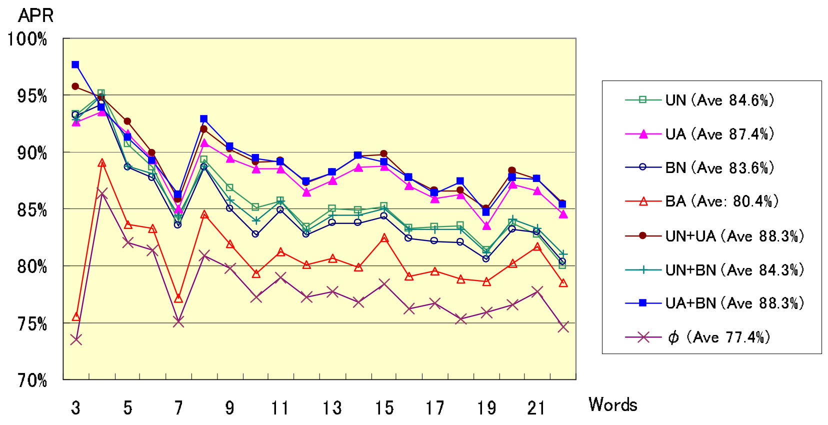


Fig.6.9 Comparison of APR w.r.t. preference knowledge (whole sentences)

UA provides the best performance 87.4% in case of single knowledge use. BA provides the worst performance 80.4% but it still outperforms 3.0% compaired with baseline ϕ. The preformances of binary relation BN and BA are relatively low. This may be caused by the data sparseness problem, i.e., the inadequecy of the training corpus data. Experiment with much larger training corpus is one of the future works. The combination of the UN and UA gives better performance compared with those obtained by independent knowledge sources.

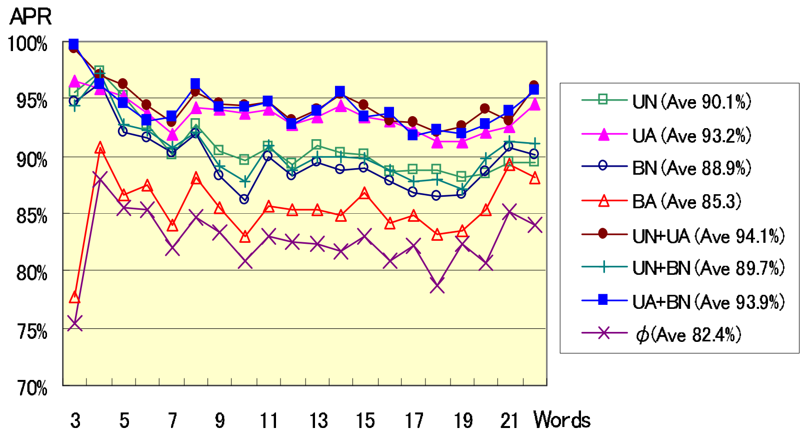


Fig.6.10 Comparison of APR w.r.t. preference knowledge (100% PCSR)

Fig.6.10 shows the results for only the possibly correct sentences, i.e., PCSR is 100%. In total, as shown in the figure, the average of APR-100PCSR/ϕ, APR-100PCSR/UN, APR-100PCSR/UA, APR-100PCSR/BN, APR-100PCSR/BA, APR-100PCSR/UN+UA, APR-100PCSR/UN+BN and APR-100PCSR/UA+BN are 82.4%, 90.1%, 93.2%, 88.9%, 85.3%, 94.1%, 89.7% and 93.9%, respectively. The differences between APR-74PCSRs and APR-100PCSR are within the range from 4.9% to 5.9%. APR-100PCSR is more constant with respect to sentence length compared with APR-74PCSR.

6.2.2 Evaluation of Disambiguation Ability

Fig.6.11 shows the comparison of ADPR for whole sentences with respect to the combination of preference knowledge sources. In total, as shown in the figure, the average of ADPR-74PCSR/ ϕ, ADPR-74PCSR/UN, ADPR-74PCSR/UA, ADPR-74PCSR/BN, ADPR-74PCSR/BA, ADPR-74PCSR/UN+UA, ADPR-74PCSR/UN+BN and ADPR-74PCSR/UA+BN are 42.7%, 65.0%, 74.6%, 62.1%, 51.6%, 77.6%, 64.6%, 77.6%, respectively. The combination of the preference score UN and UA gives 12.6% and 3.0% improvements for ADPR-74PCSR/UN and ADPR-74PCSR/UA, respectively.

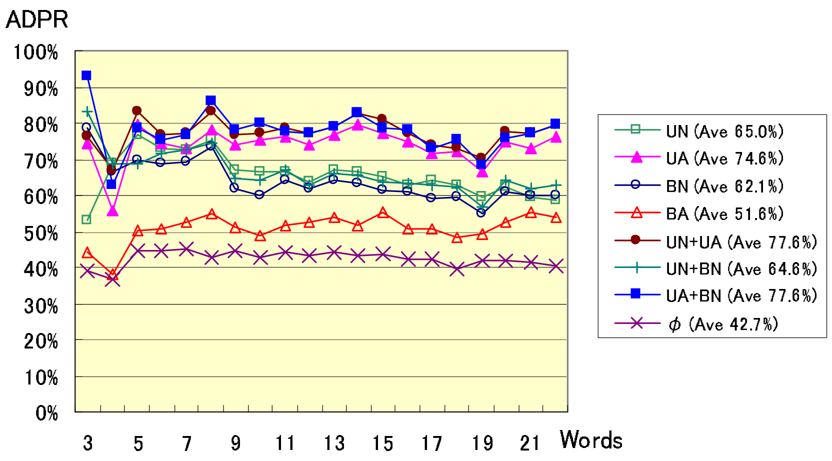


Fig.6.11 Comparison of ADPR w.r.t. preference knowledge (whole sentences)

Fig.6.12 shows the result for ADPR-100PCSR. In total, as shown in the figure, the average of ADPR-100PCSR/ϕ, ADPR-100PCSR/UN, ADPR-100PCSR/UA, ADPR-100PCSR/BN, ADPR-100PCSR/BA, ADPR-100PCSR/UN+UA, ADPR-100PCSR/UN+BN and ADPR-100PCSR/UA+BN are 42.0%, 66.1%, 76.3%, 62.3%, 50.7%, 79.1%, 65.2%, 78.5%, respectively. In contrast to the APR, PCSR has no strong effect on ADPR as described in Section 6.1.5. The differences between ADPR-74PCSRs and ADPR-100PCSRs are within the range from -1.5% to 0.9%. This difference is much smaller compared with the differences between APR-74PCSRs and APR-100PCSRs. This result suggests that ADPR is almost independent of PCSR.

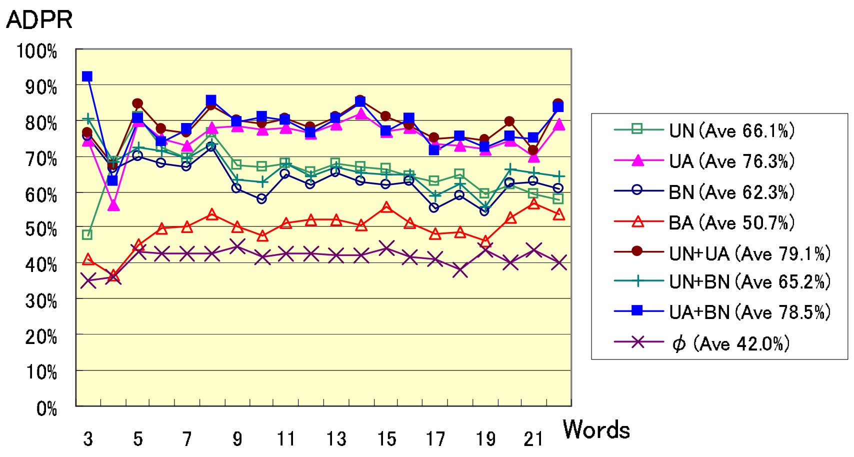


Fig.6.12 Comparison of ADPR w.r.t. preference knowledge (100% PCSR)

6.2.3 Evaluation of Selectivity Performance

[Average Optimum Solution Number]

Fig.6.13 shows the comparison of the average optimum solution number (AOSN) for whole sentences with respect to the combination of preference knowledge sources, i.e., UN, UA, BN, BA and UN+UA [*7]. In total, as shown in the figure, the average of AOSN/UN, AOSN/UA, AOSN/BN, AOSN/BA and AOSN/UN+UA are 5.1, 1.3, 5.3, 5.1 and 1.1, respectively. UN, BN and BA has clear growth of the AOSN with respect to sentence length whereas UA and UN+UA have very small growth in AOSN. The combination of preference knowledge works to decrease AOSN.

The performances of BN and BA may be improved by increasing the training corpus size to avoid the data sparseness problem. This is one of the future works.

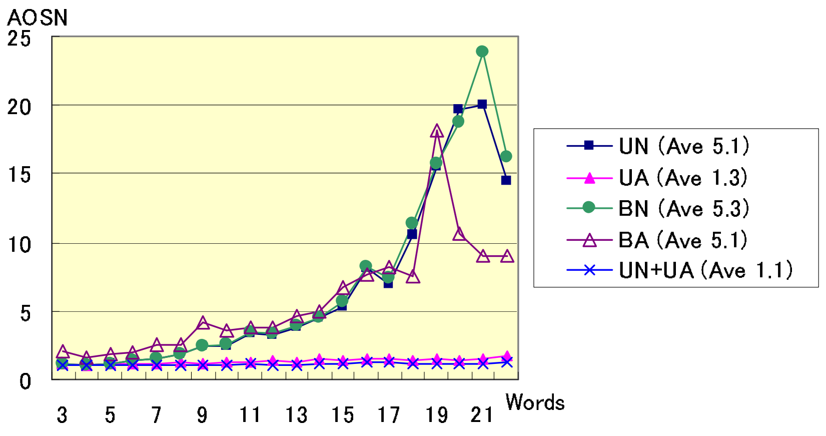


Fig.6.13 Comparison of average optimum solution number w.r.t. preference knowledge

[*7] The baseline data is not shown in the figure because the number of solutions is limited up to 100 in baseline execution.

[**Average Expanded Problem Number in Total**]

Fig.6.14 shows the comparison of the average expanded problem number (AEPN) for whole sentences with respect to the combination of preference knowledge sources, i.e., UN, UA, BN, BA and UN+UA. In total, as shown in the figure, the average of AEPN/UN, AEPN/UA, AEPN/BN, AEPN/BA and AEPN/UN+UA are 18.1, 3.7, 15.8, 15.2 and 3.3, respectively. This result shows the same tendency with the AOSN. In the AEPN evaluation, BN outperformed UN in contrast to the AOSN evaluation. AEPN/ϕ is not measured because the number of solutions is limited up to 100 in baseline execution.

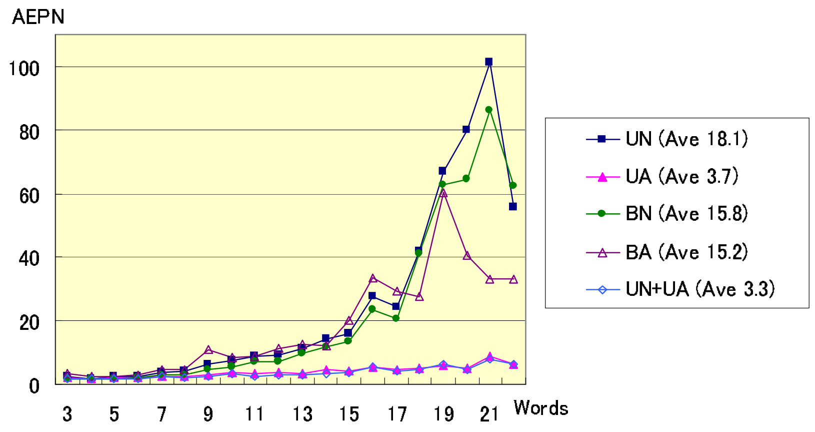


Fig.6.14 Comparison of average expanded problem number w.r.t. preference knowledge

[**Average Expanded Problem Number for the First Optimum Solution**]

Fig.6.15 shows the comparison of the average expanded problem number for the first optimum solution (AEPNF1OS) with respect to the combination of preference knowledge sources, i.e., UN, UA, BN, BA and UN+UA. In total, as shown in the figure, the average of AEPNF1OS/UN, AEPNF1OS/UA, AEPNF1OS/BN, AEPNF1OS/BA and AEPNF1OS/UN+UA are 2.3, 1.5, 1.6, 1.1 and 1.6, respectively. AEPNF1OS/ϕ (baseline) is always 1.0 because all solutions are optimum solutions.

AEPNF1OS/BA has the least value 1.1 which is close to the baseline value 1.0. This reflects the fact that BA has the biggest number of average optimum solutions as shown above. BA needs big amount of training corpus to obtain enough selectivity performance by avoiding the data sparseness problem. AEPNF1OS/UN has the biggest value 2.3. This means that a set of the most frequent WPPs for the words in a sentence is not necessarily correspond to the well-formed dependency tree. The combination of UN and UA (UN+UA) requires a bit more computation for obtaining the first optimum solution compared with UA due to the influence of UN.

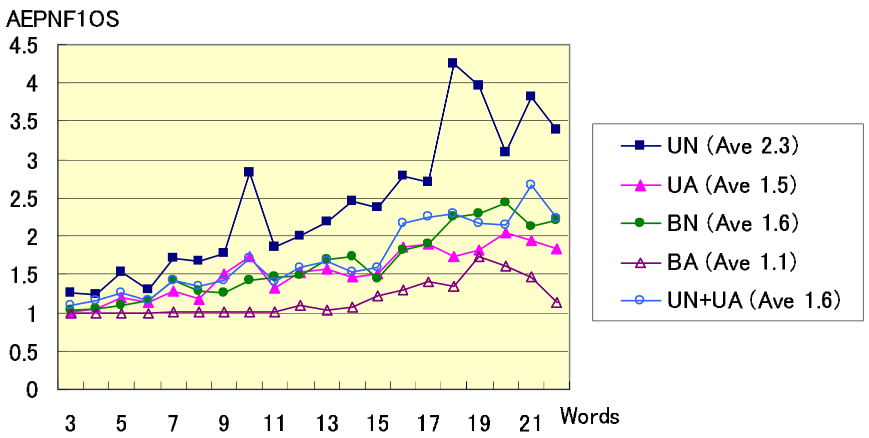


Fig.6.15 Comparison of average expanded problem number for the first optimum solution w.r.t. preference knowledge

6.3 Concluding Remarks for the Experiments

This section discussed the sentence analysis ability based on the PDG framework and proposed three kinds of evaluation measures for dependency structures. The arc precision ratio (APR) and the word dependency precision ratio (WDPR) measure comprehensive analysis ability. The possibly correct sentence ratio (PCSR) and the arc disambiguation precision ratio (ADPR) measures a part of analysis abilities, i.e., comprehensive hypothesis generation ability and hypothesis selection ability, respectively.

The experiment using English technical documents suggests that ADPR is independent of PCSR and the grammar coverage. This means ADPR has good characteristics for measuring the ability of preference knowledge and application strategies. The current simple preference knowledge and strategy shows the decrease of ADPR against sentence length.

From the experiment reported in this thesis, extending the coverage of the basic grammar is a basic task for improving the sentence analysis ability and improvement of PCSR and preference knowledge application strategy are effective for improving the system performance for longer sentences.

The evaluation measures described in this section are applicable not only to PDG-based systems but also to other dependency structure based sentence analysis systems. In addition, by ignoring arc labels and POS categories as described in 6.1.2, PCSR and ADPR can be revised to the sentence measures suitable for comparing different sentence analyzers with different node category and arc label systems.

Section 6.2.1 has shown that the evaluation of comprehensive sentence analysis ability is improved by introducing every kind of preference knowledge. Among various kinds of the combinations of preference knowledge, the best APR 88.3% is obtained by UN+UA and UA+BN in this experiment. This is 10.9% improvement compared with the baseline performance 77.4% APR. As shown in section 6.1, the disambiguation ability is well measured or compared by using

ADPR instead of APR. The ADPR of both UN+UA and UA+BN is 77.6% which shows great improvement compared with baseline performance ADPR/ϕ 42.7%.

In comparison between UN+UA and UA+BN, ADPR-100PCSR/UN+UA (79.1%) is a little bit better than ADPR-100PCSR/UA+BN (78.5%). Furthermore, UN+UA is superior to UA+BN because the unary model, i.e., UN+UA, requires less computational resource than the binary model, i.e., UA+BN, in general. The combination of UN and UA provides the best performance in this experiment. However, an experiment using big amount of training corpus should be done for the binary knowledge to obtain enough performance by avoiding the data sparseness problem in future.

Section 6.2.3 has shown that the AOSN, AEPN and AEPNF1OS of UN+UA is 1.1, 3.3, 1.6, respectively. These are very small and show the very good selectivity performance of UN+UA.

Chapter 7

Future Work

7.1 Development of Real-world PDG System

The current PDG system is a Prolog-based prototype system aimed at the feasibility study of the PDG framework. The research and development of a PDG system applicable to real-world applications is one of the important future works.

Improvements in the accuracy and efficiency of the PDG system are expected by the enhancements of the grammar description framework, such as the introduction of more detailed conditions based on feature descriptions and non-obligatory constituent description into a grammar rule. The introduction of non-obligatory constituent descriptions reduces not only the number of edges generated during a parse process but also the number of equivalent arcs in the generated initial dependency forest. The generated equivalent arcs can also be reduced if sharable arcs are identified before parsing by applying the pre-analysis of the grammar. This is an interesting research topic not only for building an efficient system but also for understanding the relation (or exploring the equivalence) between constituency and dependency. In conjunction with these methodological improvements, system implementation using programming languages like C and C++ should be conducted for the real-world PDG system along with the development of the PDG grammar.

The current prototype PDG system adopts a heuristic approach instead of a learning approach for tuning the scoring parameters. Learning technologies have been one of the most advanced areas in natural language processing for a number of years and several excellent learning methods based on annotated corpora have been proposed. In addition to the generative learning model, the discriminative learning model, which can treat the structural parameters based on the entire sentence structure, is studied in detail (McDonald et al., 2005). The introduction of such learning mechanisms in the scoring process is one of the important and promising future works to obtain the best accuracy using the PDG framework.

7.2 Research on Semantic Structure

Semantic processing is a very important but difficult NLP research topic which requires considerable research. There is no common consensus on the representation scheme for semantic sentence structures. MTT adopts the semantic graph structure representing predicate argument relations as its basic semantic representation. This kind of deep predicate argument relation will be necessary for properly representing the meanings of various sentences.

One simple but basic extension of PDG framework toward the semantic layer analysis is to introduce the semantic dependency tree as a kind of semantic layer sentence interpretation. The semantic dependency tree consists of concept nodes and arcs labeled with semantic dependency relations. The semantic dependency tree is a simple but natural expansion of the syntactic dependency tree which is in the current uppermost level of the PDG architecture. Each WPP node has some corresponding semantic nodes (concepts) and each syntactic arc has corresponding semantic arcs (semantic roles). The possible construction of the semantic dependency tree should be guided by some lexicalized predicate-argument information or the case frame structure as introduced by the semantic dependency graph (Hirakawa, 2002).

Fig.7.1 shows PDG model extended to the semantic dependency level. A semantic dependency forest is used to represent a set of semantic dependency trees that represent the semantic interpretations of a sentence. The packed shared data structure for a set of semantic dependency trees is a semantic dependency forest. The semantic dependency forest is expected to be obtained from a (syntactic) dependency forest by two kinds of semantic expansions, i.e., semantic node expansion and semantic arc expansion. Fig.7.2 shows a conceptual example of semantic expansion for the Japanese sentence “Kanojo no Me wa Ookii (彼女の目は大きい)” (Her eyes are

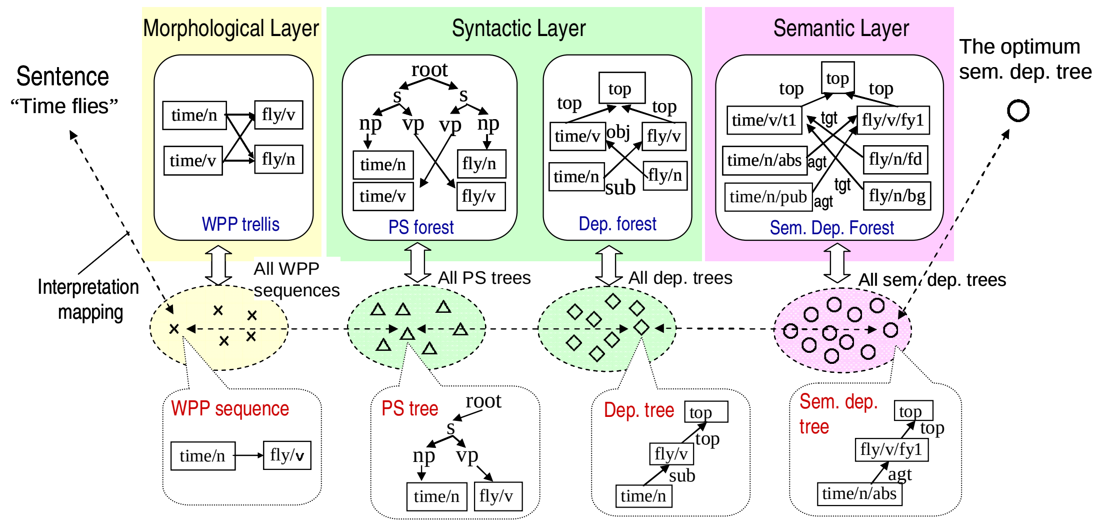


Fig.7.1 Extension of PDG to Semantic Dependency Graph

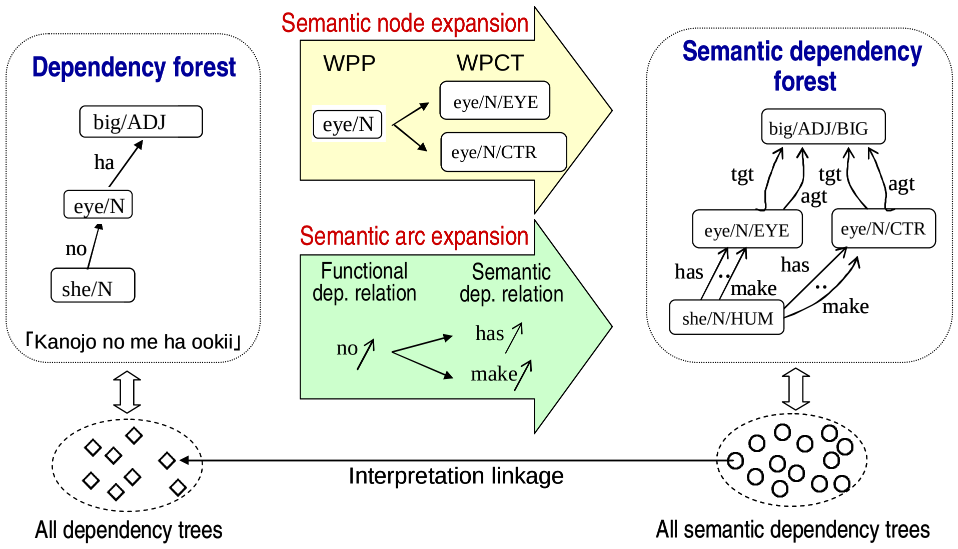


Fig.7.2 Semantic expansion process

big). By applying semantic node expansion, the WPP nodes in a (syntactic) dependency forest are expanded to concept nodes. The WPP node labeled "eye/N" in the figure corresponds to the word "Me (目)" (eye) in Japanese, which has two meanings corresponding to "eye" and "center." This WPP node is expanded to two concept nodes with labels "eye/N/EYE" and "eye/N/CTR." The lexical concepts can be provided by concept dictionaries such as the EDR dictionary (EDR, 1996), Nihongo-Goi-Taikei (Ikehara, 1999), and WordNet (Fellbaum, 1998).

The semantic arc expansion operation generates more than one semantic arc from each (syntactic) dependency arc in the (syntactic) dependency forest. One (syntactic) dependency arc labeled by the Japanese particle "no (の)" can represent various semantic relations such as "possession," "creator," "agent," and "target" between two nouns including verbal nouns. In Fig.7.2, the syntactic arc labeled with "no" is expanded to two semantic dependency arcs labeled with the semantic dependency relation "has" (possessive) and "make" (creator). The expansion from a syntactic dependency relation to semantic dependency relations is performed by consulting a mapping table which defines the mapping between them. Provided that the semantic expansion generates M concept nodes and N semantic dependency arcs from one WPP node and one syntactic dependency arc, respectively, the expanded semantic forest has around M times nodes and $M^2 * N$ times arcs as compared to the syntactic dependency forest. Thus, the combinatorial explosion in the size of the semantic dependency forest through the semantic expansion is suppressed because the semantic dependency forest is also a packed shared data structure.

It is obvious that the interpretation mapping exists between the (syntactic) dependency forest and the semantic dependency forest. Therefore, this framework satisfies the requirements of the MPDC model, because the semantic expansion maintains the mapping between nodes and arcs in the dependency forest and those in the semantic dependency forest. The scoring and the optimum search methods are applicable to both the syntactic and semantic dependency forests.

Needless to say, the approach described in this section is a very rough approximation and simply shows the research direction toward semantic processing. This requires intensive research from both linguistic and computational perspectives.

7.3 Bidirectional Model of PDG

In this thesis, PDG focuses on the sentence analysis direction. However, its basic framework, i.e., the MPDC model, is inherently bidirectional as well as it is true in MTT. For example, the partial structure mapping rule in Section 3.4.2 simply describes a mapping between a partial phrase structure and a partial dependency structure and therefore it can be used bidirectionally. The generation of a phrase structure from a dependency structure is an interesting future research topic.

Researches on the equivalence between the phrase structure grammar and the dependency grammar have not succeeded in showing the strong equivalence between CFG and Tesniere model DG under the equivalence criteria "ramification" as described in Chapter 1. If a proper formal dependency grammar framework based on the partial structure mapping is established, it may be possible to discuss the descriptive power and/or the equivalence of the phrase structure grammar and the dependency grammar based on the new framework. This would make clearer and deeper understanding for both of the two major syntactic representations and the relation between them.

Chapter 8

Conclusion

This thesis proposed a novel dependency analysis framework called the "preference dependency grammar (PDG)," which utilized the two major syntactic representations, i.e., the phrase structure and the dependency structure to obtain the benefit from both representations. Based on the discussion on the multilevel model with respect to the roles of preference and constraint knowledge, PDG adopts the three level MPDC architecture (multilevel system which adopts packing method) with two intermediate levels (morphological structure and phrase structure) and the uppermost level (dependency structure). In this architecture, the phrase structure level (CFG rules) works as a filter for the dependency structure level. This suppresses the magnification of the search space and enables PDG to include full POS ambiguities at dependency level.

In PDG design architecture, the higher description ability of the dependency level data structure is required because the uppermost level is the basis of the knowledge integration in the MPDC model. In order to realize PDG architecture, two core technologies, i.e. a new data structure "dependency forest" and a new algorithm "graph branch algorithm" are proposed in this thesis. The dependency forest fulfils the multilevel model mapping condition required for the MPDC model. The completeness and soundness of the dependency forest with respect to the phrase structure forest is assured by this thesis.

The dependency forest is a data structure with a high descriptive ability to integrate the preference and constraint knowledge by providing two matrices, i.e., the preference matrix and the constraint matrix, which represent the arbitrary arc co-occurrence preferences and constraints, respectively. This thesis proposed a new algorithm called the "graph branch algorithm" that searches the optimum well-formed dependency tree in a dependency forest based on the branch and bound principle. By adopting these data structures and algorithms, PDG enables the proper treatment of the single valence occupation constraint and the non-projective dependency structure, which were not handled properly by the traditional methods. The descriptive power of the dependency forest for ambiguous constructions is examined by using the experimental grammar with the rules that generate typical types of syntactic ambiguities as well as an non-projective construction.

In addition to the arc precision ratio (APR) (measure for the comprehensive sentence analysis ability), this thesis proposed two new evaluation measures for dependency-based NLA systems to measure the performances of their preference and constraint knowledge. The possibly correct sentence ratio (PCSR) measures the hypothesis generation ability, i.e., the performance of the constraint knowledge including the generation knowledge. The arc disambiguation precision ratio (ADPR) which measures disambiguation ability, i.e., the performance of the preference knowledge. This thesis reported an experimental result for checking these measures using the PDG prototype system. The disambiguation ability is well measured or compared by using ADPR instead of APR.

This thesis described the experimental results using the PDG prototype system with the prototype basic English grammar. Four types of preference knowledge (the WPP unigram frequency, the WPP bigram frequency, the unary dependency frequency and the binary dependency frequency) are extracted from the dependency tree corpus obtained by the oracle system (existing machine translation system). The evaluation of comprehensive sentence analysis ability is improved by introducing every kind of preference knowledge. Among various kinds of the combinations of preference knowledge, the best APR 88.3% is obtained by UA+UN (unary arc and unary node scores) and UA+BN (unary arc and binary node scores) in this experiment. This is 10.9% improvement compared with the baseline performance 77.4% APR.

This thesis introduced the foundations of PDG which utilizes the two major syntactic representations and showed its feasibility. There remain a lot of future works in PDG research toward real-world NLP applications and semantic analysis.

Appendix A

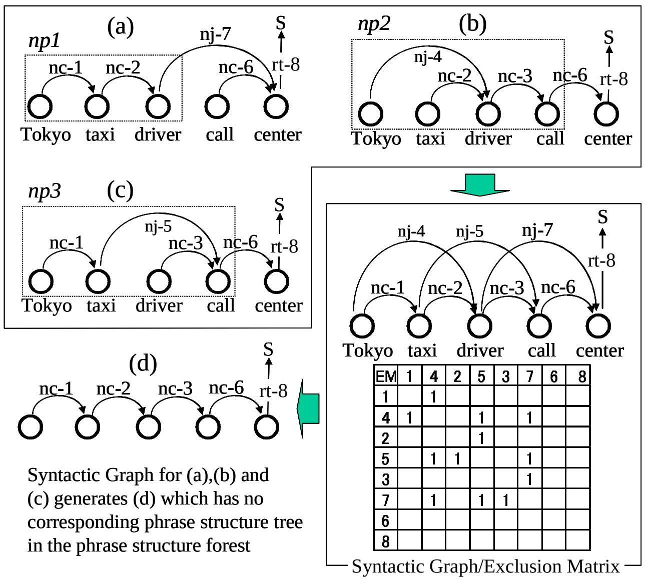
Problem in the Syntactic Graph

Consider the parsing of “Tokyo taxi driver call center” using the following grammar rules and lexicons.

```
[Grammar Rules]
  np/NP   → npc/NP                   : []
  npc/Nb  → np1/NP1,n/Na,n/Nb        : [arc(nj,NP1,Nb),arc(nc,Na,Nb)]
  npc/Na  → np2/NP2,n/Na             : [arc(nc,NP2,Na)]
  npc/Na  → np3/NP3,n/Na             : [arc(nc,NP3,Na)]
  np1/Nc  → n/Na,n/Nb,n/Nc           : [arc(nc,Na,Nb),arc(nc,Nb,Nc)]
  np2/Nd  → n/Na,n/Nb,n/Nc,n/Nd      : [arc(nj,Na,Nc),arc(nc,Nb,Nc),arc(nc,Nc,Nd)]
  np3/Nd  → n/Na,n/Nb,n/Nc,n/Nd      : [arc(nc,Na,Nb),arc(nj,Nb,Nd),arc(nc,Nc,Nd)]

[Lexicon]
  word(n,[Tokyo]).  word(n,[taxi]).  word(n,[driver]).
  word(n,[call]).  word(n,[center]).
```

This example sentence has three well-formed dependency trees shown in Fig.A.1 (a), (b) and (c). The boxes np1, np2 and np3 in the dependency trees are given only for showing the correspondences between phrase structures and dependency structures.



(a)
np1
nc-1
nc-2
nj-7
nc-6
rt-8
S
Tokyo taxi driver call center
(b)
np2
nj-4
nc-2
nc-3
nc-6
rt-8
S
Tokyo taxi driver call center
(c)
np3
nc-1
nj-5
nc-3
nc-6
rt-8
S
Tokyo taxi driver call center
(d)
nc-1
nc-2
nc-3
nc-6
rt-8
S
Syntactic Graph for (a),(b) and (c) generates (d) which has no corresponding phrase structure tree in the phrase structure forest
nj-4
nj-5
nj-7
nc-1
nc-2
nc-3
nc-6
rt-8
S
Tokyo taxi driver call center
Syntactic Graph/Exclusion Matrix

EM	1	4	2	5	3	7	6	8
1		1						
4	1			1		1		
2				1				
5		1	1			1		
3						1		
7		1		1	1			
6								
8								

Fig.A.1 Problem in the syntactic graph/exclusion matrix

Since nc-1 and nc-2 in (a), nc-2 and nc-3 in (b) and nc-3 and nc-1 in (c) have co-occurrence relation, respectively, the values of the exclusion matrix for these three pairs are 0 (“ ” in the figure). This allows the existence of the dependency tree (d), which has no corresponding phrase structure tree in the phrase structure forest in the syntactic graph/the exclusion matrix. Therefore, the syntactic graph violates the soundness condition.

Bibliography

S. Abney. 1994. Dependency Grammars and Context-Free Grammars. *Manuscript, University of Tubingen, March.*

S. P. Abney. 1997. Stochastic Attribute-Value Grammars. *American Journal of Computational Linguistics*, 23(4).

S. Amano, H. Hirakawa, H. Nogami, and A. Kumano. 1989. The Toshiba Machine Translation System. *Future Computing Systems*, 2(3):121–124.

S. Bangalore and A. K. Joshi. 1999. Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25(2).

D. M. Bikel. 2004. Intricacies of Collins' Parsing Model. *Computational Linguistics*, 30(4):479–511.

E. Black, F. Jelinek, J. Lafferty, D. Magerman, R. Mercer, and S. Roukos. 1992. Towards History-based Grammars: Using Richer Models for Probabilistic Parsing. In *Proceedings of the 5th DARPA Speech and Natural Language Workshop*, pages 134–139.

G. Bouma, G. Noord, and R. Malouf, 2001. *Computational Linguistics in the Netherlands2000*, chapter Alpino: Wide Coverage Computational Analysis of Dutch, pages 45–59. McGraw-Hill, Amsterdam, Rodopi.

E. Briscoe, J. Carroll, J. Graham, and A. Copestake. 2002. Relational Evaluation Schemes. In *Proceedings of the Beyond PARSEVAL Workshop at the 3rd International Conference on Language Resources and Evaluation*, pages 4–8.

S. Buchholz and E. Marsi. 2006. CoNLL-X Shared Task on Multilingual Dependency Parsing. *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2006.*

G. Carroll and E. Charniak. 1992. Two Experiments on Learning Probabilistic Dependency Grammars form Corpora. Technical report, Department of Computer Science, Brown University.

J. Carroll, T. Briscoe, and A. Sanfilippo. 1998. Parser Evaluation: a Survey and a New Proposal. In *Proceedings of the 1st International Conference on Language Resources and Evaluation*, pages 447–454.

E. Charniak and M. Johnson. 2005. Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 173–180.

E. Charniak. 1995. Parsing with Context Free Grammars and Word Statistics. Technical report, Department of Computer Science, Brown University, Providence, September.

E. Charniak. 1997. Statistical Parsing with a Context-Free Grammar and Word Statistics. In *AAAI/IAAI*, pages 598–603.

E. Charniak. 2000. A maximum-Entropy-Inspired Parser. In *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics*, pages 132–139.

C. Chelba, D. Engle, F. Jelinek, V. Jimenez, S. Khudanpur, L. Mangu, H. Printz, E. Ristad,

R. Rosenfeld, A. Stolcke, and D. Wu. 1997. Structure and Performance of a Dependency Language Model. *Proceedings of the Fifth European Conference on Speech Communication and Technology, Grenoble, France*, 5.

N. Chomsky. 1956. Three Models for the Description of Language. *IEEE Transactions on Information Theory*, 2(3):113–124.

N. Chomsky. 1957. *Syntactic Structure*. Mouton: De Gruyter.

N. Chomsky, 1970. *Readings in Transformational Grammar*, chapter Remarks on Nominalization. Ginn and Co, Boston.

Y. J. Chu and T. H. Liu. 1965. On the Shortest Arborescence of a Directed Graph. *Science Sinica*, 14:1396–1400.

S. Clark and J. R. Curran. 2003. Log-Linear Models for Wide-Coverage CCG Parsing. In *Proceedings of the SIGDAT Conference on Empirical Methods in Natural Language Processing (EMNLP '03)*, pages 97–104.

S. Clark and J. R. Curran. 2004. Parsing the WSJ using CCG and Log-linear Models. In *Proceedings of the 42nd Meeting of the ACL (ACL-04)*.

W. F. Clocksin and C. S. Mellish, editors. 1984. *Programming in Prolog (Second Edition)*. Springer Verlag, Heidelberg.

M. J. Collins. 1996. A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings of the 34th Annual Meeting of the ACL (ACL-96)*, pages 184–191.

M. Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia.

A. Colmerauer, H. Kanoui, P. Roussel, and R. Passero. 1973. Un Systeme de Communication Homme-Hachine en Francais. Technical report, Groupe de Recherche en Intelligence Artificielle, Universite d'Aix-Marseille.

J. Courtin and D. Genthial. 1998. Parsing with Dependency Relations and Robust Parsing. *Processing of Dependency-based Grammars. Proceedings of the Workshop, COLING-ACL*, 98:95–101.

M.A. Covington. 1990. Parsing Discontinuous Constituents in Dependency Grammar. *Computational Linguistics*, 16(4):234–236.

V. Dahl and M. C. McCord. 1983. Treating Coordination in Logic Grammars. *American Journal of Computational Linguistics*, 9(2):69–91.

B. Djordjevic, J. R. Curran, and S. Clark. 2007. Improving the Efficiency of a Wide-Coverage CCG Parser . In *Proceedings of the 10th International Conference on Parsing Technology (IWPT-07)*, pages 39–47.

J. Edmonds. 1967. Optimum Branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

EDR. 1996. EDR Electronic Dictionary Version 1.5 Technical Guide. Technical report, EDR.

J. Eisner. 1996a. Efficient Normal Form Parsing for Combinatory Categorial Grammar. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 79–86.

J. Eisner. 1996b. Three New Probabilistic Models for Dependency Parsing: An Exploration. In

Proceedings of COLING'96, pages 340–345.

J. M. Eisner. 1996c. An Empirical Comparison of Probability Models for Dependency Grammar. Technical report, Institute for Research in Cognitive Science, University of Pennsylvania.

C. Fellbaum, editor. 1998. *WORDNET – An Electronic Lexical Database.* The MIT Press, Cambridge, MA.

H. Gaifman. 1965. Dependency Systems and Phrase-structure Systems. *Information and Control*, 8:304–337.

G. Gazdar, E. Klein, G. Pullum, and I. A. Sag, editors. 1985. *Generalized Phrase Structure Grammar.* Harvard University Press, Cambridge, MA.

D. Grinberg, J. Lafferty, and D. Sleator. 1995. A Robust Parsing Algorithm For Link Grammars. *Proceedings of the Fourth International Workshop on Parsing Technologies*, pages 111–126.

R. Grishman, C. Macleod, and J. Sterling. 1992. Evaluating Parsing Strategies using Standardized Parse Files. In *Proceedings of the 3rd conference on Applied Natural Language Processing.*

K. Hall. 2007. K-best Spanning Tree Parsing. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 392–399, Prague, Czech Republic.

M. Harada and T. Mizuno. 2001. Japanese Semantic Analysis System SAGE using EDR (in Japanese). *Transactions of the Japanese Society of Artificial Intelligence*, 16(1):85–93.

M. P. Harper and R. A. Helzerman. 1995. Extensions to Constraint Dependency Parsing for Spoken Language Processing. *Computer Speech and Language*, 9:187–234.

M. P. Harper, S. A. Hockema, and C. M. White. 1999. Enhanced Constraint Dependency Grammar Parsers. In *Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing.*

M. P. Harper, C. M. White, W. Wang, M. T. Johnson, and R. A. Helzerman. 2000. Effectiveness of Corpus-Induced Dependency Grammars for Post-Processing Speech. In *Proceedings of NAACL'2000.*

S. Hashimoto, editor. 1946. *Kokugo-gaku Gairon (in Japanese).* Iwanami, Tokyo.

D. G. Hays. 1964. Dependency Theory: A Formalism and Some Observations. *Language*, 46:511–525.

M. Hearst and H. Schutze. 1993. Customizing a Lexicon to Better Suit a Computational Task. In *Proceedings of the ACL SIGLEX Workshop.*

J. Heineck, J. Kunze, W. Menzel, and I. Schroder. 1998. Eliminative Parsing with Graded Constraints. In *Proceedings of the Joint Conference COLING-ACL*, pages 526–530.

H. Hirakawa and S. Amano. 1989a. Japanese Sentence Analysis Using Syntactic/Semantic Preference (in Japanese). In *Proceedings of the 3rd National Conference of JSAI*, pages 363–366.

H. Hirakawa and S. Amano. 1989b. Method for Searching Optimum Tree in Japanese Sentence Analysis (in Japanese). In *Natural Language Processing NL-74-2,IPSJ*, pages 9–16.

H. Hirakawa, Z. Xu, and K. Haase. 1996. Inherited Feature-based Similarity Measure Based on Large Semantic Hierarchy and Large Text Corpus. In *Proceedings of COLING'96*, pages 508–513.

H. Hirakawa, K. Ono, and Y. Yoshimura. 2000. Automatic Refinement of a POS Tagger Using a Reliable Parser and Plain Text Corpora. In *Proceedings of the COLING'00*, pages 313–319.

H. Hirakawa. 2001. Semantic Dependency Analysis Method for Japanese Based on Optimum Tree Search Algorithm. In *Proceedings of the PACLING2001*, pages 117–126.

H. Hirakawa. 2002. Semantic Dependency Analysis Method for Japanese Based on Optimum Tree Search Algorithm (in Japanese). *IPSJ Journal*, 43(3):696–707.

H. Hirakawa. 2006a. Graph Branch Algorithm: An Optimum Tree Search Method for Scored Dependency Graph with Arc Co-occurrence Constraints. *Journal of Natural Language Processing*, 13(4):3–32.

H. Hirakawa. 2006b. Preference Dependency Grammar and its Packed Shared Data Structure 'Dependency Forest' (In Japanese). *Journal of Natural Language Processing*, 13(3):37–90.

T. Hitaka and S. Yoshida. 1980. A Syntax Parser based on the Case Dependency Grammar and its Efficiency. In *Proceedings of the 8th conference on Computational linguistics.*

K. Hollingshead and B. Roark. 2007. Pipeline Iteration. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 952–959.

L. Huang and D. Chiang. 2007. Forest Rescoring: Faster Decoding with Integrated Language Models. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 144–151, Prague, Czech Republic.

R.A. Hudson. 1984. *Word Grammar.* Basil Blackwell, Oxford, United Kingdom.

R.A. Hudson. 1991. *English Word Grammar.* Basil Blackwell, Cambridge, Mass., USA.

T. Ibaraki. 1978. Branch-and-Bounding Procedure and State-space Representation of Combinatorial Optimization Problems. *Information and Control*, 36,1-27.

S. Ikehara, editor. 1999. *Nihongo-Goi-Taikei CD-ROM Version (in Japanese).* Iwanami Shoten, Tokyo.

T. Inui and K. Inui. 2000. Committee-based Decision Making in Probabilistic Partial Parsing. In *Proceedings of COLING 2000*, pages 348–354. Morgan Kaufmann.

F. Jelinek, J. Lafferty, and R. Mercer, 1992. *Speech Recognition and Understanding: Recent Advances, Trends, and Applications, Series F: Computer and Systems Sciences, Volume 75*, chapter Basic Methods of Probabilistic Context-free Grammars, pages 1689–1690. Springer Verlag, New York.

R. Johnson, M. King, and L. Tombe. 1985. EUROTRA: A Multilingual System under Development. *Computational Linguistics*, 11(2-3):155–169.

M. Johnson, S. Geman, S. Canon, Z. Chi, and S. Riezler. 1999. Estimators for Stochastic "Unification-based" Grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL99)*, pages 535–541.

S. Kahane, A. Nasr, and O. Rambow. 1998. Pseudo-Projectivity: A Polynomially Parsable Non-Projective Dependency Grammar. In *Proceedings of the 36th annual meeting on Association for Computational Linguistics*, pages 646–652.

S. Kahane, 2003. *Handbooks of Linguistics and Communication Sciences 25 : 1-2*, chapter The Meaning-Text Theory, Dependency and Valency, pages 546–569. De Gruyter, Berlin/New

York.

R. M. Kaplan, S. Riezler, T. H. King, J. T. Maxwell III, A. Vasserman, and R. Crouch. 2004. Speed and Accuracy in Shallow and Deep Stochastic Parsing. In *Proceedings of the Human Language Technology Conference and the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2004)*.

R. Kaplan. 1989. The Formal Architecture of Lexical-Functional Grammar. *Journal of Information Science and Engineering*, 5:305–322.

N. Katoh and T. Ehara. 1989. A Fast Algorithm for Dependency Structure Analysis (in Japanese). In *Proceedings of 39th Annual Convention of the Information Processing Society*.

M. Kay. 1984. Functional Unification Grammar: a Formalism for Machine Translation. In *Proceedings of the 22nd annual meeting on Association for Computational Linguistics*, pages 75–78, Morristown, NJ, USA. Association for Computational Linguistics.

K. Kimura and H. Hirakawa. 2000. Abstraction of the EDR Concept Classification and its Effectiveness in Word Sense Disambiguation. In *Proceedings of LREC2000*.

T. Kodama. 1987. 依存文法の研究 *(Study in Dependency Grammar) (in Japanese)*. Kenkyu-sha Shuppan, Tokyo.

G.J. Kruijff. 2001. *A Categorial-Modal Logical Architecture of Informativity: Dependency Grammar Logic & Information Structure*. Ph.D. thesis, Faculty of Mathematics and Physics, Charles University, Prague.

G.J. Kruijff. 2002. Formal & Computational Aspects of Dependency Grammar - Heads, dependents, and dependency structures -. presentation.

V. Kubon. 2001. *Problems of Robust Parsing of Czech*. Ph.D. thesis, Faculty of Mathematics and Physics, Charles University, Prague.

T. Kudo and Y. Matsumoto. 2005. Japanese Dependency Parsing Using Relative Preference of Dependency (in Japanese). *IPSJ Journal*, 46(4):1082–1092.

S. Kurohashi and M. Nagao. 1994. A Syntactic Analysis Method of Long Japanese Sentences based on the Detection of Conjunctive Structures. *Computational Linguistics*, 20(4):507–534.

J. Lafferty, D. Sleator, and D. Temperley. 1992. Grammatical Trigrams: a Probabilistic Model of Link Grammar. In *Probabilistic Approaches to Natural Language*.

B.Y.T. Lai and C. Huang. 1994. Dependency Grammar and the Parsing of Chinese Sentences. *Arxiv preprint cmp-lg/9412001*.

A. H. Land and A. G. Doig. 1960. An Automatic Method of Solving Discrete Programming Problems. *Economerrica*, 28(3):497–520.

S. Lee and K. S. Choi. 1997. Reestimation and Best-First Parsing Algorithm for Probabilistic Dependency Grammars. In *Proceedings of the Fifth Workshop on Very Large Corpora*, pages 41–55.

D. Lin. 1998. A Dependency-Based Method for Evaluating Broad-Coverage Parsers. *Natural Language Engineering Archive*, 4(2):97–114.

V. Lombardo and L. Lesmo. 1996. An Earley-type recognizer for dependency grammar. *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 723–728.

V. Lombardo and L. Lesmo. 1998. Formal aspects and parsing issues of dependency theory. *Proceedings of the 36th conference on Association for Computational Linguistics-Volume 2*, pages 787–793.

H. Maruyama. 1990. Constraint Dependency Grammar and Its Weak Generative Capacity. *Computer Software*.

Y. Matsumoto, H. Tanaka, H. Hirakawa, H. Miyoshi, and H. Yasukawa. 1983. BUP: A Bottom-Up Parser Embedded in Prolog. *New Generation Computing*, 1(2):145–158.

S. Matsunaga and M. Kohda. 1988. Linguistic Processing using a Dependency Structure Grammar: for Speech Recognition and Understanding. In *Proceedings of the 12th conference on Computational linguistics*, pages 402–407, Budapest, Hungry.

D. McCarthy. 1997. Word Sense Disambiguation for Acquisition of Selectional Preferences. In *Proceedings of the ACL/EACL 97 Workshop Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications*, pages 52–61.

R. McDonald and J. Nivre. 2007. Characterizing the Errors of Data-Driven Dependency Parsing Models. *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 122–131.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajic. 2005. Non-projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of the Human Language Technology / Empirical Methods in Natural Language Processing conference (HLT-EMNLP)*, pages 523–530.

I.A. Mel'cuk. 1988. *Dependency Syntax: Theory and Practice.* State University of New York Press, Albany, New York.

P. Mertens. 2002. Parsing Dependency Grammar using ALE. *Proceedings COLING 2002*, 2:653–659.

Y. Miyao and J. Tsujii. 2002. Maximum Entropy Estimation for Feature Forests. In *Proceedings of Human Language Technology Conference (HLT 2002).*

U. Montanari. 1976. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Science*, 7.

T. Mori, M. Matsuo, and H. Nakagawa. 2000. Zero pronoun resolution by Linguistic Constraints and Defaults – The case of Japanese Instruction Manuals –. *SPECIAL ISSUE ON ANAPHORA RESOLUTION IN MACHINE TRANSLATION, (Ruslan Mitkov editor), Machine Translation*, 14:231–245.

S. Muresan and O. Rambow. 2007. Grammar Approximation by Representative Sublanguage: A New Model for Language Learning. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 832–839, Prague, Czech Republic.

S. Nakagawa and T. Ito. 1987. Recognition of Spoken Japanese Sentences Using Mono-Syllable Units and Backward Kakari-uke Parsing Algorithm (in Japanese). *Transactions of IEICE*, J70-D(12):2469–2478.

A. Nasr, 2000. *Selected Aspects of Dependency Theory*, chapter Selected Aspects of Dependency Theory. Benjamins Academic Publishers, Amsterdam.

T. Ninomiya, T. Matsuzaki, Y. Tsuruoka, Y. Miyao, and J. Tsujii. 2006. Extremely Lexicalized

Models for Accurate and Fast HPSG Parsing. In *Proceeding of EMNLP 2006*, pages 155–163.

T. Ninomiya, T. Matsuzaki, Y. Miyao, and J. Tsujii. 2007. A Log-linear Model with an N-gram Reference Distribution for Accurate HPSG Parsing. In *Proceedings of IWPT-2007.*

J. Nivre and J. Nilsson. 2005. Pseudo-Projective Dependency Parsing. In *Proceedings of the 43rd Meeting of the ACL (ACL-05)*, pages 99–106.

J. Nivre and K. Sandra. 2006. Dependency Parsing. In *COLING-ACL 2006 Tutorial Notes.*

J. Nivre and M. Scholz. 2004. Deterministic Dependency Parsing of English Text. In *Proceedings of COLING'04*, pages 64–70.

J. Nivre. 2005. Dependency Grammar and Dependency Parsing. Technical report, MSI report 05133, Vaxjo University: School of Mathematics and Systems Engineering.

T. Noro, A. Okazaki, T. Tokunaga, and H. Tanaka. 2002. 大規模日本語文法構築に関する一考察. In *Proceeding of The Eighth Annual Meeting of The Association for Natural Language Processing*, pages 387–390.

T. Noro, T. Hashimoto, T. Tokunaga, and H. Tanaka. 2005. Building a Large-Scale Japanese Grammar. *Journal of Natural Language Processing*, 12(1):3–32.

S. Oepen, K. Toutanova, S. Shieber, C. Manning, D. Flickinger, and T. Brants. 2002. The LinGO, Redwoods Treebank. Motivation and Preliminary Applications. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002).*

K. Ozeki and Y. Zhang. 1999. 最小コスト分割問題としての係り受け解析 (Kakari-uke analysis as minimum cost partitioning problem) (in Japanese). In *Proceeding of the Workshop of The Fifth Annual Meeting of The Association for Natural Language Processing*, pages 9–14.

K. Ozeki. 1986. A Multi-stage Decision Algorithm for Optimum Bunsetsu Sequence Selection (in Japanese). In *Speech-86-32,IEICE*, pages 41–48.

K. Ozeki. 1994. Dependency Structure Analysis as Combinatorial Optimization. *Information Sciences*, 78(1-2):77–99.

K. Ozeki. 1998. A Tabular Method of Finding the Optimal Word String together with its Dependency Structure. In *Tabulation in Parsing and Deduction (TAPD'98).*

F. Pereira and D. Warren. 1980. Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence*, 13(3):231–278.

C. Pollard and I. A. Sag. 1994. *Head-Driven Phrase Structure Grammar.* University of Chicago Press, Chicago.

O. Rambow and A. Joshi. 1995. A Formal Look at Dependency Grammars and Phrase-Structure Grammars, with Special Consideration of Word-Order Phenomena. *Current Issues in Meaning-Text Theory. Pinter, London*, pages 167–190.

P. Resnik. 1993. *Selection and Information: A Class-Based Approach to Lexical Relationships.* Ph.D. thesis, University of Pennsylvania, Philadelphia.

P. Resnik. 1995a. Disambiguating Noun Groupings with Respect to WordNet Senses. In *Proceedings of the ACL-95.*

P. Resnik. 1995b. Using Information Content to Evaluate Semantic Similarity in a Taxonomy.

In *Proceedings of the IJCAI-95*.

S. Riezler, T. H. King, R. M. Kaplan, R. Crouch, J. T. Maxwell III, and M. Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and Discriminative Estimation Techniques. In *Proceedings of the 40th Annual Meeting of the ACL (ACL-02)*, pages 271–278.

H. C. Rim, J. Seo, and R. F. Simmons. 1990. Transforming Syntactic Graphs into Semantic Graphs. Technical report, Artificial Intelligence Lab, Texas at Austin.

J.J. Robinson. 1970. Dependency Structures and Transformational Rules. *Language*, 46(2):259–285.

K. Sagae and A. Lavie. 2006. A Best-First Probabilistic Shift-Reduce Parser. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 691–698.

K. Sagae and J. Tsujii. 2007. Dependency Parsing and Domain Adaptation with LR Models and Parser Ensembles. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 1044–1050, Prague, Czech Republic.

K. Sagae, Y. Miyao, and J. Tsujii. 2007. HPSG Parsing with Shallow Dependency Constraints. In *Proceedings of the 44th Meeting of the Association for Computational Linguistics*, pages 624–631, Prague, Czech Republic.

G. Sampson. 2000. A Proposal for Improving the Measurement of Parse Accuracy. *International Journal of Corpus Linguistics*, 5(1):53–68.

M. Schiehlen. 1996. Semantic Construction from Parse Forests. In *Computational Linguistics (COLING'96)*, pages 907–912.

K. Seki, A. Fujii, and T. Ishikawa. 1997. Estimation of a Probability Distribution over a Hierarchical Classification. In *Proceedings of the Tenth White House Papers COGS - CSRP*.

K. Seki, A. Fujii, and T. Ishikawa. 2002. A Probabilistic Method for Analyzing Japanese Anaphora Integrating Zero Pronoun Detection and Resolution. In *Proceedings of COLING'02*, pages 188–195.

J. Seo and R. F. Simmons. 1989. A Syntactic Graphs: A Representation for the Union of All Ambiguous Parse Trees. *Computational Linguistics*, 15(1):19–32.

P. Sgall, E. Hajičová, J. Panevová, and J. Mey. 1986. *The Meaning of the Sentence in its Semantic and Pragmatic Aspects*. Reidel.

S. M. Shieber, H. Uszkoreit, F. C. N. Pereira, J. Robinson, and M. Tyson. 1983. The Formalism and Implementation of PATR-II. Technical report, SRI International, Menlo Park, California.

K. Shirai, M. Ueki, T. Hashimoto, T. Tokunaga, and H. Tanaka. 2000. MSLR Parser - Tools for Natural Language Analysis. *Natural Language Processing*, 7(5):93–112.

K. Shudo, T. Narahara, and S. Yoshida. 1980. Morphological Aspects of Japanese Language Processing. In *Proceedings of COLING'80*, pages 1–8.

D. Sleator and D. Temperley. 1991. Parsing English with a Link Grammar. Technical report, Department of Computer Science, Carnegie Mellon University.

B. Srinivas. 2000. A Lightweight Dependency Analyzer for Partial Parsing. *Natural Language Engineering*, 6(2):113–138.

M. Steedman. 2000. *The Syntactic Process.* The MIT Press, Cambridge, MA.

A. Stolcke, C. Chelba, G. Engle, V. Jimenez, L. Mangu, H. Printz, E. Ristad, R. Rosenfeld, and D. Wu. 1997. Dependency Language Modeling. Large vocabulary continuous speech recognition summer research workshop technical reports, John Hopkins University.

K. Y. Su, T. H. Chiang, and J. S. Chang. 1996. An Overview of Corpus-Based Statistics-Oriented (CBSO) Techniques for Natural Language Processing. *International Journal of Computational Linguistics & Chinese Language Processing (CLCLP)*, 1(1):101–157.

B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. 2004. Max-Margin Parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP).*

L. Tesnière. 1969. *Éléments de syntaxe structurale.* Klincksieck.

M. Tomita. 1987. An Efficient Augmented-Context-Free Parsing Algorithm. *Computational Linguistics*, 13(1-2):31–46.

K. Toutanova and C. D. Manning. 2002. Feature Selection for a Rich HPSG Grammar Using Decision Trees. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002).*

Y. Tsuruoka, Y. Miyao, and J. Tsujii. 2004. Towards Efficient Probabilistic HPSG Parsing: Integrating Semantic and Syntactic Preference to Guide the Parsing. In *IJCNLP-04 Workshop: Beyond Shallow Analyses- Formalisms and Statistical Modeling for Deep Analyses.*

M. Walker, M. Iida, and S. Cote. 1994. Japanese Discourse and the Process of Centering. *Computational Linguistics*, 20(2):193–232.

D. L. Waltz, 1975. *The Psychology of Computer Vision*, chapter Understanding Line Drawings of Scenes with Shadows. McGraw-Hill.

W. Wang and M. P. Harper. 2002. The SuperARV Language Model: Investigating the Effectiveness of Tightly Integrating Multiple Knowledge Sources. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.*

W. Wang and M. P. Harper. 2004. A Statistical Constraint Dependency Grammar (CDG) Parser. In *Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*, pages 42–49.

L. Wanner. 1994. *Current issues in Meaning-Text Theory.* Pinter Publishers, London.

Y. A. Wilks. 1975. An Intelligent Analyzer and Under-stander of English. *Communications of the A.C.M.*, 18:264–274.

T. Winograd. 1983. *Language as a Cognitive Process: Volume 1 Syntax.* Addison-Wesley Publishing.

F. Xia and M. Palmer. 2000. Converting Dependency Structures to Phrase Structures. *Proceedings of the First International Conference on Human Language Technology Research*, pages 1–5.

S. Xiaodong and Y. Chen. 2007. Nbest Dependency Parsing with Linguistically Rich Models. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 80–82, Prague, Czech Republic.

H. Yamada and Y. Matsumoto. 2003. Statistical Dependency Analysis with Support Vector

Machine. In *Proceedings of IWPT'03*, pages 195–206.

S. Yoshida. 1972. Syntax Analysis of Japanese Sentence based on Kakariuke Relation between Two Bunsetsu (in Japanese). *Transactions of IECE Japan*, J55-D(4).

D. Zeman and Z. Žabokrtský. 2005. Improving Parsing Accuracy by Combining Diverse Dependency Parsers. In *Proceedings of the International Workshop on Parsing Technologies*, pages 171–178, Vancouver, British Columbia.