An Analogical Parser for Restricted Domains

Donald Hindle

AT&T Bell Labs
600 Mountain Ave.
Murray Hill, NJ 07974

ABSTRACT

This note describes the current development of an approach to parsing designed to overcome some of the problems of existing parsers, particularly with respect to their utility as language models. The parser combines lexical and grammatical constraints into a uniform grammatical representation, is readily trainable (since the parser output is indistinguishable from the grammar input), and uses analogy to guess about the likelihood of constructions outside the grammar.

1. THE PROBLEM WITH PARSERS

A parser is a device that provides a description of the syntactic phrases that make up a sentence. For a speech understanding task such as ATIS, the parser has two roles. First, it should provide a description of the phrases in a sentence so these phrases can be interpreted by a subsequent semantic processor. The second function is to provide a language model – a model of the likelihood of a sentence – to constrain the speech recognition task. It is unfortunately the case that existing parsers developed for text fulfill neither of these roles very well.

It is useful to begin by reviewing some of the reasons for this failure. We can describe the situation in terms of three general problems that parsers face: the Lexicality Problem, the Tail Problem, and the Interpolation Problem.

The Lexicality Problem

The most familiar way to think of a parser is as a device that provides a description of a sentence given some grammar. Consider for example a context free grammar, where nonterminal categories are rewritten as terminals or nonterminals, and terminals are rewritten as words. There typically is no way to express the constraints among individual words.

Yet it is clear that much of our knowledge of language has to do with what words go together.[2] Merely knowing the grammatical rules of the language is not enough to predict which words can go together. So for example, general English grammatical rules admit premodification of a noun by another noun or by an adjective. It is possible to describe broad semantic constraints on such modification; so for example, *early morning* is a case of a time-adjective modifying a time-period, and *morning flight* is a time-period modifying an event. Already we are have an explosion of categories in the grammar, since we are talking not about nouns and adjectives, but about a fairly detailed subclassification of semantic types of nouns and adjectives.

But the problem is worse than this. As Table 1 shows, even this rough characterization of semantic constraints on modification is insufficient, since the adjective-noun combination *early night* does not occur. This dependency of syntactic combinability on particular lexical items is repeated across the grammar and lexicon.

The lexicality problem has two aspects. One is representing the information and the other is acquiring it. There has recently been increasing work on both aspects of the problem. The approach described in this paper is but one of many possible approaches, designed with an emphasis on facilitating efficient parsing.

The Tail Problem

Most combinations of words never occur in a corpus, but many of these combinations are possible, but simply have not been observed yet. For a grammar (lexicalized or not) the problem presented by this tail of rare events is unavoidable. The grammar will always undercover the language. The solution to the tail problem involves training from text.

The Interpolation Problem

While it is always useful to push a grammar out the tail, it is inevitable that a grammar will not cover everything encounted, and that a parser will have to deal with unforeseen constructions. This is of course the typical problem in language modeling, and it raises the problem of estimating the probabilities of structures that have not been seen – the Interpolation Problem. The rules of the grammar must be extendible to new constructions. In this parser the approach is through analogy, or memory-based reasoning.[8]

	_flight(s)	*early_*	N
morning	143	33	597
afternoon	146	25	504
evening	51	12	215
night	27	0	121

Table 1: Pre- and post- modifiers for time nominals in a 266k word ATIS sample.

2. THE PARSER

The basic parser data structure is a pointer to a node, the parser's *focus of attention*. The basic operation is to combine the focus of attention with either the preceding or following element to create a new node, updating preceding and following pointers and updating the focus of attention and then repeat. If no combination is possible, the focus is moved forward, and thus parsing proceeds from the beginning of the sentence to the end.

Some consequences of this model: no non-contiguous dependencies are picked up by this stage of the parser. The idea is that the parser is a reactive component. Its output is not the complete analysis of a sentence, but rather consists of a set of fragments that subsequent processing will glue together. (cf. [1, 3, 4]).

2.1. The Grammar

Trees in the grammar are either terminal or non-terminal. Terminal trees are a pair of a syntactic feature specification and a word. Non-terminals are a pair of trees, with a specification of which tree is head – thus, this is a binary dependency grammar.

$t \rightarrow terminal \mid (1\ t\ t) \mid (2\ t\ t)$

$terminal \rightarrow (features\ word)$

The category of a non-terminal is the category of its head.

The grammar for the parser is expressed as a set of trees that have lexically specified terminals, each with a frequency count. For example, in the ATIS grammar, the tree corresponding to the phrase *book a flight* is

(1 (V "book") (2 (XI "a")(N "flight")))

It occurs 6 times. The grammar consists of a large set of such partial trees, which encode both the grammatical and the lexical constraints of the language.

Following are examples of two trees that might be in the grammar for the parser.

```
(V 1
   (V 1
      (V 0 GIVE)
      (XP11_0 0 ME))
   (N 1
      (N 2
         (XI 0 A)
         (N 0 LIST))
      (P 1
         (P 0 OF)
         (N2 2
            (XQ 0 ALL)
            (N2 0 AIRFARES)))))
(P 1
   (P 0 FOR)
   (N2 2
      (N 0 ROUND-TRIP)
      (N2 0 TICKETS)))
```

2.2. Parsing

The basic parser operation is to combine subtrees by matching existing trees in the grammar. Consider, for example, parsing the fragment *give me a list.*

Initially, the parser focuses on the first word in the sentence, and tries to combine it with preceding and following nodes. Since *give* exists in the grammar as head of a tree with *me* as second element, the match is straightforward, and the node *give me* is built, directly copying the grammar. Nothing in the grammar leads to combining *give me* and *a*, so the parser attention moves forward, and *a list* is built, again, directly from the grammar.

At this point, the parser will is looking at the fragments *give me* (with head *give*) and *a list* (with head *list*), and is faced again with the question: can these pieces be combined. Here the answer is not so obvious.

2.3. Smoothing by analogy.

If we could guarantee that all trees that the parser must construct will exist in its grammar of trees, then the parsing procedure would be as described in the preceding section. Of course, we don't predict in advance all trees the parser might see. Rather, the parser has a grammar representing a subset of the trees it might see along with a measure of similarity between trees. When the parser finds no exact way to combine two nodes to match a tree that exists in the grammar, it looks for similar trees that combine. In particular, it looks at each of the two potential combined nodes in turn and tries to find a similar tree that does combine with the observed tree.

So in our example, although *give me a list* does not occur,

give me occurs with a number of similar trees, including:

```
a list of ground transportation
a list of the cities serve
a list of flights from philadelphia
a list of all the flights
a list of all flights
a list of all aircraft type
```

One of these trees is selected to be the analog of *a list*, thus allowing *give me* to be combined as head with *a list*.

The parser uses a heuristically defined measure of similarity that depends on: category, root, type , specifier, and distribution. Obviously, much depends on the similarity metric used. The aim here is to combine our knowledge of language, to determine what in general contributes to the similarity of words, with patterns trained from the text. The details of the current similarity metric are largely arbitrary, and ways of training it are being investigated.

Notice that this approach finds the closest exemplar, not average of behavior. (cf. [7, 8])

2.4. Disambiguation

For words which are ambiguous among more than one possible terminal (e.g. *to* can be a preposition or an infinitival marker), the parser must assign a terminal tree. In this parser, the disambiguation process is part of the parsing process. That is, when the parser is focusing on the word *to* it selects the tree which best combines *to* with a neighboring node. If that tree has *to* as, for example, head of a prepositional phrase, then *to* is a preposition, and similarly if *to* is an infinitival marker.

Of course, if a word is not attached to any other constituent in the course of parsing, this method will not apply. Disambiguation is still necessary, to allow subsequent processing. In such cases, the parser reverts to its bigram model to make the best guess about the proper tree for a word.

3. DEVELOPING A GRAMMAR

Developing a grammar for this parser means collecting a set of trees. There are 4 distinct sources of grammar trees.

General English. The base set of trees for the parser is a set of general trees for the language as a whole, independent of the domain. These include standard sentence patterns as well as trees for the regular expressions of time, place, quantity, etc. For the current parser, these trees were written by hand (though in this set will over time be developed partly by hand and partly from text). This set of trees is independent of the domain, and available for any application. It forms part of a general model for English.

The remaining three parts of the tree database are all specific to the particular restricted domain.

Domain Database Specific. Trees specific to the subdomain, derived semi-automatically from the underlying database. Included are airline names, flight names and codes, aircraft names, etc. This can also include a set of typical sentences for the domain. In a sense, this set of trees provides information about the content of the messages in the domain. the things one is likely to talk about.

Parsed Training Sentences. hand parsed text from the training sentences. These trees are fairly easy to produce through an incremental process of: a) parse a set of sentences, b) hand correct them, c) remake the parser, and d) repeat. About a thousand words an hour can be analyzed this way. (Thus for the ATIS task, it is easy to hand parse the entire training set, though this was not done for the experiment reported here.)

Unsupervised Parsed Text. also from the training sentences, but parsed by the existing parser and left uncorrected. (Note: given an existing database of parsed sentences, these could transformed into trees for the parser grammar.)

Obviously, one aim of this design is to make acquisition of the grammar easy. Indeed, the parser design is not English-specific, and in fact a Spanish version of the parser (under an earlier but related design) is currently being updated.

4. THE ATIS EXPERIMENT

For The ATIS task, a vocabulary was defined consisting of 1842 distinct terminal symbols (a superset of the February 91 vocabulary, enhanced by adding words to regularize the grammar, and by distinguishing words with features; e.g. "travel" as a verb is a different terminal from "travel" as a noun). A grammar was derived, based on 1) a relatively small general English model including trees for general sentence structure as well as trees for dates, times, numbers, money, and cities, and 2) an ATIS specific set of trees covering types of objects in the database (aircraft, airports, airlines, flight info, ground transportation) and 3) sentences in the training set. In this experiment, approximately 10% of the grammar are language general, 10% are database specific, 50% are supervised parsed trees and 30% are unsupervised.

The weighting of the various sources of grammar trees has not arisen here – all trees are weighted equally. But in the general case, where there is a pre-existing large general grammar, and a large corpus for unsupervised training, the weighting of grammar trees will become an issue.

Given this grammar consisting of 14,000 trees, derived as described above, the grammar perplexity is 15.9 on the 138 February 91 test sentences. This compares to a perplexity of 18.9 for the bigram model (where bigrams are terminals). The grammar trees derived from the unsupervised parsing of the training sentences improve the model slightly (from 16.4 to 15.9 perplexity).

5. SENTENCE PROBABILITY

The parse of a sentence consists of a sequence of N nodes. By convention, the first and last nodes in the sequence (n_1 and n_N) are instances of the distinguished *sentence boundary* node. If all the words in a sentence are incorporated by the parser under a single root node, then the output will consist of a sequence of three nodes, of which the middle one covers the words of the sentence. But remember, the parser may emit a sequence of fragments; in the limiting case, the parser will emit one node for each word.

5.1. The tree grammar

The tree grammar, consists of a set of tree specifications.

For each tree t_i, the specification records:

the shape of t_i –

- for terminals – the root and category
- for non-terminals – whether the head is on the left or right what the left and right subtrees are.

count(t_i) – number of times that t_i appears

left_count(t_i) – number of times t_i appears on the left in a larger tree

right_count(t_i) –number of times t_i appears on the right in a larger tree

lsubs_for(t_i, t_j) – for tree t_j in which t_i is the left subtree, sum of count(t_k) where t_k could realize t_i in t_j

rsubs_for(t_i,t_j) – for tree t_j in which t_i is the right subtree, sum of count(t_k) where t_k could realize t_i in t_j

lsubs(t_i) – sum of count of trees t_j such that t_i could realize the left subtree of t_j

5.2. probability calculation

In the following, rd, ld, rc, and lc mean right daughter, left daughter, right corner and left corner respectively. The probability of a sentence s consisting of a sequence of n nodes (starting with the sentence boundary node, which we call n_1) is:

$$\begin{aligned} Pr(s) &= \prod_{i=1}^{N-1} Pr(bigram(rc(n_i), lc(n_{i+1}))) \\ &\quad * Pr(not\_attached(n_i)) \\ &\quad * Pr(n_{i+1} \mid lc(n_{i+1})) \end{aligned}$$

In this formula, the bigram probabilities are calculated on the terminals (word plus grammatical features), interpolating using feature similarity.

$Pr(not\_attached(n_i))$ means the probability that n_i is not attached as the *ld* of any node. It is estimated from $count(n)$ and $left\_count(n)$.

$Pr(n_{i+1} \mid lc(n_{i+1}))$, the probability of a node given that we have seen its left corner, is derived recursively:

$Pr(n \mid lc(n)) = 1.0$, if n is a terminal node, since the *lc* of a terminal node is the node itself; otherwise,

$$\begin{aligned} Pr(n \mid lc(n)) &= Pr(ld(n) \mid lc(ld(n))) \\ &\quad * 1.0 - Pr(not\_attached(ld(n))) \\ &\quad * Pr(tree(n) \mid ld(n)) \\ &\quad * Pr(rd(n) \mid tree(n), ld(n)) \end{aligned}$$

In this formula, the first term is the recursion, which descends the left edge of the node to the left corner.

At each step in the descent, the second term in the formula takes account of the probability that the left daughter will be attached to something.

The third term is the probability that the tree $tree(n)$ will be the parent given that node $lc(n)$ is the left daughter of a node.

The fourth term is the probability that node $rd(n)$ will be the right daughter given that $ld(n)$ is the left daughter and $tree(n)$ is the parent tree corresponding to node n.

probability of $tree(n)$ given $ld(n)$ To find the $Pr(tree(n) \mid ld(n))$, we consider the two cases, depend-

ing on whether there is a substitution for the left_tree of n:

Case: no left_substitution. If the $left\_tree(tree(n))$ is equal to the $tree(ld(n))$ (i.e. if there is no substitution), then

$$Pr(tree(n) \mid ld(n)) =$$
$$(1.0 - prob\_left\_substitution(ld(n)))$$
$$Pr(tree(n) \mid ld(n), no\_left\_substitution)$$

The $prob\_left\_substitution(ld(n))$ is the probability that given the node $ld(n)$ whose tree is t_l, that node will be the left daughter in a node whose left_tree is is not the same as t_l. That is, t_l will realize the $left\_tree(n)$. We estimate this probability on the basis of the $count(t_l)$ and the $left\_count(t_l)$.

When there is no left_substitution, the probability of the parent tree is estimated directly from the counts of the trees that $tree(ld(n))$ can be left_tree of:

$$Pr(tree(n) \mid ld(n), no\_left\_substitution) =$$
$$count(tree(n))/left\_count(tree(ld(n)))$$

Case: left_substitution. If there is a substitution, then

$$Pr(tree(n) \mid ld(n)) =$$
$$prob\_left\_substitution(ld(n))$$
$$Pr(tree(n) \mid tree(ld(n)), left\_substitution)$$

To estimate the $Pr(tree(n) \mid tree(ld(n)))$ in case 2 (where we know there is a substitution for the $left\_ptree(n)$, we reason as follows. For each tree tx_{left} that might substitute for $tree(ld(n))$, it will substitute only if tx_{left} is observed as a left member of a tree that $tree(left_daughter(n))$ is not observed with, and for tx_{right}, tx_{left} is the best substitution. The total of such trees is called $lsubs(t)$.

By this account,

$$Pr(tree(n) \mid tree(ld(n)), left\_substitution) =$$
$$count(tree(n))/lsubs(tree(ld(n))).$$

The probability of the right daughter, given the left daughter and the tree similarly takes into account the probabilities of substitution.

6. FURTHER WORK

While current results for this parsing model look promising, there are several directions of further exploration.

Integration in Speech Recognition. There are two obvious ways of incorporating this parser into the speech recognition task. First, it can be used to select among a set of candidate sentences proposed by a recognizer. The second, more interesting, approach is to embed the parser in the recognition process. Given the parser's localization of information and its deterministic beginning-to-end processing, it can naturally be used to find a locally (where the domain of locality is adjacent trees) optimal path through an (appropriately sparse) lattice.

Development of Further Processing. This parser rests on the assumption, shared in a variety of recent work from quite different perspectives [1, 3, 4], that a level of underspecified syntactic description is efficiently obtainable and is useful. The current work supports a particular view of what partial syntactic descriptions are obtainable. It remains to show that the further processing components can be constructed to make these pieces useful.

Implementation Details. A number of decisions in the implementation of the current parser are arbitrary, and further development demands exploring the optimal design. For example, we need to explore what the similarity function should look like, and what function should be used for comparing potential attachments.

7. REFERENCES

1. Abney, Steven P. Rapid incremental parsing with repair. Paper presented at Waterloo conference on Electronic Text Research.
2. Church, Kenneth W., William A. Gale, Patrick Hanks, and Donald Hindle. (to appear). Using statistics in lexical analysis. in Zernik (ed.) *Lexical acquisition: using on-line resources to build a lexicon.*
3. Jacobs, Paul. 1990. To parse or not to parse: relation-driven text skimming. In COLING 90, 194-198, Helsinki, Finland.
4. Marcus, Mitchell P. and Donald Hindle. 1990. Description Theory and Intonation Boundaries. In Gerald Altmann (ed.), Computational and Cognitive Models of Speech. MIT Press.
5. Sadler, Victor. 1989. *Working with analogical semantics.* Foris: Dordrecht.
6. Parsing strategies with 'lexicalized' grammars: application to tree adjoining grammars. In Proceedings fo the 12th International Conference on Computational Linguistics, COLING88, Budapest, Hungary.
7. Skousen, Royal. 1989. *Analogical modeling of language.* Kluwer:Dordrecht.
8. Stanfill, Craig and David Waltz. 1986. Toward memory-based reasoning. Communications of the ACM 29.12.