A Deterministic Dependency Parser with Dynamic Programming for Sanskrit

Amba Kulkarni
Department of Sanskrit Studies
University of Hyderabad
apksh@uohyd.ernet.in

Abstract

We describe a Deterministic Dependency Parser for Sanskrit. The parse is developed following a Depth First traversal of a graph whose nodes represent morphological analyses of the words in a sentence. During the traversal, relations at each node are checked for local compatibility, and finally for each full path, the relations on the path are checked for global compatibility. Stacking of intermediate results guarantees dynamic programming. We also describe an interface that displays multiple parses compactly and facilitates users to select the desired parse among various possible solutions with a maximum of $n-1$ choices for a sentence with n words.

1 Introduction

Past decade has witnessed a lot of dynamism and upsurge of activities in the field of Sanskrit Computational Linguistics. Several computational tools became available to the Sanskrit community as a web service through the internet[1]. With the availability of a wide coverage grammar for Sanskrit in the form of *Aṣṭādhyāyī*, there was a natural tendency to follow the grammar based approach towards the development of these tools (Huet, 2009; Kulkarni et al., 2010; Kulkarni and Ramakrishnamacharyulu, 2013; Goyal and Huet, 2013). Nevertheless, there were also notable efforts to use pure machine learning approaches for building these tools with a small manually tagged corpus as a boot-strap (Hellwig, 2009). At the same time, a combination of the grammar based approach supported by the statistical evidences to push the most likely solution to the top were also followed (Kumar et al., 2010; Kulkarni and Kumar, 2011).

Sanskrit being influenced by the oral tradition, Sanskrit texts are typically written as a continuous string of characters. Characters at the juncture of word boundaries undergo euphonic changes thereby merging the word boundaries. This makes it challenging to split a given string into grammatically acceptable words before taking up the task of parsing. The task of joining two words is deterministic but splitting a string of characters into well-formed words is non-deterministic. This non-determinism together with splits at more than one places in a given string leads to exponential possibilities. Huet (2002, 2009) proposed a novel way of augmenting the nodes of a Finite State Transducer with appropriate sandhi rules, and achieved the segmentation in linear transitions. He also developed a shallow parser using the sub-categorisation frames, and the agreement rules. This parser is useful to rule out the non-solutions before proceeding for the full fledged parsing. A purely statistical parser for Sanskrit also exists (Hellwig, 2009).

The first full fledged parser for Sanskrit based on Pāṇinian Grammar formalism is described in (Kulkarni et al., 2010). This parser is implemented as a constraint solver. In this model, a word in a sentence is represented as a node in a graph G, and the relations between the words as directed labelled edges. The task of parsing a sentence is modelled as finding a sub-graph T of G which is a directed labelled Tree. The problem of parsing is divided into three tasks:

1. The first task is to establish labelled edges between the nodes. The information of expectancy and agreement is used to establish these labelled edges.

2. Next a sub-graph T of G is identified, such that T is a directed Tree which satisfies the

[1] http://sanskrit.uohyd.ernet.in/scl
http://tdil-dc.in/scl
http://sanskrit.inria.fr
http://kjc-fs-cluster.kjc.uni-heidelberg.de/dcs/index.php

Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013), pages 157–166,
Prague, August 27–30, 2013.

following constraints.

- Every node can have at the most one incoming arrow.
- No two edges emerging from the same node have the same label.
- There are no loops.
- The resulting Tree is projective, i.e. if the nodes are arranged linearly according to the word order, then no two links cross each other.
- It is ensured that certain relations which always occur in pairs e.g. anuyogī-pratiyogī (relata-1 and relata-2), kartṛsamānādhikaraṇam-kartā (predicative adjective and subject[2]), etc. do have their counter-relatum present in the parse.

3. Finally in case there is more than one possible directed Tree, the solutions are prioritized.

The implementation of the parser is reported in Kulkarni et al. (2010). The graph G is represented as a 5D matrix C with a typical element $([i,j],R,[l,m])$, where R is the relation from the m^{th} analysis of the l^{th} word to the j^{th} analysis of the i^{th} word. In order to prioritize the solutions, every relation is assigned a weight. A simple Cost function is defined as Cost = $\sum w * |j-i|$, where w is the weight of the relation between the nodes i and j.

The main disadvantage of this approach is the complexity. The size of the 5D matrix is $N * M * K * N * M$, where N is the total number of words in a sentence, M is the maximum number of morph analyses for a word in a given sentence and K is the maximum number of distinct possible relations among the words in a given sentence. Sanskrit words being overloaded with morphological analysis, frequently occurring words tend to have several analyses possible[3]. Similarly though the average length of the sentences[4] is around 10, the sentences from literary texts tend to be longer with more than 20 words.

Sanskrit grammar texts discuss various relations, among words, necessary to interpret the meaning of a sentence. All these relations were compiled and classified by Ramakrishnamacharyulu (2009) and further they were investigated for their suitability for automatic parsing. Out of around 90 relations listed there, only those relations which one can predict based on the syntactico-semantic information available in a sentence are considered for automatic tagging (Kulkarni and Ramakrishnamacharyulu, 2013). There are around 35 of them. Thus R is one of these 35.

As the number of words in a sentence increases, or if a sentence has even a single word with considerable number of morph analyses, the size of the 5D matrix explodes, and the use of parser in real time applications becomes impractical.

Second disadvantage of the above method is that the constraints are applied globally to the matrix. However, we notice that some of the constraints are local to a node. Separating the local constraints from the global, and applying the local constraints at an early stage to rule out non-solutions should increase the efficiency of the system.

The importance and advantage of Dependency Parser over a constituency parser has been well recognised by the computational linguistic community and we see Dependency parsers for variety of languages such as English, Japanese, Swedish to name a few. More than half a dozen parsers exist for English alone that produce dependency parse. The existence of Pāṇinian grammar for Sanskrit is the strong motivation behind developing a Dependency based parser for Sanskrit. The current trend towards developing dependency parsers is more towards following the data driven approaches over the grammar based. However, we follow the grammar based approach. Some of the factors that motivated the design of the parser and choice of the approach are the following.

- Sanskrit does not have a tree bank of reasonable size so that we can use data driven approaches for Sanskrit.
- Sanskrit has a free word order, and hence the traditional POS taggers do not make any sense. Unlike modern Indian languages which are relatively free word order,

[2]We roughly translate kartā as a subject. This is not a faithful translation. Kartā and other kāraka relations represent the semantic information which can be extracted purely from the syntactic information available in the sentence.

[3]The word *te* has 16 possible analyses corresponding to its inflectional analysis. If we take into account the derivational information, the possibilities explode further.

[4]This figure is based on the SHMT corpus developed by the SHMT consortium project sponsored by DeitY, India (2008-12).

and which have a fixed word order for the adjective-substantive sequences, Sanskrit allows even the adjectives and genitives to float around in a sentence. This makes the usability of POS tagger for Sanskrit doubtful.

- The existence of almost exhaustive grammar for Sanskrit also demands from the users a justification for the analysis in terms of grammar rules.

We describe below a Deterministic Parsing algorithm which applies the local constraints locally, and also uses Dynamic Programming for efficient parsing. This parser differs from the Deterministic Dependency Parsers for English developed by Yamada and Matsumoto (Yamada and Matsumoto, 2003) and Nivre (Nivre and Scholz, 2004) in three major ways. These parsers for English use either a bottom-up or a combination of bottom-up and top-down algorithm. Our parser traverses the sentence from left to right guided by the possible paths among the nodes. Second major difference is that these parsers use shift-reduce parsing, while we check the relations for compatibility at each node. The third major difference is that we follow the grammar based approach while the above parsers for English are data driven.

2 Left-Right Deterministic Parsing with Dynamic Programming

Let $G_1 = (N_1, E_1)$ be a graph, where N_1 is the set of nodes corresponding to morphological analyses of the words in a given sentence. E_1 is the set of all directed weighted arcs (i, j, r) such that i^{th} node is related to j^{th} node through a relation r. With every relation r a weight w is associated, which reflects the preferences of relations over the other. The total number of nodes = $\sum m_i$, where m_i is the number of possible morphological analyses of the i^{th} word.

Since a node in N_1 corresponds to a morphological analysis of a word, and not to the word, the constraint to choose only one analysis per word needs an information about how many analyses correspond to each word. This we specify through the adjacency information. For each word we provide indices of the morphological analyses of the word to the left as well as to the right. For the first word, the index of the left word is marked as 'S' (the starting node), and for the last word, the index of the right word is marked as 'F' (the final node).

From(j)	To(i)	relation(r)
2	4	karma (obj)
2	6	adhikaraṇa (loc)
2	7	adhikaraṇa (loc)
5	1	kartā (subj)
5	3	kartā (subj)
5	4	karma (obj)
6	4	karma (obj)
7	4	karma (obj)

Table 1: Possible Relations

This information of adjacency is represented as a graph $G_2 = (N_2, E_2)$, where $N_2 = N \cup \{S, F\}$ and E_2 is the set of directed edges (i, j) such that i and j correspond to the morphological analyses of adjacent words w_k and w_{k+1}. The direction of the edge is from i to j.

2.1 An example

We illustrate with an example the information content of the two graphs G_1 and G_2. Consider the following sentence.

San: *rāmaḥ vanaṁ gacchati.* (1)
gloss: Ram forest{acc.} goes.
Eng: Ram goes to the forest.

In this sentence, each of the two words *rāmaḥ* (Ram) and *vanaṁ* (forest) has two possible analyses, while the word *gacchati* (goes) has 3 possible analyses as shown below.
1. rāmaḥ = rāma {masc.} {sg.} {nom.}
2. rāmaḥ = rā {pr.} {1p} {pl.}
3. vanaṁ = vana {neu.} {sg.} {nom.}
4. vanaṁ = vana {neu.} {sg.} {acc.}
5. gacchati = gam {pr.} {3p.} {sg.}
6. gacchati = gam {pr. part.} {masc.} {sg.} {loc.}
7. gacchati = gam {pr. part.} {neu.} {sg.} {loc.}

Thus, G_1 has 7 nodes. Edges marking the relations are listed in Table 1. This is represented in the form of a graph as shown in Figure 1. The information of adjacency is shown in Table 2 and as a graph in Figure 2.

2.2 Local and Global constraints

A **path** P of a graph G_2 is a sequence of edges which connects the nodes from 'S' to 'F'. For example, S-1-3-5-F is a path in Figure 2.

Node no	node nos of left word	node nos of right word
1	S	3,4
2	S	3,4
3	1,2	5,6,7
4	1,2	5,6,7
5	3,4	F
6	3,4	F
7	3,4	F

Table 2: Adjacency

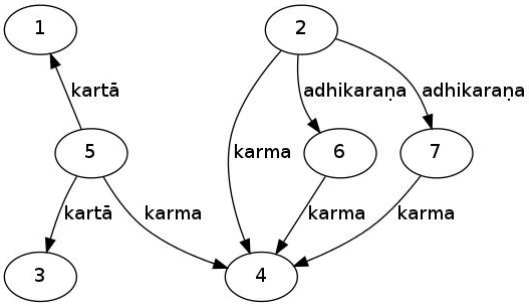

Figure 1: Possible Relations

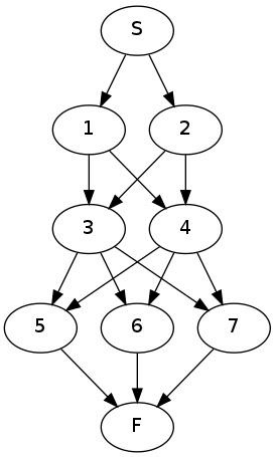

Figure 2: Adjacency and Possible paths

A relation (I, J, R) is **locally incompatible** with a set of relations R = $\{(i, j, r)|(i, j, r) \in E$ of $G_1\}$ under the following circumstances.
a. If for some $(i, j, r) \in E$ and $j = J$ and $r = R$, $i \neq I$. This ensures that no two words satisfy the same semantic role of a verb.
b. If for some $(i, j, r) \in E$ and $i = I$, either $j \neq J$ or $r \neq R$. This ensures that every word has at the most one semantic role[5].

A set of relations $R = \{(i, j, r)|(i, j, r) \in E$ of $G_1\}$ is said to be **locally compatible**, if no $(I, J, R) \in E$ is locally incompatible with the rest of the relations in R.

A set of labelled edges R = $\{(i, j, r)|(i, j, r) \in E$ of $G_1\}$ is **globally compatible** provided the following conditions are satisfied.
a. If the nodes of G_1 are arranged in an increasing order of their index, then the links do not cross.
b. For certain relations r such as *kartṛsamānādhikaraṇam* (predicative adjective) there is a matching relation *kartā* (subject).
c. The edges corresponding to the relations in R do not form a loop.

A sub-graph T of G_1 is a **parse** if
a. The nodes in T correspond to some path of G_2. This ensures that each node in T corresponds to a distinct word, and every word in the sentence is accounted for.
b. T is a Tree. This ensures that every word in a sentence is related directly to some other word.
c. The set of relations corresponding to the edge labels in T are both locally as well as globally compatible.

2.3 Parsing Algorithm

1. Starting from the node 'S' of the graph G_2 explore various paths of G_2 following the Depth-First-Traversal strategy. The stack keeps track of part of the paths visited so far.

2. At each node, refer to G_1 for various relations this node can have with other nodes. The stack, in our case, in addition to the information of paths visited so far, also keeps track of compatible relations at various nodes on this path.

[5]This condition is applied to only a few relations such as kartā (agent), karaṇa (instrument), etc.

3. For each of these relations
    - If the relation is locally compatible with the relations encountered on this path so far, add this relation to the stack and continue with the next node. The set of relations, at any point of time on the stack provides the current status of the partial solution / Tree explored.
    - If the relation is incompatible, declare this path to be incompatible, and proceed with other path, leaving this path further unexplored.
4. When you reach the node 'F', check the relations on this path for 'global compatibility'.
5. Each globally compatible set of relations, which is a sub-set of edges in G_1, forms a Tree and hence is a possible solution.
6. For each possible solution, compute the $Cost = \sum w * |j - i|$, where w is the weight associated with the relation between the nodes corresponding the j^{th} and i^{th} word.

Traversing of the graph G_2 from 'S' to 'F' is equivalent to traversing the sentence from left to right for various combinations of morphological analyses. The parser is deterministic, and it is guaranteed to terminate after $\prod m_i$ paths are explored. At each node of G_2, the number of compatibility checks is equal to the number of incoming arrows at that node in graph G_1. Stacking of intermediate results ensures the dynamic programming. A Cost function is used to prioritize the solutions. Salient features of this algorithm are:

- We follow lattice programming to explore all possible paths.
- The parser deals with one word at a time starting from the first word. This is motivated by an approach [6] in the Indian theories of verbal cogniton and also confirms with Abney's (Abney, 1989) findings that the human operates this way.
- Since we do not want to miss any possible parse, we use dynamic programming which is upto 5 times faster than the conventional beam search (Huang and Sagae, 2010).
- Unlike the traditional based parsing which are typically breadth first, we follow a depth first strategy, stacking the intermediate results ensuring the effective dynamic programming.
- At each node we use constraints to check the compatibility of new relations with the stacked one.
- The weights for each relation are determined heuristically manually in the absence of any manually tagged reasonable sized data. The n-v relations expressing the kāraka (case) relations are given preferences over the non-kāraka relations.

2.4 Evaluation

Sanskrit Tree bank corpus is developed under Government of India sponsored project 'Development of Sanskrit Computational Toolkit and Sanskrit Hindi Machine Translation system (2008-2012)'. The corpus consists of around 3000 sentences, a substantial part of it being modern short stories. A small part of the corpus contains sentences addressing various syntactic phenomena. The complete tagged corpus is still being cross checked for correctness. Hence the parser was tested only on 1316 sentences. We have a hierarchical tagset with 35 tags. Among these the subclassification of 4 types of location (adhikaraṇa) and 3 types of objects (karma) is collapsed into one each resulting into a flat tagset of 30 tags.
Our parser produces all possible parses, ordered on cost. The one with minimum cost is shown as the first parse. For evaluation, we consider only the first parse. The correctness of parses is judged on several well established parameters.

- Relations with correct label and attachment (LAS)
  With 35 relations, the labelled attachments were correct in 63.1% cases, while with 30 relations, the score was 67.4%.
- Relations with correct attachment (UAS)
  If only attachments were considered, ignoring the labels, 80.26% attachments matched with the GOLD data.
- Sentences with matching dependency trees (MDT)
  This measure tells us in exactly how many cases the first tree matches the manually

[6] ādyaṃ padaṃ vākyaṃ

tagged tree. Out of 1316 sentences, the first parse matched exactly in 569 (43.20%) sentences with a tagset of 35 tags, while with 30 tags, the first parse matched in 647 (49.1%) cases.

- Sentences with correct unlabelled dependency trees (UDT)
  Instead of complete tree match, now we check only for the attachments, and not the labels. Among 1316, the unlabelled dependency trees matched in 870 (66.05%) cases.

- Sentences with one wrong attachment (OWAS)
  It was found that out of 1316 sentences, 285 (21.6%) sentences had only one wrong labelled attachment. If this is rectified, the performance of the system for correct matches increases drastically.

3 Compact Display of Multiple Solutions

Sanskrit being a classical language demands certain special features with respect to its computational tools. Being an old classical language, most of the important texts in Sanskrit have been translated manually into several modern languages. So naturally, machine translation takes a back seat for Sanskrit. What a user needs is an access to the original text with the help of various online linguistic tools and resources so that he can himself interpret and understand the texts in original. From this aspect, displaying only the first parse does not serve the purpose. In fact, in more than 50% of the cases, the first parse is wrong. User might like to examine various possibilities and choose his own interpretation. It is also possible that the text is ambiguous with two or more readings, and user would like to go through each of them. Displaying all the parse trees would not serve any purpose, since the trees look almost similar with either a change in one or two branches, or with a change in the label.

In what follows we present a compact way of presenting all the solutions. This is an adaptation of the slim interface of Heritage segmenter (Huet and Goyal, 2013).

Let $T_i = (N_i, E_i)$, where $i = 1$ to n be n parses of a given sentence. Let $N = \cup N_i$, and $E = \cup E_i$. The display consists of 3 rows. The top row lists the words with their positions. The second row consists of morphological analyses corresponding to all the nodes in N. Analyses are written in n columns corresponding to each word. The third row consists of edges from E again displayed below the corresponding word/node.

The user can now choose either a node from the second row or an edge from the third row. Each choice calls the compatibility checker to remove the incompatible nodes and edges corresponding to the user's choice. Each choice results in the reduction of possible parses. At any point in time, a user can choose to display the graphs of current possible parses.

Here is an illustration of the interface. The input sentence is an anvaya of a śloka from Bhagvadgītā (8^{th} śloka from the 4^{th} chapter). The original śloka is
paritrāṇāya sādhūnām vināśāya ca duṣkṛtām dharma-samsthāpanārthāya sambhavāmi yuge yuge. (Bh.G.4.8)

The anvaya, an input to the parser, is:

sādhūnām paritrāṇāya duṣkṛtām vināśāya dharma-samsthāpanāya[7] *ca yuge yuge sambhavāmi*.

Fig 3 shows the summary of parses as a compact display[8]. The union of relations from all parses for each word are shown. User can choose either the correct morphological analysis or correct relation corresponding to the node. When he chooses the correct morphological analysis, all the relations in the relations row that are incompatible with this choice are removed from the display. Similarly, if a user chooses a relation in the relation row, all the relations that are incompatible with this relation, and all the morphological analyses that are incompatible with this choice of a relation are removed from the display. Thus, for example, the word *sādhūnām* has two morphological analyses in Fig 3. But, after selecting the appropriate analysis, in Fig 4, we notice that the relations under this word are also reduced. All those relations which has *sādhūnām* as one of the relata are removed from the display. Similarly, selecting the role of this word as *karma,2,2* (karma of the second analysis of the second word), not only removes all other relations below this word, but also removes the first

[7]The original word is dharma-samsthāpanārthāya, which we changed to dharma-samsthāpanāya, since the former is still not recognised by the morphological analyser.

[8]The display shows only first five columns.

Summary of Complete Parses

total filtered solutions = 556 of 26880

✓ Undo ✓ 556 filtered trees

sādhūnām(1)	paritrāṇāya(2)	duṣkṛtām(3)	vināśāya(4)	dharma-saṃsthāpanāya(5)
1. ✓ sādhu{puṃ}{6;bahu} 2. ✓ sādhu{napuṃ}{6;bahu}	1. ✓ pari_traiḷ{kta;traiṅ;bhvādiḥ;napuṃ}{4;eka} 2. ✓ pari_traiḷ{lyuṭ;traiṅ;bhvādiḥ}{napuṃ}{4;eka}	1. ✓ duskrt{puṃ}{6;bahu}	1. ✓ vināśa{puṃ}{4;eka}	1. ✓ dharma-saṃsthāpana{napuṃ}{4;eka}
1. ✓ kartā,2,1 2. ✓ kartā,2,2 3. ✓ kartā,2,1 4. ✓ kartā,2,2 5. ✓ karma,2,1 6. ✓ karma,2,2 7. ✓ karma,2,1 8. ✓ karma,2,2 9. ✓ sasthīsambandhah,2,1 10. ✓ sasthīsambandhah,2,2 11. ✓ sasthīsambandhah,2,1 12. ✓ sasthīsambandhah,2,2 13. ✓ viśesanam,3,1	1. ✓ tādarthyam,3,1 2. ✓ tādarthyam,3,1 3. ✓ samuccitamprayojanam,6,1 4. ✓ samuccitamprayojanam,6,1 5. ✓ viśesanam,5,1 6. ✓ viśesanam,5,1 7. ✓ prayojanam,9,1 8. ✓ prayojanam,9,1	1. ✓ sasthīsambandhah,4,1	1. ✓ samuccitamprayojanam,6,1 2. ✓ tādarthyam,5,1 3. ✓ prayojanam,9,1 4. ✓ prayojanam,2,1 5. ✓ prayojanam,2,2	1. ✓ samuccitamprayojanam,6,1 2. ✓ prayojanam,9,1 3. ✓ prayojanam,2,1 4. ✓ prayojanam,2,2

Figure 3: compact display of solutions

Summary of Complete Parses

total filtered solutions = 556 of 26880

✓ Undo ✓ 292 filtered trees

sādhūnām(1)	paritrāṇāya(2)	duṣkṛtām(3)	vināśāya(4)	dharma-saṃsthāpanāya(5)
1. ✓ sādhu{puṃ}{6;bahu}	1. ✓ pari_traiḷ{kta;traiṅ;bhvādiḥ;napuṃ}{4;eka} 2. ✓ pari_traiḷ{lyuṭ;traiṅ;bhvādiḥ}{napuṃ}{4;eka}	1. ✓ duskrt{puṃ}{6;bahu}	1. ✓ vināśa{puṃ}{4;eka}	1. ✓ dharma-saṃsthāpana{napuṃ}{4;eka}
1. ✓ kartā,2,1 2. ✓ kartā,2,2 3. ✓ karma,2,1 4. ✓ karma,2,2 5. ✓ sasthīsambandhah,2,1 6. ✓ sasthīsambandhah,2,2 7. ✓ viśesanam,3,1	1. ✓ tādarthyam,3,1 2. ✓ tādarthyam,3,1 3. ✓ samuccitamprayojanam,6,1 4. ✓ samuccitamprayojanam,6,1 5. ✓ viśesanam,5,1 6. ✓ viśesanam,5,1 7. ✓ prayojanam,9,1 8. ✓ prayojanam,9,1	1. ✓ sasthīsambandhah,4,1	1. ✓ samuccitamprayojanam,6,1 2. ✓ tādarthyam,5,1 3. ✓ prayojanam,9,1 4. ✓ prayojanam,2,1 5. ✓ prayojanam,2,2	1. ✓ samuccitamprayojanam,6,1 2. ✓ prayojanam,9,1 3. ✓ prayojanam,2,1 4. ✓ prayojanam,2,2

Figure 4: Selection of a morphological analysis

Summary of Complete Parses

total filtered solutions = 556 of 26880

✓ Undo ✓ 44 filtered trees

sādhūnām(1)	paritrāṇāya(2)	duṣkṛtām(3)	vināśāya(4)	dharma-saṃsthāpanāya(5)
1. ✓ sādhu{puṃ}{6;bahu}	1. ✓ pari_traiḷ{lyuṭ;traiṅ;bhvādiḥ}{napuṃ}{4;eka}	1. ✓ duskrt{puṃ}{6;bahu}	1. ✓ vināśa{puṃ}{4;eka}	1. ✓ dharma-saṃsthāpana{napuṃ}{4;eka}
1. ✓ karma,2,2	1. ✓ tādarthyam,3,1 2. ✓ samuccitamprayojanam,6,1 3. ✓ viśesanam,5,1 4. ✓ prayojanam,9,1	1. ✓ sasthīsambandhah,4,1	1. ✓ samuccitamprayojanam,6,1 2. ✓ tādarthyam,5,1 3. ✓ prayojanam,9,1 4. ✓ prayojanam,2,2	1. ✓ samuccitamprayojanam,6,1 2. ✓ prayojanam,9,1 3. ✓ prayojanam,2,2

Figure 5: selection of a relation

Summary of Complete Parses

total filtered solutions = 556 of 26880

✓ Undo ✓ Unique parse tree ✓ Save ✓ Translate into hindi

sādhūnām(1)	paritrāṇāya(2)	duṣkṛtām(3)	vināśāya(4)	dharma-saṃsthāpanāya(5)
1. ✓ sādhu{puṃ}{6;bahu}	1. ✓ pari_traiḷ{lyuṭ;traiṅ;bhvādiḥ}{napuṃ}{4;eka}	1. ✓ duskrt{puṃ}{6;bahu}	1. ✓ vināśa{puṃ}{4;eka}	1. ✓ dharma-saṃsthāpana{napuṃ}{4;eka}
1. ✓ karma,2,2	1. ✓ prayojanam,9,1	1. ✓ sasthīsambandhah,4,1	1. ✓ prayojanam,9,1	1. ✓ prayojanam,9,1

Figure 6: Unique solution

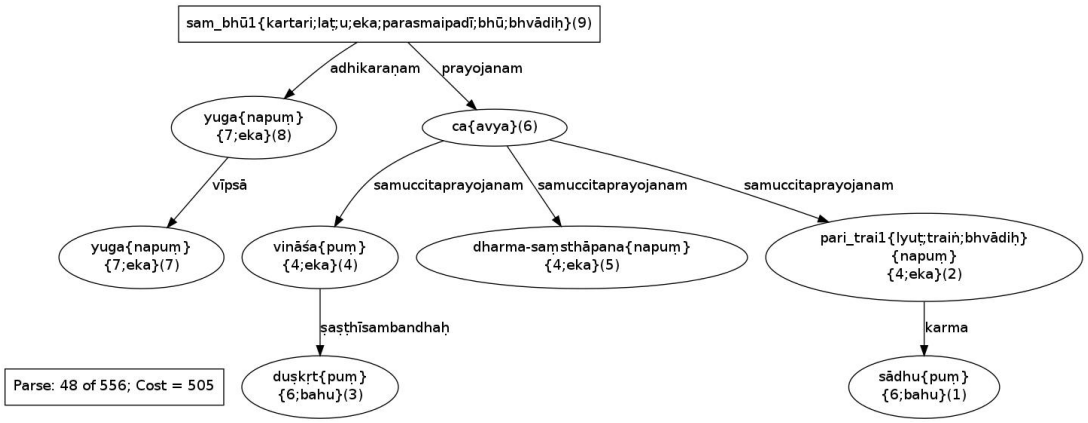

Figure 7: Dependency Graph

morph analysis of the second word, and all the relations having this analysis as one of the relata. The result of this is shown in Fig 5. Finally when we make all the choices, a unique parse is obtained (see Fig 6). Clicking on the check sign of unique parse, we get the rendering of the relations in the form of a dependency graph (see Fig 7).

The parse of a sentence with n words has $n-1$ edges corresponding to the relations. Hence one can choose the correct parse from this compact display in maximum $n-1$ choices. This interface thus can also be used for developing a tree bank for Sanskrit semi-automatically. Due to the limitations on space, we do not give the technical details of this interface here.

4 Using Shallow Parser for pruning

Normally the parsers for positional languages like English use a POS tagger to choose the morphological analysis in context before proceeding for the parsing. This reduces the search space of the parser substantially resulting in increase in the performance metric.

In case of Sanskrit which is morphologically rich and carries very little information in position, the POS taggers based on the positional information are of little value. On the other hand a shallow parser such as one developed by (Huet, 2007) makes sense. Because such a shallow parser, based on the agreement rules, subcategorisation of verbs into transitive and intransitive, co-ordination information, and certain restrictions on grammatical constructions rules out various possibilities and produces a sub-set of possible solutions. Thus it is desirable to use a shallow parser to filter out nonsensical combinations of the solutions before proceeding to a full fledged parsing. This shallow parser, in addition to resolving POS ambiguities, also does a little parsing to aid the full fledged parser.

As an example, consider the sentence (1) above. We have seen that there are 12 possible paths (Fig 2) for this sentence. The shallow parser produces two splits.

1. First split
   - 1. rāmaḥ = rāma {masc.} {sg.} {nom.}
   - 3. vanaṁ = vana {neu.} {sg.} {nom.}
   - 4. vanaṁ = vana {neu.} {sg.} {acc.}
   - 5. gacchati = gam {pr.} {3p.} {sg.}

   This corresponds to 2 paths: S-1-3-5-F and S-1-4-5-F (See Fig 8).

2. Second split
   - 2. rāmaḥ = rā {present tense} {1 per.} {pl.}
   - 3. vanaṁ = vana {neu.} {sg.} {nom.}
   - 4. vanaṁ = vana {neu.} {sg.} {acc.}
   - 6. gacchati = gam {pr. part.} {masc.} {sg.} {loc.}
   - 7. gacchati = gam {pr. part.} {neu.} {sg.} {loc.}

   This corresponds to 4 paths viz. S-2-3-6-F, S-2-3-7-F, S-2-4-6-F, and S-2-4-7-F (See Fig 8).

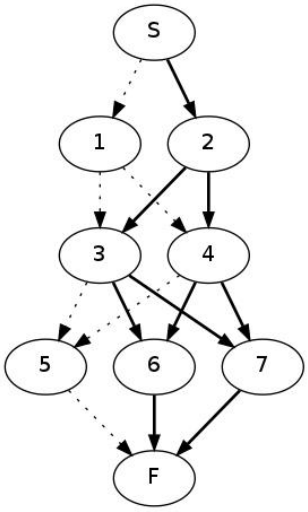

Figure 8: Partitioning of a graph

Thus the shallow parsing has reduced the number of paths from 12 to 6. Note that the POS ambiguities in the words *rāmaḥ* and *gacchati* are resolved. But the case ambiguity in the word *vanam* is not yet resolved.

5 Conclusion

The performance of the parser has confirmed our intuition that application of local constraints at an early stage improves the performance. The search space is further reduced by the use of shallow parser. Compact display is useful for a reader who wants to understand the text in original. This display can also be used for developing Sanskrit Tree bank semi-automatically. The algorithm described above is tested on Sanskrit. However it is general one and should work well for the modern Indian languages as well.

References

Steven P Abney. 1989. A computational model of human parsing. *Journal of Psycholinguistic Research*, 18:129–144.

Vāman Shivarām Apte. 1885. *The Student's Guide to Sanskrit Composition. A Treatise on Sanskrit Syntax for Use of Schools and Colleges*. Lokasamgraha Press, Poona, India.

Akshar Bharati, Vineet Chaitanya, and Rajeev Sangal. 1995. *Natural Language Processing. A Paninian Perspective*. Prentice-Hall of India, New Delhi.

Pawan Goyal and Gérard Huet. 2013. Completeness analysis of a Sanskrit reader. In *Proceedings, 5th International Symposium on Sanskrit Computational Linguistics*. D. K. Printworld(P) Ltd.

Pawan Goyal, Vipul Arora, and Laxmidhar Behera. 2009. Analysis of Sanskrit text: Parsing and semantic relations. In Gérard Huet, Amba Kulkarni, and Peter Scharf, editors, *Sanskrit Computational Linguistics 1 & 2*, pages 200–218. Springer-Verlag LNAI 5402.

Oliver Hellwig. 2009. Extracting dependency trees from Sanskrit texts. In Amba Kulkarni and Gérard Huet, editors, *Sanskrit Computational Linguistics 3*, pages 106–115. Springer-Verlag LNAI 5406.

L. Huang and K. Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics Uppsala, Sweden, July. Association for Computational Linguistics*, page 10771086.

Geŕard Huet and Pawan Goyal. 2013. Design of a lean interface for sanskrit corpus annotation. In *personal communication*.

Gérard Huet, Amba Kulkarni, and Peter Scharf, editors. 2009. *Sanskrit Computational Linguistics 1 & 2*. Springer-Verlag LNAI 5402.

Gérard Huet. 2007. Shallow syntax analysis in Sanskrit guided by semantic nets constraints. In *Proceedings of the 2006 International Workshop on Research Issues in Digital Libraries*, New York, NY, USA. ACM.

Gérard Huet. 2009. Formal structure of Sanskrit text: Requirements analysis for a mechanical Sanskrit processor. In Gérard Huet, Amba Kulkarni, and Peter Scharf, editors, *Sanskrit Computational Linguistics 1 & 2*. Springer-Verlag LNAI 5402.

S.D. Joshi, J.A.F. Roodbergen, and Bhandarkar Oriental Research Institute. 1990. *Patañjali's Vyākaraṇa-Mahābhāṣya Sthānivadbhāvāhnika: introduction, text, translation and notes*. Number v. 1 in Research Unit series. Bhandarkar Oriental Research Institute.

S.D. Joshi, J.A.F. Roodbergen, and Sāhitya Akādemī. 2004. *The Aṣṭādhyāyī of Pāṇini with Translation and Explanatory Notes*. Number v. 11 in The Aṣṭādhyāyī of Pāṇini. Sahitya Akademi.

Amba Kulkarni and Gérard Huet, editors. 2009. *Sanskrit Computational Linguistics 3*. Springer-Verlag LNAI 5406.

Amba Kulkarni and Anil Kumar. 2011. Statistical constituency parser for Sanskrit compounds. In *Proceedings of ICON 2011*. Macmillan Advanced Research Series, Macmillan Publishers India Ltd.

Amba Kulkarni and K. V. Ramakrishnamacharyulu. 2013. Parsing Sanskrit texts: Some relation specific issues. In Malhar Kulkarni, editor, *Proceedings of the 5th International Sanskrit Computational Linguistics Symposium*. D. K. Printworld(P) Ltd.

Amba Kulkarni and Devanand Shukl. 2009. Sanskrit morphological analyser: Some issues. *Indian Linguistics*, 70(1-4):169–177.

Amba Kulkarni, Sheetal Pokar, and Devanand Shukl. 2010. Designing a constraint based parser for Sanskrit. In G N Jha, editor, *Proceedings of the 4th International Sanskrit Computational Linguistics Symposium*. Springer-Verlag LNAI 6465.

Anil Kumar, Vipul Mittal, and Amba Kulkarni. 2010. Sanskrit compound processor. In G N Jha, editor, *Proceedings of the 4th International Sanskrit Computational Linguistics Symposium*. Springer-Verlag LNAI 6465.

J. Nivre and M. Scholz. 2004. Deterministic dependency parsing of english text. In *Proceedings of COLING 2004, Geneva, Switzerland, 64-70*.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT, pages 195-206, Nancy, France.*