Graph Branch Algorithm: An Optimum Tree Search Method for Scored Dependency Graph with Arc Co-occurrence Constraints

Hideki Hirakawa
Toshiba R&D Center
1 Komukai Toshiba-cho, Saiwai-ku,
Kawasaki 210, JAPAN
hideki.hirakawa@toshiba.co.jp

Abstract

Various kinds of scored dependency graphs are proposed as packed shared data structures in combination with optimum dependency tree search algorithms. This paper classifies the scored dependency graphs and discusses the specific features of the "Dependency Forest" (DF) which is the packed shared data structure adopted in the "Preference Dependency Grammar" (PDG), and proposes the "Graph Branch Algorithm" for computing the optimum dependency tree from a DF. This paper also reports the experiment showing the computational amount and behavior of the graph branch algorithm.

1 Introduction

The dependency graph (DG) is a packed shared data structure which consists of the nodes corresponding to the words in a sentence and the arcs showing dependency relations between the nodes. The scored DG has preference scores attached to the arcs and is widely used as a basis of the optimum tree search method. For example, the scored DG is used in Japanese Kakari-uke analysis[1] to represent all possible kakari-uke(dependency) trees(Ozeki, 1994),(Hirakawa, 2001). (McDonald et al., 2005) proposed a dependency analysis method using a scored DG and some maximum spanning tree search algorithms. In this method, scores on arcs are computed from a set of features obtained from the dependency trees based on the optimum parameters for scoring dependency arcs obtained by the discriminative learning method.

There are various kinds of dependency analysis methods based on the scored DGs. This paper classifies these methods based on the types of the DGs and the basic well-formed constraints and explains the features of the DF adopted in PDG(Hirakawa, 2006). This paper proposes the graph branch algorithm which searches the optimum dependency tree from a DF based on the branch and bound (B&B) method(Ibaraki, 1978) and reports the experiment showing the computational amount and behavior of the graph branch algorithm. As shown below, the combination of the DF and the graph branch algorithm enables the treatment of non-projective dependency analysis and optimum solution search satisfying the single valence occupation constraint, which are out of the scope of most of the DP(dynamic programming)-based parsing methods.

2 Optimum Tree Search in a Scored DG

2.1 Basic Framework

Figure 1 shows the basic framework of the optimum dependency tree search in a scored DG. In general, nodes in a DG correspond to words in the sentence and the arcs show some kind of dependency relations between nodes. Each arc has

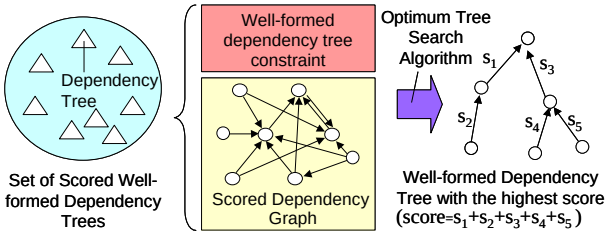

Figure 1: The optimum tree search in a scored DG

[1]Kakari-uke relation, widely adopted in Japanese sentence analysis, is projective dependency relation with a constraint such that the dependent word is located at the left-hand side of its governor word.

Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions, pages 361–368,
Sydney, July 2006.

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 DG and a well-formed dependency tree constraint prescribe a set of well-formed dependency trees. The score of a dependency tree is the sum total of arc scores. The optimum tree is a dependency tree with the highest score in the set of dependency trees.

2.2 Dependency Graph

DGs are classified into some classes based on the types of nodes and arcs. This paper assumes three types of nodes, i.e. word-type, WPP-type[2] and concept-type[3]. The types of DGs are called a word DG, a WPP DG and a concept DG, respectively. DGs are also classified into non-labeled and labeled DGs. There are some types of arc labels such as syntactic label (ex. "subject","object") and semantic label (ex. "agent","target"). Various types of DGs are used in existing systems according to these classifications, such as non-label word DG(Lee and Choi, 1997; Eisner, 1996; McDonald et al., 2005)[4], syntactic-label word DG (Maruyama, 1990), semantic-label word DG(Hirakawa, 2001), non-label WPP DG(Ozeki, 1994; Katoh and Ehara, 1989), syntactic-label WPP DG(Wang and Harper, 2004), semantic-label concept DG(Harada and Mizuno, 2001).

2.3 Well-formedness Constraints and Graph Search Algorithms

There can be a variety of well-formedness constraints from very basic and language-independent constraints to specific and language-dependent constraints. This paper 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(SRC): No two nodes in a dependency tree occupy the same input position

(C3) Projectivity constraint(PJC): No arc crosses another arc[5]

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

(C1) and (C2), collectively referred to as "covering constraint", are basic constraints adopted by almost all dependency parsers. (C3) is adopted by the majority of dependency parsers which are called projective dependency parsers. A projective dependency parser fails to analyze non-projective sentences. (C4) is a basic constraint for valency but is not adopted by the majority of dependency parsers.

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, 1996) and the algorithm based on the B&B method (Hirakawa, 2001), are used for the optimum tree search in scored DGs. The applicability of these algorithms is closely related to the types of DGs and/or well-formedness constraints. The Chu-Liu-Edmonds algorithm is very fast ($O(n^2)$ for sentence length n), but it works correctly only on word DGs. DP-based algorithms can satisfy (C1)-(C3) and run efficiently, but seems not to satisfy (C4) as shown in 2.4.

(C2)-(C4) can be described as a set of co-occurrence constraints between two arcs in a DG. As described in Section 2.6, the DF can represent (C2)-(C4) and more precise constraints because it can handle co-occurrence constraints between two arbitrary arcs in a DG. The graph branch algorithm described in Section 3 can find the optimum tree from the DF.

2.4 SVOC and DP

(Ozeki and Zhang, 1999) proposed the minimum cost partitioning method (MCPM) which is a partitioning computation based on the recurrence equation where the cost of joining two partitions is the cost of these partitions plus the cost of combining these partitions. MCPM is a generalization of (Ozeki, 1994) and (Katoh and Ehara, 1989) which compute the optimum dependency tree in a scored DG. MCPM is also a generalization of the probabilistic CKY algorithm and the Viterbi algo-

[2]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".

[3]One WPP (ex. "time/n") can be categorized into one or more concepts semantically (ex. "time/n/period_time" and "time/n/clock_time").

[4]This does not mean that these algorithms can not handle labeled DGs.

[5]Another condition for projectivity, i.e. "no arc covers top node" is equivalent to the crossing arc constraint if special root node , which is a governor of top node, is introduced at the top (or end) of a sentence.

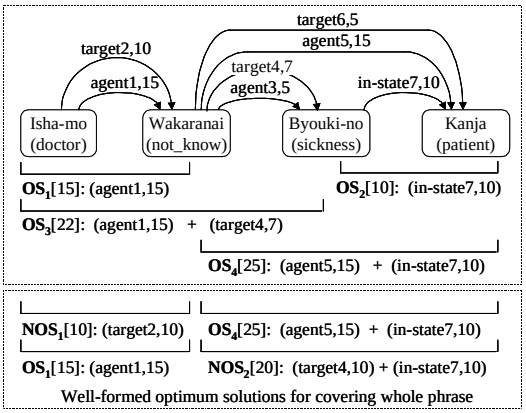

Figure 2: Optimum tree search satisfying SVOC

rithm[6]. The minimum cost partition of the whole sentence is calculated very efficiently by the DP principle. The optimum partitioning obtained by MCPM constitutes a tree covering the whole sentence satisfying the SRC and PJC. However, it is not assured that the SVOC is satisfied by MCPM.

Figure 2 shows a DG for the Japanese phrase "Isha-mo Wakaranai Byouki-no Kanja" 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. 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 $agent1$ and $target4$) for the phrase "Isha-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 SVOC because it has $agent1$ and $agent5$ simultaneously. The optimum solutions satisfying the SVOC are $NOS_1 + OS_4$ and $OS_1 + NOS_2$ shown at the bottom of Figure 2. NOS_1 and NOS_2 are not optimum solutions for their word coverages. This shows that it is not assured that MCPM will obtain the optimum solution satisfying the SVOC.

On the contrary, it is assured that the graph branch algorithm computes the optimum solution(s) satisfying the SVOC because it computes the optimum solution(s) satisfying any co-occurrence constraints in the constraint matrix. It is an open problem whether an algorithm based on the DP framework exists which can handle the SVOC and arbitrary arc co-occurrence constraints.

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

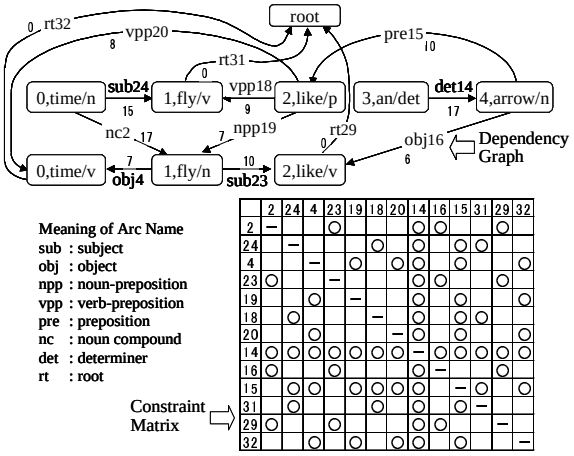

Figure 3: Scored dependency forest

2.5 Semantic Dependency Graph (SDG)

The SDG is a semantic-label word DG designed for Japanese sentence analysis. The optimum tree search algorithm searches for the optimum tree satisfying the well-formed constraints (C1)-(C4) in a SDG(Hirakawa, 2001). 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. Therefore, this method is inherently inapplicable to languages like English that require backward dependency and multiple POS analysis.

The DF described below can be seen as the extension of the SDG. Since the DF has none of the language-dependent premises that the SDG has, it is applicable to English and other languages.

2.6 Dependency Forest (DF)

The DF is a packed shared data structure encompassing all possible dependency trees for a sentence adopted in PDG. The DF consists of a dependency graph (DG) and a constraint matrix (CM). Figure 3 shows a DF for the example sentence "Time flies like an arrow." The DG consists of nodes and directed arcs. A node represents a WPP and an arc shows the dependency relation between nodes. An arc has its ID and preference score. CM is a matrix whose rows and columns are a set of arcs in DG and prescribes the co-occurrence constraint between arcs. Only when CM(i,j) is ○, arc_i and arc_j are co-occurrable in one dependency tree.

The DF is generated by using a phrase structure parser in PDG. PDG grammar rule is an extended CFG rule, which defines the mapping between a sequence of constituents (the body of a CFG rule) and a set of arcs (a partial dependency tree).

The generated CM assures that the parse trees in the parse forest and the dependency trees in the DF have mutual correspondence(Hirakawa, 2006). CM can represent (C2)-(C4) in 2.3 and more precise constraints. For example, PDG can generate a DF encompassing non-projective dependency trees by introducing the grammar rules defining non-projective constructions. This is called the controlled non-projectivity in this paper. Treatment of non-projectivity as described in (Kanahe et al., 1998; Nivre and Nilsson, 2005) is an important topic out of the scope of this paper.

3 The Optimum Tree Search in DF

This section shows the graph branch algorithm based on the B&B principle, which searches for the optimum well-formed tree in a DF by applying problem expansions called graph branching.

3.1 Outline of B&B Method

The B&B method(Ibaraki, 1978) 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)[7], or if it turns out that a partial-problem gives no optimum solutions to the original problem (maximum value test). Usually, the B&B algorithm is constructed to minimize the value of the solution. The graph branch algorithm in this paper is constructed to maximize the score of the solution because the best solution is the maximum tree in the DF.

3.2 Graph Branch Algorithm

The graph branch algorithm is obtained by defining the components of the original B&B skeleton algorithm, i.e. the partial-problem, the feasible solution, the lower bound value, the upper bound value, the branch operation, and so on(Ibaraki, 1978). Figure 4 shows the graph branch algorithm which has been extended from the original B&B skeleton algorithm to search for all the optimum trees in a DF. The following sections explain the B&B components of the graph branch algorithm.

[7]The dominance test is not used in the graph branch algorithm.

```
P0 : Initial problem, Pi : Partial problem,
AP: Active partial problem list,
O : Set of incumbent solutions, z : Incumbent value

start: /* S1(initial value setup) */
  AP := {P0}; z = -1; O := {};
  UB = get_ub(P0); /* Upper bound of P0 */

search_top: /* S2(search) */
  if(AP == {}) { goto exit; }
  else{ Pi := select_problem(AP); }
  /* Compute the feasible solution FS and the lower */
  /* bound LB (= the score of FS) for Pi.          */
  (FS,LB) := get_fs(Pi);
  /* If no feasible solution found, terminate the problem. */
  if(FS == no_solution) { goto terminate_problem; }
  /* S3(incumbent value update): If LB is better than z, */
  /* update incumbent solution and incumbent value.      */
  if(LB > z) { z := LB; O := {FS}; }
  /* S5(upper bound test): */
  if(UB < z) { goto terminate_problem; }
  /* Compute inconsistent arc pair list IAPL. */
  IAPL := get_iapl(Pi);
  /* If lower bound (score of feasible solution) is less */
  /* than upper bound, execute graph branch operation.   */
  if(LB < UB) { BACL := IAPL; goto branch; }
  /* Lower bound equals to upper bound => optimum solution */
  elsif(LB == UB) {
    O := {FS} ∪ O; /* Add this FS as incumbent solution */
    /* S8(search more optimum solutions) */
    /* (a) existence of an inconsistent arc pair */
    if(IAPL != {}) { BACL := IAPL; goto branch; }
    /* (b) existence of a rival arc */
    BACL := arcs_with_alternatives(FS);
    if(BACL != {}) { goto branch; }
    else { goto terminate_problem; } }

branch: /* S6(branching operation) */
  /* Generate child partial problems based on BACL */
  ChildProblemList := graph_branch(Pi,BACL);
  AP := AP U ChildProblemList - {Pi}; goto search_top;

terminate_problem:  /* S7(termination of Pi) */
  AP := AP - {Pi}; goto search_top;

exit: /* S9(stop) */
  if(z == -1) { Problem P0 has no solution }
  else { O is a set of the optimum solutions }
```

Figure 4: Graph branch algorithm

(1) Partial-problem

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

(a) Dependency graph DG_i
(b) Constraint matrix CM_i
(c) Feasible solution value LB_i
(d) Upper bound value UB_i
(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. For example, "$rem[b,d]$" represents a partial dependency graph $[a,c,e]$ in the case $DG = [a,b,c,d,e]$. $IAPL_i$ 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 algorithm, a well-formed dependency tree in the dependency graph DG of the partial-problem P is assigned as the feasible solution FS of P[8]. The score of the feasible solution FS is assigned as the lower bound value LB. The function for computing these values get_fs is called a feasible solution/lower bound value function. The details are not shown due to space limitations, but get_fs is realized by the backtrack-based depth-first search algorithm with the optimization based on the arc scores. get_fs assures that the obtained solution satisfies the covering constraint and the arc co-occurrence constraint. The incumbent value z (the best score so far) is replaced by the LB at $S3$ in Figure 4 if needed.

(3) Algorithm for Obtaining Upper Bound

Given a set of arcs A which is a subset of DG, if the set of dependent nodes[9] of arcs in A satisfies the covering constraint, 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 SRC and does not form a tree. In the graph branch algorithm, the score of the maximum well-covered arc set of a dependency graph G is assigned as the upper bound value UB 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.

(4) Branch Operation

Figure 5 shows a branch operation 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 of the DG of P.

(b) Create child partial-problems P_i, P_j which have new DGs $DG_i = DG - \{arc_j\}$ 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 DG of the child partial-problem is less than that of its parent.

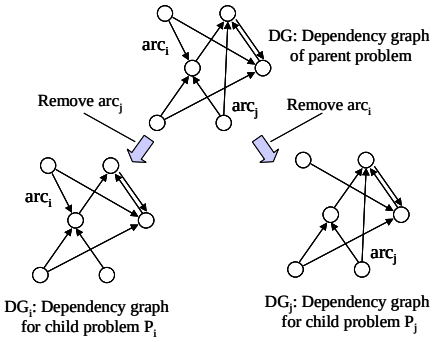

Figure 5: Graph branching

In Figure 4, 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.

(5) Selection of Partial-problem

$select\_problem$ employs the best bound search strategy, 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 $S8$ in Figure 4. 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. 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

[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.

[9] The dependent node of an arc is the node located at the source of the arc.

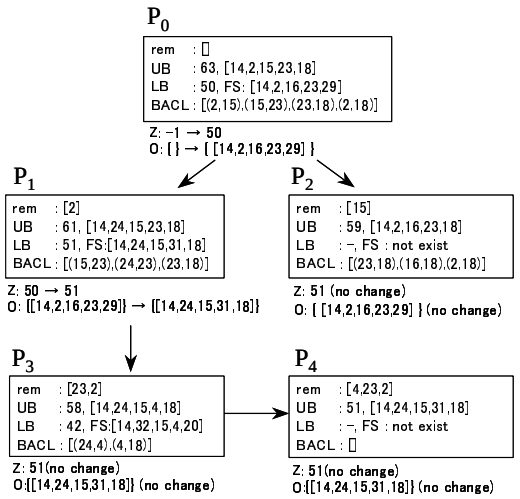

Figure 6: Search diagram

greater than or equal to the score of the optimum solutions when the computation stopped.

4 Example of Optimum Tree Search

This section shows an example of the graph branch algorithm execution using the DF in Fig.3.

4.1 Example of Graph Branch Algorithm

The search process of the B&B method can be shown as a search diagram constructing a partial-problem tree representing the parent-child relation between the partial-problems. Figure 6 is a search diagram for the example DF showing the search process of the graph branch algorithm.

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 Figure 4 are described below.

In $S1(initialize)$, z, O and AP are set to -1, $\{\}$ and $\{P_0\}$ respectively. The DG of P_0 is that of the example DF. 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 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.

P_1 updates z and O because P_1 obtained a feasible solution better than that in O obtained by P_0. P_2 and P_4 are terminated because they have no feasible solution. P_3 generates a feasible solution but z and O are not updated. This is because the obtained feasible solution is inferior to the incumbent solution in O. The optimum solution(=$\{[14, 24, 15, 31, 18]\}$) is obtained by P_1. The computation from P_2 to P_4 is required to assure that the feasible solution of P_1 is optimum.

5 Experiment

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

5.1 Environment and Performance Metric

An existing sentence analysis system[10] (called the oracle system) is used as a generator of the test corpus, the preference knowledge source and the correct answers. Experiment data of 125,320 sentences[11] extracted from English technical docu-

[10] A real-world rule-based machine translation system with a long development history

[11] Sentences ending with a period and parsable by the oracle system.

ments is divided into open data (8605 sentences) and closed data (116,715 sentences). The preference score source, i.e. the WPP frequencies and the dependency relation frequencies are obtained from the closed data. The basic PDG grammar (907 extended CFG rules) is used for generating the DFs for the open data.

The expanded problem number (EPN), a principal computational amount factor of the B&B method, is adopted as the base metric. The following three metrics are used in this experiment.

(a) EPN in total (EPN-T): The number of the expanded problems which are generated in the entire search process.

(b) EPN for the first optimum solution (EPN-F): The number of the expanded problems when the first optimum solution is obtained.

(c) EPN for the last optimum solution (EPN-L): 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 DF, 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 DF 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 and OSN have the relation OSN $\leq$ EPN-F $\leq$ EPN-L $\leq$ EPN-T. Average values for these are described as Ave:OSN, Ave:EPN-F, Ave:EPN-L and Ave:EPN-T.

5.2 Experimental Results

An evaluation experiment for the open data is performed using a prototype PDG system implemented in Prolog. The sentences containing more than 22 words are neglected due to the limitation of Prolog system resources in the parsing process. 4334 sentences out of the remaining 6882 test sentences are parsable. Unparsable sentences (2584 sentences) are simply neglected in this experiment. The arc precision ratio[12] of the oracle

[12]Correct arc ratio with respect to arcs in the output dependency trees (Hirakawa, 2005).

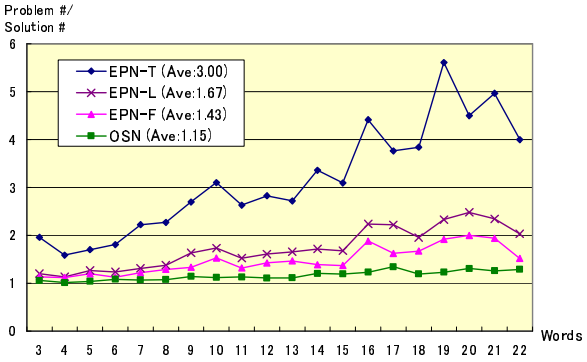

Figure 7: EPN-T, EPN-F EPN-F and OSN

system for 136 sentences in this sentence set is 97.2% with respect to human analysis results.

All optimum trees are computed by the graph branch algorithm described in Section 3.2. Figure 7 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 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.

According to (Hirakawa, 2001), the experiment on the B&B-based algorithm for the SDG shows the overall averages of AVE:EPN-T, AVE:EPN-F are 2.91, 1.33 and the average CPU time is 305.8ms (on EWS). These values are close to those in the experiment based on the graph branch algorithm. Two experiments show a tendency for the optimum solution to be obtained in the early stage of the search process. The graph branch algorithm is expected to obtain the comparable performance with the SDG search algorithm.

(Hirakawa, 2001) introduced the improved upper bound function g'(P) into the B&B-based algorithm for the SDG and found Ave:EPN-T is reduced from 2.91 to 1.82. The same technique is introduced to the graph branch algorithm and has obtained the reduction of the Ave:EPN-T from 3.00 to 2.68.

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. Figure 8 shows the ratios of the sentences obtaining the whole problem expansion, the first optimum solu-

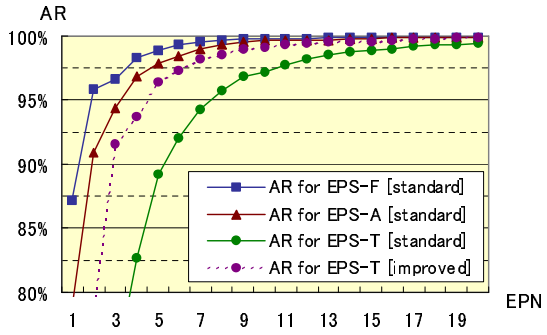

Figure 8: ARs for EPS-F, EPS-A, EPS-T

tion and the last optimum solution to whole sentences with respect to the EPNs. This kind of ratio is called an achievement ratio (AR) in this paper. From Figure 8, the ARs for EPN-T, EPN-L, EPN-F (plotted in solid lines) are 97.1%,99.6%,99.8% respectively at the EPN 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 EPN 10. However, the effect of g'(P) is quite small for EPN-F and EPN-L. This result shows that the pruning strategy based on the EPN is effective and g'(P) works for the reduction of the problems generated in the posterior part of the search processes.

6 Concluding Remarks

This paper has described the graph branch algorithm for obtaining the optimum solution for a DF used in PDG. The well-formedness dependency tree constraints are represented by the constraint matrix of the DF, 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 SVOC which has not been treated in DP-based algorithms so far. 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. The practical code implementation of the graph branch algorithm and its performance evaluation are subjects for future work.

References

Y. J. Chu and T. H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14.

J. Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

J. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of COLING'96*, pages 340–345.

M. Harada and T. Mizuno. 2001. Japanese semantic analysis system sage using edr (in japanese). *Transactions of JSAI*, 16(1):85–93.

H. Hirakawa. 2001. Semantic dependency analysis method for japanese based on optimum tree search algorithm. In *Proceedings of the PACLING2001*.

H. Hirakawa. 2005. Evaluation measures for natural language analyser based on preference dependency grammar. In *Proceedings of the PACLING2005*.

H. Hirakawa. 2006. Preference dependency grammar and its packed shared data structure 'dependency forest' (in japanese). *To appear in Natural Language Processing*, 13(3).

T. Ibaraki. 1978. Branch-and-bounding procedure and state-space representation of combinatorial optimization problems. *Information and Control*, 36,1-27.

S. Kanahe, A. Nasr, and O. Rambow. 1998. Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. In *COLING-ACL'98*, pages 646–52.

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*.

S. Lee and K. S. Choi. 1997. Reestimation and best-first parsing algorithm for probablistic dependency grammars. In *Proceedings of the Fifth Workshop on Very Large Corpora*, pages 41–55.

H. Maruyama. 1990. Constraint dependency grammar and its weak generative capacity. *Computer Software*.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of HLT-EMNLP*, pages 523–530.

J. Nivre and J. Nilsson. 2005. Pseudo-projective dependency parsing. In *ACL-05*, pages 99–106.

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. In *Proceeding of the Workshop of The Fifth Annual Meeting of The Association for Natural Language Processing*, pages 9–14.

K. Ozeki. 1994. Dependency structure analysis as combinatorial optimization. *Information Sciences*, 78(1-2):77–99.

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.