Part of Speech Tagging and Local Word Grouping Techniques for Natural Language Parsing in Hindi [1]

Pradipta Ranjan Ray **Harish V.** **Sudeshna Sarkar** **Anupam Basu**

Department of Computer Science & Engineering,
Indian Institute of Technology, Kharagpur
INDIA 721302
Email: *{ pradipta, harishv, sudeshna, anupam } @ cse.iitkgp.ernet.in*

Abstract

We present an algorithm for local word grouping to extricate fixed word order dependencies in Hindi sentences. Local word grouping is achieved by defining regular expressions for the word groups. Ambiguities occurring during word grouping are also resolved. Hindi being a free order language, fixed order word group extraction is essential for decreasing the load on the free word order parser. The parser paradigm being used is the computational Paninian model. Also, local word grouping achieved can be used to provide inputs to intonation and prosody modelling units for text to speech systems in Indian languages. Part of speech tagging is an essential requirement for local word grouping. We present another algorithm for part of speech tagging based on lexical sequence constraints in Hindi. The algorithm acts as the first level of part of speech tagger, using constraint propagation, based on ontological information and information from morphological analysis, and lexical rules.

1. Introduction

The need for computers to understand natural language, especially vernaculars, is growing by the day as human-computer interfaces become more intuitive, and machine translation gains prominence due to increasing multilingual content on the net. Natural languages may be broadly categorized into fixed word order languages (eg. English) and free word order languages (eg. Sanskrit). In fixed word order languages, words constituting a sentence can be positioned in a sentence according to grammatical rules in some standard ways (eg. The Subject-Verb-Object structure in English). In free word order languages, no fixed ordering is imposed on the sequence of words.

Parsing a natural language sentence may be looked upon as *chunking* or recognizing structural units that allow us to identify the meaning of a sentence [Manning and Schutze, 1999]. Such chunking may be achieved using different methods. For fixed word order languages, *generative grammars*, like *context*

[1] This research is funded in part by Media Lab Asia, under the auspices of the Communication Empowerment Laboratory, IIT Kharagpur.

free grammars and *tree adjoining grammars* are used for modelling sentential structures [Charniak, 1997; Joshi and Schabes, 1997]. Such techniques do not work well with free word order languages, as the number of rewrite rules required to capture the free word order phenomenon is very large. The common processing techniques for such languages is *constraint based parsing* which involves identifying relations between components of a sentence in the presence of constraints. *Case grammar* and the *computational Paninian model* are examples of such constraint-based parsing [Bharati *et al*, 1995; Bharati and Sangal, 1993; Tapanainen and Jarvinen, 1997]. Such formalisms have also been used to parse fixed order languages [Bharati *et al*, 1997].

Among Indo-Aryan languages, Sanskrit is a purely free-order language but some Indo-Aryan languages like Hindi and Bengali have partially lost the free word ordering in the course of evolution. In Hindi and Bengali, "word groups" are free-order but the internal structure of word groups are fixed-order, i.e. any sentence can be looked upon as a "bag" of word groups. Identifying these word groups correctly and maximally in such languages reduces the overhead on the core parser for the computational Paninian model. The parser then identifies relations between the verb group and noun groups by modelling the problem as an integer programming or bipartite graph matching problem [4]. The constraints and rules presented in this paper are for Hindi, but the approach is general enough to be applicable to many other Indo-Aryan languages.

Splitting up a Hindi sentence into constituent fixed order word groups has been dealt with earlier [Bharati *et al*, 1995]. The basic motivation for such group formation was to act as input to a computational Paninian parser, but part of speech (POS) tagging disambiguation, word grouping ambiguity resolution and complicated word groups [eg. word groups with conjunctions in them] need more detailed treatment. We propose to deal with these issues. In this paper, we present an algorithm to identify word groups in Hindi on the basis of lexical tags of the individual words. For this purpose, we also present an algorithm for part of speech tagging of Hindi words. Further, our basic aim for performing word grouping is two-fold. Firstly, this acts as the first phase of any computational parsing system for Hindi. Secondly, for text to speech systems in Indo-Aryan languages, it has been noted that prosodic patterns vary at the word, phrasal and clausal level. Identification of word groups and clause boundaries is hence essential for generating natural speech. The splitting of a sentence into constituent clauses also requires the formation of local word groups.

Section 2 deals with the POS tagging algorithm in Hindi while Section 3 deals with the algorithm for Local Word Grouping. Section 4 looks at future work. All Hindi words and sentences are denoted in this paper in the standard **Itrans** notation in italics and not using any Hindi glyphs.

2. POS Tagging in Hindi

2.1. Terminology and Definitions

At the outset we present three basic definitions -

The **lexicon** or the dictionary is the set of all valid words in Hindi and is denoted by L.

The **root dictionary** is the set of all valid root words in Hindi and is denoted by R.

The **set of all relevant lexical categories** is denoted by C where C ={ n,v, j, a, c, q, pp, y}

[n → noun (& pronouns) / *visheshya* , v → Verb / *kriyA* , c → Conjunction , pp → Postposition,
j → Adjective / *visheShaNa* , a → Adverb / *kriyA visheShaNa* , y → Other Avyayas,
q → Qualifier / *pravisheShaNa*]

A **morphological analyzer** takes the surface form of a word as input and gives the possible roots, POS tags and other categorical information (like Gender, Number, Person, etc for nouns) as output. A morphological analyzer (MA) is a mapping from every element of the lexicon to all possible roots, along with all possible lexical information about the root that can be obtained from the surface form. From the point of view of POS tagging, the **possible parts of speech** (PPOS) a surface form of the word can take may be obtained from the MA itself.

$$\text{MA: } L \rightarrow 2^{R \times I} \quad \text{[I: set of all possible lexical information]}$$
$$\text{PPOS: } L \rightarrow 2^{C}$$

A **POS tagging** identifies the lexical category of each word in a sentence on the basis of its context. An accurate definition of POS tagging (POST) is a mapping from a sequence of words to a sequence of lexical categories. Such a mapping is unambiguous provided the sentence itself is unambiguous.

$$\text{POST: } L^n \rightarrow C^n \qquad \text{[for a sentence of n words]}$$

For a particular POST of a sentence s, the POS of the word w in the sentence s can be defined as the projection π_w on the output of POST. Thus, for a given word w in a sentence s,

$$\pi_w \text{ (POST(s)) } \in \text{PPOS(w)}$$

2.2. Approaches to POS Tagging

POS tagging is typically achieved by rule-based systems, probabilistic data-driven systems, neural network systems or hybrid systems. For languages like English or French, hybrid taggers have been able to achieve success percentages above 98%. [Schulze *et al*, 1994]

Due to non-availability of statistical information in Hindi, purely rule-based systems are only able to partially solve the problem of POS tagging. Such systems will eliminate the large number of definitely wrong taggings which would otherwise be present if no constraints were present. The *partial* POS tagger for Hindi presented here reduces the number of possible taggings for a given sentence by imposing some constraints on the sequence of lexical categories that can typically occur in a Hindi sentence. On availability of statistical information, we can augment our algorithm by adding a statistical disambiguation module as a two-tiered structure on top of the existing algorithm for performance enhancement. The idea, however, is to use a rule base for tagging as far as possible, and to take statistical cues only where there is no other option [Tapanainen and Voutilainen, 1994].

2.3. A Trivial POS Tagger

The simplest approach to POS tagging is to use the MA on each word of the sentence, and then output a POS tagging for every possible combination of the tags returned by the MA.

e.g. In the sentence, *shyAma khAnA khAyegA*

PPOS(*shyAma*) = { n, j } ; PPOS(*khAnA*) = { n, v } ; PPOS(*khAyegA*) = { v }

Trivially, the POS tagger can output four different taggings based on all combinations of the outputs from the PPOS. The four taggings are:

< n, n, v >   < n, v, v >   < j, n, v >   < j, v, v >   … **(1)**

This rudimentary POST comes with an overhead of a huge number of possible taggings. We may disambiguate at the end of the parse, or at every step of the algorithm. At this level, for disambiguation, we define two kinds of constraints on lexical tag sequences of valid sentences in Hindi. These may be extended to other Indo-Aryan languages with some language–specific changes.

2.4. Syntactic Constraints on a Language over Lexical Tag Sequences

Let LSL be set of all lexical tag sequences corresponding to all correct POST of all correct Hindi sentences. Hence, the alphabet corresponding to the language LSL is the set of all lexical categories C. (i.e. LSL $\subset$ C*)

For the language LSL, we may define the FOLLOW set of every element of C – i.e. the follow set puts a restriction on what all lexical categories can follow a lexical category.

FOLLOW (x) = { Set of all lexical categories that can follow x in a Hindi sentence }

The follow sets may be concisely modelled as a *binary relation* on C, i.e.

FOLLOW $\subset$ C X C

From Table 1 the follow sets of all the categories may be obtained using :

FOLLOW (x) = { y : (Row x, Column y) is 1 }

Table 1 was constructed by empirical observation of Hindi and has been hand-tested and found to be correct on sentences of varying complexity.

We can then use the follow set constraints to discard some of the possible POST suggested by the MA. [eg, in the sentence *shyAma khAnA khAyegA,* the 4th POST in (1) is < j, v, v > and would be rejected because v is not in the FOLLOW set of j]

Another possible constraint is the PREV set for the LSL – which puts a restriction on what all lexical categories can be placed immediately before a certain lexical category. However, this constraint is the same as the FOLLOW set constraint, and the relation corresponding to the PREV sets can be defined using the *transpose* of the matrix pertaining to the FOLLOW relation.

PREV (x) = { Set of all lexical categories that can precede x in a Hindi sentence }
= { y : (Row y, Column x) is 1 }

	n	v	j	a	q	c	pp	y
n	1	1	1	1	1	1	1	1
v	1	1	1	1	1	1	1	1
j	1	0	1	0	1	1	0	0
a	1	1	1	1	1	1	1	1
q	0	0	1	1	1	0	0	0
c	1	1	1	1	1	1	0	1
pp	1	1	1	1	1	1	1	1
y	1	1	1	1	1	1	0	1

Table 1: The Follow Relation for LSL

2.5. Modelling Semantic Constraints for POS Tagging

Since the only disambiguation being done at this level is via lexical tags and is purely syntactic in nature, we may model some more complicated constraints on the LSL.. Disambiguation using the FOLLOW relation is based on the previous (or following) lexical tag only. We may perform disambiguation based on neighbouring lexical tags, semantic information and the *type of ambiguity.*

Such ambiguity resolution needs some semantic input usually involving cross-POS linkages. The cross POS linkage required for such disambiguation is of the form qualifier–qualified. Even though we do not have such a lexical resource for Hindi in general, for smaller domains, such semantic information (which surface form can qualify which surface forms) can be exhaustively listed out as a binary relation QUAL where QUAL(x,y) is true iff x can qualify y. For large domains, an ontology may be developed with generalization–specialization and qualifier–qualified relations between constituent entities. Hence, we may have constraints of the form of 4–tuples as follows:

< Left Context, Type of Ambiguity, Right Context, Disambiguated Category >

The following rules hold for Hindi:

1. Given a noun / adjective ambiguity, if the following word is a noun, and the adjective can qualify the succeeding noun, then the ambiguity will be resolved in favour of the adjective, otherwise it will be resolved in favour of the noun. [eg. in *moTA la.DakA, moTA* can be an adjective or noun, but as it can qualify *la.DakA*, it will be an adjective, not a noun]
2. Given an adjective / adverb ambiguity, if the following word is a noun, and the adjective can qualify the succeeding noun, then the ambiguity will be resolved in favour of the adjective, otherwise it will be resolved in favour of the adverb. [eg. in *teja la.DakA dau.DatA hai, teja* qualifies *la.DakA* and hence will be an adjective ; in (1), if the information that *shyAma* does not normally qualify *phala* is available, *shyAma* will be a noun]
3. Given an adverb / qualifier ambiguity if the following word is an adverb and the qualifier can qualify the adverb then the ambiguity will be resolved in favour of the qualifier, else it will be resolved in favour of the adverb. [eg. in *bahuta teja dau.DatA hai*, *bahuta* can be an adverb or qualifier, but *bahuta* can qualify *teja* , and hence will be a qualifier.]

Formally, these correspond to :

< - , j / n , n , j > if QUAL(w_C, w_R) is true. < - , j / n , n, n > if QUAL(w_C, w_R) is false.
< - , j / a , n , j > if QUAL(w_C, w_R) is true. < - , j / a , n , a > if QUAL(w_C, w_R) is false.
< - , q / a , a , q > if QUAL(w_C, w_R) is true. < - , q / a , a , a > if QUAL(w_C, w_R) is false.

[The left context word is w_L, the central word is w_C and the right context word is w_R.]

These semantic constraints are modelled on empirical observations, but there are rare exceptions to this set of constraints. [eg. in *vaha videshi vastroM kA vyApArI hai* , *videshi* may or may not belong to the noun group immediately following it. This is because *videshi* as a noun is very frequently used in the sense of “foreigner”. Such disambiguation is best done statistically by assigning probabilities to the lexical tag assignments.]

2.6. Algorithm for POS Tagging

```
POS_TAGGER ( sentence s )
{
  w ← the first untagged word from the right in the sentence s
  if ( some word is untagged in s )
  {
    X ← PPOS(w)  /*X is the possible set of lexical categories that w can take*/
    if w is not the last word of the sentence
      X ← X ∩ PREV(word immediately following w) /*X is also constrained by the word after w*/
      for each element e in ( X )
         if ( w tagged as e obeys semantic constraint set )
            tag w as e and call POS_TAGGER( s )
   }
   else output the tagged sentence s
}
```

3. Local Word Grouping in Hindi Tagging in Hindi

3.1. Language Structure of Hindi

Sequences of words can be grouped together to form a *word group*. Information contained in a word group can be constructed by using information obtained from each of the individual words using the MA. The information in the word group depends on the information obtained from the individual words, and the nature of relation between the words. Word groups may combine with each other to create larger word groups. In an abstract form, we may look upon the structure of these groups as a modifier – modified structure. The meaning of the composed structure is similar to the meaning of the modified or head, but is altered in different ways by different modifiers. There may be 10 different kinds of modifier – modified structures in Hindi. [Bharati *et al*, 1995]

The four local modifier-modified structures are:

i. Parsarg modifier of a noun [***mohana ko*** *achchhA lagA.*]
ii. Noun heads with adjective modifiers [***ba.DA ghara*** *khAli thA.*]
iii. Qualifier modifiers of adjective / adverb head [*Aja* ***bahuta achchhA*** *dina hai.*]
iv. Auxiliary verbs modifying the main verb [*la.DakA ghara* ***jA rahA thA***.]

The six dependencies best captured by the Paninian framework using constraint modelling are:

v. Participle verb modifiers of noun head [*mujhe* ***nAchatA huA mora*** *bahuta achchhA lagatA hai.*]
vi. Adverb modifiers of verb head [*laDaka* ***jora se ro*** *raha tha*]
vii. Verbs as arguments of verb head [*laDake ne* ***kahA*** *ki usane khAnA* ***pakAyA***.]
viii. Verb modifiers of verb heads [*wo* ***nahA kara*** *mandira* ***gaya***.]
ix. Nominal structure with verbal nouns [*bhArata kA khela* ***jitanA*** *saba ko achchhA* ***lagA***]
x. Verb heads with noun modifiers (karaka relations) [*vaha* ***Trena se*** *kashi* ***gayA***]

The first five of these language phenomena have fixed word ordering among the component words. However, the fifth phenomenon is best captured using the core Paninian parser, hence the local word grouper (LWG) we propose recognizes the first four constructs appropriately.

3.2. Local Word Group Categories in Hindi

Local word groups in Hindi are the fixed order components of the sentence and are word groups which obey the constraints that constituent words are placed contiguously, and the group of words is fixed word order. In Hindi word groups may be of three major types – Noun groups, Verb groups, and Adverb Groups. Conjunctions and other avyayas also form standalone word groups. The grouping is performed from right to left as the word group heads – nouns in noun groups, finite verbs in verb groups, and adverbs in adverb groups all occur at the extreme right end of their respective word groups. The LWG runs in four phases: *Clubbing*, *Seeding*, *Grouping* and *Filtering.* It runs on each possible POS tagged sequence generated by the POS tagger.

3.3. The Clubber

The preprocessing phase or the clubber essentially consists of clubbing certain word sequences as they need to be treated as a single unit during grouping. This consists of two kinds of clubbing:

- Clubbing duplicated words: Duplicated verbs like *chalte chalte* are clubbed together to generate an adverb. Any two consecutive *infinitive verbs* may be clubbed as an adverb.
- Clubbing avyaya sequences: Special avyaya sequences are clubbed together to act as single postpositions. Such avyaya sequences are looked up from a table and comprise of such sequences like *ke liye*, *ke dvArA, ke kAraNa,* and *mAdhyama se*. Positional avyaya sequences like *ke madhya, ke bicha* and *ke nicha* can form avyaya sequences acting as postpositions. Such sequences usually end with postpositions like *meM, para, se* and *taka* which are usually associated with positional avyayas.

The algorithm for clubbing is trivial, and is as follows:

```
Algorithm CLUB
Identify all replicated words
Identify all avyaya sequences to be clubbed by looking up a table
Clubs identified words and treat it as a single lexical unit with the corresponding lexical category
```

3.4. The Seeder

The seeder identifies all possible end points of word groups. It merely looks at the lexical tag of each word and accordingly takes a decision about whether or not to mark the word as a seed. The seeder simply marks all nouns, verbs, adverbs, conjunctions, postpositions and other avyayas – each such possible rightmost element of a group is called a **seed**.

3.5. The Grouper

Corresponding to every kind of local word group, we have derived a *regular expression* (RE). Any word group of that category has to necessarily satisfy that RE. However, satisfying the RE is not a sufficient condition for a sequence of words to be the corresponding word group. Hence, when grouping ambiguities occur, all the groupings may not be valid ones.

The grouper chooses the rightmost ungrouped seed for grouping and invokes the corresponding finite state automata (FSA) depending on the lexical type of its seed. The grouper runs the FSA on the sequence of lexical categories of the words starting from the seed backwards and continues until it encounters an invalid transition, or the sentence is completed. The last grouped word for which the FSA entered a final state is remembered at any stage - the **sentinel**. When an invalid transition or start of sentence is encountered, grouping is performed maximally upto the sentinel.

Ambiguities during grouping may occur iff the following occurs – the FSA enters a final state, and the next input to the FSA is a seed. Then the two options that a grouper has is to either continue with the current group, or to end the current group and start a new group. In such cases, the grouper outputs both groupings and which is correct is decided in the next phase (if possible).

3.6. Noun Groups

In noun groups, adjectives may occur in sequences or may be joined by conjunctions. Each adjective in turn, may be preceded by qualifiers. This is modelled by ADJGR. The basic unit of postpositions following a noun is given by POSSESSOR (for *sambandha pada*) and NSEED (for other karakas). Adjective groups immediately precede a noun – this is modelled by BASGR and POSGR. A generalized noun group consists of conjunctions of smaller noun groups or nouns, each capable of having its own *sam-*

bandha pada linkages. CONPOS models conjunctions over *sambandha padas*. The entire noun group is given by NOUNGR. [eg. *gaharA nIla AsamAna ke jhilamilAte sitAre*. *sitAre* forms the NSEED, and *AsamAna ke* the POSSESSOR. *gaharA nIla AsamAna ke* then forms POSGR, and *jhilamilAte sitAre* the BASGR. Since a single *samvandha pada* is present, the POSGR makes up the CONPOS. The CONPOS and BASGR can then form the NOUNGR.]

Verbal nouns can have adjectives, postpositions and qualifiers, eg. "***likhane kI kalA*** *sabakoM nahIM AtI"*, "*mujhe* ***khAnA aura pinA*** *bahuta pasanda hai*". Verbal nouns are not treated differently from other nouns by the LWG – but a noun group with a verbal noun head may be treated differently from other noun groups by the Paninian Core Parser.

Noun–Noun groups like "***rAnI lakShmIbAI***" are not handled by the grouper and will be grouped as contiguous noun groups, which may be clubbed by specific rules if required.

The regular expressions corresponding to a noun group is as follows:

ADJGR → (q* j (c | EPSILON))* q* j
POSSESSOR → n (ke | kI | kA)
NSEED → n (other_pp | EPSILON)
BASGR → ADJGR NSEED
POSGR → ADJGR POSSESSOR
CONPOS → (POSGR (c | EPSILON))* POSGR
NOUNGR → ((CONPOS | EPSILON) BASGR c)* (CONPOS | EPSILON) BASGR

[other_pp here refers to all postpositions beside ke, kI and kA]

3.7. Other Word Groups

A verb group is simply a sequence of consecutive or conjoined verbs containing 0 or 1 finite verb. [eg. *khAta aura pita hai*]

VERBGR → (v (c | EPSILON))* v

Noun–Verb constructs acting as verbs, called *conjunct verbs* [eg. *mAra khAnA*], are nonlocal dependencies – i.e. *mAra* and *khAnA* may not be contiguously placed, and hence is not grouped by the LWG.

Adverb groups correspond to a sequence of adverbs which may or may not be separated by conjunctions, each with possible qualifiers before them. [eg. *bahuta sundara aura madhura*]

ADVGR → (q* a (pp | EPSILON) (c | EPSILON))* q* a (pp | EPSILON)

Conjunctions and (non–conjunction, non-postposition) *avyayas* can form stand-alone groups.

3.8. The Grouping Algorithm

Algorithm GROUP

```
while there are ungrouped seeds left
{
  Choose the rightmost ungrouped seed.
  Choose the Finite State Grouping Automata (FSGA) as follows
  {
    If seed is postposition
      Choose the FSGA by looking at the word before it
    else Choose the corresponding FSGA for the seed
  }
  Set current position ← chosen seed
  Set flag ← HI
  Set sentinel ← NULL
  while ( flag is HI )
  {
    Feed the FSGA the lexical tag of the current position
     if ( transition was invalid )
       flag ← LO
    else if ( the state of the FSGA is accepting )
       {
         sentinel ← current position
         if ( next position is a seed )
           Simultaneously start a new grouping by closing the present group
           from the chosen seed    to the sentinel and opening a new group
       }
  }
  if (sentinel is not NULL)
    Create a new group by clustering all words from chosen seed to the sentinel
}
```

3.9. The Filter

All groupings which the grouper outputs are not necessarily correct. Hence, the filter discards some of the groupings based on the following constraints :

Any groupings in which there are ungrouped words are rejected.

Any groupings where noun groups do not obey the following constraints are rejected:

- If a noun group contains more than one noun (joined by conjunction), then the inflections/ postpositions of all the nouns have to be the same, or only the last noun can be inflected/ have postposition .

Any groupings where verb groups do not obey the following constraints are rejected:

- Between any closest pair of conjunction groups or between a conjunction group and the start / end of a sentence, there has to be at least one verb group and at most one finite verb group.

4. Conclusion

The performance of the POS tagger and LWG has not been statistically tested due to lack of lexical resources but it covers a wide range of language phenomenon and accurately captures the four major local dependencies in Hindi. The LWG will allow generation of near natural speech in Indian languages by enabling phrase and clause level intonation and prosody rules to be automatically applied to a sentence. Also, a sturdy LWG enormously reduces the load of any parser that follows it. An idea of local fixed-order structures also helps in language generation, as in English [Knight and Hatzivassiloglou, 1995]. Future work resides in developing verb grouping constraints, and increasing the size of the rulebase for POS tagging. The main initiative would be to develop a tagged corpus, and accordingly develop statistical models of POS tagging using a larger tag set.

5. References

[Manning and Schutze, 1999] Manning, C. and Schutze, H.; *Foundations of Statistical Natural Language Processing*, pp. 407 – 409; MIT Press, 1999.

[Charniak, 1997] Charniak, E. ; "Statistical parsing with a context-free grammar and word statistics", Proceedings of the 14th AAAI, Menlo Park, 1997.

[Joshi and Schabes, 1997] Joshi, A.K. and Schabes, Y.; "Tree-Adjoining Grammars", *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa (eds.), Vol. 3 pp. 69 - 124; Springer, Berlin, New York, 1997.

[Bharati *et al*, 1995] Bharati, A. , Chaitanya, V. and Sangal, R.; *Natural Language Processing : A Paninian Perspective*; Prentice Hall India, 1995.

[Bharati and Sangal, 1993] Bharati, A. and Sangal, R. ; "Parsing free word order languages in the Paninian Framework", *Proceedings of the Annual Meeting of Association for Computational Linguistics*, New York, 1993.

[Tapanainen and Jarvinen, 1997] Tapanainen, P. and Jarvinen, T.; "A non-projective dependency parser", *Proceedings of the 5th Conference on Applied Natural Language Processing*, Washington D.C.,1997.

[Bharati *et al*, 1997] Bharati, A., Bhatia, M., Chaitanya, V. and Sangal, R.; "Paninian Grammar Framework Applied to English", *South Asian Language Review*, Creative Books, New Delhi, 1997.

[Schulze *et al*, 1994] Schulze, B.M. et al; "Comparitive State-of-the-art Survey and Assessment of General Interest Tools", Technical Report D1B – I, DECIDE Project, Institute for Natural Language Processing, Stuttgart, 1994.

[Tapanainen and Voutilainen, 1994] Tapanainen, P. and Voutilainen, A. ; "Tagging Accurately – Don't Guess If You Don't Know", *Proceedings of 4th ACL Conference on Applied Natural Language Processing*, ACM, Stuttgart, 1994.

[Knight and Hatzivassiloglou, 1995] Knight, K. and Hatzivassiloglou, V. ; "Two Level, Many Paths Generation"; *Proceedings of the ACL-95*. Cambridge, MA, 1995.