Part-of-Speech Tagging and Partial Parsing

Steven Abney

1996

The initial impetus for the current popularity of statistical methods in computational linguistics was provided in large part by the papers on part-of-speech tagging by Church [20], DeRose [25], and Garside [34]. In contradiction to common wisdom, these taggers showed that it was indeed possible to carve part-of-speech disambiguation out of the apparently monolithic problem of natural language understanding, and solve it with impressive accuracy.

The concensus at the time was that part-of-speech disambiguation could only be done as part of a global analysis, including syntactic analysis, discourse analysis, and even world knowledge. For instance, to correctly disambiguate *help* in *give John help*$_{\mathrm{N}}$ versus *let John help*$_{\mathrm{V}}$, one apparently needs to parse the sentences, making reference to the differing subcategorization frames of *give* and *let*. Similar examples show that even world knowledge must be taken into account. For instance, *off* is a preposition in *I turned off highway I-90*, but a particle in *I turned off my radio*, so assigning the correct part of speech in *I turned off the spectroroute* depends on knowing whether *spectroroute* is the name of a road or the name of a device.

Such examples *do* demonstrate that the problem of part-of-speech disambiguation cannot be solved without solving all the rest of the natural-language understanding problem. But Church, DeRose and Garside showed that, even if an exact solution is far beyond reach, a reasonable *approximate* solution is quite feasible.

In this chapter, I would like to survey further developments in part-of-speech disambiguation ('tagging'). I would also like to consider a question raised by the success of tagging, namely, what piece of the NL-understanding problem we can carve off next. 'Partial parsing' is a cover term for a range of different techniques for recovering some but not all of the information contained in a traditional syntactic analysis. Partial parsing techniques, like tagging techniques, aim for reliability and robustness in the face of the vagaries of natural text, by sacrificing completeness of analysis and accepting a low but non-zero error rate.

1 Tagging

The earliest taggers [35, 51] had large sets of hand-constructed rules for assigning tags on the basis of words' character patterns and on the basis of the tags assigned to preceding or following words, but they had only small lexica, primarily for exceptions to the rules. TAGGIT [35] was used to generate an initial tagging of the Brown corpus, which was then hand-edited. (Thus it provided the data that has since been used to train other taggers [20].) The tagger described by Garside [56, 34], CLAWS, was a probabilistic version of TAGGIT, and the DeRose tagger improved on CLAWS by employing dynamic programming.

In another line of development, hidden Markov models (HMMs) were imported from speech recognition and applied to tagging, by Bahl and Mercer [9], Derouault and Merialdo [26], and Church [20]. These taggers have come to be standard. Nonetheless, the rule-based line of taggers has continued to be pursued, most notably by Karlsson, Voutilainen, and colleagues [49, 50, 85, 84, 18] and Brill [15, 16]. There have also been efforts at learning parts of speech from word distributions, with application to tagging [76, 77].

Taggers are currently wide-spread and readily available. Those available for free include an HMM tagger implemented at Xerox [23], the Brill tagger, and the MULTEXT tagger [8].[1] Moreover, taggers have now been developed for a number of different languages. Taggers have been described for Basque [6], Dutch [24], French [18], German [30, 75], Greek [24], Italian [24], Spanish [57], Swedish [13], and Turkish [63], to name a few. Dermatas and Kokkinakis [24] compare taggers for seven different languages. The MULTEXT project [8] is currently developing models to drive their tagger for six languages.

1.1 HMM Taggers

The standard tagger is a hidden Markov model whose states are tags or tuples of tags. HMMs are discussed in considerable detail elsewhere in this book (chap. 2) and in a number of tutorial papers [67, 66, 64], so I will assume familiarity with them here.

For a bigram tagger, the states of the HMM are tags. Transition probabilities are probabilities of a tag given the previous tag, and emission probabilities are probabilities of a word given a tag. The probability of a particular part-of-speech sequence plus sentence is the product of the transition and emission probabilities it contains. For example,

[1] As of this writing, the addresses are ⟨ftp://parcftp.xerox.com/pub/tagger⟩ for the Xerox tagger, ⟨http://www.cs.jhu.edu/∼brill⟩ for the Brill tagger, and ⟨http://issco-www.unige.ch/projects/MULTEXT.html⟩ for the MULTEXT tagger.

$$
\begin{array}{ll}
 & P\left(\begin{array}{cccccccc} \text{DT} & & \text{NN} & & \text{MD} & & \text{VB} \\ | & & | & & | & & | \\ \text{the} & & \text{garbage} & & \text{can} & & \text{smell} \end{array}\right) \\
\textbf{(1)} & \\
= & P(\text{DT})\times \\
 & P(\text{NN}|\text{DT})P(\text{MD}|\text{NN})P(\text{VB}|\text{MD})\times \\
 & P(\textit{the}|\text{DT})P(\textit{garbage}|\text{NN})P(\textit{can}|\text{MD})P(\textit{smell}|\text{VB})
\end{array}
$$

For a trigram model, states are pairs of tags, and we have for example:

$$
\begin{array}{ll}
 & P\left(\begin{array}{cccccccc} \text{\$,DT} & & \text{DT,NN} & & \text{NN,MD} & & \text{MD,VB} \\ | & & | & & | & & | \\ \text{the} & & \text{garbage} & & \text{can} & & \text{smell} \end{array}\right) \\
\textbf{(2)} & \\
= & P(\text{DT})\times \\
 & P(\text{NN}|\text{\$,DT})P(\text{MD}|\text{DT,NN})P(\text{VB}|\text{NN,MD})\times \\
 & P(\textit{the}|\text{DT})P(\textit{garbage}|\text{NN})P(\textit{can}|\text{MD})P(\textit{smell}|\text{VB})
\end{array}
$$

It should be observed that the expansion of the probability in (2) is correct only under a couple of assumptions. First, we assume that emission probabilities are conditional only on the second tag in the tag-pair representing a given state. This justifies writing e.g. '$P(\textit{can}|\text{MD})$' in place of '$P(\textit{can}|\langle\text{NN,MD}\rangle)$'. Second, we assume that the only transitions with nonzero probability out of a state $\langle\alpha,\beta\rangle$ are to states $\langle\beta,\gamma\rangle$ for some γ. This justifies writing e.g. '$P(\text{MD}|\text{DT,NN})$' in place of '$P(\langle\text{NN,MD}\rangle|\langle\text{DT,NN}\rangle)$'.

With this correspondence between tags and states, the most-likely sequence of tags can be recovered straightforwardly using the Viterbi algorithm, and the transition and emission probabilities can be estimated using the forward-backward algorithm. The error rates reported in the literature range from about 1% to 5%.

Two of the strongest selling points for HMM taggers are their accuracy and the fact that they can be trained from unannotated text. These are indeed important advantages of HMM taggers, but at the same time, there are some points to keep in mind.

If we train an HMM tagger with no hand-coded input at all, it will indeed succeed at finding a model whose cross-entropy with the corpus is low. However, the output may have little relation to the part-of-speech assignments we actually want as output. Getting good performance—as measured by assignment of the intended tags, not cross-entropy—may require a fair amount of manually prepared material. Merialdo [62] and Elworthy [29] conduct experiments to evaluate the effectiveness of forward-backward training, and conclude that the best performance is obtained by providing large amounts of pre-tagged text, and that with large amounts of pre-tagged text, forward-backward training can in fact damage performance rather than improving it.

As concerns accuracy figures—for taggers generally, not just for HMM taggers—it is good to remember the maxim, "there are lies, damned lies, and statistics."

First, tagger error rates are usually reported as percentage of words erroneously tagged. In many applications, the sentence is the more relevant unit, and a single tagging error may lead to failure of the application. Assuming 20-word sentences and independence of errors, a 4% per-word error rate translates into a $1 \quad .96^{20} = 56\%$ per-sentence error rate! Or, working backwards, to achieve a 4% per-sentence error rate, we require a per-word error rate of 0.2%.

Second, 4% error sounds good because it is so much bettern than 100% error. But, in fact, even guessing will do better than 100% error. With a very simple approach—just tagging each word with its most-frequent tag, regardless of context—one already reduces error to 10% [22].

A more general, related principle is a law of diminishing returns related to Zipf's law. A little effort goes a long way, at first: eliminating a few high-frequency error types has a big effect on per-token error rates. But the flip side is that the amount of work needed to make further progress increases exponentially.

Finally, as is true for any evaluation, a fair comparison of techniques is only possible if they are applied to the same task. In this respect, virtually none of the reported tagger error rates are comparable. Differences in tagsets, evaluation texts, and amount of training material can have significant effects on the error rate. To give a single example, some taggers do not distinguish gerunds from present participles, yet that distinction is a significant source of errors for other taggers.

1.2 Rule-Based Taggers

An alternative to the standard model is represented by rule-based taggers. Voutilainen [85, 50] describes a Constraint Grammar (CG) tagger that has similarities to TAGGIT. Sets of tags are assigned to words on the basis of a lexicon and morphological analysis, and tags are then eliminated on the basis of contextual (pattern-action) rules: for example, 'the current word is not a verb if the preceding word is a determiner'. Performance is reported to be as good as or better than that of stochastic taggers.

A criticism of rule-based taggers is the amount of effort necessary to write the disambiguation rules. However, as mentioned above, getting good performance from an HMM tagger also requires a respectable amount of manual work. Chanod and Tapanainen [18] conducted an informal experiment in which they took one month to develop a stochastic tagger for French and the same time to develop a rule-based tagger, using no annotated training material. For the rule-based tagger, the time was spent developing a rule-set, and for the stochastic tagger, the time was spent developing restrictions on transitions and emissions ('biases') to improve tagger performance. At the end of the month, the rule-based tagger had better performance: 1.9% error versus 4.1% for the stochastic tagger, averaging over two test-sets. Without more objective measures of "amount of effort", this can only be taken as an anecdote, but it is suggestive

nonetheless.

Brill [15] has developed a technique for mechanically acquiring rules for a rule-based tagger from manually tagged text. Initially, known words are tagged with their most-frequent tag, and unknown words are arbitrarily tagged 'noun'. By comparing the current tagging with a hand-tagged training text, errors are identified and candidate error-correction rules are considered. The score of a rule candidate is the net improvement it effects: the number of times it changes an erroneous tag to a correct one minus the number of times it changes a correct tag to an erroneous one. The best rule is selected and applied to the current output, and the process repeats. Two types of rules are learned: lexical rules, for assigning an initial tag to unknown words, and context rules, for correcting tags on the basis of context. All rules are of the form $\text{tag}_i \rightarrow \text{tag}_j$ if P. For lexical rules, P includes predicates like 'the word has suffix -*xyz*' or 'the word ever appears after *foo* in the training corpus'. For context rules, P includes predicates like 'the preceding tag is X' or 'the following two tags are YZ'. Training yields a sequence of rules. Tagging consists in assigning initial tags (as in training), then applying the rules in series.

According to Brill's evaluation, the taggers' error rate is indistinguishable from that of stochastic taggers, and its error rate on words not seen in the training corpus is markedly lower. An advantage over stochastic taggers is that significantly less storage is needed for 100-odd pattern-action rules than for an HMM tagger's probability matrix. Compactness is an advantage of rule-based taggers generally.

Another general advantage is speed. Unlike stochastic taggers, most rule-based taggers are deterministic. In fact, recent work in both the CG paradigm and in the Brill paradigm has been converging on the compilation of pattern-action rules into finite-state transducers, inspired in large part by the success of similar approaches to morphological analysis [48, 52]. CG rules can be re-expressed as regular expressions describing the contexts in which particular tags may legally appear. Translating the regular expressions into finite-state transducers and combining them (by intersection) yields a single transducer representing the simultaneous application of all rules [18, 86]. Roche and Schabes have also shown how the rules of Brill's tagger can be translated to finite-state transducers and combined (by composition) to yield a single transducer. The resulting transducer is larger than Brill's original tagger, but still significantly smaller than an equivalent stochastic tagger (379 KB vs. 2158 KB). It is also the fastest tagger I have seen reported in the literature (10,800 wps vs. 1200 wps for an HMM tagger).

1.3 Generative Processes vs. Classification/Regression

The Brill rule-acquisition technique can be seen as a kind of regression or classification model [68], related to classification and regression trees (CART) [14] and decision lists [70, 89]. Regression techniques can be contrasted, at least heuris-

tically, with generative-process models like HMM's. In both cases the goal is to assign a structure to an observed sentence. In a generative-process model, sentences are viewed as the output of a generative process, and tag sequences—more generally, syntactic structures—are identified with the sequence of steps by which the sentence was generated. The most-likely structure is the one associated with the sequence of steps by which the sentence was most likely generated.

In a regression or classification model, by contrast, the problem is couched in terms of a stochastic relationship between a *dependent variable* (the classification) and one or more *predictor variables* (properties of the objects that we wish to classify). In our setting, predictor variables are observable properties of sentences, and the dependent variable ranges over structures or pieces of structure. The most-likely structure is the one most likely to be the value of the dependent variable given the settings of the predictor variables.

With both sorts of models, we aim to maximize $P(S|W)$, for S a structure and W a sentence. A classification model estimates the function from W to the probability of S directly, whereas a generative-process model estimates it indirectly, by specifying $P(S)$ and $P(W|S)$, from which $P(S|W)$ can be computed using Bayes' Law.

Because the conditionalization is "backwards" in generative-process models ($P(W|S)$ instead of the desired $P(S|W)$), classification models are sometimes more intuitive. For example, a common error in describing HMM taggers is to combine lexical with contextual probabilities as the product

(3) $P(\text{tag}|\text{context})P(\text{tag}|\text{word})$

instead of the correct form $P(\text{tag}|\text{context})P(\text{word}|\text{tag})$. Intuitively, in a stochastic finite-state or context-free process, structure-building choices are conditionalized only on structure that has already been built, and though choices may be jointly conditionalized on multiple pieces of existing structure, they may not be separately conditionalized on them. We may define a generative process that uses the conditionalization $P(\text{tag}|\text{context}, \text{word})$ but that probability cannot be computed as the product (3). To illustrate, let o denote the event that a die throw comes out odd, and h denote the event that a die throw comes out high, i.e., 4, 5, or 6. Then $P(5|o) = 1/3$, $P(5|h) = 1/3$, but $P(5|o, h) = 1$ whereas $P(5|o)P(5|h) = 1/9$.

By contrast, classification models permit one to combine multiple information sources. We can define a model in which context (C) and word (W) are predictor variables and tag (T) is the dependent variable, with $T = f(C, W)$. A simple example is the linear interpolation model, $P(t) = \lambda P(t|c)+(1 \quad \lambda)P(t|w)$. The model parameter λ can be estimated from an annotated training text or via the forward-backward algorithm [45]. Clustering, decision trees, and decision lists are other general classification methods that have been applied to problems in computational linguistics [59, 89], including part-of-speech tagging [11].

A disadvantage of classification models is that they typically involve supervised training—i.e., an annotated training corpus. On the other hand, as we have seen, HMM models often require as much manually-prepared material as classification models do, if they are to perform well.

Despite the differences, it should not be supposed that generative-process and classification models are somehow in opposition. Indeed, linear interpolation can be viewed either as an HMM or as a regression [46, 45], and techniques of both types are often interspersed in a single model, as for instance when clustering is used to smooth the parameters of an HMM [17], or when forward-backward training is used to smooth decision trees [10].

As concerns rule-based and HMM taggers specifically, the differences highlighted by the contrast between classification techniques and generative-process techniques should be counterbalanced by the similarities that are brought to the fore when one re-expresses rule-based taggers as finite-state transducers. Namely, HMM's can also be viewed as stochastic finite-state transducers, as discussed by Pereira et al. [65]. This line of inquiry promises to give us a model of tagging (and partial parsing, as we shall see) of great generality, and is an area that will likely receive increasing attention.

2 Partial Parsing

Let us turn now to parsing. Traditional parsers—including standard stochastic parsers—aim to recover complete, exact parses. They make a closed-world assumption, to wit, that the grammar they have is complete, and search through the entire space of parses defined by that grammar, seeking the globally best parse. As a result, and notwithstanding 'clean-up' strategies that are sometimes applied to salvage failed parses, they do not do well at identifying good phrases in noisy surroundings.

Unrestricted text is noisy, both because of errors and because of the unavoidable incompleteness of lexicon and grammar. It is also difficult to do a global search efficiently with unrestricted text, because of the length of sentences and the ambiguity of grammars. Partial parsing is a response to these difficulties. Partial parsing techniques aim to recover syntactic information efficiently and reliably from unrestricted text, by sacrificing completeness and depth of analysis.

2.1 An Example

Many partial parsers aim only to recover the nonrecursive cores of noun phrases. A natural generalization is to recognize the nonrecursive kernels of all 'major' phrases, regardless of category ('chunks'), and to recognize simplex (i.e., nonrecursive) clauses. Here is an example of the structures to be recovered:

(4) [S

[NP The resulting formations]
[VP are found]
[PP along [NP an escarpment]]
][RC
[WhNP that]
[VP is known]
[PP as [NP the Fischer anomaly]]
]

The idea is to factor the parse into those pieces of structure that can be reliably recovered with a small amount of syntactic information, as opposed to those pieces of structure that require much larger quantities of information, such as lexical association information. Chunks and simplex clauses can be recovered quite reliably with a small regular-expression grammar. Resolving attachments generally requires information about lexical association between heads, hence it is postponed. Indeed, recovering chunks and clauses is useful for bootstrapping lexical association information. By reducing the sentence to chunks, there are fewer units whose associations must be considered, and we can have more confidence that the pairs being considered actually stand in the syntactic relation of interest, rather than being random pairs of words that happen to appear near each other. Recognizing simplex clauses serves to constrain the search space, on the assumption that attachment out of the local clause is rare enough to be negligible.

The resulting structure is not a standard syntax tree, nor are chunks and clauses necessarily even consistent with a standard tree. For example, in (4), if restrictive relatives are adjoined to $\overline{\text{N}}$, then the $\overline{\text{N}}$ *escarpment that . . . anomaly* constitutes a phrase in the standard tree that is incompatible with several of the phrases in (4), including the noun chunk *an escarpment*, the PP containing it, and the first simplex clause as a whole.

On the other hand, (4) is a sub*graph* of the standard tree, and the standard tree can be recovered via *attachment;* that is, by adding arcs to the graph (4). To be precise, we must also insert additional nodes (such as the aforementioned $\overline{\text{N}}$), but the important point is that (4) does constitute a useful intermediate representation—it is not necessary to throw it away and start over from scratch in order to recover traditional trees.

The attachment operation is not widely used in computational-linguistic parsing algorithms, the most notable exceptions being the Marcus parser [61] and Don Hindle's industrial-strength version thereof, Fidditch (see below). By contrast, attachment is widely assumed as a basic parsing action in the psycholinguistic literature. Indeed, though we have to this point considered chunks and attachment only as a pragmatic response to the exigencies of unrestricted text, there are in fact reasons to think that chunks and simplex clauses play a role in human language processing [3, 2, 4]. And, incidentally, as a nonrecursive version of phrase structure, chunks have proven useful in neural net models of

parsing [44].

2.2 Some Simple Techniques

Probably the simplest chunk-recognizer is simply to take everything delimited by function words (or stop words) as a chunk. This technique was used in a completely different context by Ross and Tukey [73]. They called stretches of stop words "chinks", and stretches of non-stop-words "chunks". A similar approach was used in earlier versions of the Bell Labs speech synthesizer (Mark Liberman, personal communication).

Bourigault [12] uses this technique for identifying noun phrases in French. Chinks are any words that can't belong to a (common) noun phrase, such as verbs, pronouns, conjunctions, prepositions, and determiners, with a few listed exceptions including *de*, *de la*, and *a*. Chunks are stretches of text between chinks. For example:

> un [traitement de texte] est installé sur le [disque dur de la station de travail]

A large set of specific part-of-speech patterns were then used to extract probable technical terms out of chunks.

A simple stochastic technique is that used by Church [20]. He constructed a noun-chunk recognizer that takes the output of an HMM tagger as input. It marks noun chunks by inserting open and close brackets between pairs of tags. For example:

(5)

	[		]		[	]	
$	DT	NN	VBD	IN	NN		CS
	the	prosecuter	said	in	closing		that

Four bracket combinations are possible between each pair of tags: ⊏, ⊐, ⊐⊏, and no brackets. We assume that there are no empty phrases, hence no need for ⊏⊐, and no nesting, hence no need for ⊏⊏, ⊐⊐, ⊐⊏⊏, etc. However, to make sure that brackets are properly paired, we must keep track of whether we are inside or outside of a noun chunk. Accordingly, we split the no-bracket condition into two states: no-brackets inside a chunk (I) versus no-brackets outside a chunk (O), yielding five states: ⊏, ⊐, ⊐⊏, I, and O. The probabilities of illegal transitions are fixed at zero, illegal transitions being ⊐ ⊐, ⊏ ⊏, ⊐⊏ ⊏, ⊐ I, etc.

The emission from a given state is a pair of tags. For example, sentence (5) is represented more accurately as:

(6)

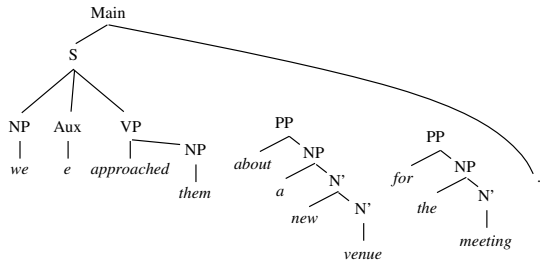
[	—	I	—	]	—	O	—	[	—	]
$,DT		DT,NN		NN,VB		VB,IN		IN,NN		NN,CS

We do not constrain the model to generate only well-formed sequences of tag-pairs, i.e., sequences in which, if $\langle \alpha, \beta \rangle$ immediately precedes $\langle \gamma, \delta \rangle$, then $\beta = \gamma$. Indeed, there is no combination of model parameters that we can fix in advance to guarantee well-formed tag pairs. This lack of constraint is harmless, however, since in training and recognition the tag pairs are given as input. We are not using the model to generate tag-pairs, but to estimate the bracket sequence.

The technique of encoding chunks as 'bracket tags' is also used by Ramshaw and Marcus [69]. Instead of HMM training, however, they apply Brill's rule-learning methods.

2.3 Fidditch

An older, and by most measures still the most successful, partial parser is Hindle's parser Fidditch [39, 41]. Fidditch was not intended as a partial parser *per se*. But it was specifically designed for use on unrestricted text, including such noisy text as transcripts of spoken language. It is based on the Marcus parser, but simplifies the rule formalism, to make it easier to write a very large grammar, and introduces a new action, 'punt'.[2] A phrase whose role cannot be determined is removed from the input, but left unattached, and the parse proceeds as if it were not there. This achieves a containment of ambiguities of much the same sort as that provided by recognition of simplex clauses. The parser recognizes the key elements of a clause—the clause boundary markers, the subject and predicate—and these attached elements surround punted phrases, preventing the degree of ambiguity from exploding. The following exemplifies a typical Fidditch tree:



A property of Fidditch that makes it useful for parsing corpora is its speed. Because it is deterministic, and more subtly, because its use of attachment as a basic action keeps the stack from growing without bound, it can be implemented as a "nearly finite state" automaton. It is one of the fastest parsers I am aware of, achieving speeds of 5600 words per second on an SGI (1200 wps on a Sparc 1). There are only a few parsers to my knowledge with speeds of the same order of magnitude: Cass2—8900 wps, UltraSparc; 1300 wps, Sparc1 [5]; Vilain &

[2]This comes from an American football term, meaning to abandon the current attempt to score, and kick the ball away. There is no relation to the British term refering to boats propelled by poles.

Palmer's implementation of the Brill parser—ave. 7400 wps, Sparc10 [personal communication]; Copsy—ca. 2700 wps, Siemens BS2000 [78]; ENGCG—1000+ wps, Sparc 10 [83]. Given the differences in hardware, it is difficult to rank these parsers, but they clearly outstrip the next fastest parsers that have been reported in the literature, whose speeds are in the 10–60 wps range. By contrast, speeds for traditional chart parsers are often well under 1 wps. Without controlled comparisons, reported speeds must be taken with a grain of salt; nonetheless, I think it is significant that the fastest parsers are all deterministic, rule-based partial parsers.

2.4 Brill, CG, Copsy, and Supertags

The transformation-based learning and constraint grammar techniques discussed earlier for tagging have also been applied to parsing. Brill [15] proposes starting with a uniformly right-branching parse and learning rules for rotating local trees in order to improve the fit to a training corpus. Learning can be time-consuming, but once the rules have been learned, parsing is very fast. Vilain & Palmer [82] explore techniques for improving learning speeds, and mention a fast parser implementation.[3]

Voutilainen [50] describes a partial parser, ENGCG, that is very similar in operation to the constraint-grammar tagger. Lexical and morphological analysis assigns a set of possible syntactic function tags to each word, in addition to part of speech. The syntactic function of each word is disambiguated in the same way that part of speech is disambiguated, via the application of pattern-matching rules to eliminate incorrect tags. Successful disambiguation provides skeletal syntactic information. The syntactic analysis is a dependency analysis, in the sense that only word-word relations are considered. Words are not explicitly associated with their governors, but the syntactic-function annotations significantly constrain the set of compatible analyses, and can be seen as representing an ambiguity class of analyses.

Copsy [78] is a dependency parser for noun phrases, designed to identify and normalize multi-word terms for information retrieval. Parsing is carried out deterministically using pattern-action rules to identify dependencies. To preserve the speed and accuracy of parsing, rules are required to be relevant, highly accurate, and cheap to apply. The parser uses only 45 rules, though over 200 candidates were considered in the course of parser development.

Joshi and Srinivas [47] describe a parser that, like the Voutilainen work, uses tagging techniques to parse. Their partial parser developed from work on lexicalized tree-adjoining grammar (LTAG), in which each elementary tree contains a unique lexical item. Substitution and adjunction in LTAG is equivalent to the attachment operation, or the insertion of an arc in a dependency graph. A word can appear in multiple elementary trees, each representing a different syntactic

[3]The cited paper reports 13,000 wps, but that does not include file I/O times; file I/O reduces speeds to 6800–7900 wps.

structure it might appear in, and a different *valency*, that is, a different set of dependents. Partial parsing consists in selecting a single elementary tree for each word, so as to permit a globally consistent dependency graph. The search for a consistent assignment of elementary trees is accomplished by viewing elementary trees as "supertags" (analogous to the syntactic-function tags of Voutilainen), and employing an adaptation of Viterbi search, as in part-of-speech tagging. As with Voutilainen, partial parsing in this sense does not produce an explicit structure, but can be seen as reducing the size of the ambiguity class of parse-trees for the sentence.

2.5 Finite-State Cascades

The idea of using cascaded finite-state machines was pursued by Ejerhed and Church [28, 27] and myself [1, 3, 5], and in a somewhat different paradigm, by Koskenniemi [53, 54].[4] Generalizing a bit from the cited papers, a finite-state cascade consists of a sequence of strata, each stratum being defined by a set of regular-expression patterns for recognizing phrases. Here is a concrete example:

(7)

1:	NP	→	D? A* N+ | Pron
	VP	→	Md Vb | Vz | Hz Vbn | Bz Vbn | Bz Vbg
2:	PP	→	P NP
3:	SV	→	NP VP
4:	S	→	(Adv|PP)? SV NP? (Adv|PP)*

The strata are numbered. The output of stratum 0 consists of parts of speech. The patterns at level l are applied to the output of level l 1 in the manner of a lexical analyzer. Multiple patterns may match, and a given pattern may match different-length prefixes of the input. The longest match is selected (ties being resolved in favor of the first pattern listed), the matched input symbols are consumed from the input, the category of the matched pattern is produced as output, and the cycle repeats. If no pattern matches, an input symbol is punted—that is, removed from the input and passed on as output.

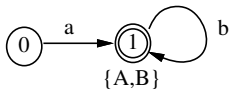
The grammar is designed such that rules, when applied using 'longest match' for disambiguation, are very reliable. There is certain linguistic information we wish to recover in the end—to a first approximation, a traditional syntax tree—and we wish it to be not too difficult to extract that information from the trees we build here, but there is no reason to insist that every phrase be linguistically motivated. For example, in (7), the NP-VP phrase SV is not linguistically motivated. Its purpose is to distinguish subject from non-subject NP's before trying to identify clause boundaries, in order to avoid e.g. having *John* be misidentified as the object of *said* in *I said John was smart.* If we omitted the SV pattern, the S pattern would consume *I said John* in *I said John was smart,* leaving a stranded VP.

[4] Though superficially similar, recursive transition networks [88] differ, as the name suggests, precisely in the question of recursion, which is crucially absent in finite-state cascades.

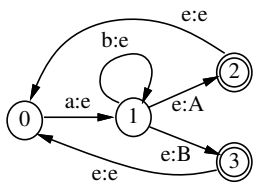
Patterns are translated by standard techniques [7] into finite-state automata. We take the union of all automata at a given stratum, yielding a single automaton. This stratum automaton is determinized and minimized. Since the stratum automaton is deterministic, each prefix of the input takes it to a unique state, hence (assuming that the input is of finite length) there is a longest prefix of the input that takes the stratum automaton into a final state, and that final state is unique. In turn, that final state corresponds to a set of final states from the pattern automata, allowing us to determine which pattern or patterns were responsible for the match.

Instead of using the longest-match heuristic, we can construct a hidden Markov model from the stratum recognizer. For concreteness' sake, let us consider the patterns $A \rightarrow ab*$, $B \rightarrow ab*$, yielding the stratum automaton (8a). First we turn the stratum automaton into a transducer by adding arcs that output A and B, leading to new final states that have no outgoing arcs. Then we add epsilon transitions from the new final states back to the initial state, to make an automaton that recognizes patterns A and B repeatedly. This yields automaton (8b).

(8) a.



b.

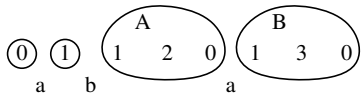


For example, running automaton (8b) against input *aba* produces (as one alternative) the state sequence

Output				A			B		
States	0	1	1	2	0	1	3	0	
Input		a	b			a			

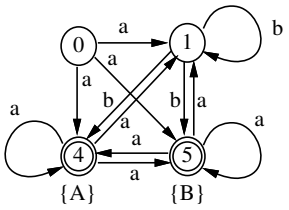
Now we eliminate transitions that consume no input by folding them into the surrounding states to create new complex states:

(9)



Applying this construction systematically to the automaton (8b) yields the following automaton, which is suitable for use in a hidden Markov model:

(10)



State 4 represents the first 'complex state' in (9), involving transitions from 1 to 2 to 0. Accordingly, it has the same incoming arcs as state 1, and the same outgoing arcs as state 0. State 5 represents the second complex state in (9). It has the same incoming and outgoing arcs as state 4; the difference being that state 4 represents the recognition of an A whereas state 5 represents the recognition of a B.

If we train the HMM (10) and then use the Viterbi algorithm to find the most-likely state sequence for a given input, the recognized phrases (A's and B's) can be read unambiguously off the state sequence. For example, suppose that the the most-likely state sequence for input *aab* is 0415. This represents the parse $[_{\mathrm{A}}$ a$][_{\mathrm{B}}$ ab$]$:

	[		$_{\mathrm{A}}$][			$_{\mathrm{B}}$]
state:	0		4		1	5
input:		a		a	b	

State 4 marks the end of an A phrase, and state 5 marks the end of a B phrase. Each phrase begins where the previous phrase ended.

Ejerhed [27] compares the performance of longest-match and stochastic versions of the stratal parser, and reports lower error rates for the stochastic version: 1.4% vs. 3.3% for noun phrase chunks, and 6.5% vs. 13% for clauses. HMM chunk parsers have also been investigated by Chen and Chen [19] and Rooth [72].

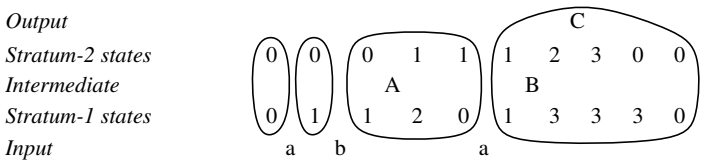
The parser just described consists of a sequence of stochastic finite-state automata (i.e., HMM's), one for each stratum. It is possible to fold all the strata together into a single HMM. The states of the new HMM are tuples of states, one from each stratum. For example, suppose the automaton for stratum 1 is as in the previous example, initially in the 'unfolded' form (8b). Let us add a second stratum with pattern $C \rightarrow AB$. Here's an example of a state sequence on input *aba*:

(11)

Output								C			
Stratum-2 states	0	0	0	1	1	1	2	3	0	0
Intermediate			A			B				
Stratum-1 states	0	1	1	2	0	1	3	3	3	0
Input		a	b			a				

(We have inserted no-op transitions where there is a transition in one stratum but no change of state in the other.) As in the previous example, we fold

transitions that involve no consumption of input into a new complex state, and we now also fold together states across strata. Continuing example (11), the folded automaton passes through the following sequence of four complex states:



To construct the entire set of complex states and transitions, we start with a state consisting of initial states in every stratum, then add a new arc and (if necessary) a new state for every possible input symbol on which there is a transition. The process is repeated until no new arcs or states can be added. On the assumption that no patterns match the empty string, termination is guaranteed. The result is a single HMM spanning all strata, such that we can read off the parse for a given input from the state sequence the automaton passes through. In this way it is possible to do stratal parsing with standard HMM training and recognition techniques.

In more formal terms, we have turned each stratum automaton into a finite-state transducer, composed the transducers, and eliminated ϵ-transitions [48, 71, 65]. The only difference from standard transducer composition is that outputs at intermediate levels matter. The standard algorithms assume that states may be merged if doing so does not affect the relationship between the input and the final output. But in stratal parsing, we wish to keep states distinct that encode different intermediate-level outputs, since different intermediate-level outputs represent different parses.

2.6 Longest Match

Despite the attractions of the HMM version of the stratal parser, we should not be too hasty to abandon the deterministic 'longest-match' version entirely. It also has advantages, including speed, the ability to do phrase-spotting, and the ability to capture a wider range of context effects.

'Phrase-spotting' refers to the ability to recognize phrases reliably without analyzing the entire sentence. Traditional parsing methods, as well as HMM's, do a global optimization. If we have a very good model of certain phrases, but a very poor model of the rest of the language, our ability to detect the phrases of interest suffers. By contrast, the methodology behind the longest-match approach is to start from 'islands of reliability', to build up larger and larger phrases that are themselves reliable, but may enclose stretches whose analysis is uncertain, such as noun-noun modification (within noun chunks), or PP-attachment (within simplex clauses).

It should be noted that the predicate 'longest match' cannot be captured by any manipulation of the probabilities in a stochastic CFG. 'Longest match'

involves a comparison across competing analyses: a phrase is a longest match only if there is no competing analysis with a longer phrase at the same level and position. It can be expressed in terms of the context in which a phrase appears, but not in terms of context-free rewrite probabilities.

Further, the measures of reliability we are interested in are global precision and recall, which also cannot be identified with the probabilities supplied by a stochastic grammar. In particular, precision is *not* the same as the conditional probability of a phrase given the input sentence. A particular pattern could have very low precision, in general, yet if a phrase it outputs happens to belong to the only parse the grammar assigns to a given sentence (perhaps because of shortcomings in the grammar), the conditional probability of the phrase is 1.

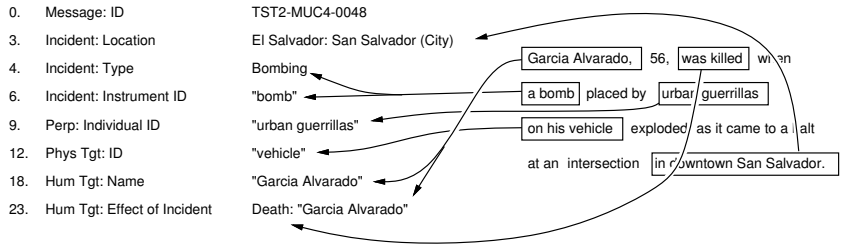
We can think of the longest-match parser as an instance of parsing on the basis of a classification model, in which 'longest match' is one predictor variable. As such, we have considerably more flexibility for bringing additional contextual information to bear than in the straightforward HMM version.

2.7 Applications

Partial parsing has been put to use in a variety of ways, including bootstrapping a more complete parser, terminology and multi-word term extraction for information retrieval, and as a component of data extraction systems.

The chief goal in bootstrapping is the acquisition of lexical information needed for more complete parsing. The type of information to be acquired is primarily collocational, particularly subcategorization frames and selectional restrictions. In an influential paper by Church et al. [21], Hindle's parser Fidditch was put to use to extract subject-verb and verb-object pairs. Measures of associativity were applied to the pairs, to provide a crude model of selectional restrictions. Two measures of association were considered: mutual information and 't-scores' (though a normal approximation was used instead of a t distribution). Hindle [40] also used Fidditch to induce a noun hierarchy, and Hindle and Rooth [42] used Fidditch to extract V-NP-PP triples, then used the 't-score' measure of association to disambiguate the attachment of the PP. Partial parsing has also been used as a preprocessor for the acquisition of verbal subcategorization frames [60], and to support finer-grained alignment in bilingual corpora [55].

A major impetus for interest in partial parsing has been provided by the series of competitions known as Message Understanding Conferences (MUC). These are U.S.-government sponsored competitions in which the task is filling in relational database templates from newswire text. Here is an example of an abbreviated template, and the textual sources for each fill from a paragraph of news text:



The competition is highly goal-oriented and systems' performance on the extraction task is exhaustively evaluated. This encourages very pragmatic approaches.

The typical MUC system goes through roughly these steps: filter out irrelevant texts, tokenize, parse around keywords, fill semantic frames, and merge frames to fill data templates. Partial parsing is a natural choice in this context, as Weischedel et al. note [87]. One can do very well by recognizing syntactic fragments around informative words, plus special constructs like dates, names, and place names, then putting fragments together using information from domain-specific semantic frames. The parallelism to chunk-and-attachment parsing is inescapable.

One group in particular created quite a stir when they replaced a traditional system that had been developed over many years with a cascaded finite-state recognizer. In a remarkably candid quote, they describe what motivated their decision:

> We were struck by the strong performance that the group at the University of Massachusetts got out of a fairly simple system. ... [And] it was simply too embarrassing to have to report at the MUC-3 conference that it took TACITUS 36 hours to process 100 messages. FASTUS has brought that time down to 11 minutes. [43]

After partial parsing, syntactic fragments are stitched together using semantic frames. Because the template-filling task keeps the semantic space quite limited, it is practical to construct a knowledge base of semantic frames by hand. The semantic frame of a chunk is defined to be the semantic frame of its head. One chunk can be attached to another only if the semantic frame of the first can fill a slot in the semantic frame of the second. Type restrictions on slots have the consequence that only a few ways of attaching chunks to one another are possible.

2.8 Acquisition

An open research question is how the grammar for a partial parser might be automatically acquired. A number of avenues are currently being pursued, though none of the current techniques yields results competitive with hand-written grammars.

There are standard supervised learning techniques for finite-state automata [32, 33, 74] and probabilistic grammars [80]. As mentioned above, Brill has applied his rule-learning techniques to phrase-structure grammars [15], though generalizations of the rules he uses for tagging might be more effective for partial parsing [69].

Techniques for unsupervised learning of phrase structure have also been proposed. The discovery procedures of Zellig Harris can be seen as an early attempt at unsupervised phrase-structure learning [36, 37, 38]. Traditionally, phrases have been defined in terms of two aspects of distribution: phrases are coherent—they move, conjoin, etc. as a unit—and phrases of the same type are intersubstitutable—they appear in the same contexts. Quantitative measures for these properties are currently well-known in computational linguistics, probably the most prevalent being mutual information as a measure of coherence, and divergence or relative entropy as a measure of substitutability.

In the 1960's, Stolz [81] proposed using mutual information (though not under that name) to identify phrases in unannotated text. A more elaborate technique for parsing by means of mutual information is described by Magerman and Marcus [58]. Finch [31] develops a general framework for induction via substitutability, and explores a range of distributional similarity functions. Work by Smith and Witten [79] is especially interesting for chunk parsing because they first identify and categorize function words, then induce a chink-and-chunk grammar on that basis.

To some extent, the need for mechanical learning methods for partial parsers is not pressing, to the extent that partial parsing is defined as recovering just that structure that can be recovered with minimal manually-supplied information. Nonetheless, practical acquisition methods would simplify the development of a parser for new languages, or new genres of text. And an acquisition method for chunks, combined with an acquisition method for attachment, could serve to further our understanding of human language acquisition.

References

[1] Steven Abney. Rapid incremental parsing with repair. In *Proceedings of the 6th New OED Conference: Electronic Text Research*, pages 1–9, Waterloo, Ontario, October 1990. University of Waterloo.

[2] Steven Abney. Syntactic affixation and performance structures. In D. Bouchard and K. Leffel, editors, *Views on Phrase Structure*. Kluwer Academic Publishers, 1990.

[3] Steven Abney. Parsing by chunks. In Robert Berwick, Steven Abney, and Carol Tenny, editors, *Principle-Based Parsing*. Kluwer Academic Publishers, 1991.

[4] Steven Abney. Chunks and dependencies: Bringing processing evidence to bear on syntax. In Jennifer Cole, Georgia M. Green, and Jerry L. Morgan, editors, *Computational Linguistics and the Foundations of Linguistic Theory*, pages 145–164. CSLI, 1995.

[5] Steven Abney. Partial parsing via finite-state cascades. In John Carroll, editor, *Workshop on Robust Parsing (ESSLLI '96)*, pages 8–15, 1996.

[6] I. Aduriz, I. Alegria, J.M. Arriola, X. Artola, A. Diaz de Illarraza, N. Ezeize, K. Gojenola, and M. Maritxalar. Different issues in the design of a lemmatizer/tagger for Basque. In *SIGDAT-95 (EACL-95 Workshop)*, 1995. Also available as cmp-lg:9503020.

[7] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[8] Susan Armstrong, Graham Russell, Dominique Petitpierre, and Gilbert Robert. An open architecture for multilingual text processing. In *EACL-95 SIGDAT Workshop*, pages 30–34, 1995.

[9] L. R. Bahl and R. Mercer. Part-of-speech assignment by a statistical decision algorithm. In *International Symposium on Information Theory*, Ronneby, Sweden, 1976.

[10] Lalit R. Bahl, Peter F. Brown, Peter V. de Souza, and Robert L. Mercer. A tree-based statistical language model for natural language speech recognition. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, pages 507–514, 1991.

[11] Ezra Black, F. Jelinek, J. Lafferty, R. Mercer, and S. Roukos. Decision tree models applied to the labeling of text with parts-of-speech. In *Darpa Workshop on Speech and Natural Language*, San Mateo, CA, 1992. Morgan Kaufman.

[12] Didier Bourigault. Surface grammatical analysis for the extraction of terminological noun phrases. In *COLING-92, Vol. III*, pages 977–981, 1992.

[13] Thorsten Brants and Christer Samuelsson. Tagging the Teleman Corpus. In *Proc. 10th Nordic Conf. of Comp. Ling.*, 1995. Available as cmp-lg/9505026.

[14] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

[15] Eric Brill. *Transformation-Based Learning*. PhD thesis, Univ. of Pennsylvania, 1993.

[16] Eric Brill. Some advances in transformation-based part of speech tagging. In *Proceedings of AAAI-94*, 1994.

[17] P. Brown, V. Della Pietra, P. deSouza, J. Lai, and R. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–480, 1992.

[18] Jean-Pierre Chanod and Pasi Tapanainen. Tagging French – comparing a statistical and constraint-based method. In *EACL-95*, 1995.

[19] Kuang-hua Chen and Hsin-Hsi Chen. Extracting noun phrases from large-scale texts: A hybrid approach and its automatic evaluation. In *Proceedings of ACL*, 1994.

[20] Kenneth Church. A stochastic parts program and noun phrase parser for unrestricted texts. In *Proceedings of the Second Conference on Applied Natural Language Processing*, Austin, Texas, 1988.

[21] Kenneth Church, William Gale, Patrick Hanks, and Donald Hindle. Parsing, word associations and typical predicate-argument relations. In *International Workshop on Parsing Technologies*, pages 389–98, 1989.

[22] Kenneth Church and Robert Mercer. Introduction to the special issue on computational linguistics using large corpora. *Computational Linguistics*, 19(1):1–24, 1993.

[23] Doug Cutting, Julian Kupiec, Jan Pedersen, and Penelope Sibun. A practical part-of-speech tagger. In *Third Conference on Applied Natural Language Processing (ANLP-92)*, pages 133–140, 1992.

[24] Evangelos Dermatas and George Kokkinakis. Automatic stochastic tagging of natural language texts. *Computational Linguistics*, 21(2):137–164, 1995.

[25] S. DeRose. Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14(1), 1988.

[26] A.M. Derouault and B. Merialdo. Language modelling at the syntactic level. In *Proc. 7th Int'l. Conference on Pattern Recognition*, 1984.

[27] Eva Ejerhed. Finding clauses in unrestricted text by finitary and stochastic methods. In *Proceedings of the 2nd Conference on Applied Natural Language Processing*, Austin, Texas, 1988.

[28] Eva Ejerhed and Kenneth Church. Finite state parsing. In Fred Karlsson, editor, *Papers from the Seventh Scandinavian Conference of Linguistics*, pages 410–432, Hallituskatu 11–13, SF-00100 Helsinki 10, Finland, 1983. University of Helsinki, Department of General Linguistics.

[29] David Elworthy. Does Baum-Welch re-estimation help taggers? In *4th Conference on Applied Natural Language Processing (ANLP-94)*, pages 53–58, 1994.

[30] Helmut Feldweg. Implementation and evaluation of a German HMM for POS disambiguation. In *EACL SIGDAT Workshop*, 1995.

[31] Steven Paul Finch. *Finding Structure in Language*. PhD thesis, University of Edinburgh, 1993.

[32] K.S. Fu. *Syntactic Methods in Pattern Recognition*. Academic Press, New York, 1974.

[33] K.S. Fu and T.L. Booth. Grammatical inference: Introduction and survey. *IEEE Trans. on System, Man and Cybernetics*, 5, 1975. In two parts.

[34] R. Garside. The CLAWS word-tagging system. In Garside R., F. Leech, and G. Sampson, editors, *The Computational Analysis of English*. Longman, 1987.

[35] Barbara B. Greene and Gerald M. Rubin. Automated grammatical tagging of English. Department of Linguistics, Brown University, 1971.

[36] Zellig Harris. From morpheme to utterance. *Language*, 22, 1946.

[37] Zellig Harris. *Methods in Structural Linguistics*. University of Chicago Press, Chicago, 1951.

[38] Zellig Harris. From phoneme to morpheme. *Language*, 31, 1955.

[39] Donald Hindle. User manual for Fidditch. Technical Memorandum #7590-142, Naval Research Laboratory, 1983.

[40] Donald Hindle. Acquiring a noun classification from predicate-argument structures. Tech. Memo. 11222-881017-15, AT&T Bell Laboratories, 1988.

[41] Donald Hindle. A parser for text corpora. In A. Zampolli, editor, *Computational Approaches to the Lexicon*. Oxford University Press, New York, 1994.

[42] Donald Hindle and Mats Rooth. Structural ambiguity and lexical relations. In *Proceedings of DARPA Speech and Natural Language Workshop*. Morgan Kaufman: New York, June 1990.

[43] Jerry R. Hobbs et al. SRI International: Description of the FASTUS system used for MUC-4. In *Proceedings, Fourth Message Understanding Conference (MUC-4)*, pages 268–275, San Mateo, CA, 1992. Morgan Kaufmann.

[44] Ajay N. Jain. *PARSEC: A Connectionist Learning Architecture for Parsing Spoken Language*. PhD thesis, CMU, Pittsburgh, PA, 1991. Available as Technical Report CMU-CS-91-208.

[45] F. Jelinek. Self-organized language modeling for speech recognition. *W & L*, pages 450–506, 1985.

[46] F. Jelinek and R. Mercer. Interpolated estimation of Markov source parameters from sparse data. In E.S. Gelsema and L.N. Kanal, editors, *Pattern Recognition in Practice*, pages 381–397. Amsterdam : North Holland Publishing Co., 1980.

[47] Aravind K. Joshi and B. Srinivas. Disambiguation of super parts of speech (or supertags): Almost parsing. In *COLING-94*, 1994.

[48] Ronald Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 1994.

[49] Fred Karlsson. Constraint grammar as a framework for parsing running text. In *COLING-90*, pages 168–173, 1990.

[50] Fred Karlsson, Atro Voutilainen, Juha Heikkilä, and Arto Anttila, editors. *Constraint Grammar*. Mouton de Gruyter, Berlin, 1995.

[51] S. Klein and R. Simmons. A computational approach to grammatical coding of English words. *JACM*, 10:334–337, 1963.

[52] Kimmo Koskenniemi. Two-level morphology: A general computational model for word-form recognition and production. Department of General Linguistics, University of Helsinki, 1983.

[53] Kimmo Koskenniemi. Finite-state parsing and disambiguation. In *COLING-90*, pages 229–232, 1990.

[54] Kimmo Koskenniemi, Pasi Tapanainen, and Atro Voutilainen. Compiling and using finite-state syntactic rules. In *COLING-92*, pages 156–162, 1992.

[55] Julian Kupiec. An algorithm for finding noun phrase correspondences in bilingual corpora. In *31st Annual Meeting of the Association for Computational Linguistics*, pages 17–22, 1993.

[56] Leech, Garside, and Atwell. The automatic grammatical tagging of the LOB corpus. *ICAME News*, 7:13–33, 1983.

[57] Fernando Sánchez León and Amalio F. Nieto Serrano. Development of a Spanish version of the Xerox tagger. CRATER/WP6/FR1 and cmp-lg/9505035, 1995.

[58] D. Magerman and M. Marcus. Parsing a natural language using mutual information statistics. In *Proceedings of AAAI-90*, 1990.

[59] David Magerman. *Natural Language Parsing as Statistical Pattern Recognition*. PhD thesis, Stanford, 1994.

[60] Christopher D. Manning. Automatic acquisition of a large subcategorization dictionary from corpora. In *31st Annual Meeting of the Association for Computational Linguistics*, pages 235–242, 1993.

[61] Mitchell Marcus. *A Theory of Syntactic Recognition for Natural Language*. The MIT Press, Cambridge, MA, 1980.

[62] Bernard Merialdo. Tagging English text with a probabilistic model. *Computational Linguistics*, 20(2):155–172, 1994.

[63] Kemal Oflazer and İlker Kuruöz. Tagging and morphological disambiguation of Turkish text. In *Fourth Conference on Applied Natural Language Processing (ANLP-94)*, pages 144–149, 1994.

[64] Douglas B. Paul. Speech recognition using Hidden Markov Models. *Lincoln Laboratory Journal*, 3(1):41–62, 1990.

[65] Fernando C.N. Pereira, Michael Riley, and Richard W. Sproat. Weighted rational transductions and their application to human language processing. In *Human Language Technology Workshop*, pages 262–267, 1994.

[66] L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, February 1989.

[67] L.R. Rabiner and B.H. Juang. An introduction to Hidden Markov Models. *IEEE ASSP Magazine*, page 4ff, January 1986.

[68] Lance A. Ramshaw and Mitchell P. Marcus. Exploring the statistical derivation of transformational rule sequences for part-of-speech tagging. In *Proceedings of the ACL Balancing Act Workshop*, 1994.

[69] Lance A. Ramshaw and Mitchell P. Marcus. Text chunking using transformation-based learning. In *ACL Third Workshop on Very Large Corpora*, pages 82–94, 1995.

[70] R.L. Rivest. Learning decision lists. *Machine Learning*, 2:229–246, 1987.

[71] Emmanuel Roche and Yves Schabes. Deterministic part-of-speech tagging with finite-state transducers. *Computational Linguistics*, pages 227–254, 1995.

[72] Mats Rooth. Unitary stochastic part-of-speech and phrase tagging. Manuscript, University of Stuttgart, 1994.

[73] Ian C. Ross and John W. Tukey. Introduction to these volumes. In *Index to Statistics and Probability*, pages iv–x. The R & D Press, Los Altos, CA, 1975.

[74] E. Sanchis, F. Casacuberta, I. Galiano, and E. Segarra. Learning structural models of subword units through grammatical inference. In *IEEE ICASSP, Vol. 1*, pages 189–192, 1991.

[75] H. Schmid. Improvements in part-of-speech tagging with an application to German. In *EACL SIGDAT Workshop*, 1995.

[76] Hinrich Schütze. Part-of-speech induction from scratch. In *31st Annual Meeting of the Association for Computational Linguistics*, pages 251–258, 1993.

[77] Hinrich Schütze. Distributional part-of-speech tagging. In *EACL-95*, 1995.

[78] Christoph Schwarz. Automatic syntactic analysis of free text. *JASIS*, 41(6):408–417, 1990.

[79] Tony C. Smith and Ian H. Witten. Language inference from function words. Manuscript, University of Calgary and University of Waikato, January 1993.

[80] Andreas Stolcke and Stephen Omohundro. Inducing probabilistic grammars by Bayesian model merging. In *Grammatical Inference and Applications, Second International Colloquium on Grammatical Inference*. Springer Verlag, 1994.

[81] W. Stolz. A probabilistic procedure for grouping words into phrases. *Language and Speech*, 8:219–235, 1965.

[82] Marc Vilain and David Palmer. Transformation-based bracketing: Fast algorithms and experimental results. In John Carroll, editor, *Workshop on Robust Parsing (ESSLLI '96)*, pages 93–102, 1996.

[83] Atro Voutilainen. NPtool, a detector of English noun phrases. In *Proceedings of the Workshop on Very Large Corpora*, pages 48–57, 1993.

[84] Atro Voutilainen. A syntax-based part-of-speech analyser. In *EACL-95*, 1995.

[85] Atro Voutilainen, Juha Heikkilä, and Arto Anttila. Constraint grammar of English: A performance-oriented introduction. Technical Report Publication No. 21, University of Helsinki, Department of General Linguistics, Helsinki, 1992.

[86] Atro Voutilainen and Timo Jarvinen. Specifying a shallow grammatical representation for parsing purposes. In *EACL-95*, 1995.

[87] Ralph Weischedel et al. Partial parsing: A report on work in progress. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 204–209, Asilomar, CA, 1991.

[88] William A. Woods. Transition network grammars for natural language analysis. *Communications of the ACM*, 13:591–596, 1970.

[89] David Yarowsky. Decision lists for lexical ambiguity resolution. Manuscript, University of Pennsylvania, 1994.