Parsing Paninian Grammar with Nesting Constraints

Akshar Bharati, Ashok K. Gupta and Rajeev Sangal
Department of Computer Science & Engineering
Indian Institute of Technology Kanpur
sangal@iitk.ernet.in

Abstract

Nesting constraints occur in dependency grammar and Paninian Grammar (PG). These constraints specify that the arcs between the verbs and their argument nouns must be properly nested. The complexity of nesting constraint solver was an open problem in PG. The solution to the open problem is outlined in this paper.

The solution is based on dynamic programming. The algorithm finds islands with optimal costs, and combines them into larger islands until the final solution is found (if one exists). Roughly, a smallest island consists of a verb and its argument nouns, with no intervening words related to other verbs. The algorithm is polynomial, though the constant factor is large.

PG and the nesting constraint are important for developing a computational theory for free word-order languages.

1 Introduction

Dependency grammars analyze a sentence in terms of dependency relations between words (Hudson, 1984). For example, dependency structure for a sentence is shown in Figure **??** (only verb noun relations are shown here ignoring determiners and prepositions). Note that the arcs are labelled from a small set of relations called karaka roles in Computational Paninian grammar (PG) following Bharati et al. (1990) (1995) as shown in Figure **??**.

PG has been applied to free word-order languages (Bharati et al., 1993). A grammar in the PG framework, specifies constraints between words in a sentence in terms of their vibhaktis.

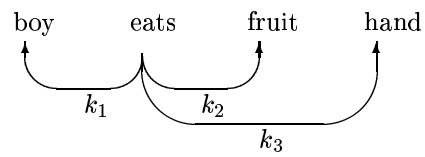

Figure 1: Karaka relations in computational Paninian grammar.

(Vibhaktis stand for collectively: case endings, prepositions, and postpositions.) In the above example sentence, the grammar (or more specifically karaka chart associated with the verb 'eat') specifies the constraints on the vibhaktis of the nouns. A constraint graph can be drawn using the karaka chart by drawing an edge between a verb and a noun that satisfies the vibhakti constraints. A constraint graph shows all the relations between the noun and the verb that are permitted by the vibhaktis. Thus, for the Hindi sentence:

```
 laDakaa   haatha se      phala     khaataa hei.
boy(nom.) hand (instr.)  fruit (nom.)   eats
(The boy eats the fruit with his hand.)
```

we have the constraint graph as shown in Figure **??**.

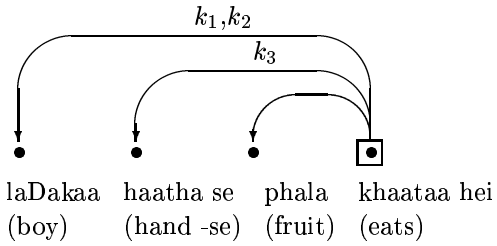

Figure 2: A constraint graph for a free word order language.

A solution graph in PG is a subgraph of the con-

straint graph which satisfies the following three constraints:

1. Each mandatory karaka in the karaka chart for each verb (group), is expressed *exactly once*. (In other words, a given mandatory karaka generates only one noun (group) with the specified vibhakti in its karaka chart.)

2. Each optional karaka in the karaka chart for each verb, is expressed *at most once*.

3. Each source word group (meaning here, a noun (group)) satisfies *some* karaka relation with some verb. In other words, there should no unconnected source word group in a sentence.

After we apply these constraints, one of the solution graphs for the example constraint graph is shown in Figure **??**.

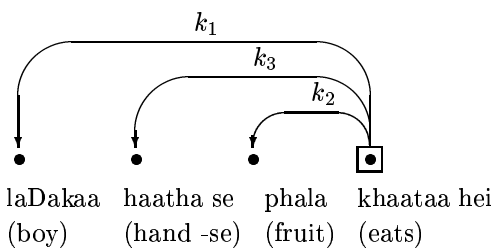

Figure 3: A solution to the previous constraint graph.

It has already been shown by Perraju (1992) that under these three constraints a solution graph can be obtained from the constraint graph in order n^3 (where n is the number of nodes in the graph or roughly the words in the sentence).

There is a fourth constraint called the *nesting constraint* or *half planarity constraint* which also seems to operate in a large number of languages. It says that if there are two or more verbs (or more generally, demand nodes), the arcs must be properly nested. In other words, if arcs are drawn between nodes (placed in a line, in order of occurrence of their corresponding words) using only one side of the plane (half-plane), the arcs can be drawn without crossing each other. This fourth constraint is important while dealing with complex sentences.

To characterize the complexity of parsing, it was posed as an open problem in ACA (1993) and later in Bharati et al. (1995). That problem has been solved and this paper presents an outline of the solution. The full solution is described in Bharati et al. (1995a).

Here, we give the formal statement of the problem. With this, the statement would be independent of the any linguistic theory. In fact, such a problem might occur in other domains too besides NLP. First, we define the constraint graph, which is obtained by applying PG to a language.

DEFINITION (Constraint Graph) A labelled graph G with the following properties is called a constraint graph:

1. G has a set N of n nodes labelled by natural numbers 1 to n.

2. N is partitioned into sets D and S (that is, D $\bigcup$ S = N; D $\bigcap$ S = ϕ).

3. Every edge (d,s) in G is such that d ϵ D, s ϵ S and label of s < label of d. (In other words, nodes in D have zero or more outgoing edges, nodes in S have zero or more incoming edges and if you place the nodes in a straight line in ascending order on label value, then all the edges will go from right to left.)

4. Every edge has a label (call it karaka label) that takes its value from a finite set of symbols $K = \{K_1, ..., K_p\}$. Number of such symbols is a constant independent of any particular constraint graph.

5. Every node $d \epsilon D$ has a pair of sets of symbols (KM, KO) where:
   ($KM \subseteq \{K_1, ..., K_p\}$)
   ($KO \subseteq \{K_1, ..., K_p\}$)
   $KM \bigcap KO = \phi$
   This pair is called its valency structure. (Note: KM names the mandatory karakas, and KO the optional karakas for d.)

We have already seen an example constraint graph earlier in Figure **??**. The valency structure for 'eats' is : ($\{k_1,k_2\}$,$\{k_3\}$) which specifies that k_1 and k_2 are mandatory and k_3 is optional. As an example of constraint graph with more than one demand node see Figure **??**.

Now we look at a parse which satisfies the first three constraints. This parse is given as a solution graph.

Valency of node 3, $v_3 = (\{k_2\}, \{\})$
Valency of node 5, $v_5 = (\{k_1, k_2\}, \{\})$

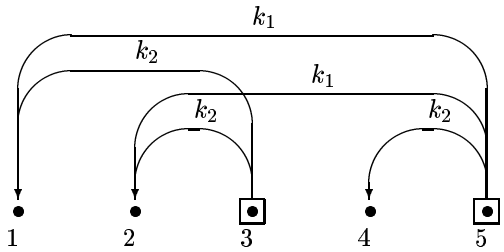

Figure 4: A constraint graph with two demand nodes.

DEFINITION (Solution Graph) A subgraph G' of G containing all the nodes of G is called a solution graph if

1. Every node in S in G' has exactly one incoming edge.
2. Every node d in D in G' with a valency structure (KM, KO) has *exactly one* outgoing arc labelled by each $k \epsilon KM$, and *at most* one outgoing arc labelled by l for each $l \epsilon KO$. (Thus, symbols in KM are mandatorily covered in (karaka) labelling the outgoing arcs, while the symbols in KO are optional. It follows that the karaka labels of the outgoing arcs from each d_i node are distinct.)

As examples consider, Figures **??** and **??** which are the solution graphs for Figure **??**. A solution graph for Figure **??** is shown in Figure **??**.

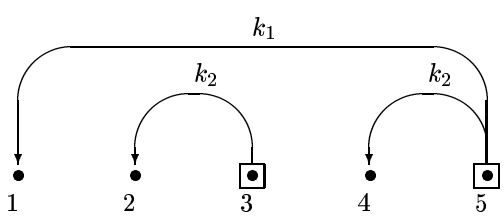

Figure 5: First solution.

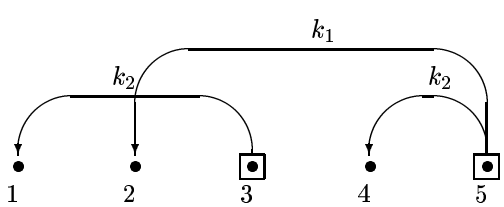

Figure 6: Second solution.

The problem addressed in this paper is that of finding restricted class of solution graphs that are half-planar. Such graphs represent parses which satisfy the nesting constraint.

DEFINITION (Half Planar Solution Graph): If the nodes of a solution graph are placed on an (infinite) straight line on a plane in ascending order of their labels, and all the arcs can be drawn without crossing each other using only half of the plane (as divided by the line), the graph is called half planar. (This property is also called the *nesting property*. It can alternatively be defined as follows: A solution graph is half planar (or satisfies the nesting property) if for any two of its arcs (d_1, s_1) and (d_2, s_2) the following holds: If $label(d_1) < label(d_2)$ then: either (a) $label(d_1) < label(s_2)$, or (b) $label(s_2) < label(s_1)$. In other words, if $label(d_1) < label(d_2)$ then s_2 must not lie in between s_1 and d_1.)

For example, in Figure **??** the solution is half planar whereas the solution in Figure **??** is not.

Now the open problem is to come up with a polynomial time algorithm (polynomial on number of nodes) for finding a half planar solution graph for a given constraint graph, if the algorithm exists. Otherwise prove that the polynomial time algorithm does not exist (ACA, 1993).

1.1 A general overview of the solution

The solution is found using dynamic programming. We will proceed to the solution by finding islands with optimal costs and combine them in the final solution. By *island* we mean a set of source and demand nodes taken sequentially, together with all the edges which start from a demand node in the set and end in a source node in the set. Ignore edges which come to a node in this set from an outside node, or go from here to outside. Every pair of source and demand nodes defines an island, which contains all the nodes lying between them and includes the arcs between these nodes. For each pair of source and demand node, we compute a utility function which depends on the optional and mandatory edges present in the island defined by them. The utility function between the two end nodes (first source node and the last demand node) of the list which make up the big island (the full constraint graph) equals an upper bound if the solution exists.

The given constraint graph will be transformed to a new graph called the *constraint kargraph*. For each demand node and each of the karaka labels in its valency structure, we have a *kardem node* in the transformed graph. An edge between a kardem node and a source node exists *if and only if* there is an edge with that karaka label between the demand node and the source node. If the karak is mandatory, the corresponding *kardem* node is *mandatory kardem node* otherwise it is *optional kardem node*. For example, the constraint graph in Figure **??** will be transformed to the graph shown in Figure **??**. Thus, note that a demand node is transformed into a sequence of kardem nodes, which can come in any order.

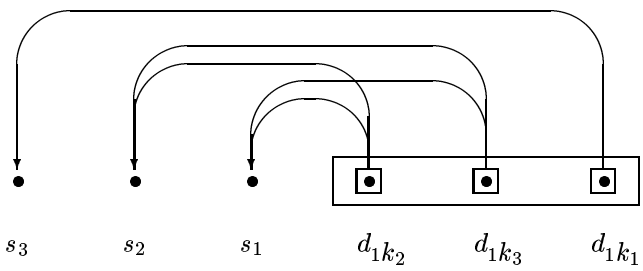

Figure 7: The constraint kargraph.

The solution will be found with respect to the constraint kargraph. The *solution kargraph* will satisfy the following properties:

1. Every source node has exactly one incoming edge.
2. Every mandatory kardem node has exactly one outgoing arc.
3. Every optional kardem node has at most one outgoing arc.
4. The graph satisfies the nesting constraint.

A solution kargraph for the constraint kargraph in Figure **??** is shown in Figure **??**.

1.2 The data structure used in the solution

We have a utility table $U(s_i,d_j)$ which is built up dynamically. The entry in the utility table $U(s_i,d_j)$ consists of two things:

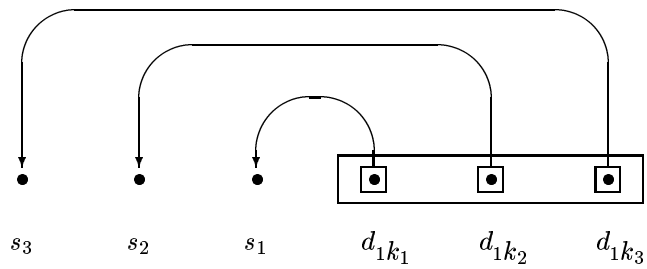

Figure 8: The solution kargraph.

- The cost $cost(s_i,d_j)$ from s_i to d_j which is defined by the optimal set of mandatory and optional edges in the island bounded by them.
- The kardem table $kar_{i,j}$ which is used to get the value of $cost(s_i,d_j)$. The *kardem table* stores the *local utility for each arrangement* of the kardem nodes.

Thus, note that each entry in the utility table contains a small table for each pair of nodes. The size of the kardem table depends on the valency structure of d_j. Besides the above, we also need to store the configuration which gives the optimal cost. Such a configuration can be easily built so we shall not explicitly show it in the procedure given.

1.3 Ordering the nodes

Given an instance of the constraint problem we arrange the nodes of the graph G in a linear order based on the label of the nodes $1 \ldots n$ and number the nodes in the partitions D and S separately as follows :

1. For nodes in set D we number them from left to right $d_1,\ldots,d_p$ where p=$|D|$.
2. For nodes in set S we number them from right to left $s_1,\ldots,s_q$ where q=$|S|$.

Thus, we have a separate ordering of the source and demand nodes. We dynamically compute the utility table using the above orderings.

1.4 Cost function

The value of the cost function between a pair of source and demand nodes is determined by the optimal number of arcs that can be present between them.

The solution is viewed as the optimal set of edges in the big island which is the full constraint graph and is obtained by considering the solution graph as a result of a maximisation problem. To ensure that mandatory arcs are always selected, we assign weights to different types of arcs. The optional arc is assigned a weight W_o of unity and the mandatory arc is assigned a sufficiently large weight W_m which is greater than $|S|$. Thus, each mandatory edge contributes more than all the optional edges put together.

We say that the cost of the island defined by the nodes s_i and d_j is

$$cost(s_i,d_j) = a_{ij} * W_m + b_{ij} * W_o$$

if a_{ij} mandatory edges and b_{ij} optional edges in the island are selected which satisfy the nesting constraint. The cost will be optimal when the maximum possible mandatory edges are selected and hence optimality in this problem is maximising the cost function. We have to find the maximum value of the cost function between d_p and s_q which defines the big island (covering the whole constraint graph).

1.5 Kardem table

Kardem table is needed to handle the fact that kardem nodes for a demand node, can come in any order. Therefore, this table keeps track of all possible permutations of these nodes.

An example kargraph has already been shown in Figure **??**. That example has only one demand node. In general, transformation for all the demand nodes have to be done and we get $|D|$ sets of at most $|K|$ nodes each. Note that the indegree for the source nodes remains unchanged.

We observe that the transformed graphs show the non-intersection property for some arrangement of the kardem nodes if the half-planar solution exists. This gives us the motivation to use the transformed graphs so that we can now use the criterion of non-intersection uniformly over demand nodes as well as kardem nodes.

2 The use of dynamic programming

In case all the karaka symbols are mandatory, the problem becomes trivial. The complexity arises because of the use of optional arcs. The local information is not enough to enable us to make a decision whether we have to keep or discard a particular arc. In case half planar solutions are not required, the problem may be solved using reduction to assignment problem (Perraju, 1992). Half planar graphs with optional arcs ask for backtracking giving us an exponential algorithm. We handle the problem using dynamic programming to compute the optimal solution and give an algorithm which is polynomial in the number of nodes. This is achieved by growing islands consisting of demand nodes and possible assignment of source nodes. Kardem table keeps local informaton associated with each demand node. Details are given in Bharati et al. (1995a).

3 Conclusion

In summary, we have presented an outline of a polynomial algorithm for parsing Computational Paninian Grammar with nesting constraints. Earlier method for solving the above problem was in terms of integer programming. The problem could be reduced to integer programming problem, which could then be solved by standard methods. However, that problem is NP-complete, and therefore, the current best algorithm is exponential. The new proposed algorithm for parsing relies on dynamic programming. Complexity of this algorithm turns out to be polynomial: $O(|S||D|^2)$. A proof is not included for want of space.

Even though the algorithm is polynomial, the constant terms are factorial on the number of karakas ($|K|$). We believe that it should be possible to remove these terms, because currently we also determine the ordering on kardem nodes for each demand node, while that is clearly not required in the solution. This line of research is being pursued.

Solving this and related problems which have actually been obtained from real grammars designed for free word-order languages is important to develop a theory of free word-order languages.

We also believe that development of constraint grammars very naturally generalizes to handling of free word-order as well as positional languages. It becomes a simple matter of selection of appropriate features: The former uses vibhakti feature and the latter uses position feature. Unlike the positional grammars such as CFG, where one pays a price in efficiency and expressibility when applying to free word-order languages, the constraint grammars will be neutral. They would specialize in a natural manner to the two respective classes of languages.

It is hoped that such work motivated by free word-order languages will yield new insight into the language structure (of all languages) on the one hand, and efficient parsing algorithms (for free word-order languages) on the other hand.

References

[1] Bharati, Akshar and Rajeev Sangal, A Karaka Based Approach to Parsing of Indian Languages, In *COLING90: Proc. of Int. Conf. on Computational Linguistics, Helsinki*, Association for Computational Liguistics, NY, August 1990.

[2] Bharati, Akshar and Rajeev Sangal, Parsing Free Word Order Languages using the Paninian Framework, In *ACL93: Proc. of Annual Meeting of Association for Computational Linguistics*. Association for Computational Linguistics, NY, 1993a.

[3] Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, *Natural Language Processing: A Paninian Perspective*, Prentice-Hall, New Delhi, 1995.

[4] Bharati, Akshar, Ashok K. Gupta, and Rajeev Sangal, *Parsing with Nesting Constraints*, Technical Report TRCS-95, Dept. of CSE, IIT Kanpur, 1995a.

[5] Lai, Bong Yeung Tom, Dependency Grammar and the Parsing of Chinese Sentences, *Proceedings of 1994 Joint Conference of 8th ACLIC and 2nd PacFoCoL*, Kyoto, Aug. 10-12, 1994.

[6] Hudson, Richard, *Word Grammar*, Basil Blackwell, Oxford, 1984.

[7] Perraju, Bendapudi V.S., *Algorithmic Aspects of Natural Language Parsing using Paninian Framework*, M.Tech. thesis, Dept. of CSE, IIT Kanpur, Dec. 1992.

[8] ACA, *A Constraint Problem in NLP*, ACA Contest, Association for Computing Activities, Dept. of CSE, IIT Kanpur, 1993.