Semantic Dependency Analysis Method for Japanese based on Optimum Tree Search Algorithm

Hideki Hirakawa

Knowledge Media Laboratory, Corporate R&D Center, Toshiba Corp.
1, Komukai Toshiba-cho, Saiwai-ku, Kawasaki, 212-8582, Japan
Email: hideki.hirakawa@toshiba.co.jp, Tel: +81-44-549-2020, Fax: +81-44-520-1308

Keywords: Japanese sentence analysis, dependency analysis, optimum search

Abstract

There is combinatorial order of ambiguities in each level of natural language analysis, such as the morphology, syntax, and semantic levels. Moreover, much of the linguistic knowledge in each level is preference knowledge and has mutual interference. Deterministic processing is usually introduced to avoid the combinatorial explosion. However, this restricts the ability of natural language processing system because of the above-mentioned features of linguistic knowledge. This paper describes a sentence analysis method which uniformly evaluates syntactic and semantic preference knowledge, and shows an algorithm (based on the branch and bounding method) to search the optimum semantic tree from a semantic dependency graph which holds syntactic and semantic ambiguities in a Japanese sentence.

1. Introduction

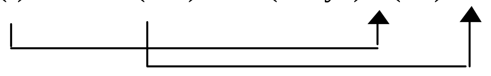
One of the most serious problems in natural language analysis is an ambiguity resolution in various levels such as morphology, syntax, semantics and pragmatics. In each analysis level, there exist a combinatorial number of interpretations of sentences. A proper interpretation of a sentence should be selected based on consideration of the following characteristics of linguistic knowledge.

a. A majority of linguistic and semantic knowledge is not restrictive knowledge but preference knowledge.
b. There is interference between the different levels of knowledge.

In the course of sentence analysis, many possible interpretations appear regarding choices of parts of speech of words and choices of dependencies of words. These possibilities are generated as hypotheses of some kind in the analysis process.

The restrictive application of linguistic knowledge causes the rejection of one or more interpretations and the preferential application gives the preferential order of the generated hypotheses. In general, Japanese word connection conditions are restrictively applicable linguistic knowledge, and semantic knowledge is preferably applicable knowledge as shown in the preference semantics [Wilks 75]. Linguistic knowledge recognized as restrictive knowledge sometimes has exceptions. For example, cross dependency restriction is considered to be a strong restriction in Japanese language, there are exceptions such as

watashi-ha sakana-wo Tokyo-ni tabe-ni itta.
(I) (fish) (Tokyo) (eat) (go)



Restrictive application of knowledge suppresses combinatorial explosions and improves process efficiency. However, if preference knowledge is applied restrictively, the system may fail to capture the correct interpretation of a sentence.

The following example shows the interference between the different levels of knowledge, i.e. syntactic knowledge and semantic knowledge.

S1. X-ha Y-ga ookii.
X-PT Y-PT big
S2. Te-ha Taro-ga ookii.
hands-PT Taro-PT big
S3. Taro-ga ookii te
Taro-PT big hands

In each of these examples, the upper sentence is a Japanese written in alphabetic characters, and the lower one shows the same sentence but content-words are replaced with English correspondences. “PT” means a particle in Japanese, which marks a case role. Here, sentence pattern S1 has two readings corresponding to (a) "X has big Y" and (b) "Y has big X". In S1, interpretation (a) is preferable since particle "ha" often marks the theme of a sentence. On the other hand, in S2, interpretation (b) is preferable since syntactic preference caused by "ha" is defeated by semantic preference caused by semantic relation (part-whole relation) between hands and Taro (human). S3 shows a conflict between preferences originated from syntactic structure and semantic relation. This example shows the need for proper treatment of different levels of preference knowledge.

In order to handle the above two characteristics, a. and b., the following three functionalities are required.

1) Incorporating possible interpretations of every analysis levels of a sentence efficiently into some structure
2) Setting various kinds of preference values to interpretations in the structure
3) Finding the optimum interpretations with respect to preference values

Tomita [Tomita 87] and Barton [Barton 85] proposed a method for packing the whole phrase structures for a sentence efficiently. In Constraint Dependency Grammar (CDG), possible dependency structures are maintained intentionally utilizing constraint matrix and constraint propagation mechanism excludes inappropriate structures [Maruyama 90]. Hirakawa proposed a data structure called a semantic dependency graph for implementing the above three functionalities for Japanese sentence analysis [Hirakawa 89a]. The third functionality inherently requires combinatorial order of computation. An efficient and practical algorithm for managing this computational problem is required. In this paper, branch and bounding method [Ibaraki 78] is adopted for obtaining the optimum interpretation of a sentence [Hirakawa 89b]. Section 2 describes a semantic dependency graph and its construction method. Section 3 shows an algorithm for obtaining the optimum solution from a semantic dependency graph. Section 4 and section 5 describes an experiment concerning the algorithm and improvement of implementation. Section 6 mentions related works.

2. Semantic dependency graph

2.1 Structure of semantic dependency graph

There are various kinds of ambiguities in Japanese sentence analysis such as ambiguities in roles of words, directions of dependency relations, semantic relations between words, word references and so on. This paper focuses on ambiguities on directions of dependency relations and their semantic interpretations.

A data structure called a semantic dependency graph (or SED-graph) is introduced to represent word dependency relations and their semantic interpretation. In this graph, a node corresponds to a word in a sentence (particles are included). An arc connecting two nodes represents a possible dependency relation between them. An arc name shows the semantic relation or the role of the dependency relation. Each arc has a preference score called a weight. Fig.1 shows a semantic dependency graph for the Japanese sentence "watashi(I)-ha anata(you)-ga suki(like)-desu". English words in curved parentheses are translations of Japanese words.

The labels "ag" and "ob" represent the semantic relations "agent" and "object", respectively. Since Japanese language has a restriction such that a dependent word is located on the left hand side of its governor, a semantic dependency graph is a directed acyclic graph (DAG). The semantic structure of a sentence is defined as a well-formed spanning tree on a semantic dependency graph. This is called a semantic tree (SEM-tree) of a sentence. The SED-graph in Fig.1 has four spanning trees. Only two of them are well-formed tree, i.e., " watashi(I) -ag-> suki-desu(like) <- ob- anata(you)" and "watashi(I) -ob-> suki-desu(like) <-ag- anata(you)". Here, a well-formed tree is a tree that does not violate the follow-

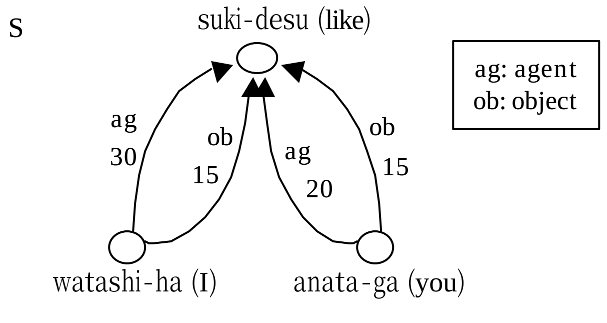


Fig.1 Semantic dependency graph for “watashi-ha anata-ga suki-desu”

ing two restrictions.

r1. No two arcs cross in a tree (cross dependency)
r2. No two arcs in a tree occupy the same valence of a predicate (multiple valence occupation)

Restriction r2 filters out two ill-formed trees " watashi(I) -ag-> suki-desu(like) <-ag- anata(you)" and " watashi(I) -ob-> suki-desu(like) <-ob- anata(you)" on the SED-graph in Fig.1.

The weight of a SEM-tree is the sum-total of weights of arcs in the tree. The SEM-tree with the maximum weight is called the optimum semantic tree or the optimum interpretation of a sentence.

2.2 Generation of semantic dependency graph

Fig.2 shows the syntactic and semantic analysis flow for the following Japanese example sentence.

kare-ha	**Taro-ga**	**katta**	**mise-wo**	**otozureta**
(he)	(Taro)	(bought)	(store)	(visited)

(He visited the store which Taro bought.)

An input word sequence (output of morphological analysis process) is analyzed by three processes. The output of each process is a syntactic dependency tree (SYD-tree), a semantic dependency graph (SED-graph), or a semantic tree (SEM-tree), respectively.

2.2.1 Syntactic analysis

The syntactic analysis process generates only one SYD-tree from an input word sequence using a extended CFG parser with an ability to construct word dependency structure. Fig.2 (2a) shows the SYD-tree for the example sentence. A SYD-tree has the following basic features.

a. A node corresponds to a word in the input sentence
b. An arc shows a syntactic dependency relation (case, nominal-modification etc.).
c. All candidates for one node's governor are in its ancestor nodes.

Feature c can be realized by applying a strategy to connect a word with its possible leftmost governor in the sentence and making a SYD-tree hold all possible syntactic dependency structures implicitly. In fact, a SYD-tree in straight line, where the rightmost word of a sentence is the root and the leftmost word of the sentence is a leaf, holds all implicit dependency relations. Furthermore, a SYD-tree has the following features.

d. Dependency relations which are not syntactically available are naturally filtered out
e. An SYD-tree itself represents information on syntactic structure of the input sentence.

An example of feature d is shown in Fig. 2. Since a particle cannot modify a noun, Fig.2 (2b) generated by syntactic parser has no implicit dependency relation between “kare-ha (he)” and “Taro-ga(Taro)”. Thus the filtering of implausible possibilities is done by syntactic knowledge. Feature e is useful in describing dependency generation rules to be explained in the next section.

2.2.2 Generation of semantic dependency graph

The second process generates a SED-graph from a

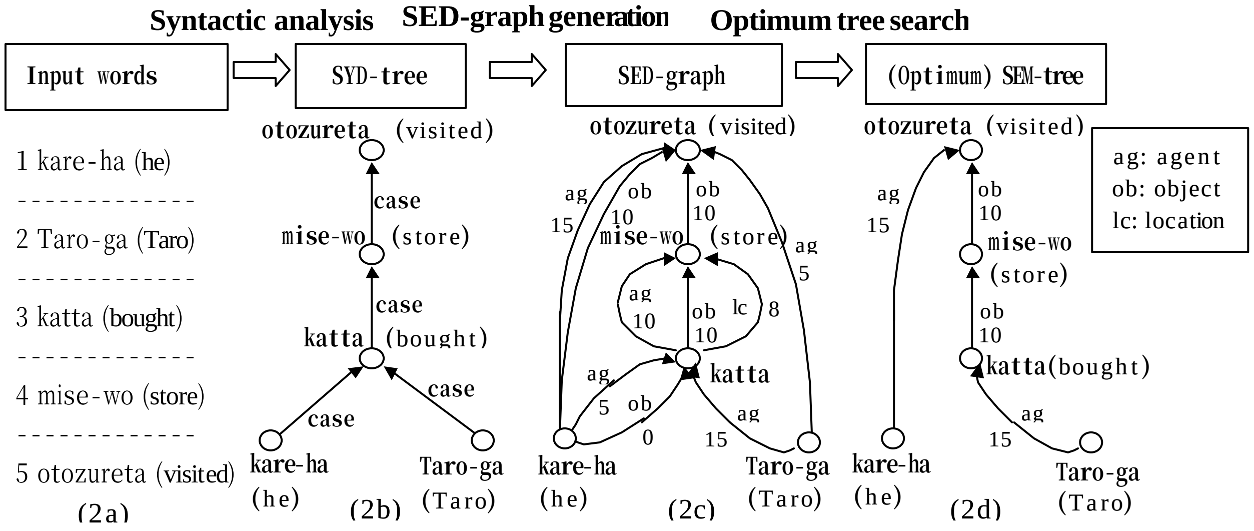


Fig. 2 Total flow of sentence analysis

SYD-tree. Fig.2 (2c) shows a SED-graph corresponding to the example sentence. SED-arcs are set by picking up two nodes in a SYD-tree along with a path in the tree, and calculating the semantic relation between the two nodes (SED generation rule). The SED generation rule in the case of the particle "ha" generates two SED-arcs ("agent", "object") between "kare(he)" and "katta(bought)" because the particle "ha" can mark these two different semantic rolls referring to the case frame of the predicate "katta(bought)". The same can be said of "utta(sold)". Weight is also attached to each arc by referring to various kinds of knowledge (syntactic, semantic, heuristic knowledge, etc.). The followings are examples of knowledge;

a. If the semantic marker of a noun matches the semantic condition of a case slot in a verb, then this semantic dependency relation is semantically preferable.
b. Particle "ha" represents a theme of a sentence, and tends to modify the rightmost predicate in a sentence.
c. Touten ("comma" in Japanese) attached to a word decreases the possibility of modification of the word to its nearest word.

3. Optimum tree search

Searching for the optimum tree is defined as a search for the maximum spanning tree for a directed acyclic graph $G=(N,A)$ (where N: set of nodes, A: set of arcs), satisfying restriction $R=\{<a_i,a_j> \mid p(a_i,a_j)=\text{true}, a_i,a_j \; A\}$ where p defines restrictions. We adopt the branch and bounding method as the basic strategy for searching the optimum tree.

3.1 Branch and bounding method

The branch and bounding 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 one is solved by solving them. Pruning called bounding operation is applied if it turns out that the optimum solution to a partial-problem is inferior to the solution to be obtained from some other partial-problem (dominance test), or if it turns out that a partial-problem gives no optimum solutions to the original one (maximum value test) [Ibaraki 78].

```
$A$ : Set of active partial-problems (not yet terminated nor
      expanded)
$z$ : incumbent value
$O$ : set of optimum solutions
$s$ : Function $s(A)$ selects one partial-problem in $A$
$l$ : Function $1(P)$ gives value of feasible solution of a partial
      problem $P$
$g$ : Function $g(P)$ gives upper-bound value of a partial
      problem $P$(maximum spanning tree)

S1 (initial-value set up) : $A:=\{P_0\}$,   $z:=-\infty$, $O:=\{\}$
S2 (search) : If $A=\{\}$ goto S8, else $P_i:=s(A)$ and goto S3.
S3 (incumbent value update) :
        If $1(P_i)>z$ then $z:=1(P_i)$, $O:=\{x\}$ ($x$ is the feasible
        solution to $P_i$), and goto S4.
S4 (G test) : If $g(P_i)=-\infty$ or $g(P_i)=1(P_i)$ then goto S7 else
        goto S5
S5 (maximum value test) : If $g(P_i)\leqq z$ then goto S7 else
        goto S6
S6 (branching operation) : Generate partial-problem $P_{i1}$ and
        $P_{i2}$ from $Pi$, and $A:= A \cup \{P_{i1}, P_{i2}\}-\{P_i\}$. goto S2.
S7 (termination of Pi) : $A:=A-\{ Pi \}$, goto S2
S8 (stop) : Computation stop. If $z = -\infty$ 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.3 The optimum tree search algorithm based on the branch and bounding method

3.2 Algorithm for optimum tree search

The optimum tree search algorithm for a SED-graph $G=(N,A)$ based on the branch and bounding method is shown in Fig.3. Two restrictive conditions (cross dependency, multiple valence occupation) are called the co-occurrence restriction collectively. This algorithm introduces a bounding operation based on the maximum bound value test. The maximum spanning tree for G gives the maximum bound value of the optimum tree for G. In fact, problem P' (the maximum spanning tree for G) satisfies the following features for the maximum bound value test with respect to the original problem P (optimum tree for G).

a. $g(P')\geqq f(P)$, where $g(P')$ is the maximum value of P', $f(P)$ is the maximum value of P.
b. If $g(P')=1(P)$ where 1 gives a value of a feasible solution to P, then the feasible solution is a solution to P.
c. If P' has no solutions then P has no solutions.
d. If a feasible solution with an incumbent value z is obtained for some partial-problem, and if $g(P')\leqq z$, then partial-problems branched from problem P have no better solutions than z.

In the case of b.-d., partial-problem P can be terminated.

In Fig.3, Sl initializes A, z, O. Initial problem P_0 is stored in A. S2 searches one active partial-problem P_i to expand, from A. In S3, a feasible solution x (well-formed tree) and its value is computed by function l. If this value is better than the incumbent value z, then x and O are updated since a better feasible solution is obtained. In S4, the maximum bound (weight of the maximum tree) is computed by function g. In S4 and S5, termination checks (b. to d. described above) are applied. Any partial-problem which cannot be terminated (i.e. the maximum bound of the partial-problem is larger than the incumbent value) is expanded into two child partial-problems in S6. And P_i in A is replaced with these problems. In this way, the problem is decomposed into partial-problems forming a binary search diagram. Each component in Fig.3 is described below.

<u>*s(A)*: Selects one partial-problem from set *A*</u>

The best bound search is employed for *s(A)*, i.e. *s(A)* selects the partial-problem which has the maximum bound value among active partial-problems. It is known that the number of partial-problems decomposed during computation is minimized by this strategy.

<u>*l(P)*: Computes a feasible solution</u>

Algorithm for *1* is shown in Fig.4. This algorithm provides depth-first search for a well-formed tree. Step1 classifies and sorts the arcs in G of P according to arc weight for obtaining a better feasible solution (greedy search). In step5, the co-occurrence restriction is checked and backtracking occurs if necessary. The choice point to backtrack is just the nearest choice point that may resolve the contradiction respecting the co-occurrence restriction. When this algorithm fails in searching for a solution, the original problem P has no solutions.

<u>*g(P)* : Calculates upper bound value</u>

This searches the maximum spanning tree for G. Since G is DAG, the maximum tree is obtained by picking up each maximum arc in S_i (1 i n-1) in Fig.4.

<u>Branch operation</u>:

Child partial-problems of P_i are constructed as follows:

a. Search arc e_l and e_2 in the maximum spanning tree for G=(N,E_i), which violate the co-occurrence restriction.
b. Create child partial-problems P_{i1},P_{i2} which have new graphs, G_{i1}=(N,E_i-{e_l}), and G_{i2}=(N,E_i-{e_2}) respectively.

Since the optimum tree of P_i cannot contain both e_l and e_2 due to the co-occurrence restriction, the optimum solution to P_i is obtained from either P_{i1} or P_{i2}.

```
EP: Array for saving arcs
BP: Array for saving the nearest backtrack points
num(S_i): number of elements in S_i
wg(EP): sum total of the weight of the arcs in EP

  stepl (Grouping and sorting arcs):
     Classify the arcs in graph G=(N,A)(|N|=n here) by their starting
    nodes, and generate the sets of arcs S_l, S_2,...,S_{n-1}. Sort elements
    in each S_i with respect to their weights in descending order. Sort
    S_l, S_2,...,S_{n-1} with respect to the maximum weight of the arcs in
    the set in descending order. This is renamed S_l, S_2,...,S_{n-1}.
  step2 (initialize): EP:= [], BP:= [], i:=1, j:=1,k:=1, l=0, w:=-∞
  step3 (termination checkl): If i=n, then w:=wg(EP) and
     terminate (EP holds a feasible solution for G),
     otherwise, goto step4.
  step4 (termination check2):   If num(S_i)≧j then goto step5 else
     EP:=[] and terminate (no solutions).
  step5 (restriction check):
     If j>num(S_i) (no arcs in S_i satisfies the co-occurrence restriction),
     goto step6. Perform the co-occurrence restriction check between
     j-th element a(i,j) of S_i and each element (e_1,e_2,... e_{i-1}) in EP in
    reverse order. If a(i,j) does not satisfy the co-occurrence restriction
    with element e_k of EP then l:=max(l,k), j:=j+1, goto step5.
    If all co-occurrence restriction checks are satisfied then goto
     step7.
  step6 (backtracking): Remove e_l, e_{l+1},...,e_{i-1} from EP,
     j:=BP[l]+1, i:=l, goto step4.
  step7 (next node):
     Add a(i,j) to the last of EP. BP[i]:=j, i:=i+1, j:=1, goto step3.
```

Fig .4 Algorithm for computing a feasible solution *l*(P_i)

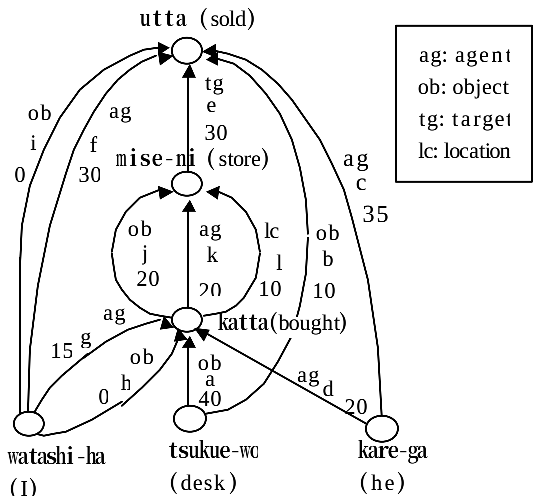


Fig. 5 Semantic dependency graph for the example sentence

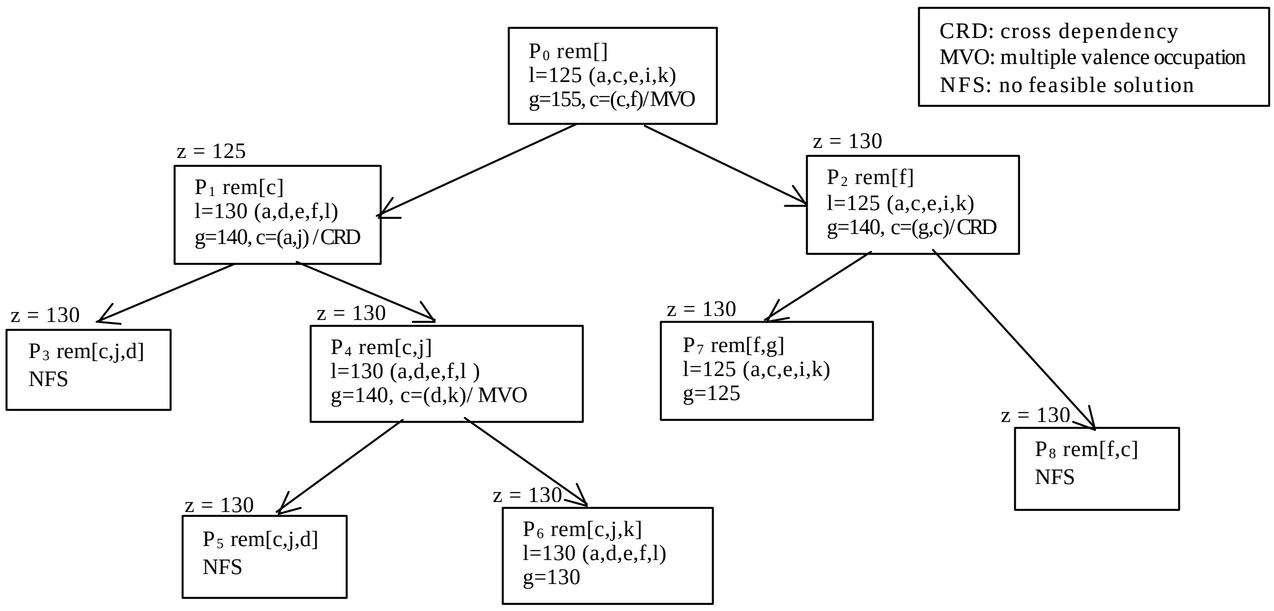


Fig.6 Search diagram for the example sentence

3.3 Example of optimum tree search

This section gives an example showing the behavior of the algorithm in 3.2. The sample sentence is "watashi-ha kare-ga tukue-wo katta mise-ni utta". This sentence has four noun phrases and two verbs. The SED-graph for this sentence is shown in Fig.5. Alphabet labels “a”-“l” are arc identifiers. Verbs are assumed to have the following simplified case frames, respectively.

```
"Utta (sold)"   ag[Agent], ob[Object], tg[Target]
"Katta (bought)"ag[Agent], ob[Object], lc[Location]
```

In Fig.5, "watashi (I)" has four outbound arcs, i.e. f, g, h, i. This means "watashi (I)" can be a filler of "ag" and "ob" case slots of verb "utta (sold)" and "katta (bought)". "katta (bought)" introduces a Japanese embedded sentence modifying “mise(store)”. In general, a Japanese embedded sentence has no surface clue for restricting semantic dependency relations. There are three possible arcs between "katta (bought)" and "mise (store)" representing semantic interpretations "bought store"(j), "store bought"(k) and "bought at store"(l). Each weight of arcs is set by the process described in section 2.

Before explaining the behavior of the algorithm, I will give the structure of a partial-problem that appears in the computation process. Partial problem P_i consists of the following three components.

a. G_i : Partial SED-graph for P_i
b. $l(P_i)$: Value of a feasible solution for P_i
c. $g(P_i)$: Upper bound value for P_i

Here, G_i is represented not by arcs that are constituents of G_i but by arcs that are not in G_i but in the whole SED-graph. "*rem[]*" shows arcs removed from the original SED-graph. For example, "*rem[*a*]*" represents a partial SED-graph consisting of arc b to l. This reduces the memory space and the computation for algorithm $l(P_i)$ in 3.2, i.e., step1 of this algorithm (Fig.4) is executed only for the first problem. The result of this process for the SED-graph in Fig.5 is as follows:

S_1:	tukue(desk)	a[ob,katta(bought),40], b[ob,utta(bought),10]
S_2:	kare(he)	c[ag,utta(sold),35], d[ag,katta(boungt),20]
S_3:	mise(store)	e[tg,utta(sold),30]
S_4:	watashi(I)	f[ag,utta(sold),30], g[ag,katta(bought),15], h[ob,katta(bought),0], i [ob,utta(sold),0]
S_5:	katta(bought)	j [ob,mise(store),20], k[ag,mise(store),20], l [lc,mise(store),10]

The SED-graph for one partial-problem is obtained by removing an arc from the SED-graph of its parent problem.

Fig.6 shows a search diagram (or branch diagram) representing the computational process for this example. In this diagram, box represents a partial-problem with "l" (value of feasible solution), "g" (upper bound value), "rem" (SED-Graph) and "c" (arc-pair list). Arc pair in "c" is found in the maximum spanning tree and does not satisfy the co-occurrence restriction. "z" shows an incumbent value for before starting the computation of the partial-problem. "P_i" is an identifier of the partial-problem and "i" shows the number of generation orders. "P_0" is an original problem and its rem is empty. The feasible solution for this problem is (a,c,e,i,k) as shown in the figure. The value of the feasible solution is 125 and the upper bound value is 155. Here, the algorithm described in 3.2 searches for the maximum spanning tree and obtains the tree (a,c,e,f,j) which has an arc pair c(c,f) violating the multiple valence occupation restriction. Since the initial value of "z" is -∞, it is set to 125. Since arc c and f are inconsistent, initial problem P_0 is branched to two child partial-problems P_1 and P_2. Values for l, g, c in P_1 and P_2 are computed and stored as shown in the figure. The next step is the search for the partial-problem to be tried next. P_1 and P_2 are candidates and either of them is available since they have the same upper bound value g (=140). In this case, P_1 is assumed to be the next target partial-problem to solve. Processing of P_3 is terminated since it has a SED-graph rem[c,a] that has no plausible solutions. Processing of P_7 is terminated due to bounding operation where upper bound value g (=125) is less than incumbent value z (=130). Finally, the semantic tree shown in Fig.7 is obtained as the optimum solution for P_0.

3.4 Computational complexity of the algorithm

The most important parameter for estimating the computational amount of the branch and bound method is the number of generated partial-problems *T*. In general, *T* increases exponential order with respect to the height of the branch diagram. This height grows according to the number of nodes in the given SED-graph, i.e. the number of words in a sentence. Therefore, the number of partial-problems grows exponentially to the sentence length. The lower bound of T is known to hold the following relation.

$$T \geqq \left| \{ P_i \mid g(P_i) > f(P_0) \} \right|$$

where $g(P_i)$ is the upper bound value for P_i, and $f(P_i)$ is the value for the optimum solution for P_i.

This means that improvement of the accuracy of $g(P_i)$ reduces the minimum number of generated partial-problems. The proposed algorithm adopts the maximum spanning tree for "g". The ability of g is one important factor for achieving the practical performance in analyzing real world sentences.

Main memory space factors of this algorithm are SED-graph and partial-problems. If each two nodes in the SED-graph with *n* nodes are connected by *k* arcs, the graph has at most $kn(n-1)/2$ arcs. This requires $O(n^2)$ memory space. On the other hand, required memory space for partial-problems is a product of “memory space for one partial-problem” and the “maximum number of leaves (partial-problems) of branch diagram during a computation”. One partial-problem requires $O(n^2)$ memory space for removed arc list.

The maximum number of the leaves of the branch diagram increases exponentially with respect to the height of the diagram. In practice, since the bounding operation suppresses the increase of the leaves of the diagram, the required space for the proposed algorithm also depends on *g* and *l*, described in the previous section, as well as on the structure of SED-graph. The next section describes an experiment to ascertain the practical performance of the algorithm.

4. Experiment of the algorithm

An experiment on the runtime efficiency of the algorithm was performed using 100 sentences extracted from a technical paper and a manual at random. The computer used was an engineering workstation AS4260. SED-graphs were provided by an existing piece of software. The result of the experiment is shown in Fig.8. A

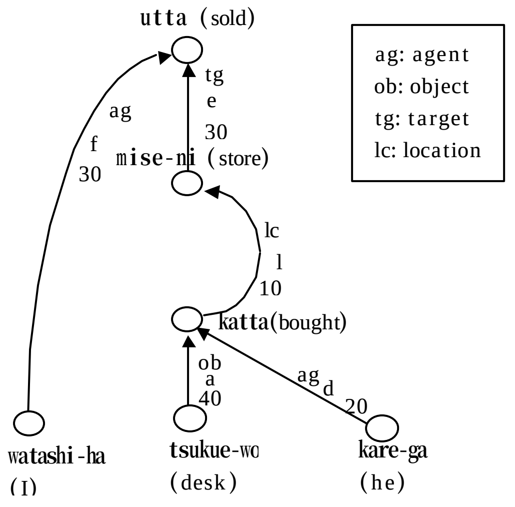


Fig.7 Optimum semantic tree for the example sentence

Number of nodes in semantic dependency graph	Sentence freq	Generated partial problems 1~5	6~10	11~15	16~25	26~
1~2	9	9 3ms				
6~10	12	12 25ms				
11~ 15	39	37 80ms	1(4/7) 293ms	1(4/11) 436ms		
16~ 20	21	20 173ms		1(1/11) 848ms		
21~ 25	10	7 312ms	1(1/9) 1088ms	1(1/11) 1231ms		8a
26~ 30	7	6 365ms				1(4/53) 11750ms
31~ 35	1			1(12/13) 3775ms	8b	
35~	1	1(1/5) 1258ms				

Fig.8 Result of the experiment

number in a cell of the figure shows the number of sentences from the sample fulfilling conditions. "(x/y)" form in a cell shows information on the solution process. The number in "x" position is the position of the partial-problem that generated the feasible solution that proved to be the optimum solution. The number in "y" position shows the number of totally expanded partial-problems. Average CPU time for the sentences in a cell is also shown in "##ms" form. The following are observations concerning Fig.8.

(1) The number of partial-problems required is relatively small

93 sentences out of 100 sentences required less than 6 partial-problems. Feasible solution search function *l(P)* based on greedy algorithm gave a good feasible solution to a given problem and suppressed the number of partial-problems.

(2) The optimum solution is obtained in the early stage of search

The sentence in a cell 8a in Fig.8 requires the maximum number (53) of partial-problems to get the optimum semantic tree. In this case, the optimum solution is obtained in the 4th partial-problem. The rest of the computation, i.e. the computation for the 5th or later partial-problems is performed only for checking if the feasible solution of the 4th problem is the optimum solution. In fact, 99% of the optimum solutions are obtained in the process of solving the first 5 partial-problems. One exception is 8b in Fig.8.

In this experiment, the average CPU time for obtaining the optimum tree for one sentence was 305.8ms. Almost all sentences require less than 1300ms, but 8a and 8b in Fig.8 required 11750ms and 3775ms, respectively. In 8a, the 4th partial-problem generated the optimum solution to the problem but the bounding operation was not applied to many succeeding partial-problems. In this situation, improvement of the upper bound function seems to be effective. In contrast, in the computation for 8b, all branch operations were necessary to get the optimum tree. In this case, improvement of feasible solution algorithm seems to be effective.

5. Improvement of the algorithm

We made two improvements to the algorithm. One is the improvement of function *g*. A new function *g'* is defined as "*g'*(*P*)=*g*(*P*)-safe penalty score". The other improvement is computation sharing between parent and child partial-problems

5.1 Improvement of upper bound value *g(P)*

In order to reduce the number of partial-problems, improvement of the upper bound value *g(P)* is introduced. *g(P)* described in 3.2 was the weight of the maximum spanning tree on G. This *g(P)* can be reduced by the safe penalty score computed from the arc pair in the maximum spanning tree on G, which violates the co-occurrence restriction. If arc pair ($S_i[k]$, $S_j[l]$) violates the co-occurrence restriction, the following score *D* can be reduced from *g(P)* in the algorithm shown in Fig.3.

$D = \min(w(S_i[k])-w(S_i[k+1]), w(S_j[l])-w(S_j[l+1]))$

where $w(A)$ gives weight of arc A

$g'(P) = g(P)-D$

5.2 Sharing of computation for partial-problems

Since a parent problem and its child problems are very similar, the same partial computations may be repeated in each problem solving process. The proposed algorithm *l(P)* adopts a backtracking method for computing a feasible solution. Since the back tracking algorithm is sometimes inefficient, sharing of the computation of feasible solution between a parent problem and its child partial-problems is introduced.

Revised with these two improvements, number of partial-problems and CPU time required in the analysis of

8a in Fig.8 improved from 53 to 9, and from 11750ms to 1326ms, respectively. In the case of 8b, number of partial-problems and CPU time improved from 13 to 11, and from 3775ms to 2826ms, respectively. Average CPU time improved from 305.8ms to 162.1ms.

6. Related work

This paper proposed a framework for analyzing Japanese sentences based on the "kakari-uke" analysis. The basic component of kakari-uke analysis is a kakari-uke relation (dependency relation) holding between two Japanese components called "bunsetsu" (basic unit consisting of contents words and functional words). Since a kakari-uke relation is subject to a restriction, namely that the dependent bunsetsu is located at the left-hand side of its governor bunsetsu, a kakari-uke grammar is considered to be a kind of the dependency grammar.

CDG is defined by lexical category, semantic role, label set, and constraint set [Maruyama 90], [Karlsson 90]. CDG assumes all dependency relations between every two words in a sentence at first, then inappropriate interpretations are eliminated based on constraint.[1] This method is called eliminative parsing to distinguish it from the conventional generative parsing method where possible parsing structures are generated in the parsing process. In this paper, we adopt the principle that sentence analysis system should be able to hold possible interpretations until they can be disambiguated, and should be able to commit an interpretation whenever it turns out to be committable. This idea is similar to CDG's eliminative parsing. However, generation of too much hypotheses causes problem in parsing efficiency, and so our method generates hypotheses obtained from the input sentence and the system's knowledge (ex. dictionary). In fact, in CDG research, restriction of hypothesis generation by using the fast filtering algorithm and the label table is introduced [White 00]. The biggest difference between CDG and our approach is that CDG treats restrictive application of linguistic knowledge (constraints) but our approach focuses on preferential application of linguistic knowledge. As discussed in the first section of this paper, the restrictive application of linguistic knowledge is insufficient for language analysis. In CDG, a preferential framework called "graded constraint" is proposed [Heinec 98].

Beale proposed to utilize the Hunter-Gatherer method (HD) for the analysis of computational semantics [Beale 96]. HD consists of a "hunting" mechanism for reducing the search space by removing sub-optimal or impossible solutions and a "gathering" mechanism for efficiently extracting solutions from search space. HD is used in the Pangloss machine translation system [Frederking 94] to obtain the optimal combination of the word sense meanings. The input for HD is a partitioned constraint dependency graph which is obtained from the "Text Meaning Representation" (TMR) produced by a syntactic analysis module called "Panglyzer" [Farwell 94]. TMR basically represents word dependency relations of a sentence and is a source of a constraint dependency graph which represents a set of constraints related to the word sense meanings. Beale introduces the concept of the preference semantics by giving a constraint "tendency" to each constraint, which is computed by using world knowledge etc. For computing the optimal combination of the word sense meanings, i.e. the optimal solution, HD utilizes a solution synthesis method enhanced with the partitioning of the constraint dependency graph and the branch and bounding method. Although the application task and the analysis framework of Beale's research is different from that of this paper, Beale showed the introduction of the preference score and use of the branch and bounding method is useful for the computational semantic application.

Various researches has been done on obtaining the optimum interpretation of Japanese kakari-uke (dependency) analysis. The main framework for optimum kakari-uke search is to represent possible kakari-uke relations with preference scores by utilizing a kakari-uke matrix, and search for the optimum interpretation based on the dynamic programming (DP) method. In kakari-uke analysis, possible interpretations of a sentence should satisfy the cross dependency restrictions. Ozeki proposed a fast kakari-uke analysis method for real-time applications such as speech recognition [Ozeki 94]. In Ozeki's method, the preference scores are basically assigned to bunsetsus (phrasal-unit) and/or kakari-uke relations. Optimum interpretation is obtained by an efficient algorithm based on DP. On the other hand, Kurohashi et al. pro-

[1] There are unary constraint and binary constraint

posed a method for analyzing coordinate structure (part of kakari-uke structure) in a long Japanese sentence [Kurohashi 94]. In this method, syntactic/pattern similarity value and semantic similarity value are totally evaluated in an analysis process based on DP. There are two main differences between conventional kakari-uke analysis method and the one proposed in this paper. One difference is that the kakari-uke relation is expanded to the semantic kakari-uke relation. The second one is that case analysis is introduced with multiple case occupation restriction. This restriction relates more than two constituents, and so efficient DP method is not applicable to the maximum solution search problem.

Seo proposed the "Syntactic Graph" for representing the possible dependency structures of a sentence [Seo 89]. Although a formal proof is not given in the paper, every dependency structure embedded in an "Syntactic Graph" seems to have one corresponding phrase structure in packed shared parse-forest. This representation has an advantage compared with the semantic dependency graph that all part of speech are represented in one structure. This feature is not so important in Japanese language analysis but is especially important for the languages, such as English, with a lot of part-of-speech ambiguities.

6. Conclusion

A Japanese sentence analysis method which uniformly evaluates syntactic and semantic preference knowledge, and an optimum solution search algorithm are described. This algorithm gives the most preferable interpretation of sentences very efficiently (disambiguation) and provides a highly accurate and efficient sentence analyzer for practical natural language systems.

[References]

[Wilks 75] Wilks, Y.A.: An Intelligent Analyzer and Understander of English, Communications of the A.C.M., vol.18, 264-274,(1975)

[Tomita 87] Tomita, M.: An efficient augmented context-free parsing algorithm, Computational Linguistics, Vol.13, (1987)

[Barton 85] Barton, G. E. and Berwick, R. C.: Parsing with Assertion Sets and Information Monotonicity, Proceedings of International Joint Conference of Artificial Intelligence-85, (1985)

[Maruyama 90] Hiroshi, M.: Constraint Dependency Grammar and its weak generative capacity, Computer Software, (1990)

[Hirakawa 89a] Hirakawa. H., Amano, S.: Japanese Sentence analysis using syntactic/semantic preference (in Japanese), In proceedings of the 3rd national conference of JSAI, pp. 363-366, (1989)

[Hirakawa 89b] Hirakawa, H., Amano, S.: Method for Searching Optimum Tree in Japanese Sentence Analysis (in Japanese), IPSJ, Natural Language Processing 74－2, (1989)

[Ibaraki 78] Ibaraki,T.: Branch-and-bounding procedure and state-space representation of combinatorial optimization problems, Information and Control,36,1-27,(1978)

[Karlsson 90] Karlsson, F.: Constraint grammar as a framework for parsing running text, 13th International Conference on Computational Linguistics, Vol.3, pp. 168-173, (1990)

[White 00] White, M. C.: Rapid Grammar Development and Parsing: Constraint Dependency Grammars with Abstract Role Value, PhD Thesis, Purdue University, (2000)

[Heinec 98] Heineck, J., Kunze, J., Menzel W., and Schroder I.,: Eliminative parsing with graded constraints, In Proceedings of the Joint Conference COLING-ACL, pp. 526-530, (1998)

[Beale 96] Beale, S., Nirenburg, S., and Mahesh, K.: HUNTER-GATHERER: Three Search Techniques Integrated for Natural Language Semantics, In Proceedings of AAAI-96, pp.1056-1061,(1998)

[Frederking 94] Frederking, R., Nirenburg, S., Farwell, D., Helmreich, S., Hovy, E., Knight, K., Beale, C., Domashnev, C., Attardo, D., Granners, D., Brown, R.: Integration Translations from Multiple Sources within the Pangloss Mark III Machine Translation System, In Proceedings of the First Conference of the Association for Machine Translation in the Americas. Columbia, Maryland, (1994)

[Farwell 94] Farwell, D., Helmreich, W., Casper, J., M., Hargrave, J., Molina, H., Weng, F.,: PANGLYZER: Spanish Language Analysis System, In Proceedings of the First Conference of the Association for Machine Translation in the Americas. Columbia, Maryland, (1994)

[Ozeki 94] Ozeki, K.: Dependency Structure Analysis as Combinatorial Optimization, Information Sciences 78(1-2), pp.77-99, (1994)

[Kurohashi 94] Kurohashi, S. and Nagao, M.: A Syntactic Analysis Method of Long Japanese Sentences based on the Detection of Conjunctive Structures, Journal of Computational Linguistics, Vol.20, No.4, pp.507-534, (1994)

[Seo 89] Seo, J. and Simmons, R. F.: A Syntactic Graphs: A Representation for the Union of All Ambiguous Parse Trees, Computational Linguistics, Vol.15, (1989)