UCSG Shallow Parsing Architecture

Kavi Narayana Murthy, G. Bharadwaja Kumar
Department of Computer and Information Sciences
University of Hyderabad, India
email: knmuh@yahoo.com,g_vijayabharadwaj@yahoo.com

Abstract

Current technologies in Information Retrieval, Information Extraction, Text Categorization, Automatic Summarization, etc. are all quite superficial. They treat orthographic units as words without regard to important linguistic phenomena such as phrase, idiom or compound. Tokenization into sequences of characters separated by spaces does not always give us linguistically meaningful units. More importantly, most current systems use the so called *bag of words* model where texts are represented as (unordered) sets of words, without regard to word order. *Form follows function* and the syntactic structure of a sentence has an important role in determining the meaning of a sentence. Substantial improvements in text processing systems can be achieved only if we include in depth analysis, at least at the level of syntax. This paper is about our efforts in building wide coverage, robust syntactic parsing systems.

Recently, there is an increasing interest in integrating rule based methods with statistical techniques for developing robust, wide coverage, high performance parsing systems. In this paper, we have proposed a methodology for building wide coverage shallow parsers by a judicious combination of linguistic and statistical techniques without need for large amount of training corpus to start with. We propose an architecture, called UCSG Shallow Parsing Architecture. In the UCSG architecture, a Finite State Grammar is designed to accept *all* valid word groups and a separate statistical component, encoded in HMMs (Hidden Markov Model), has been used to rate and rank the word groups so produced. We then use a best first search strategy to produce parse outputs in best first order, without compromising on the ability to produce all possible parses. We have proposed a bootstrapping strategy for improving HMM parameters as well as the performance of the parser. These ideas have been demonstrated by building a wide coverage partial parsing system for English.

Key Words:- Chunking, Shallow Parsing, Finite State Grammar, HMM, Best First Search, UCSG Architecture

1 Introduction

Although a lot of work has gone into developing full syntactic parsers, high performance, wide coverage syntactic parsing has remained a difficult challenge [1, 2]. In recent times, there has been an increasing interest in wide coverage and robust but partial or shallow parsing systems. Shallow parsing is the task of recovering only a limited amount of syntactic information from natural language sentences. Often shallow parsing is restricted to finding phrases in sentences, in which case it is also called chunking. Steve Abney[3], has described chunking as *finding syntactically related non-overlapping groups of words*. In CoNLL chunking task[4], chunking was defined as *the task of dividing a text into syntactically non-overlapping phrases*. The term *phrase* has come to acquire a very special technical connotation in linguistics and in order to avoid confusion, chunks are also referred to as *word groups*.

As an example, the sentence "He reckons the current account deficit will narrow to only # 1.8 billion in September" could be analyzed as follows by a chunker [4]:

[NP He] [VP reckons] [NP the current account deficit] [VP will narrow] [PP to] [NP only # 1.8 billion] [PP in] [NP September].

Note that prepositional phrases have not yet been built, let alone resolving ambiguities in prepositional phrase attachment. Nor have the thematic roles been assigned to the chunks. Partial parsing systems do a bit more than chunking while still not promising complete syntactic analysis.

Developing computational grammars is a challenging task, even if we restrict to partial parsing. There are broadly two approaches for the development of grammars - the linguistic approach which depends upon hand-crafted rules, and, the machine learning approach where grammars are learned automatically from a parsed training corpus. Developing hand-crafted grammar rules is a very slow, tedious and difficult task, requiring substantial knowledge and skill on the part of the linguist. Automatic learning of grammars requires, on the other hand, a large and representative parsed training corpus, which is rarely available. Perhaps only a good combination of linguistic and statistical approaches can give us the best results with minimal effort.

In this paper, we propose an architecture for shallow parsing, which we call UCSG Shallow Parsing Architecture. In the UCSG Shallow Parsing Architecture, a Finite State Grammar is designed to accept *all* valid word groups but not necessarily the *only* those word groups that are appropriate in context for a given sentence. Many additional word groups may also be recognized due to lexical ambiguities. The focus in this phase is only on completeness. There is a second module consist-

ing of a set of Hidden Markov Models, which will rate and rank the word groups so produced. Note that we are not pruning, we are only rating and ranking the word groups produced. Then we use a best first search module to produce parse outputs in best first order, without compromising on the ability to produce all possible parses. The aim is to produce all possible parses but hopefully in the best first order. A wide coverage partial parsing system for English has been implemented and tested on large scale data. The system has been implemented in Perl under Linux. All the experiments have been carried out on a system having Pentium IV processor and 1 GB ram.

2 A brief survey of shallow parsing systems

Steve Abney [5] proposed finite state cascade models for the chunking task. Grefenstette [6] proposed methods to use finite state transducers for partial parsing. Parsing with finite state transducers [7] was very popular in the early ninety's. Marc Vilain et al. [8] used rule based sequence processors for the chunking task. Herve Dejean [9] used ALLiS (Architecture for Learning Linguistic Structure), which is a symbolic machine learning system for the chunking task.

Miles Osborne [10] proposed maximum entropy based POS tagger for the chunking task. Veenstra and Bosch [11] used memory based learning for chunking. Zhou et al. [12] proposed error driven HMM based chunk tagger with context dependent lexicon. Rob Koeling [13] applied maximum entropy models for chunking. Christer Johansson [14] proposed context sensitive maximum likelihood approach for chunking task. Tong Zhang et al. [15] proposed generalized winnow algorithm for text chunking. Recently Fei Sha and Pereira [16] used conditional random fields for noun phrase chunking and achieved good performance.

Molina and Pla [17] proposed shallow parsing with specialized HMMs. Carreras et al. [18] used perceptrons for chunking task. Recently, Gondy et al. [19] proposed a shallow parser based on closed-class words to capture relations in biomedical text.

Taku Kudoh et al. [20] proposed SVMs for chunking. This system performed the best in CoNLL-2000 chunking task and achieved an F-measure of 93.48%. Van Halteren [21] proposed Weighted probability distribution voting algorithm (WPDV) for chunking task. Tjong Kim Sang [22] proposed combination of several memory based learning systems for chunking task.

Most of the parsers described in literature have used either only rule based tech-

niques or only machine learning techniques. Hand-crafting rules in the linguistic approach can be very laborious and time consuming. Parsers tend to produce a large number of possible parse outputs and in the absence of suitable rating and ranking mechanisms, selecting the right parse can be very difficult. Statistical learning systems, on the other hand, require large and representative parsed corpora for training.

Recently, there is an increasing interest in integrating shallow parsers with deep parsing. Berthold Crysmann et al. [23] reported an implemented system called WHITEBOARD which integrates different shallow components with a HPSG based deep parsing system. Ronald M. Kaplan et. al. proposed a hybrid architecture called XLE [24] for combining finite state machines with LFG grammar. In XLE system, first the surface forms are run through the FST morphology to produce the corresponding stems and tags. Stems and tags each have entries in the LFG lexicon. Sub-lexical phrase structure rules produce syntactic nodes covering these stems and tags and standard grammar rules then build larger phrases.

Literature survey and detailed analyses show that:

- Even shallow or partial parsing of natural languages is quite challenging.
- Testing and evaluation of shallow parsers has been carried out only on limited amount of data (say, 2000 sentences from WSJ corpus) in most cases. Performance on large scale real life data is not clear.
- Testing and evaluation of parsers is a difficult task. Parsing accuracy of trained parsers is known to depend significantly on stylistic similarities between training corpus and test data. For example, Chris Huyck's plink parser [25] was trained on Wall Street Journal portion of the Penn Tree Bank (PTB) and when it was tested on Penn Treebank and Susanne corpus, there was a significant variation in parser performance. Daniel Gildea [26] studied variation of parser performance on different corpora and observed the same effect.
- Some of the most accurate parsers namely Collins and Charniak parsers use lexical co-occurrence statistics in the parsing model. Daniel Gildea [26], in his paper quoted that "lexical co-occurrence probabilities seem to be of no benefit when attempting to generalize to a new corpus".
- High performance has been achieved only under restricted conditions. For example, in CoNLL 2000 chunking task[4] prepositions were not fully disambiguated, prepositional phrases not built and no attempt made to resolve ambiguities relating to attachment of prepositional phrases.

- In the literature, mostly, the performance of the shallow parsers is measured in terms of individual chunk types produced rather than the correct chunk sequence or parse for a whole sentence.
- A parser also needs to have good generalization capacity for other domains. Current systems have not been shown to be good at this.
- Most systems have used either a linguistic approach or a machine learning approach. There is a lot of scope for exploring combinations of linguistic and machine learning approaches in syntactic parsing.
- Given the richness of syntactic structure, large amounts of high quality parsed corpora are required for statistical approaches. The largest training corpora available for English are hardly a few hundred thousand sentences. In many languages of the world, hardly any parsed corpora are available. Further, training corpora must be suitable for a given grammar or grammar formalism. There are strong corpus effects.
- While labelled training data is difficult to build, large scale unlabelled training data (that is, plain or POS tagged text corpus) is readily available or can be easily developed. The challenge is to exploit this for developing wide coverage grammars and parsing systems.
- While several grammars and parsing systems exist for English and other major languages of the world, Indian languages are lagging far behind. There are hardly any substantial computational grammars for any of the Indian languages. Parsed corpora are also not available and hence machine learning approaches cannot be applied right away.

3 UCSG Shallow Parsing Architecture

UCSG shallow parsing architecture is set within the UCSG full parsing framework that was initiated in the early 1990's at University of Hyderabad by Kavi Narayana Murthy[27]. See figure 1 for UCSG full parsing framework. There are three modules in this architecture.

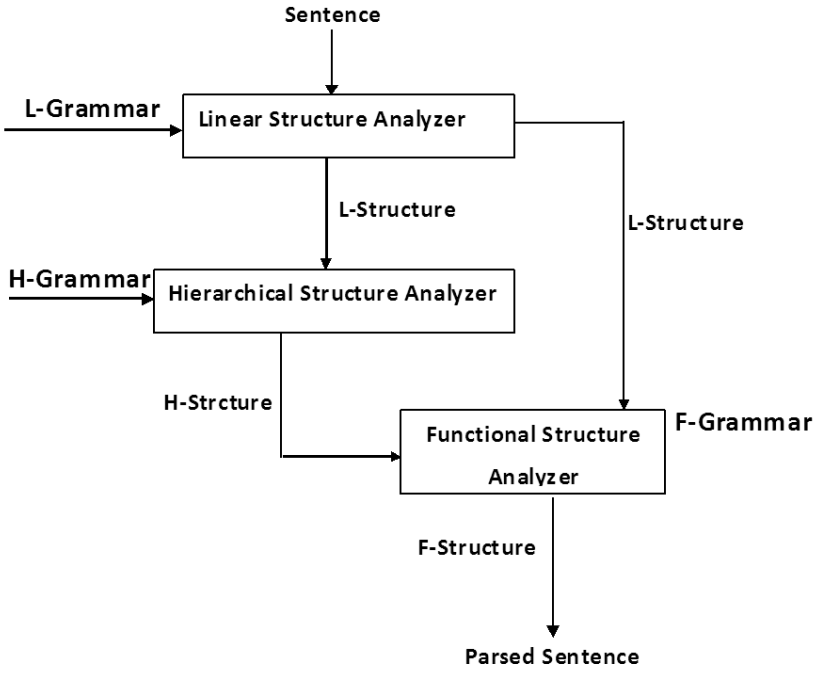

Figure 1: UCSG Full Parsing Framework

The L-module takes a sentence as input and identifies all possible word groups (chunks) in a single linear scan of the sentence using a finite state grammar in linear time. For further work, we only look at the chunks, not the individual words. The H-module uses a small number of context free grammar rules to recognize clauses, identify inter-relationships between clauses and to some extent determine clause boundaries. This is done by recognizing that every clause has a verb group and clause types and clause boundaries are marked by certain markers called sentinels. UCSG shows that a small number of simple and universal rules are sufficient. Parametric variations of the same grammar rules apply for different languages. Having recognized the hierarchical structure of clauses in a sentence, UCSG proposes a 'work from whole to part' strategy to analyze the functional structure in the last module named the F-module. This particular way of modularization has been shown to be universal and computationally simple and efficient as well. See [27] for ore details.

In this paper, the focus is only on chunking - identifying chunks or word groups, handling ambiguities, and producing parses (chunk sequences) for given sentences. This can be extended to include thematic role assignment and clause structure analysis leading towards a full parser.

Purely linguistic approaches have not proved practicable for developing wide coverage grammars and purely statistical or machine learning approaches are also impracticable in most cases due to the non-availability of large enough parsed training corpora. Only a judicious combination of the two approaches can perhaps led to wide coverage grammars and robust parsing systems. UCSG shallow parsing architecture proposes one such solution [28]. Figure 2 shows the basic UCSG Shallow Parsing Architecture.

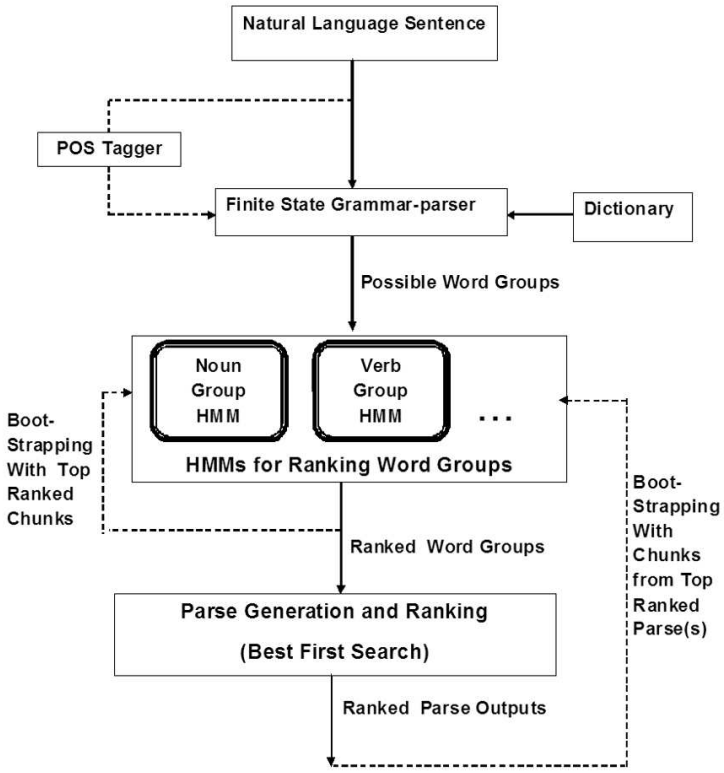

Figure 2: UCSG Shallow Parsing Architecture

The input to the parsing system is one sentence, either plain or POS tagged. Output is an ordered set of parses. The aim is to produce all possible parses in ranked order hoping to get the best parse to the top. In this work, by parse we mean a sequence of chunks. Chunks are sequences of words.

A chunk or a "word group" as we prefer to call it in UCSG, is "a structural unit, a non-overlapping and non-recursive sequence of words, that can as a whole, play a

role in some predication [27]". Note that word groups do not include clauses (relative clauses, for example) or whole sentences. Every word group has a head which defines the type of the group. Word groups can be classified into verb groups, noun groups, adjective groups and so on based on the essence of the meaning as indicated by the *head* of the word group. Thus word groups are similar to *chunks* [29, 30]. Our word groups are also very similar to the *phrases* defined in the work of Beata Megyesi [31]. It may be noted that the terms *chunk* and *phrase* have been used in substantially different connotations elsewhere in literature. The word groups we produce in UCSG are hopefully closer to ideal, semantically oriented units of full parsing, as can be seen from the examples given at the end.

In our UCSG syntax, the definition of a chunk is motivated by question-answering perspective. Consider

Sentence: I am studying at University of Hyderabad.

UCSG outputs the following word groups

```
<ng>[<PNN><i>]</ng>
<vg>[<VBB><am> <VVG><studying>]</vg>
<ng>[<PRP><at> <NN1><university> <PRF><of>
<NP0><hyderabad>]</ng>
```

The word groups produced can thus be viewed in terms of answers to basic questions such as who, whom, where, when etc. For example, if you ask a question "where are you studying", the answer is "at University of Hyderabad". Observe that many chunking systems in the world today treat prepositions as chunks in their own right. Some chunkers break 'University of Hyderabad' into two chunks. See examples below:

Memory based shallow parser [32, 33] gives the following output:

```
[NP I/PRP NP] [VP am/VBP studying/VBG VP]

{PNP [Prep at/IN Prep] [NP University/NNP NP] PNP}

[Prep of/IN Prep]Hyderabad//VBD ./.
```

Note that the word Hyderabad is not part of any chunk.

CCG shallow parser [34] gives the following chunks:

```
[NP I] [VP am studying] [PP at] [PP of] [NP Hyderabad]
```

The word “University” is missing altogether.

Thus our word groups are a bit more semantically oriented and as such, more suitable for deep parsing as also for various NLP applications. We have set for ourselves a more challenging task and our results must be viewed keeping this in mind.

3.1 Finite State Grammar-Parser

Only linear order, repetition and optional items are relevant for recognizing chunks - there are no nested or recursive structures to consider. Finite state grammars efficiently capture linear precedence, repetition and optional occurrence of words in word groups but not arbitrarily deep hierarchical nestings or general dependencies across constituents. Finite state machines are both necessary and sufficient for recognizing word groups [27]. It is also well known that finite state machines are computationally efficient - linear time algorithms exist for recognizing word groups. Finite state grammars are also conceptually simple and easy to develop and test. It may be repeated that detailed analysis of the internal structure of word groups (modifier-modified relationships, for example) is beyond the scope of the current system.

The Finite State module accepts a sentence (either already POS tagged or tagged with all possible categories using the dictionary) and produces an unordered set of possible chunks taking into account all lexical ambiguities.

During linear structure analysis all potential groups in a given sentence are to be recognized. Linear structure analysis takes care of lexical ambiguities and groups may overlap one another. The following algorithm identifies all potential word groups in a given sentence in a single left-to-right scan. This algorithm works for both deterministic and nondeterministic state transition diagrams. It simulates parallel processing. Instead of maintaining a single current state it maintains a current_state_set. Each word in the input sentence is considered only once and an amount of time bounded by the size of the grammar is spent per word. Hence the algorithm is linear in time complexity.

Pseudo Code for Linear Structure Analysis:

```
MAIN()
        initial_state_set := [ ]
        for each of the initial states s_i in the network do
            initial_state_set := union([(s_i,' ')],initial_state_set)
            current_state_set := initial_state_set
            step through the words in the given sentence and for each word w
                advance(current_state_set,w)


ADVANCE(current_state_set,w)
        new_state_set := [ ]
        for each state (s,str) in current_state_set do
            for each out going arc a do
                if any of the categories of w matches the arc a
                  begin
                        new_state_set := union( [(end_state(a),concat(str,w))]
                                                ,new_state_set)
                        if terminal_state(end_state(a)) then
                        begin
                            output(concat(str,w))
                            new_state_set := union(new_state_set,initial_state_set)
                        end
                end
        current_state_set := new_state_set
```

3.2 HMMs for Rating and Ranking Chunks

The second module is a set of Hidden Markov Models (HMMs) used for rating and ranking the word groups produced by the Finite State Grammar. The hope is to get the best chunks near the top. This way, although we are not restricting chunk generation to *only* the appropriate chunks in context, we can hope to get the right chunks near the top and push down others.

Words are observation symbols and POS tags are states in our HMMs. Formally, a HMM model $\lambda = (\pi, A, B)$ for a given chunk type can be described as follows:

Number of States (N) = number of relevant Categories

Number of Observation Symbols (M) = number of Words of relevant categories in the language

The initial state probability

$$\pi_i = P\{q_1 = i\} \tag{1}$$

where $1 \leq i \leq N$, q_1 is a category (state) starting a particular word group type.

State transition probability

$$a_{ij} = P\{q_{t+1} = j | q_t = i\} \tag{2}$$

where $1 \leq i, j \leq N$ and q_t denotes the category at time t and q_{t+1} denotes the category at time t+1.

Observation or emission probability

$$b_j(k) = P\{o_t = v_k | q_t = j\} \tag{3}$$

where $1 \leq j \leq N$, $1 \leq k \leq M$ and v_k denotes the k^{th} word, and q_t the current state.

While building HMMs, a manually checked and certified chunked corpus can be used if available. In this case, HMM parameters can be estimated right away. However, such labelled training data is rarely available. When no parsed corpus is available, we can rely on a POS-tagged corpus. In the latter case, a bootstrapping strategy is proposed to refine the HMM parameters later. See figure 3. We first pass a large POS tagged corpus through the Finite State module and obtain all possible chunks. Taking these chunks to be equiprobable, we estimate the HMM parameters by taking the ratios of frequency counts. One HMM is developed for each major category of chunks, say, one for noun-groups, one for verb-groups, and so on. The B matrix values are estimated from a dictionary that includes frequency counts for each word in every possible category.

We simply estimate the probability of each chunk using the following equation :

$$P(O, Q|\lambda) = \pi_{q_1} b_{q_1}(o_1) a_{q_1,q_2} b_{q_2}(o_2) a_{q_2,q_3} \cdots a_{q_{t-1},q_t} b_{q_t}(o_t) \tag{4}$$

where q_1 ,q_2, $\cdots$, q_t is a state sequence, o_1 , o_2,$\cdots$, o_t is an observation sequence. Note that no Viterbi search involved here and the state sequence is also known. Thus even Forward/Backward algorithm is not required and rating the chunks is therefore

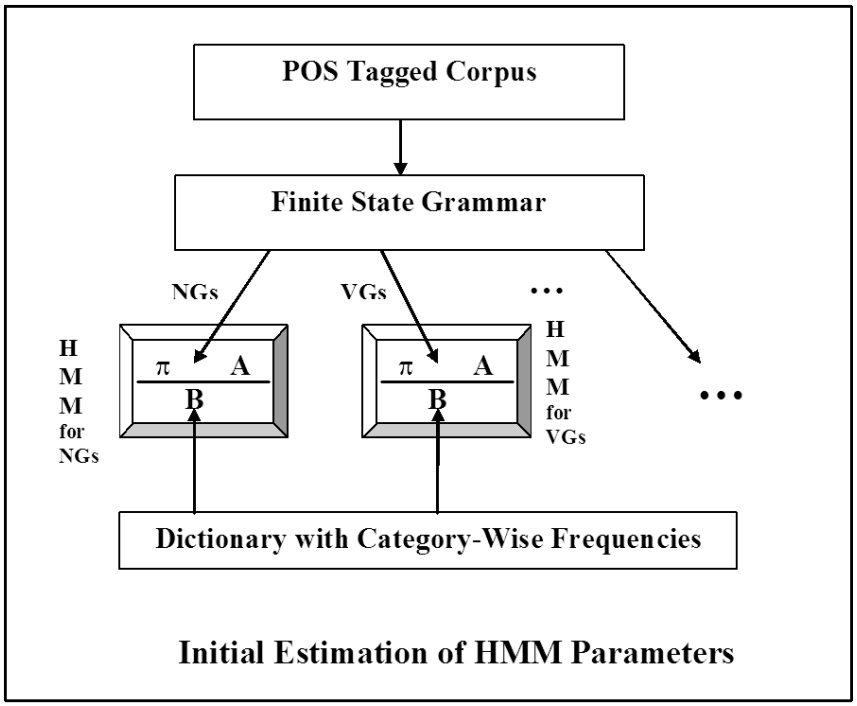

Figure 3: Initial Estimation of HMMs

computationally efficient.

The aim here is to assign the highest rank for the correct chunk and to push down other chunks. Since a final parse is a sequence of chunks that covers the given sentence with no overlaps or gaps, we evaluate the alternatives at each position in the sentence in a left-to-right manner.

Here, we use *Mean Rank Score* to evaluate the performance of the HMMs. Mean Rank Score is the mean of the distribution of ranks of correct chunks produced for a given training corpus. Ideally, all correct chunks would be at the top and hence the score would be 1. The aim is to get a Mean Rank Score as close to 1 as possible.

3.3 Parse Generation and Ranking

The third module is for identifying the best chunk sequence or global parse for a given sentence. This module generates all possible parses, hopefully in best first order. We can of course limit the number of parses generated if required but the ability to produce all possible parses is fundamental to the architecture. Note that we do not produces all possible parses first and then rate and rank them - the parse generation process inherently incorporates best-first search.

Choosing the locally best chunks at each position in a given sentence does not necessarily give us the best parse (chunk sequence) in all cases. The HMMs are local to chunks and global information such as the probability of a chunk of a given type starting a sentence or the probability of a chunk of a particular type occurring next to a chunk of a given type are useful. These probabilities can be obtained from a fairly small chunked corpus. We have used best first search algorithm to get the best parse (chunk sequence) for a given sentence.

Best First Search Algorithm

In this section, we map our parse selection problem into a graph search problem and show how best first search algorithm can be used to find the best first parse.

Words and chunks in a sentence are referred to in terms of the positions they occupy in the sentence. Positions are marked between words, starting from zero to the left of the first word. The very first word is between positions 0 and 1. A word group containing the third and fourth words in the sentence can be referred as $W_{2,4}$.

The following steps describe how we map a given sentence and word groups present in the sentence into a graph.

- The positions in the sentence are treated as nodes of the resulting graph. If a sentence contains N words then the graph contains $N+1$ nodes corresponding to the $N+1$ positions in the sentence.
- Word group $W_{i,j}$ is represented as an edge form node i to node j.
- The probability of a word group $W_{i,j}$ given by HMM module and the transition probability from previous word group type to current word group type are combined to estimate the cost of an arc between the nodes i and j.
- We always start from the initial node 0. Length of the sentence N is the goal node.

Now our parse selection problem of a sentence containing N words becomes the task of finding an optimal path from node 0 to node N.

Pseudo Code for Best First Search Algorithm:

start_node = 0
goal_node = N #(length of the sentence)
cur_best = $< 0, 0, , , > \# < pos, prob, chunktype, path, parse >$
open_set = $\emptyset$
for i = 1 to k do
    repeat
        open_set = add_successor (cur_best, open_set)
        cur_best = find_best (open_set)
    until (cur_best.pos = goal_node)
    open_set = open_set − cur_best
    print cur_best
done
function add_successor(cur_best, open_set)
chunkset = $\{x | x \in$ *CHUNKS and x.from = cur_best.pos*$\}$
foreach (chunkset) do
    elem.pos = chunkset[i].to
    elem.prob = cur_best.prob + chunkset[i].prob +
        P (cur_best.chunktype, chunkset[i].type)
    elem.chunktype = chunkset[i].type
    elem.path = update (cur_best.path, chunkset[i])

elem.parse = update_parse (cur_best.parse, chunkset[i])
open_set = open_set $\cup$ *elem*
done
open_set = open_set $-$ *cur_best*

In best first search, we can inspect all the currently-available nodes, and rank them on the basis of our partial knowledge. Here high rank means that the node looks most promising in relation to the goal. At each step, we select the most promising of the nodes we have generated so far. We then expand the chosen node to generate it successors. If one of them is a solution, we can quit. If not, all those new nodes are added to the set of nodes generated so far. Again the most promising node is selected and the process continues. In the worst case, the best first search algorithm runs in exponential time because it expands many nodes at each level. In big-O notation, this is stated as $O(b^m)$, where b is the branching factor (i.e., the average number of nodes added to the open list at each level), and m is the maximum length of any path in the search space. Memory consumption is also a big problem, apart from time complexity. The number of nodes that are stored in memory rapidly increases as the search moves deeper into the graph and expanding too many nodes can cause the algorithm to run out of memory.

Beam search is a heuristic search algorithm that is an optimization over best-first search. Like best-first search, it uses a heuristic function to estimate the promise of each node it examines. Beam search, however, only unfolds the first m most promising nodes at each depth, where m is a fixed number, the "beam width". While beam search is space-bounded as a function of m, it is neither optimal nor complete when m is finite. As m increases, beam search approaches best-first search in complexity.

Here we propose a modified beam search strategy. We do not restrict the beam width to a fixed number, instead we put bounds on the probabilities of the alternatives available. If the next best available alternative is not promising enough in relation to the promise of the best alternative, we prune. It is of course possible to incorporate a wide variety of other statistical and machine learning techniques for optimum chunk sequence selection. We would need a reasonable sized high quality chunked corpus for training. We have also explored A* best first search strategy. Linguistic constraints should be expected to play an important role in parse generation and ranking.

3.4 Bootstrapping

The HMM parameters can be refined through bootstrapping. Since we need to work with large data sets running into many hundreds of thousands of sentences,

Baum-Welch parameter re-estimation would not be very practical. Instead, we can use parsed outputs to re-build HMMs. It may be recalled that originally HMMs were built from chunks obtained from the over-general finite state parser taking all chunks as equi-probable. By parsing a given sentence using the system and taking the top few parses only as training data, we can re-build HMMs that will hopefully be better. We can also simply use the top-ranked chunks for re-building the HMMs. This would reduce the proportion of invalid chunks in the training data and hence hopefully result in better HMM parameters. In the next section, we shall see that this idea works and we can improve HMM parameters and improve parser performance as well.

These ideas and claims are substantiated with experimental work as detailed in the next section.

4 Experiments and Results

4.1 Lexicon

Lexicon is the heart of any natural language parser. We have developed a lexicon of 138,000 head words including frequency of occurrence for each tag for each word. The lexicon has been obtained from the British National Corpus(BNC) [35], an English text corpus of about 100 Million words, after a considerable amount of analysis and pre-processing. It may be noted that the BNC corpus is POS tagged but not parsed. Closed class words have been manually checked. The lexicon has a coverage of 98% on the BNC corpus itself, 86% on the Reuters News Corpus [36] (about 180 Million words in size), 96.36% on the Susanne parsed corpus [37] and 95.27% on link parser dictionary.

4.2 Sentence Boundary Detection

We have developed a sentence segmentation module using BNC corpus as training data. We have used features such as delimiter, prefix, suffix and after-word and extracted patterns from BNC corpus. We have divided instances of features collected into 20 different random sets where each set contains 50000 samples. We have tested these samples using decision tree algorithms including ID3 and J4.8 using WEKA. Each test set has been subjected to 10 fold cross validation. We have obtained the F-measure for each random set. An average F-Measure of 98.70% has been obtained, comparable to other published results. See [38] for more details.

4.3 Tag Set

We have studied various tag sets including BNC C5, BNC C7, Susanne and Penn Tree Bank tag sets. Since our work is based on BNC 96 edition with C5 tag set, we have made some extensions as and when required. We have totally 71 tags in the extended tag set.

Examples of Tags Added:

There is no distinction between nominative, accusative and possessive pronouns in the C5 tag set. This distinction is very much required in eliminating many ungrammatical sentences. We have introduced four tags for accusative pronoun(PNA), nominative pronoun(PNN), both nominative and accusative pronoun(PNC) and possessive pronouns(PPS).

There is no distinction distinction between interrogative pronoun and relative conjunction in C5 tag set. Hence, we have added one more tag "CJR" for relative conjunctions.

We have introduced new tag for the pre-determiners called "DTP".

We have distributed the frequencies for the newly introduced tags by manual observation of some random samples either from our own manually parsed corpus or the BNC corpus itself. For example, the word 'which' is tagged only as "DTQ" in BNC corpus. According to UCSG grammar, it can be either of the three tags, namely, relative conjunction, pronoun and determiner. We have taken examples from manual parsed corpus and studied the distribution of tags in manual parsed corpus. We found that 60% of the times the word 'which' is tagged as "CJR", 25% of the times as "PNQ" and 15% of the times as "DTQ". The frequencies are distributed accordingly.

4.4 Manually Parsed Corpus Development

We have developed a manually parsed corpus of 4000 sentences, covering a wide variety of sentence structures. 1000 sentences have been randomly selected from BNC corpus, 1065 sentences from 'Guide to Patterns and Usage in English' (hereinafter referred to as GPUE corpus) [39] and 1935 sentences from CoNLL-2000 test data. This corpus is thus very useful for evaluating the various modules of the parsing architecture and also for bootstrapping.

This corpus was developed by parsing the sentences using the UCSG shallow

parser for English and then manually checking the top parse and making corrections where required. We felt this was far easier than parsing the sentences entirely by hand.

4.5 Preprocessing Steps: Tagging

In the preprocessing step, plain sentences are tagged using the dictionary. Here, we have considered all possible tags in the dictionary for a given word. In case, the word is not found in the dictionary we have used morphological rules to find its tag. The most important aspects of inflectional morphology of English including plurals for nouns, past tense, gerundial and participial forms of verbs and degrees of comparison for adjectives are handled. Derived forms are directly found in the dictionary.

The following are the most productive rules for generating inflectional forms in English:

- plural forms of noun and -s form of lexical verbs
- Superlative forms of adjectives (e.g. oldest, hottest, gravest)
- Comparative forms of adjectives (e.g. better, older)
- -ing forms of lexical verbs (e.g. forgetting, living, returning)
- Past and Past participle forms of lexical verbs (e.g. lived, returned, whetted)

Finally, if the word is directly not found in the dictionary and the root of that word from morphological analysis also not found in the dictionary, we have considered the word as proper noun and assigned NP0 tag for the word.

A POS tagger can be included.

4.6 Finite State Grammar

We have developed a nondeterministic finite state grammar for identifying English word groups. The Finite State Machine has a total of 50 states of which 24 are final states. See [40] for further details.

4.6.1 Example

Sentence: The sun rises in the east.

Actual word groups in the given sentence

```
<ng><0-2><AT0><the>##<NN1><sun> <vg><2-3><VVZ><rises>
<ng><3-6><PRP><in>##<AT0><the>##<NN1><east>
```

The following word groups are produced by our FSM:

```
<ng><0-2><AT0><the>##<NN1><sun>
<ng><0-3><AT0><the>##<NN1><sun>##<NN2><rises>
<ng><0-6><AT0><the>##<NN1><sun>##<NN2><rises>##<PRP><in>
##<AT0><the>##<NP0><east>
<ng><0-6><AT0><the>##<NN1><sun>##<NN2><rises>##<PRP><in>
##<AT0><the>##<NN1><east>
<ng><1-2><NN1><sun>
<ng><1-3><NN1><sun>##<NN2><rises>
<ng><1-6><NN1><sun>##<NN2><rises>##<PRP><in>##<AT0><the>
##<NN1><east>
<ng><1-6><NN1><sun>##<NN2><rises>##<PRP><in>##<AT0><the>
##<NP0><east>
<vg><2-3><VVZ><rises>
<ng><2-3><NN2><rises>
<vg><2-4><VVZ><rises>##<AVP><in>
<ng><2-6><NN2><rises>##<PRP><in>##<AT0><the>##<NN1><east>
<ng><2-6><NN2><rises>##<PRP><in>##<AT0><the>##<NP0><east>
<part><3-4><AVP><in>
<ng><3-6><PRP><in>##<AT0><the>##<NN1><east>
<ng><3-6><PRP><in>##<AT0><the>##<NP0><east>
<ng><4-6><AT0><the>##<NN1><east>
<ng><4-6><AT0><the>##<NP0><east>
<ng><5-6><NN1><east>
<ng><5-6><NP0><east>
```

We have evaluated the performance of the FSM module on various corpora - Susanne Parsed Corpus, CoNLL 2000 test data set and on our manually parsed corpus of 4000 sentences. The evaluation criteria is Recall alone since the aim here is only to include the correct chunks.

The Susanne corpus [37] is a manually parsed corpus containing about 130,000 words in 6891 sentences. Some preprocessing was necessary. Ambiguities with

apostrophes have been resolved. Spelling errors mentioned in the Susanne documentation have been corrected. Since the structure of the parse output in the Susanne corpus differs somewhat from that of UCSG, suitable mapping schemes had to be developed and validated [41]. Plain text sentences were extracted and given as input to the UCSG shallow parser.

In Susanne Corpus, phrases are classified into eight types [37] namely, verb phrase, noun phrase, adjective phrase, adverb phrase, prepositional phrase, determiner phrase, numeral phrase, genitive phrase.

Results are given in table 1 for Noun, Verb, Adjective and Adverb groups.

Table 1: Performance of the Finite State Parser on Susanne Corpus

Word Group Type	No. of Groups in Test Data	No. of Groups Recognized	% Recall
Noun Group	47735	41016	85.92
Verb Group	17559	17179	97.83
Adjective Group	2619	1733	66.17
Adverb Group	5516	4701	85.22
Overall	73429	64629	88.02

Overall, 88.02% of phrases in the Susanne corpus have been correctly identified. 97.83$\%$ of all the verb groups could be correctly identified. Failures in the case of verb groups are limited to complex cases such as "have never, or not for a long time, had".

We have done analysis of the word groups that are not covered by our FSM grammar. The main reason for failures we found is that in Susanne corpus the definition of phrases are very much different from the chunks we are using here. Some phrases in Susanne corpus are recursive in nature. We have given a few examples of failures here.

The examples given below are the noun phrases in Susanne corpus, which include other phrases or clauses within the noun phrases themselves.

- $< ng >$ of the little pink woman who chose to be called auntie
- $< ng >$ the largest majority given a candidate in recent years
- $< ng >$ in a society deeply fissured by antagonisms

The examples given below are the adjective phrases in Susanne corpus, which include other phrases within the adjective phrases themselves.

- $< ajg >$ comfortable about her child
- $< ajg >$ as neat as i can

As another example of the kinds of differences, the word "today" is considered as noun in UCSG dictionary, where is it is treated as adverb in Susanne corpus.

The CoNLL 2000 test data set consists section 20 of the Wall Street Journal corpus (WSJ) and includes 47377 words and 23852 chunks. In the current evaluation, LST chunks (list items) have been excluded. Also, in the UCSG framework, there are no separate PPs - PPs are included in noun groups. Table 2 gives the performance in the first set of experiments [28].

Table 2: Evaluation of Finite State Parser on CoNLL 2000 Test Data Set

CoNLL Chunk Type	UCSG Terms	Chunks in Test Data	Chunks Recognized	% Recall
NP	ng	12422	10588	85.24
VP	vg,infg,vgs	4658	3786	81.28
ADVP	avg	866	698	80.60
ADJP	ajg,ags	438	398	90.87
SBAR	sub,rel	535	507	94.77
PRT	part	106	105	99.06
CONJP	sub	9	9	100.00
INTJ	intg	2	1	50.00
Total		19036	16092	84.53

There are a few minor differences in the way chunks are defined in the CoNLL 2000 chunking task and UCSG. Punctuation marks are removed by a pre-processor and handled separately elsewhere in UCSG. Currency symbols such as $ and # are considered part of numbers in UCSG while they become separate words in CoNLL. CoNLL splits chunks across the apostrophes in genitives as in *Rockwell International Corporation's tulsa unit* while UCSG does not. To-infinitives as in *continue to plummet* are recognized separately in UCSG while they may form part of a VP in CoNLL. Also, in keeping the UCSG philosophy, PPs are not recognized separately in UCSG, they are included in noun groups. In order to get a better feel for the true performance of the UCSG shallow parser, the above differences were discounted for and performance checked again. The results are given in Table 3. There is no

change in the performance for other groups. Overall, 18185 out of 19130 chunks have been correctly identified, giving a Recall of 95.06%.

Table 3: Evaluation of the Finite State Parser on CoNLL Data Set after mapping

CoNLL Chunk Type	UCSG Terms	Chunks in Test Data	Chunks Recognized	Recall (%)
NP,PP	ng	12261	11605	94.65
VP	vg	4283	4223	98.60
-	infg	625	610	97.6
ADVP	avg	866	710	82.56
ADJP	ajg	438	414	94.52
SBAR	sub	544	517	95.03
PRT	part	106	105	99.06
INTJ	intg	2	1	50

Table 4 gives the performance of the FSM module on the manually parsed corpus. From the table 4, we can observe that very high recall (99.56%) is achieved on manually parsed corpus.

Table 4: FSM Evaluation on Manually Parsed Corpus

Chunk type	Symbol	No of Chunks in Corpus	No. of Chunks Found	Recall (%)
Noun	ng	15648	15627	99.86
Verb	vg	6827	6817	99.85
Adverb	avg	908	836	92.07
Adjective	ajg	869	863	99.31
Coordinate conjunction	coord	460	457	99.35
Subordinate conjunction	sub	1048	1048	100
Relative conjunction	rel	460	460	100
Particle	part	31	31	100
To infinitive	infg	955	948	99.27
Interjection	intg	7	7	100
Adjective special	ags	15	15	100
Verb special	vgs	475	475	100
Total	-	27703	27584	99.56

We have done analysis of the word groups that are not covered by our FSM grammar. The main reason we found that in CoNLL corpus, some of the words

have tag differences. For example, the word “according to” is a single preposition in UCSG dictionary where as the words are tagged as separate prepositions in CoNLL corpus. Multi-token adverbs such as ‘at last’, ‘no longer’ are not identified by our grammar as on date. There are also tag differences between CoNLL and UCSG tag set. We have considered the word ‘today’ as noun in our dictionary whereas in CoNLL it is considered as adverb.

The table 5 shows the number of extra phrases produced by the over generalization of FSM grammar. In manually parsed corpus, there are 27703 correct chunks.

Table 5: Analysis of FSM Module - Test Data of 4000 sentences having 27703 phrases in Manually Parsed Corpus

	Plain	POS tagged
Number of phrases produced by FSM module	313306	136926
% of correct chunks recognized by FSM module	99.56	99.96

We may conclude that our finite state grammar is very good in recognizing the correct chunks in most cases. By design, the FSM also produces other possibilities and the UCSG architectures provides a separate module for rating and ranking the chunks produced by the FSM so that the best ones can be selected for further processing.

4.7 Developing HMMs

HMMs were initially developed from 3.7 Million POS-tagged sentences taken from the BNC corpus. Sentences with more than 40 words were excluded. Since we use an extended C5 tag set, POS tags had to be mapped to the extended set where necessary. HMM parameters are estimated from the chunks produced by the Finite State grammar, taking all chunks to be equi-probable. Separate HMMs are built for noun groups, verb groups, adjective groups, adverb groups, infinitive groups and one HMM for all other chunk types.

The probability of a given chunk $P(O, Q|\lambda)$ has been calculated using the equation

$$P(O, Q|\lambda) = \pi_{q_1} b_{q_1}(o_1) a_{q_1,q_2} b_{q_2}(o_2) a_{q_2,q_3} \cdots a_{q_{t-1},q_t} b_{q_t}(o_t) \quad (5)$$

where q_1 ,q_2, $\cdots$, q_t is a state sequence, o_1 , o_2,$\cdots$, o_t is an observation sequence.

The chunks are ranked accordingly. It is interesting to observe the Recall and Mean Rank Score within the top k ranks, where k is a given cutoff rank. Table 6

shows that there is a clear tendency for the correct chunks to bubble up close to the top. For example, more than 95% of the correct chunks were found within the top 5 ranks. Nearly 99% of the correct chunks are within a rank of 10.

Table 6: Performance of the HMM Module on the Manually Parsed Corpus of 4000 sentences - Plain Sentences as Input

Cutoff	Plain Sentences as Input		POS Tagged Sentences as Input	
	Mean Rank Score	Cumulative Recall (%)	Mean Rank Score	Cumulative Recall (%)
1	1	43.06	1	62.74
2	1.38	69.50	1.28	86.97
3	1.67	84.72	1.43	95.64
4	1.85	91.69	1.50	98.31
5	1.96	95.13	1.54	99.25
6	2.04	96.91	1.55	99.61
7	2.08	97.80	1.56	99.72
8	2.12	98.39	1.56	99.79
9	2.14	98.70	1.57	99.81
10	2.16	98.93	1.57	99.82

4.7.1 Example

Sentence: The sun rises in the east.

The following word groups and their ranks given by HMM module

```
<ng><0-2><AT0><the>##<NN1><sun> <-10.8199668891226><1><4><1>

<ng><0-3><AT0><the>##<NN1><sun>##<NN2><rises>
<-22.645126557751><2><4><1>

<ng><0-6><AT0><the>##<NN1><sun>##<NN2><rises>##
<PRP><in>##<AT0><the>##<NN1><east> <-35.0961918977221><3><4><1>

<ng><0-6><AT0><the>##<NN1><sun>##<NN2><rises>##
<PRP><in>##<AT0><the>##<NP0><east> <-36.6074325860112><4><4><1>

<vg><2-3><VVZ><rises>   <-10.3267169484799><1><5><2>

<ng><2-3><NN2><rises>   <-11.7411565945832><2><5><2>

<vg><2-4><VVZ><rises>##<AVP><in> < -16.744490507491><3><5><2>
```

```
<ng><2-6><NN2><rises>##<PRP><in>##<AT0><the>## <NN1><east>
<-24.1922219345543><4><5><2>

<ng><2-6><NN2><rises>##<PRP><in>##<AT0><the>## <NP0><east>
<-25.7034626228434><5><5><2>

<part><3-4><AVP><in>    <-5.39011993798651><1><3><3>

<ng><3-6><PRP><in>##<AT0><the>##<NN1><east>
<-13.3023793305427><2><3><3>

<ng><3-6><PRP><in>##<AT0><the>##<NP0><east>
<-14.8136200188319><3><3><3>

<ng><4-6><AT0><the>##<NN1><east> <-10.6864467975293><1><2><4>

<ng><4-6><AT0><the>##<NP0><east> <-12.1976874858185><2><2><4>
```

Each entry includes the chunk type, the starting and ending positions, the chunk itself with the POS tags of all the words, log probability given by HMM, rank, number of items in the set, and the serial number of the branching points. It may be noted that the correct chunks have been ranked at 1, 1 and 2 respectively.

We have also done some experiments to see the effect of the size of training data used to build HMMs on HMM performance. We have found that as we use more and more training data, the HMM performance is improving significantly. Since we are dealing with very large data sets, even a change in the second decimal place is very significant. The results are shown in table 7.

Table 7: Effect of the size of training data on HMM performance

Size of the data (No. of sentences)	Mean Rank
0.1Million	2.29
1 Million	2.27
3.7 Million	2.26

4.8 Parse Generation and Ranking

The parse generation module has been evaluated on the manually parsed corpus in terms of rank of the fully correct parse and also in terms of percentage of correct chunks in the top parse. Plain sentences and POS tagged sentences have been considered separately for input. The results are summarized in table 8. Here, we have restricted the parsing time taken by the best first search algorithm to 3 epoch

seconds for each sentence because the time and space complexity increases exponentially as branching factor (b) and length of the sentence (n) increases.

Table 8: Performance of the Best First Search Module - Test Data of 4000 Sentences

Rank	No. of correct Parses	
	(Plain Sentences)	(POS tagged Sentences)
1	1130	1774
2	351	487
3	185	194
4	85	137
5	70	129
% of Correct parses in top 5	45.52	68.02
% of Correct chunks in top parse	78.70	78.42
Total Recall	54.67	86.45
Time taken to parse	1h:55m:33sec	0h:31m:49sec

From the table 8, we can see that when we restrict best first search module to give best five parses and time limit to 3 epoch seconds, we have 45.52% correct parses within top 5 for plain sentences and 68.02% of correct parses within top 5 for POS tagged sentences. The total number of sentences parsed by the best first search module is only 54.67% for plain sentences and 86.45% for the POS tagged sentences within the stipulated time. It must be noted that since the finite state grammar is recognizing correct chunks with a very high recall and since the HMM modules are used only for ranking and no pruning is done, correct parses will surely be generated in most cases provided we have no time limits.

We have analyzed the complexity involved in exhaustive search to produce all the parses for a given sentence. We have summarized the results in table 9. We can see that the total number of parses for each sentence increases exponentially with the length of the sentence and also branching factor. The results have also shown that POS tagging greatly helps in parsing by reducing the complexity.

Table 9: Analysis of Complexity - Plain Sentences

Corpus	Average Sentence Length	Average No. of Parses	
		Plain Sentences	POS Tagged Sentences
GPUE	7.02	381	5
BNC	15.78	12,428,029	401
CoNLL	20.06	786,473,522,192	81,794

It may be noted that the performance of the parser in terms of its ability to produce the correct parse is limited only by the Finite State Grammar and the dictionary, since the other modules do not resort to any pruning. However, it is conceivable that in practical usage, we may impose a cutoff and attempt to produce only the top k parses. In this latter case, the percentage of cases where the fully correct parse is included could also be observed.

4.8.1 Example

Sentence: The sun rises in the east.

The following parses are the ranked order given by BFS module if we use dictionary tags

```
<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>##<PRP><in>##<AT0><the>##
<NN1><east>]</ng>   -- -35.2345922674581

<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>]</vg>
<ng>[<PRP><in>##<AT0><the>##<NN1><east>]</ng>      --
-35.9226802717702

<ng>[<AT0><the>##<NN1><sun>]</ng>
<ng>[<NN2><rises>##<PRP><in>##<AT0><the>##<NN1><east>]</ng>      --
-36.504440120609

<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>##<PRP><in>##<AT0><the>##
<NP0><east>]</ng>   -- -36.7458329557472

<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>]</vg>
<ng>[<PRP><in>##<AT0><the>##<NP0><east>]</ng>      --
-37.4339209600594

<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>]</ng>
<ng>[<PRP><in>##<AT0><the>##<NN1><east>]</ng>      --
-37.4397571852259

<ng>[<AT0><the>##<NN1><sun>]</ng>
```

```
<ng>[<NN2><rises>##<PRP><in>##<AT0><the>##<NP0><east>]</ng>      --
-38.0156808088982

<ng>[<AT0><the>##<NN1><sun>]</ng> <ng>[<NN2><rises>]</ng>
<ng>[<PRP><in>##<AT0><the>##<NN1><east>]</ng>      --
-38.7096050383768

<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>]</ng>
<ng>[<PRP><in>##<AT0><the>##<NP0><east>]</ng>      --
-38.950997873515

<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>##<AVP><in>]</vg>
<ng>[<AT0><the>##<NN1><east>]</ng>      -- -39.724521297768

<ng>[<AT0><the>##<NN1><sun>]</ng> <ng>[<NN2><rises>]</ng>
<ng>[<PRP><in>##<AT0><the>##<NP0><east>]</ng>      --
-40.220845726666

<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>##<AVP><in>]</vg>
<ng>[<AT0><the>##<NP0><east>]</ng>      -- -41.2357619860571

<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>]</ng>
<part>[<AVP><in>]</part> <ng>[<AT0><the>##<NN1><east>]</ng>      --
-46.222079476789

<ng>[<AT0><the>##<NN1><sun>]</ng> <ng>[<NN2><rises>]</ng>
<part>[<AVP><in>]</part> <ng>[<AT0><the>##<NN1><east>]</ng>      --
-47.4919273299399

<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>]</ng>
<part>[<AVP><in>]</part> <ng>[<AT0><the>##<NP0><east>]</ng>      --
-47.7333201650782

<ng>[<AT0><the>##<NN1><sun>]</ng> <ng>[<NN2><rises>]</ng>
<part>[<AVP><in>]</part> <ng>[<AT0><the>##<NP0><east>]</ng>      --
-49.0031680182291

<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>]</vg>
<part>[<AVP><in>]</part> <ng>[<AT0><the>##<NN1><east>]</ng>      --
-62.3971218959318

<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>]</vg>
<part>[<AVP><in>]</part> <ng>[<AT0><the>##<NP0><east>]</ng>      --
-63.908362584221
```

It may be observed that there are 18 parses and the fully correct parse is in rank two.

We have also implemented a modified beam search algorithm to improve the

parser efficiency in terms of time and space. Here, we have kept a threshold on the probability so that the word groups which are having probability less than the threshold can be pruned. In this way, we can reduce the number of combinations the parser has to explore and also save a good deal of memory. But this may cause pruning of some of correct parses. If we do not want to loose the correct parse, we have to increase the threshold accordingly. As the threshold increases, the complexity approaches that of the best first search. The results in table 10 have been obtained for a beam threshold of 1.

Table 10: Performance of the modified Beam Search - Test Data of 4000 Sentences

Rank	No. of correct Parses	
	Plain	POS tagged
1	1262	1796
2	259	240
3	67	36
4	35	22
5	9	4
% of Correct parses in top 5	40.8	52.45
% of Correct chunks in top parse	67.98	74.31
Total Recall	100.00	99.87
Time taken to parse	0h:15m:31sec	0h:0m:18sec

We have also studied effect of increase in threshold on parse generation. As we increase threshold, the performance approaches that of the best first search, but the time taken to parse will also increase. The results are shown in table 11.

Table 11: Performance of the modified Beam Search with increasing threshold - Test Data of 4000 Sentences

Rank	Threshold 1	Threshold 3
1	1796	1796
2	240	497
3	36	200
4	22	142
5	4	119
% of Correct parses in top 5	52.45	68.85
% of Correct chunks in top parse	74.31	74.31
Time taken to parse	0h:0m:18sec	6h:07m:20sec

We have also studied the percentage correct tags assigned to the words in the top parse of modified beam search module. We have observed that 96.01% of the words are assigned correct POS tags in the top parse. This shows that most of the times the top parse given by the parse generation module is almost correct in terms of POS tags and may only have problems with chunk boundary detection. The results are shown in table 12.

Table 12: Evaluation of the POS tags in the top parse of parse generation module (modified beam search)

Number of words	62268
Number of words assigned Correct POS tags:	59784
% of correct POS tags	96.01

It may be observed that the only kind of linguistic constraints we have used so far is the structure of chunks as captured by the Finite State Grammar. It is in fact interesting to see fully correct parse (that is, chunk sequence) being produced by the system in many cases before applying any sentence level linguistic constraints at all. We have not included a grammar of clause structure, hierarchical structure of clauses and phrases in sentences, or functional structure constraints such as sub-categorization and selectional restrictions or even simple agreement rules. Further improvements to the parser performance will critically depend on judicious application of relevant linguistic constraints within the overall architecture.

Also, more work is needed to assign thematic roles to the chunk sequences produced by the parser.

4.9 Bootstrapping

We hypothesize that the HMM parameters can be refined through bootstrapping. Initial HMMs were developed from chunks produced by the over-general Finite State Grammar, taking all chunks to be equi-probable. Once the HMMs have been built, we can use the same HMMs to rate and rank the chunks and further produce parses using best first search. From the results obtained, it is clear that the top ranked chunks and chunks from the top ranked parses will give us better data for re-building HMMs. The new data sets so generated contain a higher percentage of correct chunks. In other words, noise is reduced. However, the size of the data set also comes down as shown in table 13.

Table 13: Bootstrapping: Data Set Size

HMM development Phase	No. of Sentences	No. of Chunks
Initial HMM building with FSM Output	3770917	122748054
Bootstrapping with HMM Top Ranked Chunks	2008877	22368823
Bootstrapping with Best first search Top Parse	1804827	11061598

To prove the bootstrapping hypothesis, we have carried out several experiments. Plain text sentences from BNC corpus, 5 to 20 words in length, have been used. All possible chunks are obtained using the Finite State State Grammar-Parser and HMMs built from these chunks. In one experiment, only the chunks rated highest by these very HMMs are taken as training data for bootstrapping. In a second experiment, best first search is also carried out and chunks from the top ranked parse alone are taken for bootstrapping. In a third experiment, data from these two sources have been combined. Best results were obtained when the chunks from the top parse alone were used for bootstrapping. Table 14 shows the effect of bootstrapping on HMM module.

Table 14: Effect of Bootstrapping after iteration-1: on 4000 sentences from Manually Parsed Corpus containing a total of 27703 chunks

	Initial			Iteration-1			Iteration-2		
Cut--off	No. of Chunks	Recall	Mean Rank	No. of Chunks	Recall	Mean Rank	No. of Chunks	Recall	Mean Rank
1	11929	43.06	1.0	12611	45.52	1.0	13090	47.25	1.0
2	19254	69.50	1.38	19787	71.43	1.36	20170	72.81	1.35
3	23470	84.72	1.67	23609	85.22	1.63	23811	85.95	1.60
4	25402	91.69	1.85	25418	91.75	1.80	25541	92.20	1.77
5	26356	95.13	1.96	26303	94.94	1.90	26401	95.30	1.87
6	26848	96.91	2.04	26805	96.75	1.98	26863	96.97	1.94
7	27096	97.80	2.08	27078	97.74	2.03	27108	97.85	1.99
8	27257	98.39	2.12	27226	98.28	2.06	27249	98.36	2.02
9	27344	98.70	2.14	27326	98.63	2.09	27336	98.68	2.04
10	27406	98.93	2.16	27393	98.88	2.11	27407	98.93	2.06

It may be observed that the percentage of correct chunks is increasing in the top 4 positions and decreasing thereafter, clearly showing that bootstrapping has helped to rate and rank chunks better.

There is also some improvement in the final parse when the HMMs obtained through bootstrapping are used. See table 15.

Table 15: Effect of Bootstrapping on Parse Generation - (Best First Search - Epoch Time limit 3)

Rank	No. of correct Parses					
	Plain Sentences			POS Tagged Sentences		
	Initial	Iter-1	Iter-2	Initial	Iter-1	Iter-2
1	1130	1172	1210	1774	2113	2193
2	351	308	352	487	470	495
3	185	152	157	194	186	164
4	85	82	83	137	132	129
5	70	72	68	129	89	91
% of Correct parses in top 5	45.52	44.65	46.75	68.02	74.75	76.80
% of Correct chunks in top parse	78.70	83.17	83.92	78.42	87.51	88.26

4.9.1 Example

Sentence: The sun rises in the east.

The following are the top 5 parses in ranked order given by BFS module after bootstrapping if we use dictionary tags

```
<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>]</vg>
<ng>[<PRP><in>##<AT0><the>##<NP0><east>]</ng>      --
-37.1886283215909

<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>]</vg>
<ng>[<PRP><in>##<AT0><the>##<NN1><east>]</ng>      --
-38.8822563306516

<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>]</ng>
<ng>[<PRP><in>##<AT0><the>##<NP0><east>]</ng>      --
-39.357919583567

<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>]</ng>
<ng>[<PRP><in>##<AT0><the>##<NN1><east>]</ng>      --
-41.0515475926276

<ng>[<AT0><the>##<NN1><sun>]</ng> <ng>[<NN2><rises>]</ng>
<ng>[<PRP><in>##<AT0><the>##<NP0><east>]</ng>      --
-41.7994535067038
```

It may be observed that the correct parse is still in second position but the top parse is far better.

In the table 16, we have shown the effect of bootstrapping on modified beam search algorithm results. Interestingly, bootstrapping also improved the performance of beam search. This is due to the fact that the distribution of probabilities among the phrases has improved with bootstrapping.

Table 16: Effect of Bootstrapping on Parse Generation - modified Beam Search with threshold 1

Rank	No. of correct Parses			
	Plain Sentences		POS Tagged Sentences	
	Initial	Iteration-2	Initial	Iteration-2
1	1262	1386	1796	2267
2	259	259	240	188
3	67	57	36	17
4	35	33	22	13
5	9	14	4	6
% of Correct parses in top 5	40.08	43.72	52.45	62.27
% of Correct chunks in top parse	67.98	76.84	74.31	86.32

The performance figures given above need to interpreted with care. We have seen that the percentage correct tags assigned to the words in the top parse is over 96%. This shows that most of the times the top parse given by the parse generation module is almost correct in terms of POS tags and may only have minor problems with chunk boundary detection. The very definition of chunks is much more demanding in UCSG - we expect prepositions to be combined with the appropriate noun groups, we expect correct handling of adverb particles which may be ambiguous with a preposition, etc. A quick check by manual observation shows that in most cases the top parse is reasonably good if not 100% perfect. Also, the top parse may be more or less adequate for applications such as IE. More thorough examination of this aspect is planned and all that we wish to say now is that one should not be disheartened by the not-so-high performance figures depicted here.

4.10 Comparison with other Systems

1. Plain Sentence:

Concern for the environment has always topped our agenda.

Tagged Sentence:

```
<VVB_NN1><concern>##<PRN_PRP_CJS_AVP><for>##<AT0><the>##
<NN1><environment>##<VHZ><has>##<AV0><always>##<VVN_VVD><topped>
##<DPS><our>##<NN1><agenda>##
```

UCSG output:

The chunk types in UCSG shallow parsing system are: 1) ng: noun group, 2) vg: verb group, 3) vgs: verb group special, 4) avg: adverb group, 5) ajg: adjective group, 6) ags: adjective group special, 7) coord: coordinate conjunction, 8) sub: subordinate conjunction, 9) rel: relative conjunction, 10) part: particle group, 11) infg: infinitive group, 12) intg: interjection group.

The top 5 parses from UCSG shallow parser in ranked order are given below. Top parse is fully correct.

```
<ng>[<NN1><concern>##<PRP><for>##<AT0><the>##<NN1><environment>]</ng>
<vg>[<VHZ><has>##<AV0><always>##<VVN><topped>]</vg>
<ng>[<DPS><our>##<NN1><agenda>]</ng>

 <ng>[<NN1><concern>]</ng>
<ng>[<PRP><for>##<AT0><the>##<NN1><environment>]</ng>
<vg>[<VHZ><has>##<AV0><always>##<VVN><topped>]</vg>
<ng>[<DPS><our>##<NN1><agenda>]</ng>


<ng>[<NN1><concern>##<PRP><for>##<AT0><the>##<NN1><environment>]</ng>
<vg>[<VHZ><has>]</vg> <ajg>[<AV0><always>##<VVN><topped>]</ajg>
<ng>[<DPS><our>##<NN1><agenda>]</ng>


<ng>[<NN1><concern>##<PRP><for>##<AT0><the>##<NN1><environment>]</ng>
<vg>[<VHZ><has>##<AV0><always>]</vg> <ajg>[<VVN><topped>]</ajg>
<ng>[<DPS><our>##<NN1><agenda>]</ng>

<ng>[<NN1><concern>##<PRP><for>##<AT0><the>##<NN1><environment>]</ng>
<vg>[<VHZ><has>]</vg> <avg>[<AV0><always>]</avg>
<vg>[<VVD><topped>]</vg> <ng>[<DPS><our>##<NN1><agenda>]</ng>
```

Memory Based Shallow Parser Output:

```
[NP concern/NN NP] {PNP [Prep for/IN Prep] [NP the/DT environment/NN
NP] PNP} [VP has/VBZ always/RB topped/VBN VP] [NP our/PRP agenda/NN
NP] ./.
```

By now it should be very clear as to why it is very important to combine prepositions with noun groups appropriately to get a clear reading of the given sentence. UCSG output is generally far better than the output of other parsing systems.

Cognitive Computation Group Shallow Parser Output:

```
[NP concern] [PP for] [NP the environment] has [ADVP always] [VP
topped] [NP our agenda] .
```

UCSG requires that every word in the given sentence is included in the final parse. Leaving out words like makes the parse output so much less useable.

2. Plain Sentence:

He is one of the authors who are destined to be immortal.

Tagged Sentence:

```
<PNN><he>##<VBZ><is>##<CRD_PNI><one>##<PRN_PRF_AVP><of>
##<AT0><the>##<NN2><authors>##<NP0_CJR_PNQ><who>##<VBB><are>##
<VVN><destined>##<PRN_TO0_PRP_AVP><to>##<VBB><be>##<AJ0><immortal>##
```

UCSG output:

The top 5 parses from UCSG shallow parser in ranked order are given below. Top parse is fully correct. 'who' is treated as a conjunction introducing a relative clause, thereby facilitating extensions to a full parsing system. Compare with the outputs of other parsers below.

```
<ng>[<PNN><he>]</ng> <vg>[<VBZ><is>]</vg>
<ng>[<CRD><one>##<PRF><of>##<AT0><the>##<NN2><authors>]</ng>
<rel>[<CJR><who>]</rel> <vg>[<VBB><are>##<VVN><destined>]</vg>
<infg>[<TO0><to>##<VBB><be>]</infg> <ajg>[<AJ0><immortal>]</ajg>

<ng>[<PNN><he>]</ng> <vg>[<VBZ><is>]</vg>
<ng>[<PNI><one>##<PRF><of>##<AT0><the>##<NN2><authors>]</ng>
<rel>[<CJR><who>]</rel> <vg>[<VBB><are>##<VVN><destined>]</vg>
<infg>[<TO0><to>##<VBB><be>]</infg> <ajg>[<AJ0><immortal>]</ajg>

<ng>[<PNN><he>]</ng> <vg>[<VBZ><is>]</vg> <ajg>[<CRD><one>]</ajg>
<ng>[<PRF><of>##<AT0><the>##<NN2><authors>]</ng>
<rel>[<CJR><who>]</rel> <vg>[<VBB><are>##<VVN><destined>]</vg>
<infg>[<TO0><to>##<VBB><be>]</infg> <ajg>[<AJ0><immortal>]</ajg>

<ng>[<PNN><he>]</ng> <vg>[<VBZ><is>]</vg> <ng>[<CRD><one>]</ng>
<ng>[<PRF><of>##<AT0><the>##<NN2><authors>]</ng>
<rel>[<CJR><who>]</rel> <vg>[<VBB><are>##<VVN><destined>]</vg>
<infg>[<TO0><to>##<VBB><be>]</infg> <ajg>[<AJ0><immortal>]</ajg>

<ng>[<PNN><he>]</ng> <vg>[<VBZ><is>]</vg>
<ng>[<CRD><one>##<PRF><of>##<AT0><the>##<NN2><authors>]</ng>
<ng>[<PNQ><who>]</ng> <vg>[<VBB><are>##<VVN><destined>]</vg>
<infg>[<TO0><to>##<VBB><be>]</infg> <ajg>[<AJ0><immortal>]</ajg>
```

Memory Based Shallow Parser Output:

```
[NP He/PRP NP] [VP is/VBZ VP] [NP one/CD NP] {PNP [Prep of/IN Prep]
[NP the/DT authors/NNS NP] PNP} [NP who/WP NP] [VP are/VBP
destined/VBN to/TO be/VB VP] [ADJP immortal//JJ ADJP] ./.
```

Cognitive Computation Group Shallow Parser Output:

```
[NP He]  [VP is]  [NP one]  [PP of]  [NP the authors]  [NP who] [VP
are destined to be]  [ADJP immortal]  .
```

Only a few simple examples have been included here. It can observed that UCSG Shallow Parse output is generally superior.

5 Conclusions:

In this paper we have described an architecture for partial parsing called the UCSG shallow parsing architecture. We have shown that UCSG successfully combines linguistic constraints expressed in the form of finite state grammars with statistical rating using HMMs built from a POS-tagged corpus and a best first search strategy for global optimization. We have also shown that finite state grammars are good enough to produce word groups. Finite State Grammars are easy to understand and visualize. Recognition with Finite State Grammars is computationally efficient. We have shown that HMMs can be built from POS tagged corpora and can be used for rating and ranking the word groups. We have shown that best parse(chunk sequence) can be selected using best first search strategies. Finally, we have also shown that bootstrapping can improve HMMs and in turn parse generation modules. From the results, we can conclude that it would hopefully be possible to develop wide coverage and robust partial parsing systems without the need for parsed corpora which are not easily available in many cases. The UCSG Shallow Parsing Architecture is also computationally efficient.

Chunkers are usually evaluated just for the percentage of correct chunks they produce, not for the correctness of the complete parse (chunk sequence) for the whole sentence. We have placed greater demands on ourselves and we expect our parser to produce optimal chunk sequence for the whole sentence. No word can be left out and there can be no overlaps either. Further, we produce all (or top few) combinations and that too in hopefully a best first order. Since we are aiming at chunks that correspond to answers to questions that can be asked of the given sentence, the very nature of the chunking task here is more semantic and hence more demanding. More over, we have used a fairly fine grained tag set with more than 70 tags. The data we have started with, namely the BNC POS tagged corpus, has

tagging errors, multiple tags are given in many cases, some words are not tagged, and the tag set had to be extended. Given these factors, the performance we are able to achieve both in terms of percentage of correct chunks in the top parse and rank of the fully correct parse is very encouraging. We are demanding perfect match with manually parsed sentences and in most cases we have observed that the top parse is nearly correct. For example, just could be just a NN1/NP0 error in one of the chunks - not a serious problem for many applications.

The UCSG parser developed for English is a wide coverage shallow parsing system. The system has been built and tested on very large data sets, covering a wide variety of texts, giving us confidence that the system will perform well on new, unseen texts. The system is general and not domain specific, but we can adapt and fine tune for any specific domain as and when needed. We are confident that wide coverage and robust shallow parsing systems can be developed using the UCSG architecture for other languages of the world as well. We plan to continue our work on English parsing while we also start our work on Telugu.

References

[1] Doran, C., Egedi, D., Hockey, B.A., Srinivas, B., Zaidel, M.: XTAG system – a wide coverage grammar for english. In: Proceedings of the 15th. International Conference on Computational Linguistics (COLING 94). Volume II., Kyoto, Japan (1994) 922–928

[2] Li, X., Roth, D.: Exploring evidence for shallow parsing. In: Proceedings of the Annual Conference on Computational Natural Language Learning. (2001)

[3] Abney, S.P.: Parsing by Chunks. Principle-based parsing: Computation and psycholinguistics edn. Kluwer (1991)

[4] Tjong Kim Sang, E.F., Buchholz, S.: Introduction to the conll-2000 shared task: Chunking. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 127–132

[5] Abney, S.: Partial parsing via finite-state cascades. In: Workshop on Robust Parsing, 8th European Summer School in Logic, Language and Information, Prag (1996) 8–15

[6] Grefenstette, G.: Light parsing as finite state filtering. In: Workshop on Extended finite state models of language, Budapest, Hungary (1996)

[7] Roche, E.: Parsing with finite state transducers. Finite–state language processing edn. MIT Press (1997)

[8] Vilain, M., Day, D.: Phrase parsing with rule sequence processors: an application to the shared conll task. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proc. of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 160–162

[9] Dejean, H.: Learning rules and their exceptions. In: Journal of Machine Learning Research, volume 2. (2002) 669–693

[10] Osborne, M.: Shallow parsing as part-of-speech tagging. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 145–147

[11] Veenstra, J., van den Bosch, A.: Single-classifier memory-based phrase chunking. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 157–159

[12] Zhou, G., Su, J., Tey, T.: Hybrid text chunking. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 163–166

[13] Koeling, R.: Chunking with maximum entropy models. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 139–141

[14] Johansson, C.: A context sensitive maximum likelihood approach to chunking. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 136–138

[15] Zhang, T., Damerau, F., Johnson, D.: Text chunking based on a generalization of winnow. In: Journal of Machine Learning Research, volume 2. (2002) 615–637

[16] Sha, F., Pereira, F.: Shallow parsing with conditional random fields. Technical Report CIS TR MS-CIS-02-35, University of Pennsylvania (2003)

[17] Molina, A., Pla, F.: Shallow parsing using specialized hmms. In: Journal of Machine Learning Research, volume 2. (2002) 595–613

[18] Carreras, X., Marquez, L.: Phrase recognition by filtering and ranking with perceptrons. In: Proceedings of the International Conference on Recent Advances in Natural Language Processing, RANLP-2003, Borovets, Bulgaria (2003) 127–132

[19] Gondy, L., Hsinchun, C., Jesse, M.: A shallow parser based on closed-class words to capture relations in biomedical text. In: Journal of Biomedical Informatics 36. (2003) 145–158

[20] Kudoh, T., Matsumoto, Y.: Use of support vector learning for chunk identification. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 142–144

[21] van Halteren, H.: Chunking with wpdv models. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 154–156

[22] Erik F. Tjong Kim Sang: Memory-based shallow parsing. In: Journal of Machine Learning Research, volume 2. (2002) 559–594

[23] Berthold Crysmann et al.: An integrated archictecture for shallow and deep processing systems. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), University of Pennsylvania, Philadelphia (2002)

[24] Kaplan, R.M., III, J.T.M., King, T.H., Crouch, R.: Integrating finite-state technology with deep lfg grammars1. In: Proceedings of the Workshop on Combining Shallow and Deep Processing for NLP(ESSLLI). (2004)

[25] Yong, K.K., Huyck, C.: Robust parsing using plink. In: Proceedings of the 1st International Conference on Vision, Information and Parallel Processing for Industrial Automation (ROVISP),. (2003) 269–275

[26] Gildea, D.: Corpus variation and parser performance. In: Proceedings of Conference on Empirical Methods in Natural Language. (2001)

[27] Murthy, K.N.: Universal Clause Structure Grammar. PhD Thesis, University of Hyderabad (1995)

[28] Kumar, G.B., Murthy, K.N.: Ucsg shallow parser. Proceedings of CICLING 2006, LNCS **3878** (2006) 156–167

[29] Molina, A., Pla, F.: Shallow parsing using specialized hmms. In: Journal of Machine Learning Research, volume 2. (2002) 595–613

[30] Sang, E.F.T.K., Buchholz, S.: Introduction to the CoNLL-2000 shared task: Chunking. In: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 127–132

[31] Megyesi, B.: Shallow parsing with pos taggers and linguistic features. In: Journal of Machine Learning Research, volume 2. (2002) 639–668

[32] Daelemans, W., Zavrel, J., Berck, P., Gillis, S.: Memory-based shallow parsing. In: Proceedings of CoNLL-99, Bergen, Norway (1999)

[33] Memory Based Shallow Parser. (`http://ilk.uvt.nl/cgi-bin/tstchunk/demo.pl`)

[34] Cognitive Computation Group Shallow Parser. (`http://l2r.cs.uiuc.edu/~cogcomp/eoh/index.html`)

[35] Burnard, L.: The users reference guide for the British National Corpus. Oxford University Computing Services, Oxford (2000)

[36] Rose, T., Stevenson, M., Whitehead, M.: The reuters corpus volume 1 - from yesterday's news to tomorrow's language resources. In: Proceedings of the Third International Conference on Language Resources and Evaluation, Las Palmas de Gran Canaria (2002)

[37] Sampson, G.: English For The Computer. Clarendon Press (the scholarly imprint of Oxford University Press) (1995)

[38] Htay, H.H., Kumar, G.B., Murthy, K.N.: Constructing english-myanmar parallel corpora. In: Proceedings of Fourth International Conference on Computer Applications, Yangon,Myanmar (2006) 231–238

[39] Hornby, A.S.: Guide to Patterns and Usage in English. Oxford University Press (1975)

[40] Kumar, G.B.: UCSG Shallow Parser: A Hybrid Architecture for a Wide Coverage Natural Language Parsing System. PhD Thesis, University of Hyderabad (2007)

[41] Nagesh, K.: Towards a robust shallow parser. Masters thesis, Department of Computer and Information Sciences, University of Hyderabad (2004)