Statistical Constituency parser for Sanskrit compounds

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

Anil Kumar
Deptartment of Sanskrit Studies
University of Hyderabad
Hyderabad
anil.lalit22@gmail.com

Abstract

Sanskrit is very rich in compound formation. Typically a compound does not code the relation between its components explicitly. To understand the meaning of a compound, it is necessary to identify its components, identify the way the components group together, discover the relations between them and finally generate a paraphrase of the compound. In this paper, we discuss our efforts in building a constituency parser for Sanskrit compounds. The average performance of this parser is 85%.

1 Introduction

Sanskrit is very rich in compound formation. The compound formation being productive, forms an open-set and as such it is not possible to list all the compounds in a dictionary. The compound formation involves a mandatory *sandhi*[1]. But mere *sandhi* splitting does not help a reader in identifying the meaning of a compound. Typically a compound does not code the relation between its components explicitly. To understand the meaning of a compound, thus, it is necessary to identify its components, identify the way the components are grouped together, discover the relations between them, and finally produce a paraphrase(*vigrahavākya*[2]) of the compound.

There have been significant efforts in the area of segmentation (Huet 2009, Mittal 2010), paraphrase generation(Kumar, 2009) and compound type identification(Kumar, 2010) in the past few years. It has been pointed out by Huet(2009) that the number of ways the constituents of a compound can be grouped is a catalan number. This shows the complexity involved in parsing and the feeble chances of getting a correct parse as the number of components in a compound increases.

In what follows, we first describe the compound formation in Sanskrit and the reasons behind the complexity in compound processing. In the third section, we present a model for processing Sanskrit compounds. In the fourth secion we present some statistical facts about the manually tagged corpus and we describe our approach to constituency parsing taking advantage of the tagged corpus. Fourth section presents the results of the parser along with the precision, recall and F-measure for compounds with different number of components. Finally we conclude by pointing out various improvements and the significance of this work in the broad context of handling Multi Word Expression (MWE) in Indian Languages that may lead to better results.

2 Sanskrit Compounds

The Sanskrit word *samāsaḥ* for a compound means *samasanam* which means "combina-

[1] *Sandhi* means euphony transformation of words when they are consecutively pronounced. Typically when a word w_1 is followed by a word w_2, some terminal segment of w_1 merges with some initial segment of w_2 to be replaced by a "smoothed" phonetic interpolation, corresponding to minimising the energy necessary to reconfigure the vocal organs at the juncture between the words.(Huet, 2006)

[2] An expression providing the meaning of a compound is called a *vigrahavākya*.

"Proceedings of ICON-2011: 9th International Conference on Natural Language Processing, Macmillan Publishers, India. Also accessible from http://ltrc.iiit.ac.in/proceedings/ICON-2011"

tion of more than one word into one word which conveys the same meaning as that of the collection of the component words together". While combining the components together, a compound undergoes certain operations such as loss of case suffixes, loss of accent, etc.. Kumar (2010) describes various features a Sanskrit compound may have. One of the important features among them is that the word order of components in a compound is fixed.
Though compounds of 2 or 3 words are more frequent, compounds involving more than 3 constituent words with some compounds even running through pages is not rare in Sanskrit literature. Here are some examples of Sanskrit compounds involving more than 3 words.

1. वेदवेदाङ्गतत्त्वज्ञः[3] = वेद–वेद–अङ्ग–तत्त्व–ज्ञः
(vedavedāṅgatattvajñaḥ = veda-veda-aṅga-tattva-jñaḥ)

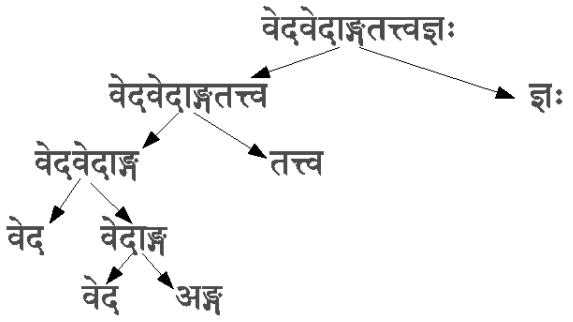

Figure 1: Constituency representation of वेदवेदाङ्गतत्त्वज्ञः

2. प्रवरमुकुटमणिमरीचिमञ्जरीचयचर्चितचरणयुगलम्[4] = प्रवर–मुकुट–मणि–मरीचि–मञ्जरी–चय–चर्चित–चरण–युगलम्
(pravaramukuṭamaṇimarīcimañjarīcayacarcitacaraṇayugalam = pravara-mukuṭa-maṇi-marīci-mañjarī-caya-carcita-caraṇa-yugalam)

3. जलादिव्यापकपृथिवीत्वाभावप्रतियोगिपृथिवीत्ववती[5] = जल–आदि–व्यापक–पृथिवीत्व–अभाव–प्रतियोगि–पृथिवीत्ववती
(jalādivyāpakapṛthivītvābhāvapratiyogipṛthivītvavatī = jala-ādi-vyāpaka-pṛthivītva-ābhāva-pratiyogi-pṛthivītvavatī)

[3] *Rāmāyaṇam 1-1-14*
[4] Paṇctantra - Kathāmukham
[5] Kevalavyatirekī Prakarṇam : Maṇikaṇaḥ P. 22

The compounds are formed with two words at a time and hence they can be represented faithfully as a binary tree, as shown in figure 1.

3 Compound Processor

Understanding a compound involves

1. Segmentation,
2. Constituency Parsing,
3. Compound-Type-Identification and
4. Paraphrasing.

3.1 Segmenter

The task of this module is to split a compound into its constituents. For instance, the compound

sumitrānandavardhanaḥ
is segmented as
sumitrā-ānanda-vardhanaḥ
(Sumitra-joy-growth)

Each of the constituent component of a compound except the last one is typically a compounding form (a bound morpheme)[6].
A Segmenter handling segmentation of compounds should therefore handle these special forms. A segmenter augmenting the nodes of a Finite State Transducer (FST) by the sandhi rules (Euphonic changes) is described by Huet(2006). Another segmenter is developed by Mittal(2010)(Kumar 2010) which is optimised for sandhi and samāsa separately. This segmenter uses the statistical model of the annotated corpus for ranking the possible segmentations.

3.2 Constituency Parser

Constituency parser takes an output of the segmenter and produces a binary tree showing the syntactic composition of a compound corresponding to each of the possible segmentations. Each of these compositions show the possible ways various segments can be grouped. To illustrate various possible parses that result from a single segmentation, consider the segmentation a-b-c of a compound. A compound being binary, the three components a-b-c may be grouped in two ways as

[6] with an exception of components of an *'aluk'* compound.

<*a*-<*b*-*c*>> or <<*a*-*b*>-*c*>. Only one of the ways of grouping may be correct in a given context as illustrated by the following two examples.

1. <*eka*-<*priya*-*darśanaḥ*>>
   (<one - <who is dear to all>>)

2. <<*tapas*-*svādhyāya* >-*niratam*>
   (<<Penance-self-study>-constantly engaged >)

With 3 components, only these two parses are possible. But as the number of constituents increase, the number of possible ways the constituents can be grouped grows very fast. The constituency parsing is similar to the problem of completely parenthesizing *n+1* factors in all possible ways. Thus the total possible ways of parsing a compound with $n + 1$ constituents is equal to a *Catalan number,* C_n (Huet, 2009) where for $n \geq 0$,

$$C_n = \frac{(2n)!}{(n+1)!n!}$$

The task of the constituency parser is then to choose the correct way of grouping the components together, in a given context.

3.3 Type Identifier

After getting a constituency parse of a compound, the next task in the compound analysis is to assign an appropriate operator (tag) to each non-leaf node.

Semantically *Pāṇini* classifies the Sanskrit compounds into four major types:

- Tatpuruṣaḥ: (Endocentric with head typically to the right),
- Bahuvrīhiḥ: (Exocentric),
- Dvandvaḥ: (Copulative), and
- Avyayībhāvaḥ: (Endocentric with head typically to the left and behaves as an indeclinable).

This classification is not sufficient for generating the paraphrase. For example, the paraphrase of a compound *vṛkṣamūlam*(tree-root) is *vṛkṣasya mūlam* (root of a tree) and *gṛhagataḥ* (home-came) is *gṛham gataḥ*(the one who has come home, here home takes an accsative marker in Sanskrit), though both of them belong to the same class of *tatpuruṣaḥ*. Based on their paraphrase, these compounds are further sub-classified into 55 sub-types. The details of this subclassification and the way a paraphrase can be generated for this are described in Kumar(2010).

The tag identifier determines the type on the basis of the components involved. For instance,

<<*sumitrā*-*ānanda*>-*vardhanaḥ*>
is tagged as
<<sumitrā-ānanda>T6-vardhanaḥ>T6.

where T6 stands for a compound of type *ṣaṣṭī-tatpuruṣa*, an endocentric compound with the 1^{st} component taking a genitive (sixth) case. This module needs an access to the semantic content of its constituents, and possibly even to the wider context. A type identifier trained on manually annotated data is discussed in Kumar(2010).

3.4 Paraphrase Generator

Finally after the tag has been assigned, the paraphrase generator(Kulkarni, 2009) generates a paraphrase for the compound. For the above example, the paraphrase is generated as:
sumitrāyāḥ ānandaḥ = *sumitrānandaḥ*,
Gloss: Sumitra's joy
sumitrānandasya vardhanaḥ = *sumitrānandavardhanaḥ*.
Gloss: the growth of Sumitra's joy

The rules for paraphrase generation are simple, and except for a few exceptional cases, the paraphrases can be generated automatically, as described in Kulkarni(2009).

4 Constituency parser

In this paper we describe our constituency parser for Sanskrit compounds. As discussed earlier the total number of ways *n+1* components can be grouped in a binary tree is C_n where C_n is a catalan number. Though mathematically all these combinations are possible, the meaning compatibility among the components rule out the possibility of most of the

parses, eventually leading to one or may be a small number of possible parses.

Manually tagged Sanskrit corpus of around 150K words is available, which has around 30K instances of compound words. These compounds are split into components and also tagged and parsed manually. Table 1 describes the corpus statistically.

Table 1: Statistical details of corpus

Total size in words	150K
Total compounds	30K
Compounds with 2 components	21,384
Compounds with 3 components	6,809
Compounds with 4 components	1,321
Compounds with 5 components	319
Compounds with more than 5 components	133

The compounds with more than 2 components need parsing. Though Sanskrit is a free word order language, the components in a compound have a fixed word order, and they also show natural tendency towards left branching. In other words, in case of a compound with 3 components, the number of compounds with <a-<*b-c*>> pattern were less compared to <<a-*b*>-c>. The manually tagged data supports with this observation. Table 2 and 3 show the number of occurences of different parsed structures with 3 components and 4 components.

Table 2: Compounds with 3 components

Pattern no.	Patterns	No. of cases	cases (in %)
1	<a-<b-c>>	466	6.8
2	<<a-b>-c>	6343	93.2

Table 3: Compounds with 4 components

Pat. no.	Patterns	No. of cases	cases (in %)
1	<a-<b-<c-d>>>	5	0.3
2	<a-<<b-c>-d>>	33	2.3
3	<<a-<b-c>>-d>	127	9.7
4	<<a-b>-<c-d>>	324	24.7
5	<<<a-b>-c>-d>	832	63

4.1 Base line

As is clear from Table 2, the data is skewed and even a simple algorithm assigning the most frequent pattern will result into 93% and 63% accuracy in case of compounds with 3 components and 4 components respectively.

4.2 Our algorithm

The main task here is to decide the compatibility (sāmarthya or yogyatā) between the two words. The conditional probabilities are used to decide the compatibility between a pair of words. Let us take an example of 3 component compound viz a-b-c. Now, to decide the parse of this compound essentially means to decide whether the component 'b' joins with 'a' or with 'c'. In a compound <a-b>, the component 'a' is termed as *iic* (in initio compositi or 'samāsa pūrvapada) and the component 'b' is termed as *ifc* (in fine compositi or samāsa uttarapada). Thus to decide the parse of a compound a-b-c, one should know whether it is more likely to use 'b' as an iic or ifc. However, only the unigram frequencies to this effect are not sufficient, since the context, or the affinity of other words viz. 'a' and 'c' in the context plays a role in determining the parse. So what is needed is the comparison between two conditional probabilities viz. the probability of 'b' as an ifc given that 'a' is an iic and the probability of 'b' as an iic given that 'c' is the ifc. If the difference between these conditional probabilities is above a certain threshold, we use this information to decide the parse. When the conditional probabilities are not available or if the difference between the two conditional probabilities is not significant, we resort to the unigram frequencies, and based on the probabilities of 'b' as ifc versus iic, we take the decision.

Thus the algorithm may be summarised as:

let p(ab) = probability of b as ifc given a is iic,
let p(bc) = probability of b as iic given c is ifc,
let p(bf) = probability of b as ifc,
let p(bi) = probability of b as iic,
and let T = threshold.
if((p(ab) - p(bc)) >T) then the parse = <<a-b>-c>
elsif((p(bc) - p(ab)) >T) then the parse = <a-<b-c>>
elsif (p(bf) - p(bi) >T) then the parse = <<a-

b>-c>
elsif (p(bi) - p(bf) >T) then the parse = <a-<b-c>>
else the default parse = <<a-b>-c>

We used the manually tagged corpus for training as well as testing. The data was randomised and split into 5 sets, and a 5-fold testing method was adopted, eachtime reserving one set for testing and using other 4 sets for training. The average results of this experiment using the above algorithm are shown in Table 4 below.

Table 4: Performance of compounds with 3 components

Patterns	No. of cases	Pecision (in %)	Recall (in %)	F-measure
<a-<b-c>>	471	57	45	0.503
<<a-b>-c>	6408	95.8	97.27	0.965

The average performance is 93.66% which is very close to the baseline, and at the same time the F-measure for the frequent pattern is also close to 1.

This algorithm was extended to more than 3 components, and the performance for 4 components is shown in the Table 5.

Table 5: Performance of compounds with 4 components

Patterns	No. of cases	Pecision (in %)	Recall (in %)	F measure
<<<a-b>-c>-d>	832	72	87	0.788
<<a-<b-c>>-d>	127	80	3	0.57
<<a-b>-<c-d>>	324	79	40	0.53
<a-<b-<c-d>>>	5	50	20	0.28
<a-<<b-c>-d>>	33	-	0	-

The average performance for 4 components is 65.4% which is again just above the base line performance.

Examples with more than 4 component compounds being very small in number, their precision and recall are not measured. The overall performance for 5 test data is given below.

Sr no.	Total compounds	Correct Parse	Wrong Parse	% success
1	1738	1493	245	85.9
2	1734	1503	231	86.6
3	1729	1497	232	86.5
4	1759	1532	227	87.0
5	1737	1500	237	86.3

The average success rate is 86.5%.

5 Analysis of results

Analysis of wrong results of 3 component compounds is carried out manually to understand the reasons behind failure. The failures were of two types.

(a) Where the instances of <<a-b>-c>type were parsed by machine as <a-<b-c>>, in most of these cases 'a' was an adjective of 'b'. Since the evidences of <b-c>were found in the corpus, whereas the evidences of <a-b>were not found, machine did not have the information that 'a' can be an adjective of 'b'. Hence these compounds were wrongly classified as '<a-<b-c>>'.

(b) The cases where the instances of <a-<b-c>>were classified as <<a-b>-c>, as such instances of <b-c>were not found in the training data and machine produced the default left branching parse viz <<a-b>-c>.

6 Conclusion

1. The $iifs$ or the final components in the compound are inflected. Instead of the inflected words, if the probabilities are calculated taking into account the roots, the performance should increase.

2. The compounds are of 4 different types with head being different in each of them. The algorithm described in section 4, should take into account the compound type to decide its association. For example, consider the compound <<a-b >-c >where <a-b >is an avyayībhāva (left word to be the head), then it is not 'b'

which will be associated with 'c' but it is 'a' which will be associated with 'c'. The current implementation since does not take into account the type of a compound it produces wrong grouping in such cases. The algorithm needs to be thus modified to take into account the compound type as well along with the association.

3. The components of compound together convey a unique meaning which is over and above its components meanings. So had the components of a Sanskrit be written seperately, they are very close to a Multi Word Expression (MWE). So the problem of constituency parsing of Sanskrit compounds then is directly relevent for determining the association of words within a MWE with each other. The insights gained in handling compounds will be directly available for handling MWEs of Indian languages.

References

Bharati, A., Kulkarni, A.P., and Sheeba, V. 2006 . *Building a wide coverage Sanskrit Morphological Analyser: A practical approach.* In: The First National Symposium on Modelling and Shallow Parsing of Indian Languages, IIT-Bombay.

Bhat, G.M. 2006. *Samāsaḥ.* Samskrita Bharati, Bangalore, Karnataka.

Gillon, B.S. 2007 *Exocentric Compounds in Classical Sanskrit.* In: Proceeding of the First International Symposium on Sanskrit Computational Linguistics(SCLS-2007), Paris, France.

Gillon, B.S. 2009. *Tagging Classical Sanskrit Compounds.* In: Sanskrit Computational Linguistics 3, pages 98-105, Springer-Verlag LNAI 5406.

Hellwig, O. 2007. *Sanskrit Tagger: A Stochastic Lexical and POS Tagger for Sanskrit.* In: Sanskrit Computational Linguistics 1 & 2, pages 266-277, Springer-Verlag LNAI 5402.

Hellwig, O. 2009. *Extracting Dependency Trees from Sanskrit Texts.* In: Sanskrit Computational Linguistics 3, pages 106-115, Springer-Verlag LNAI 5406.

Huet, G. 2006 *Shallow syntax analysis in Sanskrit guided by semantic nets constraints.* In: Proceedings of International Workshop on Research Issues in Digital Libraries, Kolkata.

Huet, G. 2006.*Lexicon-directed Segmentation and Tagging of Sanskrit.* XIIth World Sanskrit Conference, Helsinki, Finland, Aug. 2003. In Themes and Tasks in Old and Middle Indo-Aryan Linguistics, Eds. Bertil Tikkanen and Heinrich Hettrich. Motilal Banarsidass, Delhi, pp. 307-325.

Huet, G. 2007: *Formal structure of Sanskrit text: Requirements analysis for a Mechanical Sanskrit Processor.* In: Sanskrit Computational Linguistics 1 & 2, pages 162-199, Springer-Verlag LNAI 5402.

Huet, G. 2009. *Sanskrit Segmentation.* In: South Asian Languages Analysis Roundtable XXVIII, Denton, Ohio.

Jha, G.N. and Mishra, S.K. 2007. *Semantic Processing in Pāṇini's Kāraka System.* In: Sanskrit Computational Linguistics 1 & 2, pages 239-252, Springer-Verlag LNAI 5402.

Kulkarni, A.P., Shukla, D. 2009. *Sanskrit Morphological Analyser: Some Issues.* In: To appear in Bh.K Festschrift volume by LSI.

Kulkarni, A.P., Kumar, A., Sheeba, V. 2009. *Sanskrit compound paraphrase generator.* In: Proceedings of ICON-2009: 7th International Conference on Natural Language Processing, Macmillan Publishers, India.

Kumar, A., Mittal, V., Kulkarni, A.P. 2010. *Sanskrit Compound Processor.* In : Proceedings of 4i-SCLS 2010: 4^{th} International Sanskrit Computational Linguistics Symposium, Springer-Verlag LNAI 6465.

Mahavira. June 1978. *Pāṇini as Grammarian (With special reference to compound formation).* Bharatiya Vidya Prakashan [Delhi - Varanasi], India.

Mittal, V. 2010. *Automatic Sanskrit Segmentizer using Finite State Transducers.* In: Proceeding of Association for Computational Linguistics - Student Research Workshop.

Murty, M.S. 1974. *Sanskrit Compounds-A Philosophical Study.* Chowkhamba Sanskrit Series Office, Varanasi(India).

Pandit Ishvarachandra. 2004. *Aṣṭādhyāyī.* Chaukhamba Sanskrit Pratisthan, Delhi.

Sarma, E.R.S. 1960. *Maṇikaṇa : A Navya-Nyāya Manual.* The Adyar Library and research Centre, Madras.

Scharf, P.M. 2009. *Levels in Pāṇini's Aṣṭādhyāyī.* In: Sanskrit Computational Linguistics 3, pages 66-77, Springer-Verlag LNAI 5406.

Yuret, D., Biçici, E. 2009. *Modeling Morphologically Rich Languages Using Split Words and Unstructured Dependencies.* In: ACL-IJCNLP, Singapore.