Chapter 5

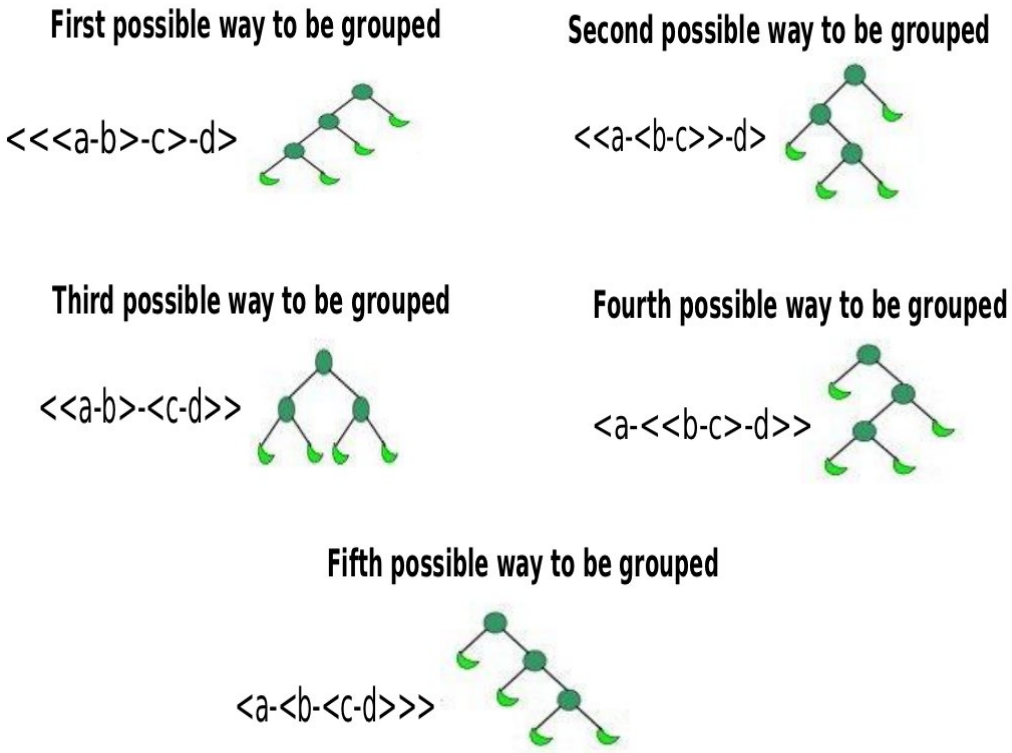
Constituency parser

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 <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 >>
   (Gloss : <one - <who is dear to all>>)
2. << tapas-svādhyāya > - niratam >
   (Gloss : <<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. For instance a compound with 4 components *a-b-c-d* can be grouped into 5 possible ways. See the figure below.



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 $n \geq 0$,

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

5.1 Developing the constituency parser

We observed that the correctness of parse is governed by the semantics. It is the semantic compatibility (सामर्थ्य) between two words which decides the parse of a compound. We found that there are two approaches to decide semantic compatibility (समार्थ्यनिर्धारण) :-

1. Use the semantically rich lexicon and follow a ruled based approach.
2. Use statistical patterns of manually parsed compounds to decide the more likely parse.

We follow the second approach to parse the compounds.

5.2 Statistical approach

The task of the constituency parser is then to choose the correct way of grouping the components together, in a given context. 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. The Sanskrit Consortium has developed a manually tagged Sanskrit corpus of around 600K words, which has around 80K instances of compound words. These compounds are split into components and also tagged and parsed manually. Table 1 describes the corpus statistically.

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 5.1 : 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

Table 2 and 3 show the number of occurences of different parsed structures with 3 components and 4 components.

Table 5.2 : Compounds with 3 components

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

Table 5.3 : Compounds with 4 components

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

5.2.1 Base line

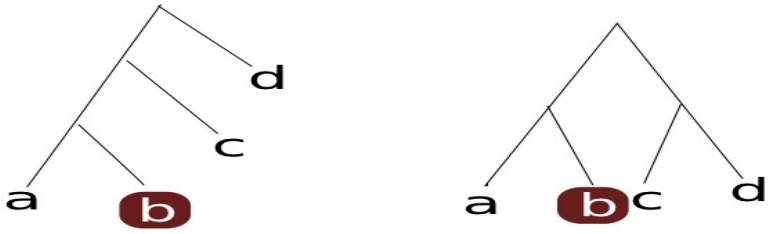
As is clear from Table 2, the data is skewed and even a simple algorithm assigning the most frequent pattern will result into 93% and 63% accuracy in case of compounds with 3 components and 4 components respectively.

5.2.2 Our algorithm

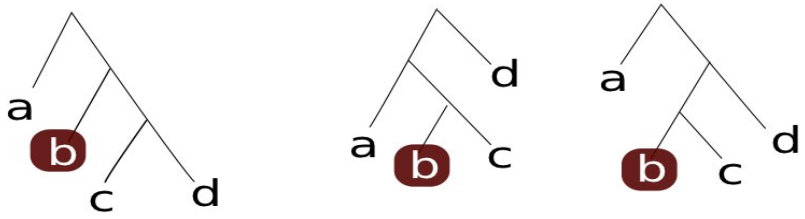
The main task here is to decide the compatibility (sāmarthya or yogyatā) between the two words. The conditional probabilities are used to decide the compatibility between a pair of words. Let us take an example of 3 component compound viz a-b-c. Now, to decide the parse of this compound essentially means 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 refer the figure below.

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. For examples in case of four components as a-b-c-d, the component b is a final component in two parses and initial component in three parses as shown in figure -

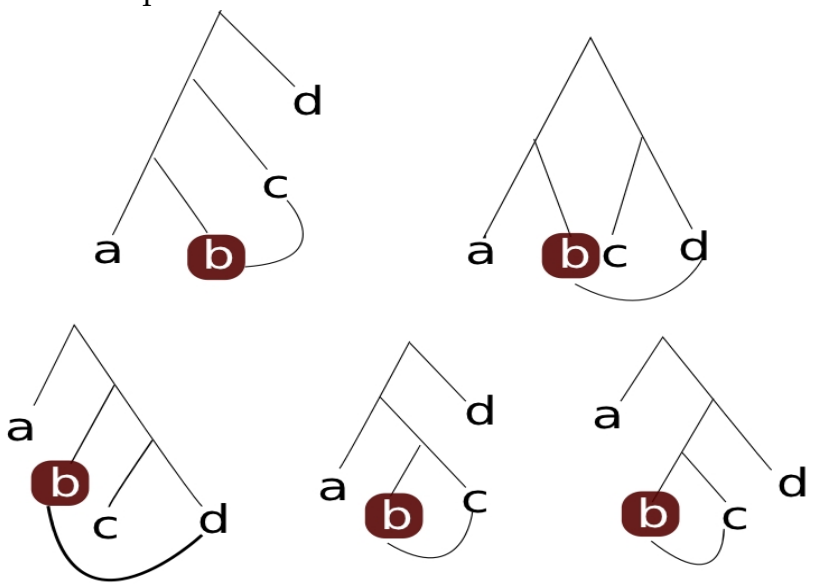
b as final component :



b as initial component :



So to fix the parse, we need one more parameter viz whether 'b' is related to 'c' or 'd', in other words whether 'b' has more affinity towards 'c' or 'd', and this fixes the parse as shown below -



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>
else if((p(bc) - p(ab)) > T) then the parse = <a-<b-c>>
else if (p(bf) - p(bi) > T) then the parse = <<a-b>-c>
else if (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 5.4 : Performance of compounds with 3 components

Patterns	No. of Instances	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 5.5 : Performance of compounds with 4 components

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

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.

The average success rate is 86.5%.

Sr no.	Total compounds	Correct instances	Wrong instances	% 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

5.3 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' is 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>.

5.4 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.