Head-Driven Probabilistic Parsing for Word Lattices

by

Christopher Collins

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Head-Driven Probabilistic Parsing for Word Lattices

Christopher Collins

Master of Science

Graduate Department of Computer Science

University of Toronto

2004

This thesis presents the first application of the state-of-the-art head-driven statistical parsing model of Collins (1999) as a simultaneous language model and parser for large-vocabulary speech recognition. The model is adapted to an online left to right chart-parser for word lattices, integrating acoustic, N-gram, and parser probabilities. The parser uses structural and lexical dependencies not considered by N-gram models, conditioning recognition on more linguistically-grounded relationships. By preferring paths through the word lattice for which a probable parse exists, word error rate can be reduced and important syntactic and semantic relationships can be determined in a single step process. New forms of heuristic search and pruning are employed to improve efficiency. Experiments on the Wall Street Journal treebank and lattice corpora show word error rates competitive with the standard N-gram language model while extracting additional structural information useful for speech understanding.

Acknowledgements

Gerald Penn, my advisor, has been supportive and encouraging throughout my time at the University of Toronto. During some very difficult times, Gerald reassured me that I would complete my degree successfully and, perhaps more importantly, reminded me of the exciting nature of this research.

Bob Carpenter provided the framework source code for probabilistic parsing of word lattices, and many initial ideas for this project. I am lucky to have had the chance to build upon a project started by such a great scholar. Keith Hall suggested the training and test data for this work, and helped me obtain the necessary tools for evaluation. This work builds upon the lexicalized head-driven statistical parser presented in Michael Collins' PhD thesis. He has been helpful by answering questions I had about this reimplementation of the model.

A great deal of thanks also to my parents for their encouragement, financial assistance, and for their helpful editing of this work. Also thanks to my grandparents, my sister Lesley, and friends Andrea, Sylvia, and Dan, who have always been there for me through the personal difficulties of the past 2 years. They even politely smile and nod when I start talking about language modelling.

Acknowledgements are due to NSERC for funding support, and my colleagues in the CL Group for their friendship, LaTeX advice, and for creating an exciting environment in which to work.

Finally, I would like to offer special acknowledgement to Suzanne Stevenson for her thorough reading of this work and very helpful advice. Suzanne was instrumental in convincing me that the University of Toronto would be a welcoming and friendly place to study, and her continual support of graduate students goes beyond what could be expected of any faculty member.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

The question of how to integrate high-level knowledge representations of language with automatic speech recognition (ASR) is becoming more important as speech recognition technology matures, the rate of improvement of recognition accuracy decreases, and the need for additional information (beyond simple transcriptions) becomes evident. Most of the current best (here we define "best" in terms of word error rate (WER)) ASR systems use an N-gram language model of the type pioneered by Bahl *et al.* (1983). The prediction of the next word is based on frame-level[1] acoustic information and lexical dependence of Markov order N (*i.e.*, the N previous words). This model has long been thought linguistically deficient and far from appropriate by the computational linguistics community. However, only recently has research begun to show progress towards application of new and better models of spoken language.

One significant barrier, according to Bourlard *et al.* (1996), has been the continuous quest for decreasing word error rate (WER). By focusing on gains relative to this metric, current techniques are continually adapted and incrementally improved, but innovation

[1] A frame is usually a 10ms interval of acoustic data.

is limited. In this work, we suggest that there are other ways to measure success of spoken language processing, such as ability to educe semantic features, syntactic structure, and other high-level knowledge. These are areas for which standard N-gram models provide little information. High-level knowledge is crucial when the ultimate goal is speech understanding. Jurafsky and Martin (2000) explains the concept of expanding "automated speech recognition" (discovering the words a person says) to "automated speech understanding" by extracting the syntax, lexical semantics, and compositional semantics of an utterance. For example, the same sequence of words can have different meaning, depending on syntactic, pragmatic, prosodic, and contextual features. Only by applying unconventional techniques — which may increase WER in the short term — will progress towards better speech understanding be achieved.

We present new integration of head-driven lexicalized parsing with acoustic and N-gram models for speech recognition. Our goal is to extract high-level structure from speech, while simultaneously selecting the best path in a word lattice. Parse trees generated by this process will be useful for automated speech understanding, perhaps for use in higher semantic parsing (Ng and Zelle, 1997). In addition to discovering higher-level syntactic structure useful for semantic interpretation, WER should also decrease with use of a parsing language model. An N-gram model uses only the previous few words as conditioning context for selection of the next word. This has proved to be a powerful model, but it does have severe limitations. For example, consider the word `after` in the sentence:

`He will buy the Foof Inc. stocks after the price falls below $5.00.`

A trigram would need to predict `after` from the uninformative pair `(stocks, Inc.)`, whereas a more intuitive model would capture the word `will` as a powerful predictor. The distance in this case is 7, but the dependency could have been arbitrarily long in the string. If we were to consider longer distance relationships, we may have a better chance at predicting the next word. Expanding to a higher N-gram is not practical, as

the parameter space increases as $|\mathcal{V}|^N$, where $\mathcal{V}$ is the vocabulary. Sparse data problems soon arise.

Collins (1999) presents three lexicalized models which do consider long-distance dependencies within a sentence. Grammar productions are conditioned on head-words (briefly, the most important or meaningful word in a sequence). The conditioning context is thus more focused than that of a large N-gram covering the same span, so the sparse data problems arising from the sheer size of the parameter space are less pressing[2]. The head-driven probabilistic word lattice parser described in chapter 4 is based on parsing model II of Collins (1999), reviewed in chapter 3. Collins (1999) presents an example of the model's ability to distinguish between candidate sentences of the type found in speech recognition, which we repeat below:

- Actual Utterance: He is a resident of the U.S. and *of* the U.K.
- Speech Recognizer Hypothesis: He is a resident of the U.S. and *that* the U.K.

The parsing model assigns 78 times higher probability to the correct string, whereas a simple bigram trained on the same data assigns over 10 times greater probability to the incorrect string — the bigram `(and that)` is 15 times more frequent than `(and of)`. Despite the promise of this example, this model has not been previously applied to parsing word lattices for speech recognition.

New parsing models and integration techniques have been reported recently, such as the Structured Language Model (Chelba and Jelinek, 2000) and the lexicalized parser of Eugene Charniak (Hall and Johnson, 2003; Charniak, 2001). These focus on potential ways of using more linguistically-meaningful dependence relations to reduce WER for automatic speech recognition with little attention to high-level information extraction. Roark (2001) is the work closest in thesis to that presented in this dissertation. It reports

[2]However, sparse data problems arising from the limited availability of annotated training data become a problem.

on the use of a lexicalized probabilistic top-down parser for word lattices, evaluated both on parse accuracy and WER. Our work is different from Roark (2001) in that we use a bottom-up parsing algorithm with dynamic programming based on the parsing model II of Collins (1999).

The largest improvements in WER have been seen with N-best list rescoring (Xu *et al.*, 2002). The best N hypotheses of a simple speech recognizer are processed by a more sophisticated language model and re-ranked. This method is algorithmically simpler than parsing lattices, as one can use a model developed for strings, which need not operate strictly left to right. However, in this dissertation we confirm the observation of (Ravishankar, 1997; Hall and Johnson, 2003) that parsing word lattices saves computation time by only parsing common substrings once.

A word lattice is a compact representation of a large number of utterance hypotheses (*e.g.*, the most compact combination of the elements of an N-best list). Words are linked by edges. Word lattices have a unique start and end node, so that any path from the start to the end represents a complete utterance hypothesis. A standard lattice format (SLF), developed and documented by the HTK project has been adopted, and will be explained further in section 2.6.1.

Past work on word lattices has been met with limited success, when measured by WER improvement, and the systems have been unable to compete with N-gram systems for speed and efficiency. Many (*e.g.*, Hall and Johnson, 2003; Weber *et al.*, 1997) have introduced a pre-parsing stage to prune the input, which is then passed to the more sophisticated parser. This is in agreement with the stepwise model integration paradigm of Harper *et al.* (1994) which recommends that each step overgenerate and pass results to a subsequent more sophisticated (and often more computationally intensive) step for additional pruning, eventually arriving at a final hypothesis. The experience of others reveals that the size of word-lattices, sparseness of training data, and the time and space complexity of chart-parsing will present a significant challenge in our work.

We test the head-driven statistical lattice parser with word lattices from the NIST HUB-1 corpus, which has been used by others in related work (*e.g.*, Hall and Johnson, 2003; Roark, 2001; Chelba and Jelinek, 2000). Parse accuracy and word error rates are reported. We present an analysis of the effects of pruning and heuristic search on efficiency and accuracy and note several simplifying assumptions common to other reported experiments in this area, which present challenges for scaling up to real-world applications.

1.2 Statement of Thesis and Objectives

1.2.1 Thesis

The state-of-the-art in speech understanding can be advanced by extracting linguistic structure from word lattices during speech recognition. Parsing models, such as a head-driven statistical model, can be formulated in a left-to-right manner in order to work as language models for speech recognition. It can be shown that the success rate of finding the true word sequence for a given utterance can be improved upon using parsing, as the model selects conditioning which is more linguistically salient than that used by the currently standard N-gram model. The parameters used by the head-driven parsing model are roughly orthogonal to lexical N-gram parameters, and thus can be combined with those of an N-gram model. The combination model will select not only a probable word sequence, but also one with a probable sentence structure. Assuming a parameterization and training corpus adapted to the challenges of spoken natural language, a parsing system can improve on word-level recognition and extract important structure in a single step.

1.2.2 Objectives

This document supports the thesis by making the following specific contributions:

1. **Overview of existing work:** A thorough review of methods for coupling language and acoustic models for speech recognition, of new language models incorporating better understanding of linguistic structure, and of recent work in the area of lattice parsing, is conducted. The review reveals difficulties in comparing results in this area, a lack of focus on extracting linguistic structure from spoken language, and a need for a standard word lattice corpus. The overview also presents promising results from some preliminary studies.

2. **New standards for evaluation:** The inadequacy of word error rate as the only measure of speech recognition success is described. Other measures, such as parsing accuracy, are suggested as important measures for speech understanding tasks.

3. **A chart parser for lattices:** A modular chart parser for word lattices is presented. This has been implemented using Java. The grammar and tagger modules can be easily changed to incorporate other parameterizations of statistical parsing.

4. **Investigation of head-driven statistical parsing as a model of language:** The chart parser for lattices is implemented with a current state-of-the-art statistical parsing model — model II of Collins (1999). The implementation is tested on strings and word lattices, and both parse accuracy and word error rate are compared with related work. An evaluation of how heuristic search and pruning affects efficiency and accuracy is presented.

1.3 Overview of the Dissertation Structure

The remainder of this dissertation is structured as follows:

Chapter 2: State of the Art in Language Modeling for Speech reviews the basics of language modelling, including a critical evaluation of how success is currently measured. Methods for coupling language models with speech recognition systems

are described, and examples from the current literature, including recent attempts to parse word lattices, are compared.

Chapter 3: Review of the Probabilistic Model presents the parameterization of the head-driven statistical parsing model used in the word lattice parser. Smoothing methods for parameter estimation are presented as a way to ameliorate sparse data problems.

Chapter 4: Head-Driven Parsing for Speech Recognition begins with a description of the parsing model of chapter 3 as a language model for speech recognition. The algorithms and main data structures for word-lattice tagging and left-to-right probabilistic lattice chart parsing are explained. The parser operates on trees with at most binary branching, so the mapping from Penn-Treebank-style trees to parser format trees is explained. Finally we discuss the dynamic programming restrictions imposed on the model by computational resource limitations, and the practical details and achievements of the implementation are outlined.

Chapter 5: Experimental Results presents an evaluation of the word lattice parser over strings and word lattices. Standard measures for parsing and speech recognition performance are presented, along with an analysis of the impact of the restrictions imposed on the model. The time and resources required to parse N-best lattices and N-best lists are compared and the efficiency and utility of lattice parsing is confirmed.

Chapter 6: Conclusions is an analysis of the main contributions and a presentation of opportunities to build upon this research.

Chapter 2

State of the Art in Language Modeling for Speech

2.1 Background

Speech recognition by humans is an extremely complex process, drawing on an intricate acoustic sensory system, a vast knowledge base, and a powerful reasoning system (Brill *et al.*, 1998). Emulating this behaviour with a computer has long been a goal of computational linguistics and engineering research, and one assumed to be a future possibility by everyone from scientist Alan Turing to film-maker Stanley Kubrick. The realization of this goal has proved much more difficult than anticipated.

The task is traditionally divided into two levels — automatic speech recognition (ASR) and its extension, automatic speech understanding (ASU). ASR has been defined as the automated mapping from a spoken utterance to a word string; the use of high-level knowledge has been distinguished as automated speech understanding (Chelba, 2000). However, we take the view that the *recognition* of an utterance can be improved by integrating the high-level information directly into the search for the true word string, thus the traditional senses of recognition and understanding cannot be decoupled. Speech

understanding in this work will be considered the task of using the high-level information extracted during recognition — such as using headwords in parse trees to query a database.

We define ASR as an automated mapping from a spoken utterance to a range of possible levels of information — lexical, syntactic, semantic, and pragmatic. These levels of information each have a variety of uses, such as transcription of speech (many commercial software applications attempt this), translation to another natural language (such as the VERBMOBIL project; `http://verbmobil.dfki.de`), question answering (such as with the TINA semantic analysis system (Seneff, 1992)), emotion detection (Polzin and Waibel, 1998), or in assistive technology devices for persons with disabilities.

The question of how to integrate language models with speech recognition systems can currently be answered with three coupling paradigms: **tight**, **incremental**, and **sequential**. First, we will review general speech recognition terminology and structure common to all three coupling methods.

The different types of knowledge useful for speech recognition are shown in figure 2.1. Harper *et al.* (1994) arranges these into the hierarchy shown, based on evidence that to use (or extract) a type of information, the previous must be available.

Prosody is a special level of knowledge, grouped as low-level knowledge and as high-level knowledge. Broadly, prosody is accepted as properties of speech related to segments larger than phones (Jurafsky and Martin, 2000). These properties include pitch, pauses, relative duration, intonation, voice quality, and energy, which more generally can be considered to compose accentuation, phrasing, and pauses. Ladd and Cutler (1983) further refines this definition in terms of concrete features, such as acoustic parameters of pitch, duration, and intensity, and abstract features of phonological organization on the suprasegmental level. Prosody can be considered high-level knowledge, for example, when it is used to indicate a question sentence, which can affect syntactic and semantic interpretation. Prosodic features can also aid the acoustic model on a low level. For example,

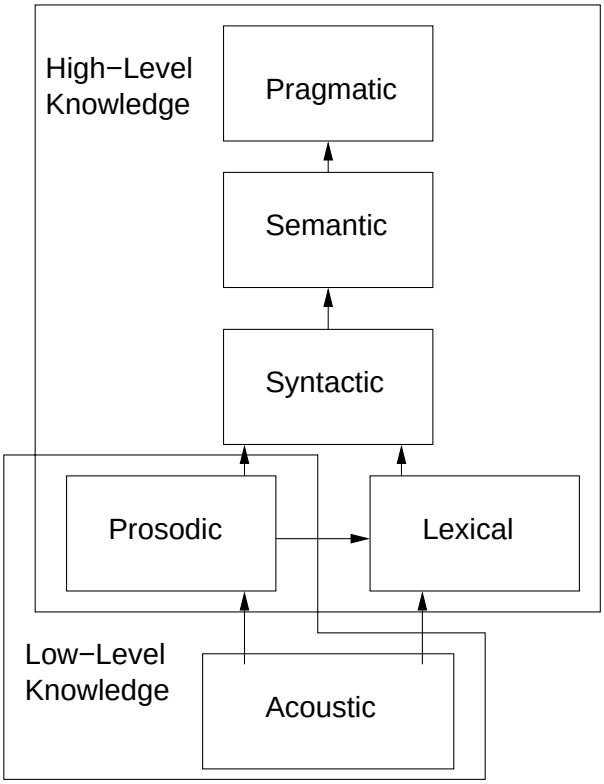

Figure 2.1: Hierarchy of knowledge precedence between spoken language sources

homographs and homophones (*e.g.*, *per*mit vs. per*mit*) in English can be disambiguated by pitch accent.

Speech parsing for understanding is syntactically more difficult than text parsing for understanding. One reason is that punctuation is not supplied, so phrasal boundaries are not delineated[1]. Prosody can be used to detect phrasal and sentential boundaries, sometimes to a greater extent than punctuation does. Semantic and pragmatic analysis can be aided by use of accentuation (through prosodic phrasing and pitch range reset). For example, information on focus (Beckman, 1997) can disambiguate scope ambiguities (*e.g.*, "[old men] and women are holding a protest." vs. "[old men and women] are holding a protest."). Dialogue analysis can be aided by prosodic information in the classification of dialogue acts (*e.g.*, "fifteen." vs. "fifteen?".)

Despite the great potential, there are also a variety of difficulties that arise when prosody is added to an ASR system (Kompe, 1997). For example, pitch accent can have at least two meanings — as emphasis or indicating a question. Different speakers have been found to realize prosodic events through different means. Prosodic information may be redundant if semantic and syntactic analysis can disambiguate an utterance, leaving the benefit of additional information debatable. One negative result of multiple prosodic realizations of events is that it is difficult (and thus-far impossible) to define a functional mapping between prosodic boundaries and syntactic boundaries.

ASR systems use different levels of knowledge, depending on their goals, and, as we will see in coming sections, are sometimes restricted by their coupling paradigm. We have explored the use of prosody in detail. Pragmatics have also played a role in speech recognition (*e.g.*, Levin *et al.*, 1995; Polzin and Waibel, 1998; Swerts *et al.*, 2000). Specifically, dialogue context as a pragmatic information source has been shown to aid dynamic generation of expectations of what a user is likely to say (Young *et al.*, 1989;

[1]This is only the beginning of the problem — word boundaries are also unknown. In addition, the signal can be very noisy.

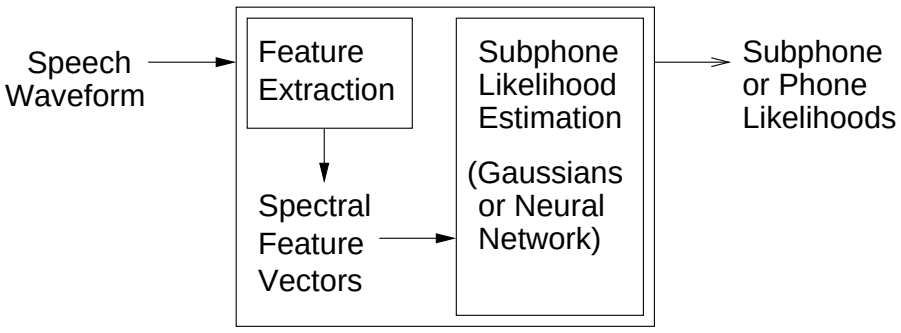

Figure 2.2: Acoustic pre-processing of waveform speech data.

Roy *et al.*, 2000). Use of acoustic, lexical, syntactic, and semantic levels of knowledge will be explored in the remainder of this chapter.

Each of the coupling schemes described in the following sections includes an acoustic pre-processing stage, shown in figure 2.2. This stage includes signal processing — extracting acoustic features such as pitch, duration, and intensity from digitally encoded speech. We also group phone likelihood estimation in this stage. Gaussian models and neural networks are two standard models for computing these phone (or sub-phone) probabilities from the spectral vectors extracted in signal processing. Our discussions of language model coupling methods will assume this pre-processing has occurred. The interested reader can find further detail about signal processing, phone-likelihood estimation, and general ASR in (Jurafsky and Martin, 2000).

2.2 Basic Language Modelling

The traditional view of ASR is to formulate the task as automatic transcription from speech to text. This basic task underlies the extraction of other high-level knowledge as well, so we will use it to develop our definition of a language model. Chelba (2000) gives a full description of language modelling. Following in this section is a summary of that work.

Given some vector of acoustic features $A = (a_1, a_2, \ldots, a_n)$ extracted by acoustic pre-processing, the task of speech recognition is to find the most likely word sequence $W = (w_1, w_2, \ldots, w_n)$ which would produce A. Note that we simplify by assuming a 1:1 correspondence between acoustic features and words — in reality, the acoustic features are usually generated in 10ms intervals and must be combined into groups corresponding to words. This process of finding word-boundaries is called segmentation and is a challenging problem in itself (Jurafsky and Martin, 2000). The most successful approach to the problem of finding W given A has been the Bayesian statistical approach introduced by (Bahl *et al.*, 1983). In this formulation, the probability of $P(W|A)$ is maximized, yielding sentence hypothesis $\hat{W}$:

$$\hat{W} = \underset{W \in \mathcal{L}}{\operatorname{argmax}} P(W|A) = \underset{W \in \mathcal{L}}{\operatorname{argmax}} P(A|W) \cdot P(W) \tag{2.1}$$

The acoustic pre-processing step (including extracting the features) models the *acoustic probabilities* $P(A|W)$. The task of the language model is then to calculate *prior probabilities* of word sequences $P(W)$. An additional non-trivial task is how to search the space of W word sequences in language $\mathcal{L}$ that could give rise to A, to find the best hypothesis $\hat{W}$. The search space for some string W of length n is $|\mathcal{V}|^n$ where $\mathcal{V}$ is the vocabulary. Practical considerations usually require restriction of this vocabulary to some fixed word set (in tens of thousands). Words not in the vocabulary are then mapped to one or more classes of *unknown* words. Probabilities are often estimated from a training corpus. The corpus can also be used to build the vocabulary.

The language model is defined by its parameterization θ of the source that generates the language, where $\theta \in \Theta$, Θ is the *parameter space*. This parameterization, θ, is also referred to as the *modelling assumption* about the language source.

The task of language modelling then becomes the selection of a parameterization of the source. Generally, a source model should operate in a sequential left-to-right manner, to allow for efficient searching as word hypotheses become available from the acoustic model. This, as we will see, is not a requirement for successful language modelling,

but rather a requirement for tight and incremental coupling with the acoustic model, as defined below. Left to right computation of probabilities $P(W)$ can be formulated as a conditional probability chain $P(w_1, w_2, \ldots, w_n) = P(w_1) \cdot \prod_{i=2}^{n} P(w_i|w_1 \ldots w_{i-1})$.

Parameterization is then restricted to models which compute prefix probabilities, the probability of some word w_i given the history $(w_1 \ldots w_{i-1})$:

$$P_\theta(w_i|w_1 \ldots w_{i-1}), \theta \in \Theta, w_i \in \mathcal{V} \tag{2.2}$$

The most successful models of this type are N-gram models, which condition words on a restricted history, making a Markov assumption of order N about the language source:

$$P_\theta(w_i|w_{i-1} \ldots w_1) \approx P_n(w_i|w_{i-1} \ldots w_{i-n+1}) \tag{2.3}$$

Note that, following Chappelier *et al.* (1999), we refer to all probabilistic models applied to speech recognition as *language models*, although they may not strictly fit the definition given above. We use *high-level language model* to refer to language models using linguistic knowledge above the level used by N-gram models. When we refer to the strict definition of language model, adhering to equation (2.2), we will use *word-predictive language model.*

2.3 Expanding the Measures of Success

Bourlard *et al.* (1996) states that the focus on continual small incremental improvements in WER and the resulting avoidance of techniques which may result in increased WER are traps which can result in a lack of innovation in ASR. They suggest many creative ways to improve ASR which may result in increased WER during development — including incorporating high-level syntactic knowledge. It is important to have new measures of success, and not to abandon early work when increased WER is the result. First, we should examine the generally accepted metrics — WER and perplexity.

Given the task of simply generating a transcription of speech, WER is a useful and direct way to measure language model quality for ASR. WER is the count of incorrect words in hypothesis $\hat{W}$ per word in the true string W. For measurement, we must assume a priori knowledge of W and the best *alignment* of the reference and hypothesis strings[2]. Errors are categorized as insertions, deletions, or substitutions.

$$\text{Word Error Rate} = 100\frac{\text{Insertions} + \text{Substitutions} + \text{Deletions}}{\text{Total Words in Correct Transcript}} \tag{2.4}$$

An example of WER calculation, taken from the HUB-1 corpus used in this work:

```
REF:     he  said  THE  YEN  could  RISE   due   to  the  REALIGNMENT
HYP:     he  said  AND       could  ARISE  due   to  the  REALLY       MEANT
ERRORS:           S    D           S                      S            I
5 errors per 10 words in transcription:  WER=50%
```

It is important to note that most models — Mangu *et al.* (2000) is an innovative exception — minimize *sentence error*. Sentence error rate is the percentage of sentences for which the proposed utterance has at least one error. Thus models (such as ours) which optimize prediction of test sentences W_t, generated by the source, minimize the sentence error. Thus even though WER is useful practically, it is formally not the appropriate measure for the commonly used language models. Unfortunately, as a practical measure, sentence error rate is not as useful — it is not as fine-grained as WER.

Perplexity is another measure of language model quality, measurable independent of ASR performance (Jelinek, 1997). The perplexity of the language modelling task is, roughly, the average number of choices at any decision point. The perplexity is minimized when the true source model is known and exactly modelled. For a given perplexity, recognition accuracy can be improved by improving the acoustic model or by incorporating additional high-level information into the recognition process. Perplexity is related to the entropy of the source model which the language model attempts to

[2]SCLITE (`http://www.nist.gov/speech/tools/`) by NIST is the most commonly used alignment tool.

estimate. For an N-word utterance, perplexity of the model can be measured:

$$\mathrm{PPL(M)} = \exp\left(-1/\mathrm{N}\sum_{\mathrm{i=1}}^{\mathrm{N}} \ln[\mathrm{P_M}(\mathrm{w_i}|\mathrm{w_1}\ldots\mathrm{w_{i-1}})]\right) \tag{2.5}$$

These measures, while informative, do not capture success of extraction of high-level information from speech. Task-specific measures should be used in tandem with extensional measures such as perplexity and WER. For example, for a speech recognition system with the goal of personalized interaction, we should measure speaker identity recognition success, which may be based on word-choice, prosodic cues, syntactic structure, acoustic features, etc. For a dialogue system, measures of success on a pragmatic level would be needed. These measures are closely linked to recognition because the search for the true utterance can be directed by the type of information sought, resulting in a simultaneous search, such as the search for the best parse tree and utterance carried out in this work.

Roark (2002), when reviewing parsing for speech recognition, discusses a modelling trade-off between producing parse trees and producing strings. Most models are evaluated either with measures of success for parsing or for word recognition, but rarely both. Parsing models are difficult to implement as word-predictive language models due to their complexity, so perplexity is not easy to measure. Traditional (*i.e.*, N-gram) language models do not produce parse trees, so parsing metrics are not useful. However, Roark (2002) argues for using parsing metrics, such as labelled precision and recall[3], along with WER, for parsing applications in ASR. Weighted WER is also a useful measurement, as the most often ill-recognized words are short, closed-class words, which are not as important to speech understanding as phrasal head words. For example, Weber *et al.* (1997) uses unification grammars to extract meaning by parsing word lattices in the context of speech-to-speech translation in VERBMOBIL. In this case, weighted word error, focusing on content words, is an appropriate measure.

[3]Parse trees are commonly scored with the PARSEVAL set of metrics (Black *et al.*, 1991).

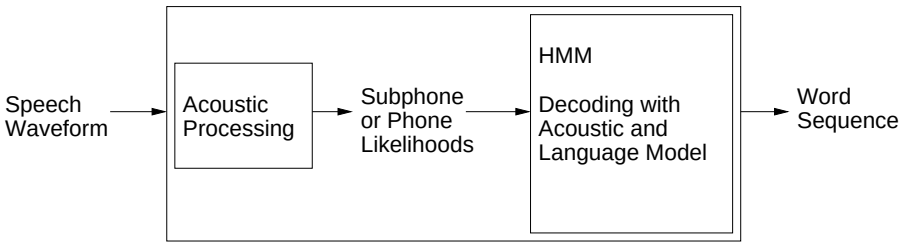

Figure 2.3: Tight coupling of acoustic and language model.

Successful extraction of high-level information has been shown to be important in real-world applications of speech recognition. The TINA system (Seneff, 1992), a widely-used syntactic and semantic analyzer, also acts as a high-level language model to rescore N-best lists for speech recognition (see section 2.6.3 for an explanation of N-best list rescoring). The system uses a hand-written probabilistic context free grammar with feature unification, trained on example sentences. It produces parse trees, annotated with semantic information, such as thematic role assignment. The relationships in the parse trees have been used in dialogue systems of the VOYAGER project to answer questions and complete tasks in restricted domains, such as booking an airline flight or checking the weather. Clearly, for such applications, the ability to gather enough semantic information to correctly carry out the task is more important than simple WER.

For real-world applications, time and space efficiency are also important measurements of success.

2.4 Tight Coupling

A tightly coupled system is one which integrates all the sources of knowledge in a highly interdependent set of processes which cannot be decoupled. Figure 2.3 shows a general schematic of a tightly coupled ASR system. Language model probabilities are integrated with phone likelihoods in decoding of hidden markov models (HMM). Other methods, such as phone-level parsing have been abandoned in favour of HMM. Decoding algorithms

such as the Viterbi algorithm are used for the simplest language models, such as the lexical bigram model — the most common form of language model used in tight coupling. Trigram probabilities have been integrated with the acoustic probabilities using A* search (Jelinek, 1969).

The main challenge for tight coupling is effectively integrating knowledge sources. There have been several attempts, with limited success — the resulting systems are usually orders of magnitude slower and produce sub-optimal results. Moore *et al.* (1989), applies tight coupling using a unification-based CFG to generate word transition probabilities for a Viterbi decoder. Goddeau (1992) uses a probabilistic LR parser to generate word transition probabilities for HMM decoding. Zue *et al.* (1991) introduces a method to generate bigram probabilities from a higher-language model by using the language model to generate random sentences from which the bigram probabilities are trained. More recently, Jurafsky *et al.* (1995) extends that model to general stochastic context free grammar (SCFG)s by extending the top-down Earley parsing algorithm to compute word transition probabilities. That work uses the SCFG to smooth the bigram grammar and add structural constraints. The bigram probabilities are generated from the SCFG by computing the *characteristic N-gram* in closed form by the method of Stolcke and Segal (1994). The characteristic N-gram method calculates expected bigram counts for strings generated by nonterminals in the grammar by solving a system of linear equations derived from the SCFG rule probabilities. The smoothed bigram model is then used as the language model coupled with the acoustic model in the HMM decoding process. A 5% absolute improvement in WER is achieved with the SCFG-smoothed bigram over the standard bigram. Direct combination of the dynamic programming computations from the Earley parser and Viterbi decoder are computationally infeasible, given the cubic time of the parser, and an input length equal to the number of 10ms frames. It is generally agreed that incorporation of acoustic/phonetic probabilities with high-level syntactic and semantic constraints is computationally intractable (Harper *et al.*, 1994).

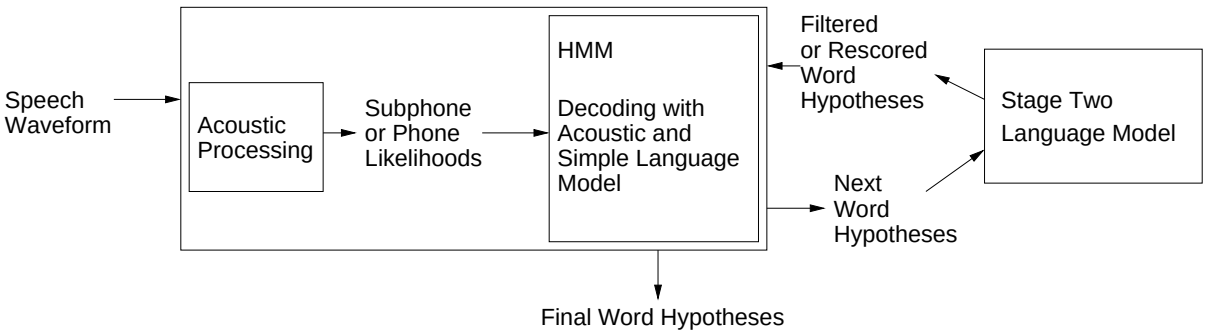

Figure 2.4: Incremental coupling of basic ASR with an additional language model.

Even if it were possible, it would be challenging to scale up the system to realistic tasks.

2.5 Incremental Coupling

Incremental coupling is defined by the use of time-synchronous feedback from the language model to the acoustic model, as shown in figure 2.4. The components cannot be procedurally separated; the coupling requires that component models work in a bidirectional cooperative fashion, filtering the set of hypotheses at each time step. Unlike tight coupling, there are two or more distinct language models or components, rather than compiling the higher level language models directly into the HMM decoding process. It differs from sequential coupling by allowing two way communication — the acoustic model generally sends next word hypotheses to the language model, which generates word transition probabilities based on the prefix sequences already proposed. Components of a sequential system are procedurally separate, and communication is one-way. Chappelier *et al.* (1999) includes incremental coupling within tight coupling due to its time-synchronous nature, but we believe there is a fundamental difference deserving of separate consideration.

Incremental coupling of LR parsing with a HMM has been reported (*e.g.*, Lavie and Tomita, 1993; Kita *et al.*, 1989) — the LR parser predicts phones, which are verified

by the HMM. Wachsmuth *et al.* (1998) proposes an extension of LR parsing with a robust integration of a (non-probabilistic) context free grammar with statistical acoustic modelling. Violations of the grammar rules are handled in a flexible way; penalty scores are assigned by the parser to ungrammatical constructions during the word-verification phase. Earlier work used binary grammar decisions, completely eliminating ungrammatical constructs from consideration. The technique of Wachsmuth *et al.* (1998) is useful for domains where parser training corpora are not available, and statistical parsing cannot be used. A tree of parse stacks is used to track the progress of recognition and parsing.

Weber (1994) reports incremental coupling with a pseudo-probabilistic typed unification grammar, which can operate in predictive mode (supplying next word hypotheses to the acoustic model), or verification mode (calculating word transition probabilities for hypotheses generated by the HMM).

The SCFG which was used in pseudo-tight coupling by Jurafsky *et al.* (1995) to smooth bigram probabilities was also applied to incremental coupling with the Viterbi decoder. The SCFG provided approximate probabilities (true transition probabilities could not be calculated given only a prefix). The parser probabilities were then used to filter next word hypotheses of the decoder. The resulting system showed the same performance as the SCFG-smoothed bigram used in tight coupling — 5% absolute decrease in WER over the standard bigram model, with a 36% increase in computation time.

2.6 Sequential Coupling

Sequential coupling, as shown in figure 2.5, consists of one way communication between the acoustic model and one or more language models which successively filter hypotheses to arrive at a final proposal for an utterance, parse tree, or representation from some other level of knowledge (see figure 2.1).

This form of coupling has been the most successful in terms of computational effi-

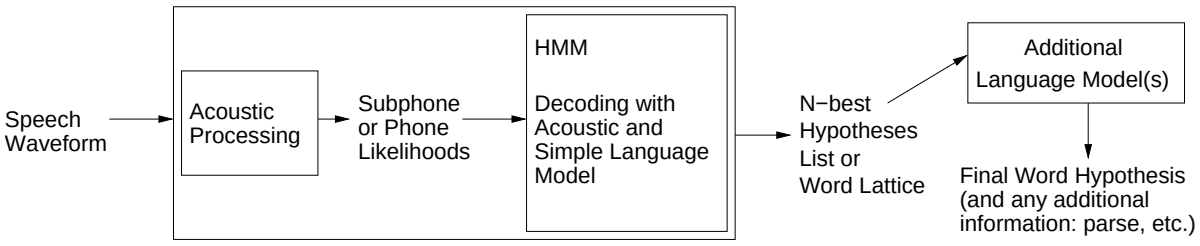

Figure 2.5: Sequential coupling of basic ASR with additional language model(s).

ciency, WER reduction over standard N-gram models, and ability to apply high-level knowledge to the task of ASR. Sequential coupling does have several challenges not present in other forms of coupling. The models in the stream must overgenerate, providing ample hypotheses for the next models to choose from. Otherwise, one risks bias from earlier models influencing decisions downstream. Overgeneration is also important in order to reduce risk of eliminating the correct string, parse tree, or other form of information early in the sequence. Measures such as the intermediate oracle word error rate[4] can be used to evaluate each of the stages separately.

Two major variations of this coupling scheme have been developed — N-best list rescoring and parsing, and lattice rescoring and parsing. Both use high-level language models operating at the word level. They differ in the format of the data transferred between stages. These variations are described in following subsections. First, we will define word lattices, and explore the significance of and ways to select N for N-best lists and lattices used in sequential coupling.

2.6.1 Word Lattices

The output from any level of an ASR system can be formed into a word-lattice. Ortmanns and Ney (1997) describes an algorithm constructing word lattices for large vocabulary,

[4]The WER of the hypothesis which best matches the true utterance, *i.e.*, the lowest WER possible given the hypotheses set.

continuous speech recognition. A word lattice is simply a compact representation of a large number of full utterance hypotheses. A standard lattice format (SLF), developed and documented by the HTK project (`http://htk.eng.cam.ac.uk`), has been generally adopted. A word lattice $L = (V, E)$ in SLF format is a directed acyclic graph (DAG) where:

V The set of vertices, or nodes. Vertices are defined by a timestamp, measured from the unique start time of the lattice, and labelled with a word.

E The set of labelled, weighted edges, representing the word utterances. A word w is hypothesized over edge e if e ends at a vertex v labelled w. The word starts at the time associated with vertex $v_{e_{start}}$ and ends at the time associated with vertex $v_{e_{end}}$. Edges are associated with transition probabilities. In SLF, multiple scores (not necessarily strict probabilities) can be associated with each edge:

acoustic score is the score assigned by the acoustic model, such as by a series of HMMs in the acoustic pre-processing step. The score is an estimate of $\mathcal{P}(a_i|w_i)$, the conditional probability of the acoustic observation a_i for the time frame of the current word, given word hypothesis w_i.

language model (prior) score is the score assigned by a language model, using one of the several forms of coupling. The lattices of the HUB-1 corpus, for example, are annotated with trigram scores trained using a 20 thousand word vocabulary and 40 million word training sample (see section 5.3).

Each path through a word lattice represents a string hypothesis. A simplified example of a lattice is shown in figure 2.6[5]. As discussed in section 2.6.2, word lattices can include many paths resulting in the same string hypothesis. This occurs when a word boundary is unclear. For example, if the boundary between `really` and `meant` in figure 2.6 was

[5]The actual lattice on which this example is based contains 36,814 paths.

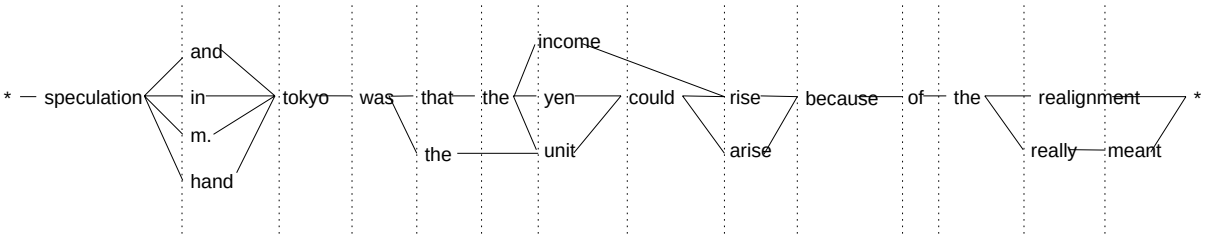

Figure 2.6: A simplified word lattice. Dotted lines represent time alignments.

unclear, many vertices for `really` (a vertex is labelled with a time and the word ending at that vertex) would be added to the lattice (one for each possible end-time for the word), resulting in many edge pairs producing `really meant` over the same total time span. Hall and Johnson (2003) minimizes the impact of this by converting a SLF lattice to a finite state machine (FSM) and minimizing the FSM to have only one path for each unique string.

Chelba (2000) reports WER reduction by rescoring word lattices with scores of a structured language model (Chelba and Jelinek, 2000), interpolated with trigram scores. Word predictions of the structured language model are conditioned on the two previous phrasal heads not yet contained in a bigger constituent. This is a computationally intensive process, as the dependencies considered can be of arbitrarily long distances. All possible sentence prefixes are considered at each extension step. Roark (2002) directly rescores word lattices using a parser that makes a Markov assumption akin to the assumption of N-gram models. Lattice reentrancies (points at which divergent paths rejoin) are used to determine Markov order. Once a reentrant point is passed, the parse prefixes up to that point are fixed. The Markov restriction allows rescoring of the entire acoustic lattice to be computationally feasible. Lattice rescoring improves the posterior probabilities of edges in the lattice. By rescoring the lattice, no information is eliminated, and the entire lattice can be used as input to another system. For example, modern multi-pass recognition systems use lattices to adapt to individual speakers.

Extracting a single best guess of the true utterance which created the lattice is more common than lattice rescoring. Bottom-up chart parsing, through various forms of extensions to the CKY algorithm, has been applied to word lattices for speech recognition (*e.g.*, Hall and Johnson, 2003; Chappelier and Rajman, 1998; Chelba and Jelinek, 2000). Full acoustic and N-best lattices filtered by trigram scores have been parsed. Hall and Johnson (2003) uses a best-first probabilistic context free grammar (PCFG) to parse the input lattice, pruning to a set of *local trees* (candidate partial parse trees), which are then passed to a version of the parser of Charniak (2001) for more refined parsing. WER for parsing the full acoustic lattices is higher than that for the N-best lattices, even though oracle WER is higher for N-best lattices. This suggests the trigram model, through pruning, is contributing to the WER improvement. However, unlike (Roark, 2001; Chelba, 2000), Hall and Johnson (2003) achieves improvement in WER over the trigram model without interpolating its lattice parser probabilities directly with trigram probabilities.

Weber *et al.* (1997) presents bottom-up parsing using a pseudo-probabilistic HPSG unification grammar with features including speech phenomena. The result is unacceptably slow on a full acoustic lattice, so a stage-one approximation of the grammar is used to create an N-best lattice to pass to the full grammar in stage two. Acoustic phenomena are also considered in the parser of Zhou and Lua (1999), a PCFG for Mandarin Chinese word lattices which include tonal features.

Lattice parsing (with the exception of Weber *et al.* (1997) and Roark (2001)) has focused on using parsers to reduce word error rates, with little attention to the quality of the parse result. Xu *et al.* (2002), using N-best list rescoring, shows a positive correlation between parse accuracy and improved WER. Brill *et al.* (1998) shows that humans use language structure to perform recognition. We have also discussed (see section 2.3) the need to develop new measures of success for speech recognition, and new approaches, that — at least in their initial development — may result in increased WER. Motivated

by this, in the following chapters, we discuss application of the current best lexicalized parsing model to the task of parsing word lattices in a sequential coupling system.

2.6.2 The Incomparability of N

Acoustic vs. N-best Lattices

Many of the works cited in this section (*e.g.*, Hall and Johnson, 2003; Roark, 2001; Chelba and Jelinek, 2000) compare rescoring of an *N-best* lattice of hypotheses to results produced from rescoring an *acoustic* word lattice. It is important to understand the distinction. An acoustic lattice is the union of all hypothesized strings produced by the acoustic model, for example, by decoding the HMM in figure 2.5. An N-best lattice is the union of N paths through the acoustic lattice. However, the lattices often differ by more than scale. If scale were the only factor, then an equivalent to the N-best lattice could be created by tightening the beam search of the HMM decoder. Commonly, additional knowledge (*e.g.*, trigram model scores) is used to prune the acoustic lattice to its N-best paths.

Whether or not the high-level language model applied to the N-best lattice explicitly uses the trigram probabilities, it is still informed by them. Improvements due to the trigram filtering cannot be decoupled from the usefulness of the high-level model. Comparisons are often made between results of a high-level model applied to an acoustic lattice and the same model applied to an N-best lattice (Hall and Johnson, 2003). This comparison is not informative about the contribution of high-level model alone, since the N-best lattice is informed by the trigram model used to prune. We propose a better comparison to isolate the contribution of the high-level model — the WER of the trigram model applied to the acoustic lattice compared with the high-level model applied to the acoustic lattice. Unfortunately, application of the high-level model to the full acoustic lattice is often computationally intractable. Therefore, we too compare the best path

through the N-best lattice, using the same trigram probabilities used for pruning to N-best, with the high-level language model applied to the N-best lattice. This yields the improvement achieved by using both models, over the trigram alone.

Selection of N

The selection of N is highly variable across work in this area. Often, one measures the oracle WER for various N, and chooses some N which gives a balance between the size of the lattice passed to the high-level model, and the risk of eliminating the true hypothesis during N-best path selection. However, this process is difficult, even for the small HUB-1 corpus used in this work. Word lattices from different speakers, with different levels of background noise, and with different words, will differ greatly in their packing. Ambiguous words (especially long words), poor conditions, or unclear speech can generate a highly packed lattice. Such a lattice has many parallel edges and orders of magnitude more paths than a lattice generated from a short, clear utterance. For example, if a word boundary is unclear in a speech signal, the resulting word lattice may contain hundreds of hypotheses, each differing only by milliseconds in the time of the word-boundary, in addition to many paths containing different word hypotheses. It is the case for the HUB-1 lattices that the paths differing by the time of one word-boundary often have similar total probabilities (the product of the acoustic and trigram scores of each edge). Figure 2.7 (A) illustrates how it is possible that for fixed N, we could get an N-best lattice or list with only one surface string realization, even if the $(N+1)$-st path is of similar probability and has a different surface string realization. Conversely, figure 2.7 (B) shows an example for which one could unnecessarily burden a high-level language model with the union of N paths if a small number of paths are strongly preferred by the trigram and acoustic models. As fixed N results in different numbers of string hypotheses in a lattice (depending on acoustic lattice structure), choosing a fixed N is certain to provide different oracle WER for each sample. One can solve this problem by attempting

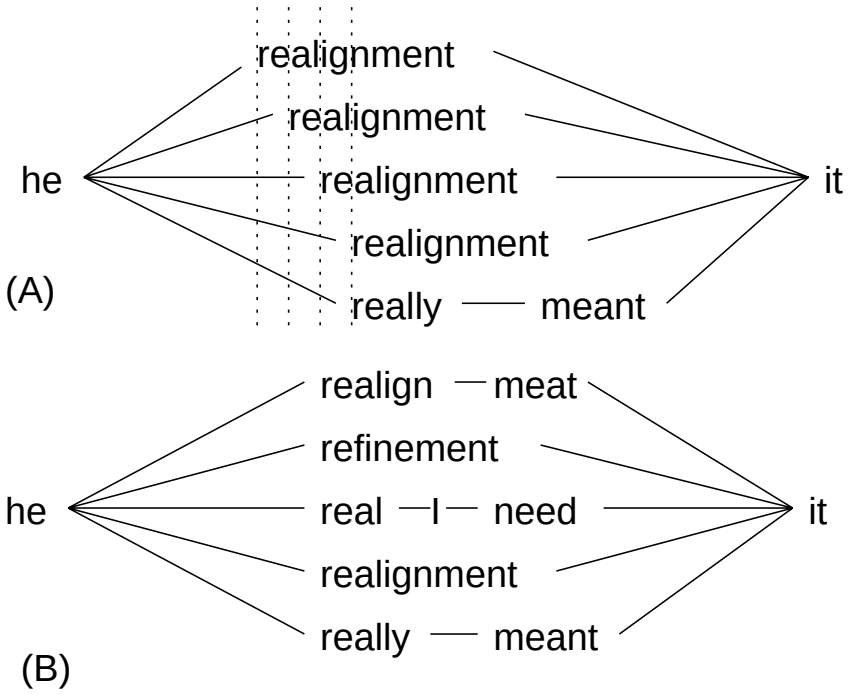

Figure 2.7: Problems with fixed N for N-best lattices: (A) This lattice has an ambiguous time for the *realignment* word-boundary. We assume all paths in the lattice have very similar total probability, and that the correct path, *really meant*, has the lowest total probability. An N-best lattice with $N \leq 4$ would contain only one string hypothesis, eliminating the correct hypothesis from consideration by a higher language model. (B) If the paths in this lattice have widely differing total probabilities, with the correct *really meant* path receiving the highest probability, a fixed N-best lattice with $N \geq 1$ would unnecessarily burden a high-level language model with multiple unlikely hypotheses. A variable N, based on a beam-search, could set N=5 for (A) and N=1 for (B).

to select some N large enough to reduce the average N-best oracle WER to within some factor of the oracle WER of the acoustic model. This would be difficult to select, and would likely result in lattices larger than could be processed by most high-level language models. Dynamic selection of N, such as choosing the union of all paths with a total score (the product of acoustic and trigram scores along each edge) within some beam of the best, would provide a better balance of computational burden and bias of the trigram model. We leave this as a consideration for future creation of lattice corpora.

The lattice corpus used in this work was previously pruned to the union of the 50-best paths (Roark, 2001) using the A* search of Chelba (2000). We parse on these lattices, for comparability, and because using the full acoustic lattices is currently too large for parsing by our system. Reimplementing the A* search to create dynamic size N best lattices is beyond the scope of this project.

2.6.3 N-Best Lists

High-level language models have been applied to speech recognition through N-best list rescoring[6]. The N complete utterance hypotheses with the best total score are passed from earlier stages to later stages. An N-best list can be thought of as an "unpacked" N-best word lattice; each element of the N-best list is a path through the corresponding lattice. The total score for a complete utterance hypothesis, or path through the word lattice, is the product of the probabilities along the edges.

For very long utterances, N will have to be very large to achieve an acceptably high probability of the true utterance being within N. Ravishankar (1997) uses articulation points (points of high confidence in a word hypothesis) to extract N-best segmented lists from word lattices for very long utterances. N-best hypotheses are extracted for each segment. Segments meet at articulation points, creating a packed representation of the N-best lists, similar to a lattice.

[6]We will consider rescoring to include rescoring by parsing.

Rayner *et al.* (1994) combines several knowledge sources, including N-gram scores, binary features related to grammaticality, and scores based on relationship types between phrasal headwords (similar to relationships used in our work) as a basis for rescoring. They separate the contribution of each knowledge source to the improvement in WER; headword relationships coupled with acoustic scores are found to be as useful as N-grams, however the differences found may not be significant, given the small size of the test corpus used.

There can be an arbitrary number of rescoring steps, each using a different kind of information or a different model. In most systems, there are only two stages — the N-best hypotheses from HMM decoding are rescored by a higher-level language model. The benefit of this process over word lattice rescoring is that the language model need not be strictly left-right (*i.e.*, it does not have to operate on sentence prefixes). Word lattice rescoring can operate on prefixes (partial lattices), *i.e.*, rescoring can begin before the end of the utterance is processed by the acoustic model. This requires adaptation of the language model to operate on-line (as edges become available) and left-right. N-best list rescoring need not operate on-line or left-right, as the N-best list is not generated until the utterance has been completely processed by the acoustic model. Once the entire word lattice is available, the N paths are extracted. Therefore, complete (*i.e.*, spanning the time of the utterance) hypotheses become available to the parsing model in a single step. Thus high-level parsing models, such as the probabilistic lexicalized parsers of (Charniak, 2001; Collins, 1999) can be directly applied to N-best lists, and the sentence (and parse tree) with the highest score is selected. Scores generated by this process can also be interpolated with scores assigned by earlier stages. Xu *et al.* (2002) shows a reduction in WER through N-best list rescoring with the structured language model (Chelba and Jelinek, 2000) interpolated with a trigram model.

Roussel and Halber (1997) applies additional filtering only to ill-recognized words, based on confidence scores assigned by analysing agreement between members of N-best

lists. This is a novel way to restrict the choice of the parsing model, based on the assumption that parsing (even lexicalized parsing) can accept an incorrect hypothesis which happens to be grammatically well-formed, and that more control should lie with the acoustic model.

Brill *et al.* (1998) presents an application of N-best list rescoring using humans as the rescoring system. Participants are presented with N-best lists and asked to choose the item most likely to be the true utterance. In addition, data are gathered about the kinds of knowledge used by the human subjects when making decisions. Closed class word choice is the most effective way humans make decisions about plausible hypotheses, given the N-best lists (covering several genres) in that study. This is not surprising, given that closed-class words are often short, and provide very little acoustic evidence for an ASR system. Other proficiencies humans use to make decisions for N-best list rescoring include number agreement, complete sentence vs. incomplete sentence, topic, world knowledge, and predicate-argument/semantic agreement. The human subjects were able to achieve 17–59% relative improvement in WER over scoring with combined trigram and acoustic models. This work is significant in that it shows recognition can be improved with high-level knowledge — however, we may have to solve the general artificial intelligence problem to get there.

Chapter 3

Review of the Probabilistic Model

3.1 Syntactic Structure in Natural Language

Syntactic structure in natural language is a continually debated topic in linguistics. However, there has been some reasonable agreement on a structural representation within the computational linguistics community — the parse tree. Furthermore, a standard format of parse tree has been adopted — the Penn Treebank format. Although the Penn Treebank format is widely used, it has several shortcomings. For example, the trees are rather flat (*e.g.*, dependence is often not specified in noun-noun compounds). The format standard derives from the Penn Treebank (Taylor *et al.*, 2003), a corpus consisting of sentences for which parse trees were manually created, using a set of guidelines. There are some inconsistencies and errors in the Treebank corpus, including structural inconsistencies and part-of-speech (POS) tagging errors (Ratnaparkhi, 1996). For parsers trained on the Treebank, these inconsistencies can be problematic. However, statistical systems, such as the one used in this work, are usually robust. Each kind of Treebank error is usually a unique occurrence, contributing a low probability to our model.

Statistical parsers use a treebank to train a model, which is then applied to test sentences, with the goal of automatically producing trees that fit the guidelines. An

example of a Penn Treebank tree is shown in figure 3.1. Each word in the tree bears a POS tag, *e.g.*, `years` is annotated as a plural noun, `NNS`. Word / POS pairs are combined using parentheses into groupings, called *parse tree constituents*. For example, `(NP (CD 55) (NNS years))` is a *noun phrase* containing a cardinal number and a plural noun. Some non-terminals, such as `NP-SBJ-1`, are labelled with additional *features* — in this case the role assignment `SBJ` indicating the `NP` is the subject. Parse trees may also contain *empty constituents*, such as `(-NONE *-1)` in the example. These are *traces* — placeholders for the corresponding numbered constituent (`NP-SBJ-1` in this case). We will eliminate features and empty constituents in a pre-processing step, as they are not considered by the parsing model II of Collins (1999), which we implement for word lattice parsing.

```
( (S
    (NP-SBJ-1
      (NP (NNP Rudolph)(NNP Agnew) )
      (, ,)
      (UCP
        (ADJP
          (NP (CD 55)(NNS years) )
          (JJ old) )
        (CC and)
        (NP
          (NP (JJ former)(NN chairman) )
          (PP (IN of)
            (NP (NNP Consolidated)(NNP Gold)(NNP Fields)(NNP PLC) ))))
      (, ,) )
    (VP (VBD was)
      (VP (VBN named)
        (S
          (NP-SBJ (-NONE- *-1) )
          (NP-PRD
            (NP (DT a)(JJ nonexecutive)(NN director) )
            (PP (IN of)
              (NP (DT this)(JJ British)(JJ industrial)(NN group) ))))))
    (. .) ))
```

Figure 3.1: A Penn Treebank Parse Tree.

The trees of the Penn Treebank are not annotated with *headwords*. Headwords are roughly the most important or meaningful word in a constituent. For example, in figure 3.1, `(VBN named)` is the headword of the `VP` constituent which contains it. We annotate the non-terminal nodes of parse trees used for training with their headwords, following the deterministic rules of (Collins, 1999, appendix A), which are a modified version of the rules of (Magerman, 1994).

3.2 Parameterization

The parameterization of a parsing model is the choice of how to break down the tree into parts, and how to choose conditional probabilities to represent the choice. Three head-driven probabilistic parsing models are presented in (Collins, 1999), each of which includes POS tagging as part of the model. *Head-driven* refers to the significant role of headwords conditioning the generative process. The three models differ in the level of syntactic detail they employ. We choose model II of (Collins, 1999) as the parsing parameterization for the lattice parser of this work, and attempt a faithful re-implementation of that model, adapted to word-lattice parsing. Where strict adherence to the model of Collins (1999) is not possible, either because of differences inherent in the task of lattice parsing, or due to ambiguity in the original model, the differences will be noted.

The basic model represents a parse tree as a sequence of decisions based on a head-centred, top-down derivation of the tree. This model captures dependencies between pairs of headwords, incorporates POS tagging, lexical dependencies, influences of distance between head words and modifiers, and differences between non-recursive NPs and other NPs, using a set of parameters trained over a treebank corpus. Outside probabilities (the prior probability of a given head non-terminal, headword, and headtag) based on (Caraballo and Charniak, 1998) are also incorporated into the probability model. Model II extends the basic model to include parameters modelling a complement/adjunct dis-

tinction and subcategorization frames for headwords. Model III incorporates empty constituents (traces) and *wh*-movement — these are not considered in our parser. The specific parameters used in our parser are described in the remainder of this section. Parameters are estimated by linear interpolation of various levels of *back-off*, described fully in section 3.3. More detail on the motivation leading to the parameter selection, including examples, can be found in chapter 7 of (Collins, 1999). The remainder of this chapter, except where noted, is a summary of that work[1], as it pertains to our re-implementation of the model. Readers familiar with Collins (1999) may wish to skim ahead to chapter 4.

3.2.1 The Basic Model

The basic process of the parsing model is to break traditional PCFG rules into a set of smaller steps — the addition of modifiers to a head, and to model the probabilities of those dependency relationships. A typical PCFG rule has the form:

$$P(h) \rightarrow L_n(l_n) \ldots L_1(l_1) H(h) R_1(r_1) \ldots R_m(r_m) \tag{3.1}$$

where H is the head child of the left-hand-side constituent, and h is the associated headword and headtag[2]. $P(h)$ is the *parent* of the rule, with its headword and headtag. L and R are the left and right modifiers of the head, with their associated headwords and headtags. Left and right modifier sequences are extended with a `STOP` symbol.

The probability of the PCFG rule, given the left hand side, can be decomposed exactly,

[1] Thanks to Michael Collins for permission to repeat the pertinent details of the model and parameter estimation technique.

[2] The headword and POS tag of the headword (the headtag), are noted by the single specifier h, l, or r in this discussion. They will usually be decomposed to (ht, hw), (lt, lw), and (rt, rw) when considered in different levels of back-off context for parameter estimation.

using the chain rule for probabilities:

$$\begin{aligned}
&\mathcal{P}(L_n(l_n)\ldots L_1(l_1)H(h)R_1(r_1)\ldots R_m(r_m)|P(h)) = \\
&\mathcal{P}_h(H|P(h)) \times \\
&\prod_{i=1\ldots n+1} \mathcal{P}_l(L_i(l_i)|L_1(l_1)\ldots L_{i-1}(l_{i-1}), P(h), H) \times \\
&\prod_{j=1\ldots m+1} \mathcal{P}_r(R_j(r_j)|L_1(l_1)\ldots L_{n+1}(l_{n+1}), R_1(r_1)\ldots R_{j-1}(r_{j-1}), P(h), H) \quad (3.2)
\end{aligned}$$

Thus the top-down generation of parse trees has three phases per rule. First, generate the head child of the parent, with probability $\mathcal{P}_h(H|P,h)$. Second, generate modifiers to the left with probability $\prod_{i=1\ldots n+1}(\mathcal{P}_l(L_i(l_i)|P(h),H)$, where $L_{n+1}(l_{n+1}) = \texttt{STOP}$. Finally, generate modifiers to the right in an analogous way.

Note that conditioning modifier generation on all previously modifiers would be difficult, due to the sparsity of annotated training data. Collins (1999) simplifies the modifier probabilities by assuming independence:

$$\mathcal{P}_l(L_i(l_i)|L_1(l_1)\ldots L_{i-1}(l_{i-1}), P(h), H) = \mathcal{P}_l(L_i(l_i)|P(h), H) \quad (3.3)$$

$$\mathcal{P}_r(R_j(r_j)|L_1(l_1)\ldots L_{n+1}(l_{n+1}), R_1(r_1)\ldots R_{j-1}(r_{j-1}), P(h), H) = \mathcal{P}_r(R_j(r_j)|P(h), H) \quad (3.4)$$

3.2.2 Distance

The basic model assumes independence of all modifiers. In general, the modifier generation can depend on any function of the previous modifiers, head/parent category, and headword/headtag, making a *history-based parameterization* (Black *et al.*, 1992). The generation process is ordered — depth-first and outward from the head modifier, so when generating modifier $L_i(l_i)$, modifier $L_{i-1}(l_{i-1})$ has already been fully generated. So, we can also condition on any structure *below* the previous modifiers. Collins (1999) extends the basic model to a history-based model (model I), with the inclusion of a distance

measure:

$$\mathcal{P}_l(L_i(l_i)|P(h), H, L_1(l_1)\ldots L_{i-1}(l_{i-1})) = \mathcal{P}_l(L_i(l_i)|P(h), H, distance(i-1)) \tag{3.5}$$

$$\mathcal{P}_r(R_i(r_i)|P(h), H, R_1(r_1)\ldots R_{i-1}(r_{i-1})) = \mathcal{P}_r(R_i(r_i)|P(h), H, distance(i-1)) \tag{3.6}$$

The distance measure is a vector with two elements: (1) is the string between the head and modifier of zero length? (2) does the string between the head and modifier contain a verb? These questions allow the model to learn a preference for modification of the most recent verb. We decompose this vector into two fields in the parser implementation — the first becomes the distance measure, D, and the second becomes the boolean "contains a verb" flag, V.

3.2.3 Subcategorization

The assumption that modifiers are generated independently was proved to be a poor assumption by evaluation of the basic model (Collins, 1999). Although complements and adjuncts can be distinguished through a set of rules in a post-processing step, it was shown by Collins (1999) that making the distinction before parsing improved model performance. Collins (1999) extends model I to include the distinction between complements and adjuncts, using *subcategorization frames* — multisets[3] specifying left and right complements for each rule head constituent. This extension introduces dependence between complements, and we adopt it in the model used in this work.

The first addition to the basic model is annotation of complements with the "`-C`" suffix, creating new non-terminals. This is accomplished by adding a suffix to all non-terminals in the training data that satisfy a set of conditions specified by Collins (1999).

[3]A set which may contain duplicate elements.

Second, subcategorization frames, S, are gathered in training for all headwords. A subcategorization frame is an unordered multiset of the complement non-terminals modifying the head. These counts are used to train two new estimators for left and right subcategorization frames: $\mathcal{P}_{lc}(S|P(h), H)$ and $\mathcal{P}_{rc}(S|P(h), H)$. The conditioning context for $\mathcal{P}_l$ and $\mathcal{P}_r$ is expanded to include the subcategorization frame:

$$\mathcal{P}_l(L_i(l_i)|P(h), H, L_1(l_1)\ldots L_{i-1}(l_{i-1})) = \mathcal{P}_l(L_i(l_i)|P(h), H, distance(i-1), S) \tag{3.7}$$

$$\mathcal{P}_r(R_i(r_i)|P(h), H, R_1(r_1)\ldots R_{i-1}(r_{i-1})) = \mathcal{P}_r(R_i(r_i)|P(h), H, distance(i-1), S) \tag{3.8}$$

As each complement is generated, its non-terminal is removed from the subcategorization multiset. When adjuncts are generated, the subcategorization frame is unchanged. Note that the probability of generating the `STOP` symbol is 0 when the subcategorization frame is non-empty. Similarly, the probability of generating a complement is 0 when the subcategorization frame is empty.

In the implementation of the parser, left and right modifiers are added in separate steps. Therefore, our notation is simpler than that of Collins (1999) — a distinction between left and right subcategorization frames LC and RC is not made, and the subcategorization frame S is the left or right frame, depending on the type of edge (see section 4.4).

3.2.4 Non-Recursive NPs

A *non-recursive noun phrase* (referred to as a "baseNP", with non-terminal label `NPB`), is a `NP` that does not directly dominate another `NP`, unless that `NP` is a possessive `NP` (*i.e.*, it directly dominates the possessive tag, `POS`) (Collins, 1999). Some example baseNPs are:

- Pierre Vinken

- the Tuesday edition
- today's stock prices

Collins (1999) applies a special probabilistic treatment to baseNPs. They merit special consideration for several reasons. First, the boundaries of baseNPs are often strongly marked — the start points are often marked by a determiner or adjective. Thus, the probability of generating the `STOP` symbol should be greatly increased if the previous modifier is a determiner or adjective. The independence of modifiers does not capture this information. Second, the internal structure of baseNPs in the Penn Treebank is underspecified. Multi-noun compounds usually have no internal structure (*e.g.*, `NP` $\rightarrow$ `NN NN NN`). Therefore, there is no justification to condition on the "head" any more than on the previous modifier.

The model is augmented to generate baseNPs differently than other non-terminals. Training data are pre-processed to provide data for baseNP-specific parameters — `NPB` nodes are inserted into training parse trees where appropriate. All `NPB` nodes inserted into training parse trees are directly dominated by an `NP`. Equations (3.5) and (3.6) are modified for instances of $P =$ `NPB`, to include conditioning on the previous modifier:

$$\mathcal{P}_l(L_i(l_i)|H, P, h, L_1(l_1)\ldots L_{i-1}(l_{i-1})) = \mathcal{P}_{NPBl}(L_i(l_i)|L_{i-1}(l_{i-1})) \tag{3.9}$$

$$\mathcal{P}_r(R_i(r_i)|H, P, h, R_1(r_1)\ldots R_{i-1}(r_{i-1})) = \mathcal{P}_{NPBr}(R_i(r_i)|R_{i-1}(r_{i-1})) \tag{3.10}$$

As the modifier and previous modifier are always adjacent, the distance measure is omitted. The previous modifier is initialized to $H(h)$ (*i.e.*, $L_0(l_0) = H(h) = R_0(r_0)$).

3.2.5 Coordination

Coordination constructions normally have the form of `X` $\rightarrow$ `X CC X` in the Penn Treebank. This is evidence that generation of modifiers `X` and modifier `CC` are not independent

— in the vast majority of cases, exactly one modifier precedes and follows a conjunction. Collins (1999) thus extends the generative model to give appropriately low probability to highly unlikely constructions such as `NP` $\rightarrow$ `NP CC`. The generation of the coordinator is delayed until the next modifier is generated by setting a coordination flag (c). The flag is generated by the probability estimator $\mathcal{P}_{cc}(c|L, P, R; lt, rt; lw, rw)$. When no coordinator is generated `c = none` and $\mathcal{P}_{cc} = 1$.

Note that although the coordination relationship discussed above is the standard in the Penn Treebank, sentences which violate the `X CC X` pattern are common (*e.g.*, "And he said the stock prices would fall.", "Prices were high, but the mood was anything but."). Collins (1999) does not specify how to handle such cases — which would not be parsed by the model as outlined thus far. Such cases of non-standard coordination occur often-enough in the Penn Treebank to have a significant impact on performance. We suggest three alternatives to adapt the model:

1. Allow for generation of incomplete coordinations (*i.e.*, allow addition of a `STOP` modifier when $c \neq none$). Add the probability of generating incomplete coordination to the parameters: When a `STOP` is generated and the coordination flag is not `none`, add $\mathcal{P}_{cc}(c|STOP, P, R; STOP, rt; STOP, rw)$ or $\mathcal{P}_{cc}(c|L, P, STOP; lt, STOP; lw, STOP)$, depending on the direction. This technique requires little change to the statistical model, but allows generation of incomplete coordination at any point in the sentence.

2. Treat sentence-initial (and sentence-final, should it occur) coordination as a different POS — `CCSTART` (and `CCEND`, respectively). Treat these cases as regular modifiers, not coordination. This suggestion introduces a deterministic tagging which is based on the position of words in the sentence — a departure from the statistical nature of the parsing model. The advantage is that it restricts non-standard coordination to sentence-initial or final positions.

3. For all coordinations, generate two edges — one using the coordination flag and another using coordination as a regular modifier. The model will decide which to use, based on the scores assigned by the parameters. This technique was suggested by Michael Collins (p.c.), but may defeat the purpose of generating the coordination flag, as it removes the restriction on coordination by allowing any form of relationship to compete with the standard specified by the Treebank.

The performance of these three techniques was evaluated with the parser described in section 4.6 using the development test set. The first technique was the most successful (the others admitted non-standard coordinations anywhere in the sentence) and is used in experiments reported in chapter 5.

3.2.6 Punctuation

Although punctuation does not occur in speech lattices, in order to test our implementation of the model in comparison to the implementation of Collins (1999) (using test sections of the Penn Treebank) we implement the punctuation-handling of the model, as described in detail in Collins (1999). We consider only punctuations tagged as comma or colon — others are removed from training data.

First, punctuation in the training data is raised as required to ensure that all punctuation appears between two non-terminals (figure 3.2). Second, punctuation is treated in a similar manner to coordination, with generation of a punctuation flag (p), a variable indicating punctuation to be generated with the next modifier. The flag is generated by the probability estimator $\mathcal{P}_p(p|L, P, R; lt, rt; lw, rw)$. If no punctuation is present, the flag is `none` and $\mathcal{P}_p = 1$ is omitted. Sentence-initial and sentence-final punctuation are handled in the same way as coordination.

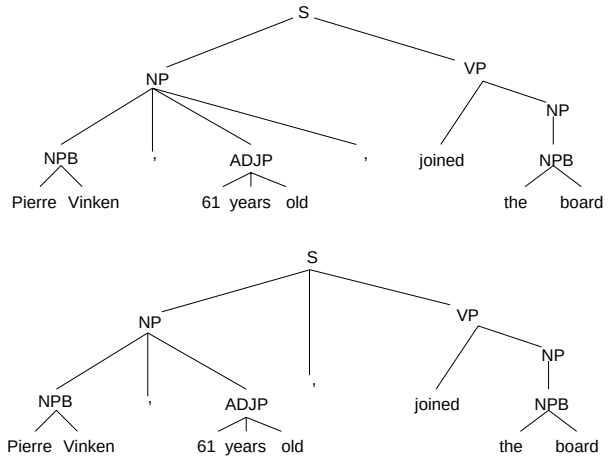

Figure 3.2: A parse tree before and after punctuation raising. Reprinted, with permission, from Collins (1999).

3.2.7 Empty (PRO) Subjects

Sentences in the Penn Treebank frequently have an empty (PRO) subject, which may or may not be controlled. For example, in the sentence "Selling stocks is profitable", *selling* is analyzed as having an empty subject. As sentences with and without empty subjects are labelled `S` in the Penn Treebank, the probability for an empty left subcategorization frame, $\mathcal{P}_{lc}(\{\}|\texttt{S}, \texttt{VP}, verb)$ will be rather high. Sentences with and without subjects appear in different syntactic environments, *i.e.*, they do not have the same parent distribution. For these reasons, Collins (1999) introduces a pre-processing modification to training data, relabelling all sentences with empty subjects as `SG`. This allows for a clear division of subcategorization, depending on the type of sentence generated: $\mathcal{P}_{lc}(\{\}|\texttt{S}, \texttt{VP}, verb) \approx 0$, while $\mathcal{P}_{lc}(\{\}|\texttt{SG}, \texttt{VP}, verb) = 1$.

3.2.8 Inside and Outside Probabilities

The *inside probability* — the probability derived using the internal structure of a parse tree — is an insufficient measure of the probability of that subtree because it takes no account of the prior probability of seeing a constituent with a given non-terminal

label, headword, and headtag (Caraballo and Charniak, 1998; Goodman, 1997). For the Collins (1999) models, the inside probability for a parse can be considered the product of the probabilities invoked during tree generation, using the parameters outlined in the previous sections. For example, for the rule `NP(NNP,Vinken)` $\rightarrow$ `NBP(NNP,Vinken) ,(,) ADJP(JJ,old)`, the inside probability is:

$$\begin{aligned}
&\mathcal{P}_h(\texttt{NPB}|\texttt{NP},\texttt{NNP},\texttt{Vinken}) \times \\
&\mathcal{P}_{lc}(\{\}|\texttt{NP},\texttt{NBP},\texttt{NNP},\texttt{Vinken}) \times \\
&\mathcal{P}_{rc}(\{\}|\texttt{NP},\texttt{NBP},\texttt{NNP},\texttt{Vinken}) \\
&\mathcal{P}_l(\texttt{STOP}|\texttt{NP},\texttt{NPB},\texttt{NNP},\texttt{Vinken}) \times \\
&\mathcal{P}_r(\texttt{ADJP}(\texttt{JJ},\texttt{old}),\texttt{c}=\texttt{none},\texttt{p}=,|\texttt{NP},\texttt{NPB},\texttt{NNP},\texttt{Vinken}) \times \\
&\mathcal{P}_r(\texttt{STOP}|\texttt{NP},\texttt{NPB},\texttt{NNP},\texttt{Vinken}) \times \\
&\mathcal{P}_p(,|\texttt{NBP},\texttt{NP},\texttt{ADJP},\texttt{NNP},\texttt{Vinken},\texttt{JJ},\texttt{old})
\end{aligned} \tag{3.11}$$

The lack of considering of the prior probability of P, h can result in constituents receiving a high inside probability, given some unlikely P, h. For example, if some VP headed by a preposition `of` is generated, the subsequent subtree may have a high *conditional* probability $\mathcal{P}_{inside}(subtree|\texttt{VP},\texttt{IN},\texttt{of})$.

The score for a parse can be improved by combining this with an additional *outside* (or *prior*) probability, yielding a combined score ("figure of merit" in Caraballo and Charniak (1998)):

$$\text{Score} = \mathcal{P}_{inside}(subtree|P,h) \times \mathcal{P}_{outside}(P,h) \tag{3.12}$$

Caraballo and Charniak (1998) describe a complicated method of determining the prior probability, but the simpler method of Goodman (1997) is adapted by Collins (1999) for the parsing model we use:

$$\mathcal{P}_{outside}(P,h) = \mathcal{P}_{outside1}(ht,hw) \times \mathcal{P}_{outside2}(P|ht;hw) \tag{3.13}$$

Recall that h is the headword, headtag pair. The counts are gathered over all training events (unary and binary dependencies).

At each step in the iterative parsing process, we have a partial parse tree with an inside and outside probability. We extend the tree, adding the probability of the additional internal tree structure to the inside probability, then recalculating the outside probability to get the score for the new tree.

3.3 Parameter Estimation

In order to obtain the best estimates of probabilities in the face of sparse annotated training data, a *back-off* strategy is used by Collins (1999), which we adopt and summarize here.

Maximum likelihood estimates are calculated given a *conditioning context*. The conditioning context is a set of values for various context features, such as distance (D), parent (P), previous modifier, etc. The conditioning context is expressed as X, a vector of $X_1, X_2, \ldots, X_n$. X is a member of the set of possible contexts $\mathcal{X} = \mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n$. The maximum likelihood estimate for some set of outcomes Y is:

$$\hat{\mathcal{P}}_{ML}(Y|X) = \frac{Count(Y, X)}{Count(X)} \tag{3.14}$$

A back-off strategy is a way to smooth probability estimates and ensure they are based on a reasonable number of samples. In a high dimensional parameter space (large n), $Count(X)$ may be very low or 0, leaving 3.14 undefined or inaccurate. To compensate, Collins (1999) introduces a linear interpolation over maximum likelihood estimates using subsets $\Phi_i(X)$ of X, where $\Phi_i(X)$ is the ith back-off context level.

The parameters of the model are summarized in table 3.1, along with the context considered at each level of back-off (see section 3.3.2). The general three-level back-off strategy is:

Level 1: full context, including parent categories, modifier tags, and modifier words

Probability	Context Variables (Levels of Back-off)
$\mathcal{P}_h(H\|\ldots)$	P, ht, hw
	P, ht
	P
$\mathcal{P}_{lc}(S\|\ldots)$	P, H, ht, hw
	P, H, ht
	P, H
$\mathcal{P}_{l1}(L_i(lt_i), c, p\|\ldots)$	S, D, P, H, ht, hw
	S, D, P, H, ht
	S, D, P, H
$\mathcal{P}_{l2}(lw_i\|\ldots)$	$lt_i, L_i, c, p, S, D, P, H, ht, hw$
	$lt_i, L_i, c, p, S, D, P, H, ht$
	lt_i
$\mathcal{P}_{ls}(\texttt{STOP}\|\ldots)$	D, P, H, ht, hw
	D, P, H, ht
	D, P, H
$\mathcal{P}_{NPBl1}(L_i(lt_i), c, p\|\ldots)$	$L_{i-1}, lt_{i-1}, lw_{i-1}$
	L_{i-1}, lt_{i-1}
	L_{i-1}
$\mathcal{P}_{NPBl2}(lw_i\|\ldots)$	$lt_i, L_i, c, p, L_{i-1}, lt_{i-1}, lw_{i-1}$
	$lt_i, L_i, c, p, L_{i-1}, lt_{i-1}$
	lt_i
$\mathcal{P}_{NPBls}(\texttt{STOP}\|\ldots)$	$L_{i-1}, lt_{i-1}, lw_{i-1}$
	L_{i-1}, lt_{i-1}
	L_{i-1}
$\mathcal{P}_p(p\|\ldots)$ $\mathcal{P}_{cc}(c\|\ldots)$	L, P, R, lt, rt, lw, rw
	L, P, R, lt, rt
	L, P, R
$\mathcal{P}_{outside1}(ht, hw)$	
$\mathcal{P}_{outside2}(P\|\ldots)$	ht, hw
	ht

Table 3.1: The model parameters and conditioning variables for each level of back-off.

Level 2: eliminate modifier words from the level 1 context

Level 3: eliminate modifier tags from the level 2 context

Notable exceptions are the second modifier parameters $\mathcal{P}_{l2}$, $\mathcal{P}_{r2}$, and their NPB equivalents, described in the next section. These parameters contribute the POS tagging probability to the model, thus the final level of back-off context contains the POS tag instead of the parent non-terminal.

3.3.1 Modifier Parameters

Note that the left and right modifier probabilities, $\mathcal{P}_l$, $\mathcal{P}_r$, $\mathcal{P}_{NPBl}$, and $\mathcal{P}_{NPBr}$ are divided into 2 parts, which are smoothed separately. For example:

$$\begin{aligned}
&\mathcal{P}_l(L_i(lt_i, lw_i), c, p|S, D, P, H; ht; hw) = \\
&\mathcal{P}_{l1}(L_i(lt_i), c, p|S, D, P, H; ht; hw) \times \\
&\mathcal{P}_{l2}(lw_i|lt_i; L_i, c, p, S, D, P, H, ht; hw)
\end{aligned} \tag{3.15}$$

3.3.2 Smoothing and Back-off

The probability estimate is calculated by a linear combination of maximum likelihood (ML) estimates at each level of back-off. The interpolation method was chosen by Collins (1999) for its ability to balance the effects of two competing types of error — bias and variance. Briefly, the variance (sampling error) is higher for estimates based on a low number of samples. The bias is roughly the difference between a parameter estimate using a restricted context and the "true" maximum likelihood estimate considering the full context. The bias is therefore 0 for $\hat{\mathcal{P}}_{ML}(Y|X)$. However, the variance is possibly high for this ML estimate, as the number of samples matching the full context X may be small. One can reduce the variance by increasing the number of samples contributing to the ML estimate. This is accomplished by decreasing the size of the conditioning context through different back-off levels $\Phi_i(X)$. This, in turn, results in increased bias —

the trade-off between the two types of error. The linear interpolation method of Collins (1999) (an adaptation of Jelinek (1990)) was found to minimize both types of errors. For some $\mathcal{P}(Y|X)$, and a back-off context $\Phi_i(X) \subset X$, Collins (1999) calculates the ith level estimate:

$$\hat{\mathcal{P}}_i(Y|X) = \hat{\mathcal{P}}_{ML}(Y|\Phi_i(X)) = \frac{Count(Y, \Phi_i(X))}{Count(\Phi_i(X))} \tag{3.16}$$

where *Count* is a function which counts the number of occurrences of its argument, in the training data. There are two constraints on how the context subset function Φ_i can be chosen:

1. $\Phi_1(X)$ must be sufficiently small (general) such that $\hat{\mathcal{P}}_1(Y|X)$ is defined for all contexts $X \in \mathcal{X}$ (*i.e.*, $\forall X \in \mathcal{X}, Count(\Phi_1(X)) > 0$)

2. if $i < j, \Phi_i(X) \subset \Phi_j(X)$ (*i.e.*, greater index, more specific context)

The levels of back-off are combined through linear interpolation, using a weight λ_i for each level. The ith level smoothed estimate $\tilde{\mathcal{P}}_i$ is defined recursively:

$$\begin{aligned} \tilde{\mathcal{P}}_1 &= \hat{\mathcal{P}}_1 \\ \tilde{\mathcal{P}}_i &= \lambda_i \hat{\mathcal{P}}_i + (1 - \lambda_i)\tilde{\mathcal{P}}_{i-1}, 1 < i \leq n \end{aligned} \tag{3.17}$$

Each λ_i must take a value from 0 to 1 for each $\tilde{\mathcal{P}}_i$ to define a distribution over the outcome-space $\mathcal{Y}$. The value of λ_i is calculated based on the number of samples contributing to the parameter estimate ($f_i = Count(\Phi_i(X))$) and the number of unique outcomes (values of Y which occur once), u_i, in the distribution $\Phi_i(X)$) at the ith level of context. Intuitively, additional weight should be given to probabilities based on many samples. However, if for some sample set, most outcome occurrences are unique (*i.e.*, the number of unique samples (u_i) is similar to the number of total samples (f_i)), the estimate will have greater variance. In this case, $\lambda(i)$ should be lower. The form for λ_i

used in this work satisfies these conditions[4]:

$$\begin{aligned} \lambda_i &= 0 \qquad \text{If } Count(\Phi_i(X)) = 0 \\ \lambda_i &= \frac{f_i}{f_i + 5u_i} \qquad \text{If } Count(\Phi_i(X)) > 0 \end{aligned} \tag{3.18}$$

Given three levels of back-off, as we have for most of the parameters in table 3.1, the recursion for estimation can be expanded:

$$e = \lambda_1 e_1 + (1 - \lambda_1)(\lambda_2 e_2 + (1 - \lambda_2)e_3) \tag{3.19}$$

We can see that if there are a large number of (non-unique) samples contributing to the probability estimate at the full (most specific) level of context, e_1, the value of $\lambda_1 \approx 1$, and the other levels of back-off contribute little to the final estimate. On the other hand, if there are relatively few samples contributing to e_1, or if there are a high fraction of unique outcomes for the context $\Phi_i(X)$, $\lambda_1 \approx 0$, and the other back-off levels contribute more. The back-off levels of context in table 3.1 conform with the constraints on the context subset function outlined above.

[4]The constant 5 was optimized by Collins (1999) on a development test set.

Chapter 4

Head-Driven Parsing for Speech Recognition

As we have discussed in section 2.6.1, parsing models based on headword dependency relationships have been reported, such as the structured language model of Chelba and Jelinek (2000). These models use much less conditioning information than the parsing models of Collins (1999), and do not provide Penn Treebank format parse trees as output. In this chapter we explain adaptation of the probability model outlined in chapter 3 for left-to-right operation as a language model. The lattice parsing algorithm, based on framework source code supplied by Bob Carpenter, is also presented. The intended action of the parser is illustrated in figure 4.1, which shows parse trees built directly upon a word lattice.

4.1 Parsing as a Predictive Language Model

The models of Collins (1999) are not only parsing models, but also assign probabilities to strings in the language. The model employed in this work assigns a probability $P(W,T)$ to a word sequence W and parse tree T such that $\sum_{\forall W,\forall T} P(W,T) = 1$. In the implementation of (Collins, 1999), the model is applied in chart parsing for complete strings. We

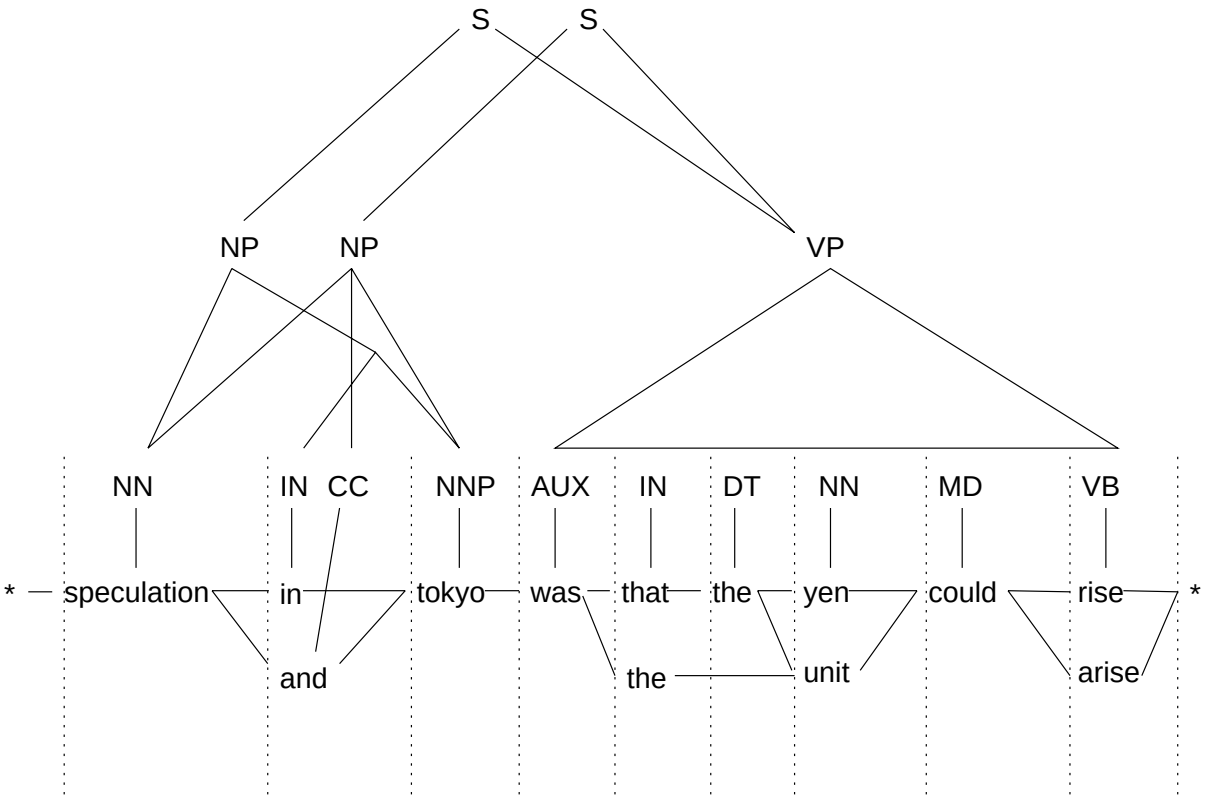

Figure 4.1: Example of a partially-parsed word lattice. Different paths through the lattice are simultaneously parsed. The example shows two final parses, one of low probability (S^*) and one of high probability (S).

adapt the model to left-to-right parsing, which completely parses each prefix of a string or path through a lattice. We approximate the search for the best next word w_k, given some word and parse prefix W_{k-1}, T_{k-1} by searching for the best word and parse extension W_k, T_k using the parameters of the model. We do this for all parse prefixes W_{k-1}, T_{k-1} in the chart (*i.e.*, those not eliminated by dynamic programming constraints, described in section 4.7). The model does not have an explicit formulation of the standard language modelling probability $P(w_k|W_{k-1})$.

4.2 Penn Treebank – Binary Mapping

As shown in figure 4.2, Penn Treebank format trees are not strictly binary branching. Although we can gather counts for our parameter estimators directly from Penn Treebank format trees with any branching, the parsing algorithm we use creates trees with only unary or binary branching at each node. This allows us to iteratively add one modifier at at time, and restricts the grammar size to unary and binary rules. The result is the same as if we had not forced unary/binary structure — the trees produced by our algorithm can be mapped to a Penn Treebank format tree, with any branching pattern. Note that *nodes* in a parse tree as shown in figure 4.2 correspond to *edges* in the chart parser.

Briefly, parse trees are produced by the parsing model using unary extension operations, which serve to change the parsing "mode", and binary join operations, which add modifiers. The order in which modifiers are added to a headword is fixed — first left modifiers then right modifiers. Unary extend operations are used to signal the end of addition of left modifiers and the beginning of addition of right modifiers. For example, in figure 4.2, we can see modifiers to the left are added to the `LEFT_NPB(NPB,apples)` edge, followed by unary extension to `RIGHT_NPB`, to which only modifiers to the right can be added. When addition of modifiers is complete, unary extension to a `STOP` edge takes place. To convert trees created this way to the Penn Treebank format, intermediate

(non STOP) nodes are omitted, and the STOP nodes (marked by boxes in figure 4.2) are connected. This process is explained in greater detail in the remainder of this section. The basic element of chart parsing is the *edge*. An edge consists of:

- a type (*e.g.*, STOP)
- a non-terminal label (*e.g.*, NP)
- a category (a set of properties used to calculated the parsing score)
- an associated probabilities (*e.g.*, acoustic model score, parsing score)
- links to its child edges

An edge may have one, two, or no children, depending on its type. The types and categories of edges will be outlined in detail in section 4.4. The parsing operations are controlled by the edge type. The probabilities assigned to edges are a function of the contents of the category and non-terminal label. The lattice parser operates with a simple deterministic grammar, based on three groupings of edge types — EXTEND, FINAL, and WORD. The parser grammar defines an internal representation of parse trees with, at most, two children per edge:

1. FINAL → EXTEND[RIGHT] | WORD
2. EXTEND[RIGHT] → EXTEND[LEFT]
3. EXTEND[LEFT] → FINAL
4. EXTEND[RIGHT] → EXTEND[RIGHT] FINAL
5. EXTEND[LEFT] → FINAL EXTEND[LEFT]

Productions of grammar rules 1-3 are referred to as unary extend operations. Productions of grammar rules 4-5 are called binary join operations. Each edge is composed of a series of related data fields, including the grammatical label corresponding to the non-terminal label in the Penn Treebank format tree. We refer to the Penn Treebank non-terminal as the *non-terminal* (*e.g.*, NP, VP), whereas we refer to the non-terminals of the simple parser

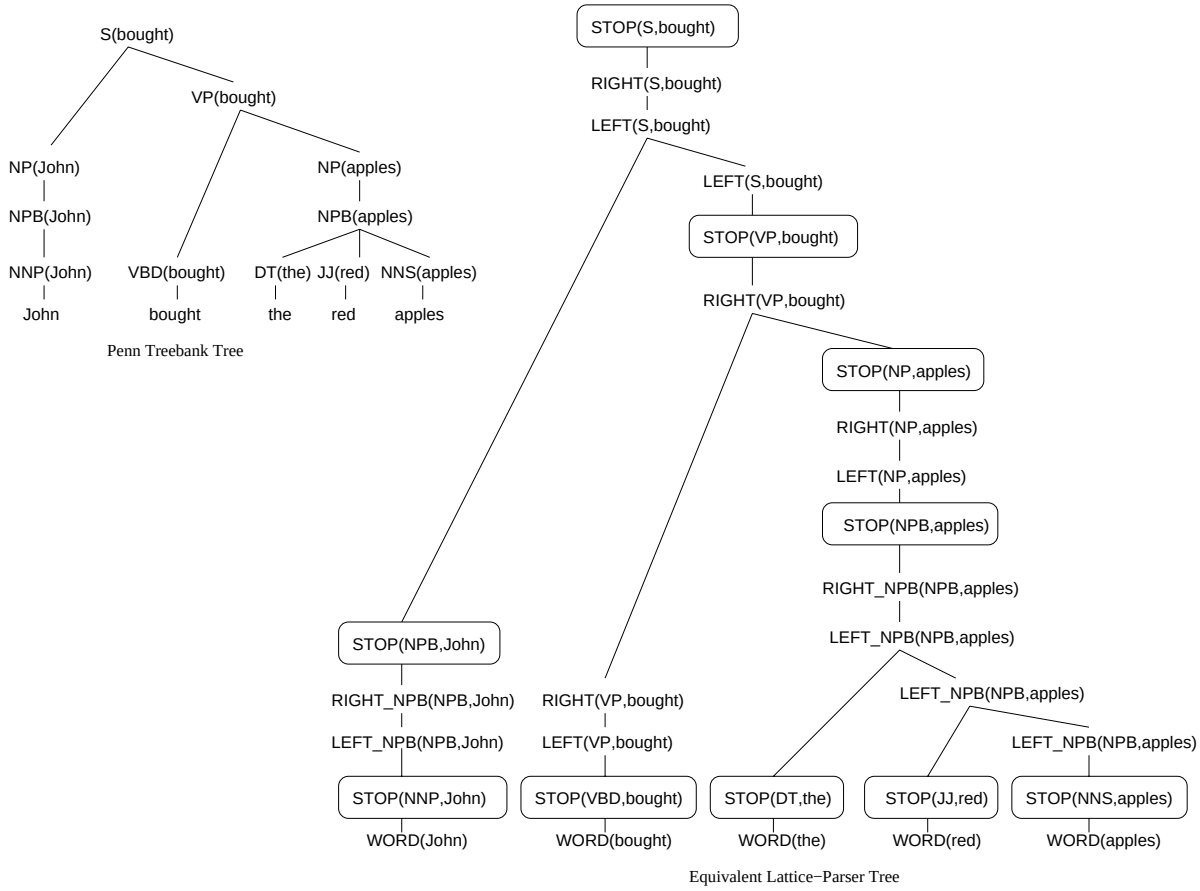

Figure 4.2: Example equivalence of Penn Treebank and parser tree formats. Labels of edge types in the `FINAL` group are enclosed in boxes. These are the edges collected on output to make a Penn Treebank format tree. The headwords are shown in parentheses for both formats. For the parser format, the edges are labelled by their edge type, and grammar non-terminal labels are shown in parentheses with the headword.

grammar given above as the *edge groups*, (*e.g.*, FINAL). The *edge types* are the different kinds of edges which form the edge groups (*e.g.*, STOP is an edge type member of the FINAL edge group). Thus LEFT[VBD]→STOP[NP]LEFT[VBD] is an instantiation of rule 5 with non-terminals and edge types given in place of edge groups. It is important not to confuse the parser grammar for edge operations with the "grammar" implicitly defined by the probabilities in the parameterization of the language model. The language model defines its grammar by allowing for any non-terminal, headword construction which has a non-zero probability given the parameters.

Trees created by the lattice parser contain three groupings of edge types, described in more detail in following sections. Briefly, the groups are:

- FINAL (edges labelled STOP, COORD, and PUNCT): equivalent to nodes in a Penn Treebank format parse tree; have exactly one child
- EXTEND (edges labelled LEFT, RIGHT, LEFT_NPB, and RIGHT_NPB): intermediate nodes inserted to create a parse tree with maximum two children per edge; further divided into two subgroups, identified with LEFT and RIGHT features in the parser grammar
- WORD (edges labelled WORD): the lexical edges which form the leaves of the tree; can be seen as a special type of FINAL edge, which cannot be extended

By creating edge groups, parse trees formed by our parser assume a structure with maximum two children per edge. This allows us to assign the calculation of the parameters of chapter 3 to individual parse operations, depending on the edge types involved (see section A.2).

Edges from the EXTEND group do not correspond to nodes in a Penn Treebank format parse tree. To convert a parse tree in the format of our parser to the Penn Treebank format, we search the tree from our parser depth-first, linking FINAL and WORD edges, bypassing intervening EXTEND edges. That is, for some FINAL edge e, all FINAL and WORD edges below it (either directly connected or through a sequence of EXTEND edges)

are collected and become direct descendants of e in the Penn Treebank format. Tree equivalence is shown by the example in figure 4.2. For the join operations (`EXTEND[LEFT]` → `FINAL EXTEND[LEFT]` | `EXTEND[RIGHT]` → `EXTEND[RIGHT] FINAL`), the headword of the child `EXTEND` edge is percolated up to the new `EXTEND` edge. For unary extension operations, the headword of the child is percolated up. Headwords of `WORD` edges are simply the word they dominate.

Note that we include the `NPB` (base-NP) edges in both forms of tree, to show their use in internal representations of trees — the calculation of edge scores depends on use of NPB specific parameters. We insert `NPB` edges in the Penn Treebank format trees used in training, but they are not part of the standard Penn Treebank format. Therefore, these intermediate edges are eliminated in final trees for evaluation against the original Penn Treebank trees.

4.3 Part-of-speech Tagging

The parsing model used in this work, described in chapter 3, has the selection of POS tags for words built in, thus does not require a separate POS tagger. Recall that a POS tagger assigns some part-of-speech to an instance of a word, *e.g.*, `NNS` for plural nouns. The Penn Treebank uses a 45-tag set. We do not consider punctuation except comma and semicolon, so our tag set consists of 40 parts of speech. In order to operate as a POS tagger, our parser must consider all POS possibilities for each word. That corresponds to up to 40 edge considerations for each word. POS conditioning in our model is implicitly included in the conditioning context for the various parameters, for example, the probability of adding a complement modifier to the left ($P_l(L_i(lw_i, lt_i), c, p|S, D, P, H; T; W)$) includes the probability of the modifier word and tag pair. So, pruning of unlikely word-tag pairs will occur during higher modifier-join operations, rather than at the word level. This leads to a geometric growth in the size of the chart, and increases the rate of chart

growth.

As we face shortages of time and memory resources running the lattice parser, we employ a pre-processing step to reduce the POS search space for each word edge considered by the parser. This module is referred to as the `TAGGER`. The tagging model was developed by Carpenter (2000), and is similar to a trigram tagger. Given the tag sequence (tag_1, tag_2, tag_3), where tag_3 is proposed for some word w:

$$P(w, tag_3|tag_2, tag_1) \approx P(tag_3|tag_2; tag_1) \cdot P(w|tag_3) \tag{4.1}$$

Note we estimate parameters using back-off and smoothing as described in section 3.3[1]. All paths through the lattice are considered — *i.e.*, all possible tag_3 for all previous tag pairs (tag_2, tag_1) on all paths entering a given `WORD` edge. Tag sequences are initialized with two `START` symbols, so the probabilities for initial words are well-defined. A search is carried out over the space of possible tags for each word to find the best tagging:

$$P_{best}(w, tag_3|tag_2, tag_1) = \max_{tag_3 \in \mathcal{T}, (tag_2, tag_1) \in tagSet(w_{-1}, w_{-2})} P(w, tag_3|tag_2, tag_1) \tag{4.2}$$

In order to provide flexibility for the more sophisticated tagging built into the Collins (1999) parsing model II, we retain more than just the 1-best tag sequence found by our tagger. For some beam β, word w and best tagging $P_{best}(w, tag_3|tag_2, tag_1)$, all those POS with $P(w, tag_3|tag_2, tag_1) \geq P_{best}(w, tag_3|tag_2, tag_1)/\beta$ are accepted and passed to the parser.

In preliminary experiments, tagging probabilities were passed to the parser module and added to the total edge score assigned by the parameters of the parsing model. This resulted in difficulties converging to a complete spanning parse. This was because the tagging probabilities varied widely in their values. Adding them to the score overwhelmed the selectivity of the parser scores. Rather than apply a scaling factor, we decided not to pass the tagging scores to the parser, because tag selection is already built into the parsing model. The tagger is used only to filter potential POS.

[1] As in earlier sections, semicolon separates levels of back-off.

4.4 *Edge* data type

There are eight edge types which form the three edge groups:

- **`FINAL` group**
  - `STOP` edges represent complete categories — all possible modifiers have been joined under the non-terminal.
  - `COORD` edges are a special case of `STOP` edges, headed by coordinating words (*i.e.*, `and`).
  - `PUNCT` edges are a special case of `STOP` edges, headed by punctuation (*i.e.*, `;`).
- **`EXTEND` group**
  - `LEFT` edges represent categories to which modifier edges (`STOP`, `COORD`, `PUNCT`) to the left may be added. (`EXTEND[LEFT]` group)
  - `LEFT_NPB` edges are the `NPB` equivalent of `LEFT` edges (`EXTEND[LEFT]` group)
  - `RIGHT` edges represent categories to which modifier edges (`STOP`, `COORD`, `PUNCT`) to the right can be added. (`EXTEND[RIGHT]` group)
  - `RIGHT_NPB` edges are the `NPB` equivalent of `RIGHT` edges. (`EXTEND[RIGHT]` group)
- **`WORD` group**
  - `WORD` edges are parse tree leaves containing the string yield of the tree. (`WORD` group)

In addition to its type, an edge is specified by several fields:

left node number (l): left end of left child

right node number (r): right end of right child

acoustic score (a): assigned by the acoustic processor, and included in the input lattice

lattice language model score (lm): assigned by stage one (usually trigram) language model, and included in the input lattice

inside score (s): edge score assigned by the parameters of the parsing model; the inside probability, *e.g.*, $\mathcal{P}_{ls} \cdot \mathcal{P}_{rc}$

weight (w): the weighted log sum of the log acoustic, lattice language model, and inside scores of all edges in the best partial parse (subtree) rooted at this edge, *i.e.*, $w = \alpha(\log a) + \beta(\log lm) + \log s$

total score (t): the weight plus the log outside probability for this edge, *i.e.*, $t = w + \log \mathcal{P}_{Pr}$

children (c_1, c_2): one child (c_1) for `FINAL` edges, one or two children for `EXTEND` edges, and no children ($c_1 = c_2 = null$) for `WORD` edges

category (cat): the properties of the edge as related to the Collins (1999) parsing model, such as headword, parent non-terminal, coordination flag, etc.

The edge category contains the information used to calculate edge parser scores using the parameters of the parsing model outlined in section 3.2. The various fields of the category are outlined in table 4.1.

4.5 Input Ordering

The parsing algorithm operates purely bottom-up — a phrase structure category is only considered if all children have been created. The algorithm also operates strictly left-to-right, evaluating lexical hypotheses online. This requires incoming edges from the word graph (lattice or string) to arrive in topological order. Thus we topologically sort the nodes in the input lattices. The topological ordering is a total ordering on the nodes that places the left node of an edge before the right node of an edge. This ordering of the nodes also results in edges being ordered such that an edge ending at $r = i$ will be

Category Property	Symbol	Definition
edge type	*type*	*e.g.*, `STOP`
non-terminal label	P	also called the rule parent, *e.g.*, `VP`
head child category	H	the parent of the head child, *e.g.*, `VP`
headword	W	the headword of the head child, *e.g.*, `bought`
headtag	T	the POS tag of the headword, *e.g.*,VBD
coordination flag	c	the coordination word, *e.g.*, *and*; set to *none* when not used
punctuation flag	p	the punctuation word (p_w), *e.g.*, (*;*); set to *none* when not used
distance	D	distance (as defined in section 3.2.2), from headword to the modifier being added, *e.g.*, between "bought" and "week"
contains verb flag	V	boolean indicator of whether this edge dominates a verb
# unary extensions	u	restricted to two consecutive extensions. A single unary extension is defined as a full (`STOP` $\rightarrow$ `RIGHT` $\rightarrow$ `LEFT` $\rightarrow$ `STOP`) sequence, *i.e.*, a unary extension in the Penn Treebank format tree
subcategorization	S	the listing of expected modifiers to the left for `LEFT` edges, and to the right for `RIGHT` edges
previous head	H_{-1}	the head child category of the previously added modifier[a], used for `NPB` edges
previous word	W_{-1}	the word associated with the head child category of the previously added modifier, used for `NPB` edges
previous tag	T_{-1}	the POS tag of the previous word, used for `NPB` edges.

[a]The *previously added* modifier is towards the head, not lexically previous. Therefore, it is the modifier to the right when extending `LEFT_NPB`, or to the left for `RIGHT_NPB`. For example, for the `NPB (NPB((DT the)(NN company)(NN policy))`, if `(NN policy)` is the head, then `(NN company)` is the *previously added* modifier H_{-1} when adding `(DT the)` with an `LEFT_NPB` edge.

Table 4.1: The fields of an edge category. Examples are for an edge connecting the left-modifier `NP(week,NN)` to the head `VP(bought,VBD)` using the rule `LEFT[VP(bought,VBD)]` $\rightarrow$ `STOP[NP(week,NN)] LEFT[VP(bought,VBD)]`

processed before edge ending at $r = j$ if $i < j$.

Bottom-up parsing also requires that for any edge of span (l, r), all edges of span $(l + i, r - j)$ (*i.e.*, any smaller edge within the span) have been processed. In a word lattice, word edges may have different lengths and overlap. Consider an edge for the word (*realignment*) with the same end points as an edge pair (*really,meant*). The time span for the edge (*meant*) is a subset of the span of (*realignment*). However, the edge (*meant*) is not dominated by (*realignment*), as they appear in different lattice paths. Therefore, the bottom up parsing restriction does not apply and the edges may be considered in any order.

4.6 Bottom-up Parsing

The algorithm described in this section — online, bottom-up, left-to-right lattice chart parsing — is based on preliminary work described in Carpenter (2000) and an incomplete code framework supplied by Bob Carpenter. The algorithm is a variation of probabilistic bottom-up Cocke-Kasami-Younger parsing similar to Chappelier and Rajman (1998).

Our parser produces trees (bottom-up) in a right-branching manner: starting with a proposed headword, left modifiers are added first using right-branching, then right modifiers using left-branching. For some grammar rule $Z \rightarrow Y_1 \ldots Y_h \ldots Y_n$, where Y_h is the head child, we get ordered intermediate steps, resulting in the tree shown in figure 4.3.

The basic algorithm for our system, including the `TAGGER` and `PARSER` modules, is summarized in figure 4.4.

4.6.1 Parser Initialization

Word lattices are input from standard lattice format (SLF) files or a single-path lattice can be formed from an input string. The lattice can be initially pruned to N-best paths

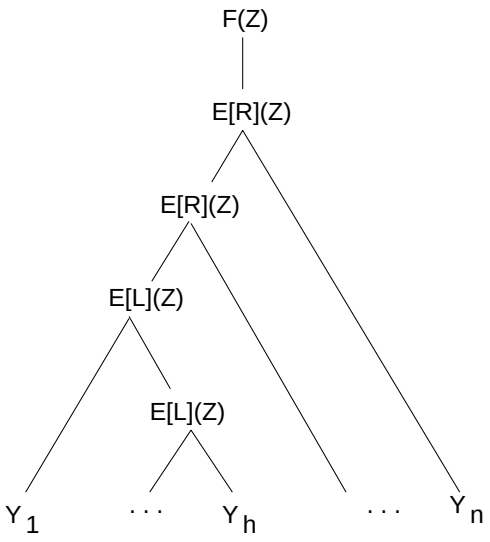

Figure 4.3: Edge join operations: sequence of steps to attach modifiers to head for grammar rule $Z \rightarrow Y_1 \ldots Y_h \ldots Y_n$. Nodes marked `E[L]` are `EXTEND[LEFT]` and `E[R]` are `EXTEND[RIGHT]`. Node `F(Z)` is a `FINAL` node, headed by non-terminal Z.

using the acoustic and language model (*i.e.*, trigram) scores stored in the file. After pre-processing, the lattice is topologically sorted, and edges are arranged in a data stream.

4.6.2 Agenda and Chart Initialization

The `AGENDA` is a collection of edges which have not been processed (*i.e.*, have not been fully extended and joined using closure). It is initialized as an empty collection of edges. The `AGENDA` is implemented as a priority queue which is constantly ordered (smallest-first) by edge span. Other implementations (such as a first-in, first-out (FIFO) queue and a priority queue ordered by score) were also used in testing (see chapter 5).

The `CHART` is a collection of edges which have been processed and form partial parse trees. It is initialized as an empty collection of edges and grows as `WORD` edges are parsed. The edges in the `CHART` are partitioned into nodes by their right endpoint. `CHART` nodes are further sorted by score (best first), for ease of comparison and pruning (see section 4.7).

```
// INITIALIZATION
input word lattice, prune to N-best
topologically sort edges, form into input stream

// ITERATIVE PARSE
// until iteration limit is reached or a complete parse is found,
// iteratively grow beam (reduce amount of pruning) and repeat parse
while ((parses.size == 0) && (baseBeam < beamLimit)) {
    clear chart;
    parses = tagAndParse(input stream);
    if (parses.size == 0)
        baseBeam = baseBeam + beamIncrement;
}

// TAG AND PARSE FUNCTION
edge[] tagAndParse(input stream, baseBeam) {
    while input stream has more edges {
        // TAGGER
        tags = tag(next WORD edge);
        // PARSER
        for each (tag in tags) {
            // new edge created using rules of table B.1
            if (isCoord(tag))
                newEdge = COORD(child=WORD,T=tag);
            else if (isPunct(tag))
                newEdge = PUNCT(child=WORD,T=tag);
            else
                newEdge = STOP(child=WORD,T=tag);
            initialize agenda with newEdge;
        }
        // binary join and unary extend using rules of appendix A
        // until agenda is empty
        chart closure(baseBeam);
    }
    return chart.parses();
}
```

Figure 4.4: Overview of lattice chart-parsing algorithm.

The AGENDA is empty before (and after) processing a WORD edge. The TAGGER receives an edge, and, using the model described in section 4.3, compiles a list of potential tags with probability within a beam of the best tagging. The tag list and WORD edge are used to initialize the AGENDA with new FINAL edges, as shown in figure 4.4. The properties of each type of extension of a WORD edge are outlined in table A.1.

4.6.3 Chart Closure

The closure operation is called after the agenda is initialized with a set of FINAL edges created from a WORD edge. Closure is carried out iteratively, for each edge popped from the AGENDA, until the AGENDA is empty.

There are two basic operations — unary extension and binary join — corresponding to the simple parser grammar rules described in section 4.2. For each operation, the appropriate log probabilities (the parameters of Collins (1999) Model II, as specified in section A.2) are added to the parser scores of the children, giving the parser score for the new edge.

A unary extension creates a new edge with the original edge as its child. The new edge has the same span as its child. For a binary join operation, two adjacent edges, edge1 and edge2 are combined under a new edge parent. Depending on the type of join (left or right), the headword of either edge1 or edge2 is passed to parent. Adjacency is defined by the WORD edges forming the leaves of edges in the chart.

Modifiers are added using the binary join operation. The direction of adjunction is governed by the edge group (EXTEND[LEFT] or EXTEND[RIGHT]). Unary extensions change the processing mode. For example, an EXTEND[LEFT] edge can join with FINAL edges in binary adjunction, adding modifiers to the left. By unary extension using the rule EXTEND[RIGHT] $\rightarrow$ EXTEND[LEFT], addition of left modifiers stops[2]. Now the

[2]That is, the new EXTEND[RIGHT] edge does not join with left modifiers. The original EXTEND[LEFT] edge remains active on the chart.

EXTEND[RIGHT] edge can join with FINAL edges in binary adjunction, adding modifiers to the right. Unary extension using the rule FINAL → RIGHT stops addition of modifiers under the current non-terminal label and completes an join/extend cycle[3].

The specific unary extension and binary join operations of the parser, as defined at edge group level by the grammar of section 4.2, are specified at edge type level, along with associated parameters, in the tables of section A.2.

Edges are sequentially popped off the AGENDA and unary extension and binary join operations are carried out to completion. *Completion* means the edge is extended as much as allowed by the unary rules, and joined to all adjacent edges in the chart, as allowed by the binary parser grammar rules. All newly created edges are added to the CHART and AGENDA. This continues until the AGENDA is empty. Binary join operations can only continue as long as there are adjacent edge pairs which have not been processed. The number of adjacent pairs is restricted by the length of the string or lattice path read so far. Unary extension is restricted to, at most, K consecutive extensions, so the closure operation is guaranteed to stop ($K = 3$ for experiments of chapter 5). Once the AGENDA is empty, it is initialized again using the next WORD edge from the sorted word edge stream. The detailed algorithms for chart closure, unary extension, and binary adjunction are outlined in appendix A.1.

Note that, similar to the standard CKY parsing algorithm, only one edge for each unique *edge signature* is kept in the chart. Edge equality, for the purpose of dynamic programming, is defined by equality of a subset of edge properties (the edge signature), such that for any edge e with total score t_e, if there exists some edge f with total score $t_f > t_e$ and edges e and f are equal in terms of their edge signatures, then only f is kept. It is not possible that edge e can be part of some parse with overall better score than the best parse containing f. In terms of the higher parse tree structure built upon e or f,

[3]Extension is bottom up, although grammar rules are written in conventional top-down format. For grammar rule STOP → RIGHT we create a STOP edge as parent to the RIGHT edge.

they are completely interchangeable. Edge properties defining a unique edge signature are (l, r, cat), where cat contains $(type, P, H, T, W, c, p, D, V, u, S, H_{-1}, T_{-1}, W_{-1})$.

Bottom-up parsing requires that when processing any edge of span (l, r), all edges of span $(l + i, r - j)$ for any (i, j) have already been processed. That is, any smaller edge within the larger span has already been added to the chart and fully extended by chart closure. This usually is achieved by considering edges of sequentially greater span. Our online (left-to-right) parser performs complete chart closure after each word edge is received. The closure operation is similar to the standard chart closure which is usually performed over a chart initialized with all word edges simultaneously. To ensure that the bottom-up restriction is met, chart closure is carried out working backwards from r, increasing the span considered iteratively from the `WORD` edge span (l, r) to full path length $(1, r)$.

4.6.4 Parse Completion

After all `WORD` edges have been processed from the word lattice, the `CHART` is queried for all *spanning edges* — edges with left endpoint at the start node of the lattice, and right endpoint at the end of the lattice. These are further filtered to only those *complete parse* edges, which are `STOP` edges with parent P equal to `TOP`, the unique "complete parse" non-terminal which we add to all Penn Treebank training trees. Note that `(S)` cannot be used, as not all Penn Treebank trees are full sentences.

If there are no complete parse edges, then the beam is increased (see section 4.7) and the parse is attempted again. Any parses that are found are converted to Penn Treebank format and output.

4.7 Heuristic Search and Pruning

The parsing model we employ finds the most probable parse given a word lattice. It is important to note that the sentence yielded by the most probable parse is not necessarily the most-probable sentence (Chappelier *et al.*, 1999). For example, assume that parse tree with the highest probability, p_{best}, yields some sentence s. If p_{best} is the only parse of s, then we can say the probability of s is that of p_{best}. There may be some other sentence x, for which several (lower probability) parse trees exist. It is possible that sum of the probabilities of all the parse trees of x will be greater than the probability of p_{best}. Thus x is a more probable sentence than s. Conducting this sort of summation is not practical given a dynamic programming search. In fact, finding the most-probable sentence is an NP-hard problem (Sima'an, 1996).

The parsing model (Collins, 1999) used in this work generates all possible parse trees, using relationships observed in the training data. Therefore, our model could be used to approximate the search for the most probable sentence, by calculating probabilities for each word as the sum of probabilities contributed by all parses they participate in. However, this would change our task to a two-stage task — a search for the best sentence using our model as a pure language model, followed by a search for the best parse of that sentence. Such a process may not work well with the deficient probabilities we use — we do not account for the entire probability mass, as we use heuristics to restrict the search space.

Generating all potential parse trees results in a chart which quickly becomes unmanageable in size. The worst case asymptotic time and space complexity of the models is $O(n^5T^2N^3D^2LR)$ (Collins, 1999), where:

- n = length of input (length of path in lattice)
- T = maximum number of different tags seen for any word in vocabulary
- N = the number of possible labels for an edge (the number of non-terminals in the grammar)

- D = the number of values possible for the distance
- L = the number of left subcategorizations seen in training data
- R = the number of right subcategorizations seen in training data

If we trace the generation of edges, we can see this complexity at work. For example, given a `STOP` edge headed with H, a `LEFT` edge is created for all parents P and all left subcategorizations S observed for H in the training data. Extension of all these `LEFT` edges to `RIGHT` edges creates a new edge for all observed right subcategorizations. Binary adjunction takes place between all adjacent edges. The chart size continues to grow in this way. Fortunately, the vast majority of the edges we create have probability much lower than the best edges for their span, and therefore have low likelihood of appearing in the best parse. There are several techniques to reduce chart growth with varying impact on the quality of results.

4.7.1 Beam Search

The main technique we employ is a variation of the beam search of Collins (1999) to restrict the chart size by excluding low probability edges. The drawback to this process is that we can no longer guarantee that a model-optimal solution will be found. In practice, these heuristics have a negative effect on parse accuracy, but the amount of pruning can be tuned to balance relative time and space savings against precision and recall degradation (Collins, 1999).

One can think of the beam search as searching a darkened space with a flashlight, following clues. One can only see the area around the flashlight beam, but if one follows the clues, one can still reach a goal. The chance of seeing clues depends proportionally on the diameter of the flashlight beam. The cost of the search also grows proportionally as we illuminate more space to consider. We can consider the space of all possible parse trees, given the parameter set and the sentence or lattice to parse, as the search space. The basic idea of beam search is to only consider those candidate edges (the "clues"

in our analogy) which have total scores (acoustic, lattice language model, inside, and outside probabilities combined) within some factor of the best candidate we have seen for any span:

$$edge.t \geq bestTotal(edge.l, edge.r)/beam \tag{4.3}$$

Collins (1999) uses a fixed size beam (10000). We experiment with several variable beam sizes, where the beam is some function of a base beam and the edge *width*. The width is the number of terminals (`WORD` edges) dominated by an edge. The base beam starts at a low beam size and increases iteratively by a specified *beamIncrement* if no parse is found, as shown in figure 4.4. This allows parsing to operate quickly (with a minimal number of edges added to the chart). However, if many iterations are required to obtain a parse, the utility of starting with a low beam and iterating becomes questionable (Goodman, 1997). It may be better to set a larger initial base beam, which may be slower for some inputs, and require fewer iterations for others. The base beam size is limited by *beamLimit* to ensure the chart size is not allowed to increase infinitely. As an edge gets wider (and as the associated tree gets deeper), it is informed by greater context and more model parameters, and has survived more pruning steps. The variable beam function is based on the intuition that due to these factors informing wider edges, we have increased confidence in them, when compared to narrow edges. Therefore, we do not need to retain as many of them, so the beam size can be smaller. Additionally, fewer steps remain between a wide edge and a spanning edge (complete parse). Therefore, we can use a more restrictive pruning on wide edges as we are less likely to use sub-optimal edges in building the remaining structure. An example variable beam function is:

$$variableBeam = baseBeam/\log(edge.width + 1) \tag{4.4}$$

Varying the beam in this way can cause thrashing — the parser will form small edges, but will not combine them into larger ones because the beam for large edges is too small. The balance then is to select a variable beam function and baseBeam which allow for

reasonable speed and chart size, while maintaining parse accuracy and recall and avoiding thrashing.

The algorithm for adding edges to the chart, including the comparison with the variable beam, is outlined in figure 4.5. Note that we use log-space for scores, so the beam is subtracted from the best score rather than divided.

A Note on Removing Parents

The algorithm of figure 4.5 includes a call to the `chart.removeEdgeAndParents` method. The dynamic programming algorithm requires that, for every edge e, when an edge with a matching signature is found, if e has a score lower than the new edge, e is replaced with the new edge. In a standard bottom-up parser, no parents of e would yet exist in the chart, because they would have a span larger than the edge we are replacing, and spans are parsed by increasing size. In addition, edges in a traditional parser point only to an edge signature as a child — not a specific edge. Thus any edges on the agenda whose child is replaced remain valid, as they will link automatically to the new edge. The implementation we use is different in two ways:

- Closure is carried out after each `WORD` edge is received by the `PARSER`. Thus it is possible that the old edge we are removing has parent edges which have already been added to the chart.

- The edges added to the agenda are hard-linked to children in the chart. Each stores its score (weight w and total t), rather than recomputing it by tracing back through the chart each time. This allows for faster edge comparison. The score is based on the score of child edges, so if child edge scores change, the score of the parent becomes invalid. Rather than recompute scores and reassign children, we simply remove the obsolete parent edges and rebuild them through closure on the new edge.

```
boolean chart.dynamicAdd(edge) {
if (edge.t > threshold) {
    if (edge.t > bestTotal(edge.l,edge.r) - beam(baseBeam,edge.width)) {
        if (chart.getNode(edge.r).hasEdge(edge.signature)) {
            oldEdge = chart.getNode(edge.r).get(edge.signature);
            if (edge.t > oldEdge.t) {
                // remove old edge with matching signature
                // remove any edges which dominate old edge
                chart.removeEdgeAndParents(oldEdge);
                // matching edge may still be scheduled for processing
                agenda.remove(oldEdge);
                chart.add(edge);
                return true;
            } else {
                return false;
            }
        } else {
            // no matching edge already in chart
            chart.add(edge);
        }
    } else {
        return false;
    }
}
else
    return false;
}
```

Figure 4.5: Dynamic programming and pruning algorithm for adding edges to CHART. Given an edge, dynamic programming conditions are checked to ensure the probability of the edge is above the threshold and (if a competing edge with the same signature exists in the chart) within the beam. If conditions are met, the edge is added to the CHART.

When an edge is created and added to the `AGENDA`, an upward link is created from its daughters. The parent-removal operation is carried out recursively by tracing these links. An example is shown in figure 4.6

4.7.2 Thresholds

Collins (1997) reports a 36% average speed increase for parsing when an absolute threshold is used in addition to the beam search. Only edges with total $t \geq threshold$ are added to the chart. In a manner similar to iteratively growing the beam, if no parse is found after complete processing of all word edges, the threshold is decreased. We implement this type of pruning, but, similar to Goodman (1997), could not gain a significant increase in speed as a result.

4.7.3 Overparsing

Hall and Johnson (2003) introduces *overparsing* as a technique to ensure that early stages of parsing do not strongly bias later stages. They use a chart parser with a PCFG to perform a first stage of parsing on word lattices, and supply the trees found by this stage to a more sophisticated lexicalized probabilistic parser (that of Charniak (2001)). The first stage of parsing is continued until a complete parse is found. At this point they let n equal the number of edges in the chart. Parsing with the PCFG is continued (overparsing) until the chart size is some factor k times n. Now there are presumably many complete parses for the second stage model to rescore.

We adapt this idea to a single stage process. Due to the restrictions of beam search and thresholds, the first parse found by the model may not be the model optimal parse. We therefore employ overparsing, further extending the base beam by the beam increment and parsing again. We continue this process as long as extending the beam results in an improved best parse score. The basic algorithm is adapted and shown in figure 4.7. Note that extending the beam can sometimes lead to a decrease in the best parse score, due

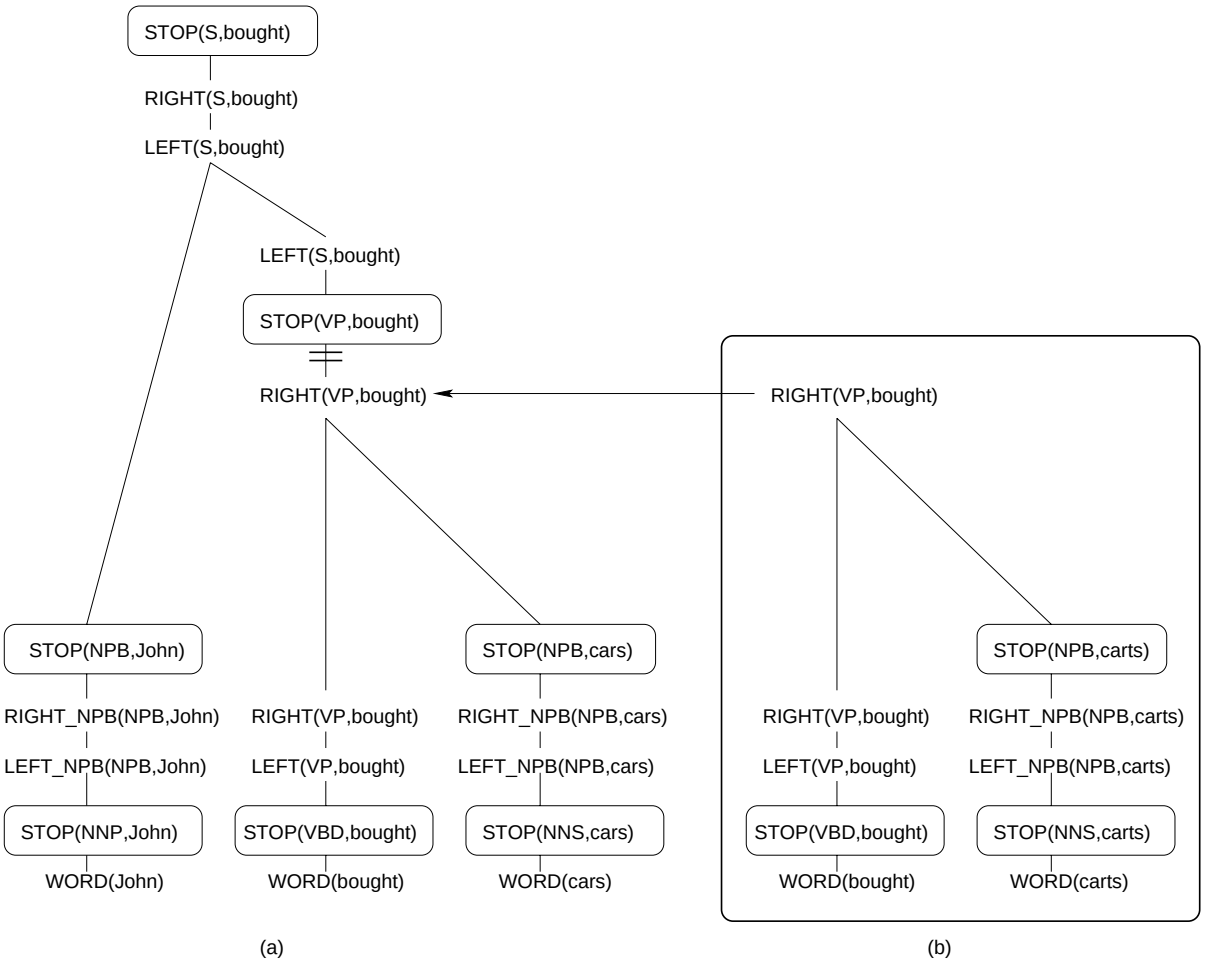

Figure 4.6: Example of removal of parent edges from chart. After the `WORD` edge for (cars) is read, the parse tree (a) is added to the chart on closure. On reading the next `WORD` edge (carts) from the lattice, the new VP edge (b) is created, and replaces the VP of the same signature in (a). At this point, the connection in (a) is broken, as shown. The higher edges must be removed from the chart. (Note that the link cannot simply be moved to the new edge, as the weights and totals for the higher edges are based on the weights and totals of the old `RIGHT(VP,bought)` edge.)

to the nature of beam search. If this occurs, the beam is reset to the beam which gave the best parse score, and the parse is repeated, ending the process.

4.8 Implementation Details

The lattice parser is implemented using Java, which was chosen for its ease of prototyping, an extensive existing class library (including file I/O using GZip compression, hashing and resizable arrays, sorted sets), its well integrated debugging and exception handling, its class-based inheritance structure, its support of literate programming by generating documentation (using `javadoc`), and its garbage collection model which manages memory without the need for express deallocation. Java also is portable and widely available on most platforms. One drawback is that Java is often slower than other object-oriented languages, such as C++.

Standard object-oriented design, Java code style, and javadoc documentation guidelines were followed. The implementation was optimized using a profiler (Borland OptimizeIt) to eliminate memory leaks and to discover and streamline any bottlenecks in the parsing process.

A notable optimization, suggested by Bob Carpenter, is the calculation of parameter values during training. The original implementation of Collins (1999) retains counts gathered from training data, and uses the counts to calculate parameter values as needed at runtime. We first implemented this type of parameter calculation, but found it was too slow, using Java. Instead, values of all $P(Y|\Phi_i(X))$ for all observed outcomes Y, all contexts X, and all back-off levels $\Phi_i(X)$ are calculated and their log value is stored in a hash table (the *estimator*), during training. At runtime, values can be quickly recovered using the context and outcome as a hash key. This results in more memory usage, but faster implementation as all parameters are only calculated once, during training. Estimators are created for each parameter type in table 3.1.

```
// INITIALIZATION
input word lattice, prune to N-best
topologically sort edges into input stream

// ITERATIVE PARSE
while ((parses.size == 0) && (baseBeam < beamLimit)) {
    parses = parse(input stream, baseBeam);
    if (parses.size == 0)
        baseBeam = baseBeam + beamIncrement;
}

// OVERPARSING (only if we exited first loop with a parse)
if (parses.size != 0) {
    // start previous best score at lowest value
    previousBestScore = -1 * Double.MAX_VALUE;
    bestScore = bestScore(parses);
    while ((bestScore > previousBestScore) && (baseBeam < beamLimit)) {
        previousBestScore = bestScore;
        baseBeam = baseBeam + beamIncrement;
        parses = tagAndParse(input stream, baseBeam)
        bestScore = bestScore(parses);
    }
    // repeat best parse if overparsing results in a lower probability
    if (bestScore < previousBestScore) {
        baseBeam = baseBeam - beamIncrement;
        parses = tagAndParse(input stream, baseBeam);
    }
}
```

Figure 4.7: Addition of overparsing to the lattice chart-parsing algorithm.

The lattice parser operates with an interactive interface, supporting several functions:

- training the tagger and parser using Penn Treebank format files — model parameters are saved to a file for later recall
- selecting a previously trained set of model parameters
- parsing sentences as they are input by a user
- extracting sentences from Penn Treebank format files and parsing them
- parsing word lattices, N-best lattices, and N-best lists from SLF files
- runtime modification of the beam search, overparsing, and various output variables (*e.g.*, headword printing, NPB printing, etc.)

Chapter 5

Experimental Results

One of the goals of this work is to implement the first word lattice parser based on the head-driven probabilistic model of Collins (1999). Parsing word lattices directly provides useful syntactic and semantic information, such as parse trees, which are not supplied by the simpler language models, such as the N-gram model. By changing the focus of the search from the most probable word sequence to the most probable parse, we expect to extract the most probable syntactic structure from the parse tree, which will be rooted in a probable word sequence. Most work in syntactic language modelling for speech recognition focuses on achieving the lowest word error rate (WER) — a measure of the correctness of a proposed word sequence. To evaluate our model, we propose measuring both the parsing accuracy and the accuracy of the word sequence extracted from the lattice, as compared to reference parse trees and utterance transcriptions. Only Roark (2001) has previously reported both kinds of evaluation on a single system.

5.1 Overview

The word lattice parser is evaluated with several metrics — WER, labelled precision and recall, crossing brackets, and time and space resource usage. We conduct evaluations using two experimental sets — strings and word lattices. First we optimize settings

(thresholds, variable beam function, base beam value) for parsing using development test data consisting of strings for which we have annotated parse trees. Second, we measure the performance of the parser for several acoustic model / language model mixtures using word lattices and N-best lists for which we have the true utterance transcriptions.

The parsing accuracy for parsing word lattices is not directly evaluated as we do not have annotated parse trees for comparison. Furthermore, standard parsing measures such as labelled precision and recall are not directly applicable in cases where the number of words differs between the proposed parse tree and the gold standard.

We also examine the impact of our sub-optimal beam search and the use of overparsing on parse accuracy, WER, and the time required for parsing. Results show scores for parsing strings which are lower than the original implementation of Collins (1999). The WER scores for this, the first application of the Collins (1999) model to parsing word lattices, are comparable to other recent work in syntactic language modelling, and better than a simple N-gram model trained on the same data.

The training and test data, and evaluation results are explained in more detail in the following sections.

5.2 Parsing Strings

The lattice parser can parse strings by creating a single-path lattice from the input (all word transitions are assigned an input probability of 1.0). Whereas many language modelling works for speech recognition report only WER, we are also interested in the parsing accuracy of the model. We measure parsing accuracy using sections of the Penn Treebank (Taylor *et al.*, 2003), for which we have annotated reference parses for comparison.

5.2.1 Training and Test Data

The lattice parser was trained on sections 02-21 of the Wall Street Journal portion of the Penn Treebank (Taylor *et al.*, 2003) (39,832 sentences; 989,860 words). Development testing was carried out on section 23 (2,416 sentences; 59,100 words) in order to select model thresholds and variable beam functions. Final testing was carried out on section 00 (1,921 sentences; 88,494 words), and scores are reported for sentences with ≤ 40 words.

5.2.2 Evaluation Metrics

We use the PARSEVAL measures (Black *et al.*, 1991) to measure the parsing performance:

Labelled Precision (LP) $= \frac{number\ of\ correct\ constituents\ in\ proposed\ parse}{number\ of\ constituents\ in\ proposed\ parse}$

Labelled Recall (LR) $= \frac{number\ of\ correct\ constituents\ in\ proposed\ parse}{number\ of\ constituents\ in\ reference\ parse}$

Crossing Brackets (CB) = number of constituents in proposed parse which violate constituent boundaries with a constituent in reference parse

0 Crossing Brackets (0 CB) = percentage of sentences for which there are no crossing brackets

$\leq$ 2 Crossing Brackets ($\leq$ 2 CB = percentage of sentences for which there are 2 or fewer crossing brackets

5.2.3 Variable Beam Function

The variable beam functions were developed by starting with the intuition that the beam should be smaller at larger widths, as described in section 4.7.1. We initially used $\hat{b} = b/width$, and found that the rate of decrease in beam size was too large — the beam became too small as width increased. This meant a large base beam (admitting many narrow edges) was required for convergence. We worked from this result towards

a relationship which strongly pruned at large widths, to speed up parsing, and had a beam/width relationship of order less than linear, to ensure enough narrow edges were admitted. The resulting format, where w is the edge width, b is the base beam, and k is a constant, is:

$$\hat{b} = \frac{b}{\log((w+k)/2)} \tag{5.1}$$

The rate of decrease in the beam slows as width increases. The rate is controlled by offsetting the width by k — the application of k has greater effect on smaller widths. Performance for various values of k were compared against the original (Collins, 1999) implementation with constant beam $b = 10000$, or, in the natural log domain we use, $b \approx 9$.

5.2.4 Thresholds

Collins (1997) reports a 36% increase in parsing speed when an absolute threshold is applied during parsing. All edges with score below the threshold are pruned. As with Goodman (1997), we were unable to duplicate this speed-up.

When a constant threshold is applied, it usually prunes edges that would otherwise be removed by the beam pruning. Since the beam search only admits one edge per edge signature (left node, right node, and category) to the chart, the benefit of applying a threshold in this case is limited to pruning that single edge. That is, the threshold will only prune out edge signatures for which no edge exists with score above the threshold. With relatively little savings in chart size, this technique risks eliminating important edges from the chart. For instance, assume for some sentence or lattice, only one complete parse (with non-zero probability) exists. The threshold can eliminate some very low probability edge e within that parse, resulting in no complete parse being found. Application of the beam does not eliminate e unless a more probable edge with the same signature and span is found. The problem of not finding any parse outweighs the benefit of further reduction in chart size, so we do not apply an absolute threshold in the

Exp.	Beam Function	OP	LP (%)	LR (%)	CB	0 CB (%)	≤2 CB (%)
Ref	b	N	88.7	89.0	0.95	65.7	85.6
1a	b	N	56.8	59.9	3.81	25.9	59.3
1b	b	Y	79.6	74.8	2.12	42.0	69.1
2a	$b/log((w+1)/2)$	N	71.3	67.6	2.75	34.2	56.5
2b	$b/log((w+1)/2)$	Y	81.3	81.3	1.88	50.0	75.0
3a	$b/log((w+2)/2)$	N	79.4	80.6	1.89	46.2	74.5
3b	$b/log((w+2)/2)$	Y	80.8	81.4	1.70	44.3	80.4
4a	$b/log((w+4)/2)$	N	58.9	64.4	3.43	27.6	62.6
4b	$b/log((w+4)/2)$	Y	69.8	72.1	2.14	35.8	70.4

Table 5.1: Results for parsing section 0 ($\leq$ 40 words) of the WSJ Penn Treebank: OP = overparsing, LP/LR = labelled precision/recall. CB is the average number of crossing brackets per sentence. 0 CB, $\leq$ 2 CB are the percentage of sentences with 0 or $\leq$ 2 crossing brackets respectively. The beam function gives the dependence of the beam $\hat{b}$ on the base beam, b, and width w. *Ref* is Model II of (Collins, 1999).

experiments reported in this chapter.

5.2.5 Experimental Results and Analysis

Experiments were carried out while varying the variable beam function (section 4.7.1), and with or without overparsing (section 4.7.3). The variable beam is a function of the base beam and the edge width (number of words dominated by the edge). The base beam is a tuple in the log domain: $(base, increment, final)$, where the *base* is increased by the *increment* until a parse is found, the $final$ limit is exceeded, or until overparsing is complete. The base beam was set to $(8, 1, 20)$, which was found to be a good balance between speed and the probability of finding a valid parse on most sentences. Experimental results are given in table 5.1.

The scores for all experiments are lower than the scores of the original implementation of model II (Collins, 1999). Experiment 1 of table 5.1 corresponds to the experimental

Exp.	Beam Function	OP	Time	Number of Edges (per word)
1a	b	N	22h41m	3025
1b	b	Y	38h04m	3516
2a	$b/log((w+1)/2)$	N	19h32m	1920
2b	$b/log((w+1)/2)$	Y	28h20m	2920
3a	$b/log((w+2)/2)$	N	15h01m	1491
3b	$b/log((w+2)/2)$	Y	22h51m	2055
4a	$b/log((w+4)/2)$	N	11h03m	871
4b	$b/log((w+4)/2)$	Y	15h33m	1220

Table 5.2: Parsing times and chart size for section 0 of the WSJ Penn Treebank. OP = overparsing.

conditions of the reported reference work. We have confirmed the reported PARSEVAL scores for the reference work. Comparing experiment 1b against the reference, there is a difference of approximately 10% in the labelled precision and recall scores, and a difference of 1.2 in the average number of crossing brackets, with our implementation scoring lower on all metrics. Our model performs best with beam function $b/log((w+2)/2)$. This is likely because the beam functions b and $b/log((w+1)/2)$ admit many narrow edges, and are less selective. They can quickly converge to a low probability parse, and stop. The more restrictive beam function $b/log((w+4)/2)$ prunes too strongly, reducing the search and the probability of finding the model-optimal parse. The difference in scores for all experiments with our model, compared to the reference work, is likely due in part to differences in POS tagging. Tag accuracy for our model is 93.2%, whereas for the original implementation of Collins (1999), model II achieved tag accuracy of 96.75%. We restrict the tag-set for each word to those suggested by a simple first-stage tagger (section 4.3). For `unknown` words, we use the tag predicted by the model. Collins (1999) considers all tags for each word, except `unknown`, for which he uses the tagger of (Ratnaparkhi, 1996). Note that we cannot fall back to the tagging of Ratnaparkhi (1996) for `unknown` words,

as that tagging system is not adapted for tagging word lattices.

The utility of the overparsing extension can be seen in table 5.1. Each of the PARSEVAL measures improves when overparsing is used. Table 5.2 reports the times required for parsing section 0 with and without overparsing. In addition, the average number of edges added to the chart per word is reported. This measure shows how overparsing and the variable beam function affect the chart size. Generally, we see that larger chart sizes and longer parse times correspond with improved performance. The greater k is in the variable beam function, the fewer edges are admitted by the beam. This also corresponds to faster parsing, and lower accuracy. One exceptional case is the use of a constant beam (experiments 1a/1b in table 5.2). The total number of edges admitted (per word) by the constant beam is lower because convergence occurs at a lower base beam: more narrow edges are admitted during parsing, resulting in faster convergence. Overparsing results in parsing times approximately 1.5 to 2 times longer than equivalent best-first parsing. Thus we have an instantiation of the common accuracy/speed trade-off.

5.3 Parsing Word Lattices

The success of the parsing model as a language model for speech recognition is measured both by parsing accuracy (parsing strings with annotated reference parses), and by WER. WER is measured by parsing word lattices and comparing the parse tree sentence yield to the reference transcription (using NIST SCLITE for alignment and error calculation). We assume the parsing performance achieved by parsing strings (see section 5.2) carries over to parsing word lattices. We cannot measure PARSEVAL scores for parsing word lattices as we do not have annotated parse trees for the true utterances of the HUB-1 corpus. Also, the PARSEVAL scores are not adapted to compare parses for which the number of words differs. For parsing lattices we use the same model and implementation as we used for parsing strings (a string is simply parsed as a single-path word lattice).

Additionally, the same parse output (PTB-format trees) is reported, and the training and test corpora are from the same domain (Wall Street Journal). Thus we expect parsing scores to be similar for parsing lattices. However, Roark (2001) (using a different parsing model) reports that parsing accuracy (LR/LP) decreases by approximately 3% when using a pruned version of the Penn Treebank for training and testing in which all punctuation is removed. Punctuation is a strong indicator of phrasal boundaries, so the impact of removing it is severe. As our training and test data for the word lattice tasks do not contain punctuation, a similar decrease in parse accuracy is likely. The parsing task is further complicated by the joint nature of the search — the parser finds the best parse and word sequence simultaneously. There exists the possibility for competition between finding the best word sequence (as scored by the acoustic model and lattice language model) and finding the best parse tree (as scored by our model).

The remainder of this section describes the training and test corpora for word-lattice parsing and word-list rescoring, and the evaluation of the parser using N-best lattices and lists.

5.3.1 Training and Test Data

Two different corpora are used in training the parsing model on word lattices:

- sections 02-21 of the WSJ Penn Treebank (the same sections as used to train the model for parsing strings) [1 million words]
- section "1987" of the BLLIP corpus (Charniak *et al.*, 1999) [20 million words]

The BLLIP corpus is a collection of Penn Treebank-style parses of the three-year (1987-1989) Wall Street Journal collection from the ACL/DCI corpus (approximately 30 million words)[1]. The parses were automatically produced by the parser of Charniak

[1] The sentences of the HUB-1 corpus are a subset of those in BLLIP. We removed all HUB-1 sentences from the BLLIP corpus used in training.

(2001). As the memory usage of our model corresponds directly to the amount of training data used, we were restricted by available memory to use only one section (1987) of the total corpus. Using the BLLIP corpus as training data, we expect to get lower quality parse results due to the higher parse error in the automatically annotated BLLIP corpus, when compared to the manually annotated Penn Treebank. The WER is expected to improve, as the BLLIP corpus has much greater lexical coverage (*i.e.*, random error will be lower because the parameters are based on more training samples). Training on the BLLIP corpus shows the relationship between the amount of training data and the accuracy of our model for the speech recognition task. For better comparison against the trigram model stored in the HUB-1 lattices (the "Lattice Trigram", trained on 40 million words), and the work of Hall and Johnson (2003) (trained on the entire BLLIP corpus) we train on the BLLIP corpus, section 1987.

The corpora are modified using a utility provided by Keith Hall to convert newspaper text to speech-like text, before being used as training input to the model. Specifically:

1. numbers are converted to words (`60` $\rightarrow$ `sixty`)

2. all punctuation is removed

We test the performance of our parser on the word lattices from the NIST HUB-1 evaluation task of 1993. This corpus was chosen for several reasons:

- it has been used in other works on syntactic language modelling (e.g Chelba, 2000; Roark, 2001; Hall and Johnson, 2003)

- it is in the HTK standard lattice format, with silences and other disfluencies removed

- 50-best paths found by A* search are available (Chelba, 2000; Roark, 2001)

- the lattices are derived from a set of utterances produced from Wall Street Journal text — the same domain as the Penn Treebank and the BLLIP training data

- the corpus was produced by professional readers, thus the speech is relatively clear and the resulting lattices are sparse (an advantage for our system in terms of available memory resources)

There is an important difference in the tokenization of the HUB-1 corpus and the Penn Treebank format. Contractions (*i.e.*, `he's, wasn't, jones'`) are split in the Penn Treebank (*i.e.*, `he 's, was n't, jones '`), but not in the word lattices. The Treebank format cannot easily be converted into the lattice format, as often the two parts fall into different parse constituents. We use the lattices modified by Chelba (2000) in dealing with this problem — contracted words are split into two parts. The first part takes the original time span and is assigned an acoustic probability of 1.0. The second part is assigned zero time and the original acoustic probability. We differ from Chelba (2000) and instead follow Hall and Johnson (2003) in that the Treebank tokenization is also used for measuring the WER — recognition of at least part of a conjunction should be credited. The reference transcriptions are processed and conjunction words are split to correspond to the modified lattices.

The word lattices of the HUB-1 corpus are annotated with an acoustic probability, a, and a trigram probability, lm, for each edge. The lattice trigram model was trained on 40 million words with a 20,000 word open vocabulary. These scores are used to guide the parsing, as described in section 4.4. Recall that word edges of single-path lattices (created when parsing strings) are assigned an input probability of 1.0. For other word lattices, the input probability is $\log(\mathcal{P}_{input}) = \alpha(\log a) + \beta(\log lm)$, where a is the acoustic score and lm is the trigram score stored in the lattice. The edge weight is a combination of these scores with the model parameters:

$$w = \alpha(\log a) + \beta(\log lm) + \log s \tag{5.2}$$

where w is the log edge weight, and s is the inside probability assigned by the parameters of the parsing model. Some, such as Roark (2001), set α to 1.0 and instead apply

a scaling factor to increase the score assigned by their model (*i.e.*, they use a factor $\gamma(\log s)$). Chelba and Jelinek (2000) optimizes γ at 16 for the HUB-1 corpus. This factor is adopted by Roark (2001) and Hall and Johnson (2003). However, according to Mangu *et al.* (1999), models converge to a parse faster if a factor is used to reduce the dynamic range of acoustic score rather than increase that of the language model. Our experiments on a development test subset (the first 10 lattices of the HUB-1 corpus) confirmed this. We use $\alpha = 1/16$ as the optimal acoustic model weight — this is inversely related to a γ value of 16, used in other work. We do not apply any scaling factor, γ, to the inside score, s, of our model. The factor β is applied to the lattice trigram score, lm, stored in the lattice in order to adjust the relative weight of lm against the score of our model, s. Chelba (2000) and Roark (2001) interpolate the language model score (s) of their parsers with the lattice trigram score (lm) and find that an interpolation value of $\beta = 0.4$ is optimal. Hall and Johnson (2003) does not use the lattice trigram scores, *i.e.*, $\beta = 0$. In our work, values of β from 0 to 2.0 in 0.5 increments were tested using the variable beam function $\hat{b} = b/log((w+2)/2)$ and base beam of $(5, 1, 20)$.

5.3.2 Test Results and Comparison to Related Work

The performance of the parsing model as a language model for speech recognition is evaluated on the NIST HUB-1 corpus, a set of 213 word lattices in the HTK Standard Lattice Format, previously pruned to the 50-best paths by Roark (2001). The model was tested with and without overparsing, and for various variable beam functions and trigram weighting factors (β).

Parameter Optimization

The variation of lattice trigram weighting factors (β) is shown in table 5.3. The optimal value of $\beta = 1$ shows that the language model probabilities of the lattice trigram are complementary to those of our model. Factors of $\beta > 1.0$ strongly influence the choices

Exp.	β	Training Size	WER			
			S	D	I	T
1	0	1M	10.9	3.6	1.5	16.0
2	0.5	1M	10.5	3.6	1.5	15.6
3	1	1M	10.4	3.3	1.5	15.2
4	1.5	1M	9.4	3.3	1.2	13.9
5	2	1M	9.3	3.2	1.2	13.7
6	0	20M	9.3	3.2	1.2	13.7
7	0.5	20M	9.1	3.1	1.1	13.3
8	1	20M	9.0	3.1	1.0	13.1
9	1.5	20M	9.1	3.2	1.2	13.5
10	2	20M	9.2	3.2	1.2	13.6

Table 5.3: Comparison of different values of the trigram weighting factor: Parsing N-best word lattices with variable beam function: $\hat{b} = b/log((w+2)/2)$, no overparsing. S = substutitions (%), D = deletions (%), I = insertions (%), T = total WER (%). 1M training = Penn Treebank, sections 02-21. 20M training = BLLIP, section 1987.

of the parser, and make the contribution of our model's parameters less significant (*i.e.*, the same result is obtained as with the lattice trigram alone). The score for our model interpolated with the lattice trigram ($\beta = 1$) is better than that for our model alone ($\beta = 0$). This is likely due to the incorporation of additional training data by mixing the models — our model is trained on 1 million or 20 million words and the interpolated lattice trigram is trained on 40 million words. This is supported by the fact that the improvement gained by interpolating the lattice trigram is greater for interpolation with our model trained on only the Penn Treebank (1 million words). We choose $\beta = 1$ for the remaining experiments.

The application of the model to 50-best word lattices is compared to rescoring the 50-best paths individually (50-best list parsing). The results are presented in table 5.4.

Exp.	Training Size	Lattice/List	OP	WER				Number of Edges (per word)
				S	D	I	T	
1	1M	Lattice	N	10.4	3.3	1.5	15.2	1788
2	1M	List	N	10.4	3.2	1.4	15.0	10211
3	1M	Lattice	Y	10.3	3.2	1.4	14.9	2855
4	1M	List	Y	10.2	3.2	1.4	14.8	16821
5	20M	Lattice	N	9.0	3.1	1.0	13.1	1735
6	20M	List	N	9.0	3.1	1.0	13.1	9999
7	20M	Lattice	Y	9.0	3.1	1.0	13.1	2801
8	20M	List	Y	9.0	3.3	0.9	13.3	16030

Table 5.4: Results for parsing HUB-1 N-best word lattices and lists: OP = overparsing, S = substutitions (%), D = deletions (%), I = insertions (%), T = total WER (%). Variable beam function: $\hat{b} = b/log((w+2)/2)$. Training corpora: 1M = Penn Treebank sections 02-21; 20M = BLLIP section 1987.

Parsing N-best Lattices vs. N-best Lists

The number of edges added to the chart per word for N-best lists is an order of magnitude larger than for corresponding N-best lattices, in all cases. Since the WERs are similar, we see that parsing in N-best lists, we are doing more work that parsing N-best lattices, for the same result. Thus parsing lattices is more efficient. This is because common substrings are only considered once per lattice. For N-best list parsing, these common substrings can be parsed up to N times. We see from table 5.4 that overparsing has little effect on the WER. The word sequence most easily parsed by the model (*i.e.*, generating the first complete parse tree) is likely also the word sequence found by overparsing. Although overparsing may have little effect on WER, we know from the experiments of section 5.2.5 that overparsing increases parse-accuracy. This introduces another, more complex, speed-accuracy trade-off. Depending on what type of output is required from the model (*i.e.*, parse trees or strings), the additional time and resource requirements of overparsing may or may not be warranted.

Comparison to Previous Work

The NIST HUB-1 corpus has been previously parsed using syntactic language models (*e.g.*, Hall and Johnson, 2003; Chelba, 2000; Roark, 2001). The results of our best experiments for lattice- and list-parsing are compared with previous results in table 5.5. The oracle WER for the HUB-1 corpus is 3.4%. For the pruned 50-best lattices, the oracle WER is 7.8%. We see that by pruning the lattices using the trigram model, we already introduce additional error. Because of the memory usage and time required for parsing word lattices, we were unable to test our model on the original "acoustic" HUB-1 lattices, and are thus limited by the oracle WER of the 50-best lattices, and the bias introduced by pruning using a trigram model (see section 2.6.2). Where available, we also present comparative scores of the sentence error rate (SER) — the percentage of sentences in the test set for which there was at least one recognition error. Note that due to the small (213 samples) size of the HUB-1 corpus, the differences seen in SER may not be significant.

We see significant improvement in WER for our model trained on 1 million words of the Penn Treebank compared to a trigram model trained on the same data (Roark, 2001) — the "Treebank Trigram" noted in table 5.5. This indicates that the larger context considered by our model allows for performance improvements over the trigram model. The improvement seen for our model trained on BLLIP 1987 over the lattice trigram is not as large. As we reach lower WER scores, it is likely that the remaining errors are quite difficult (large number of ambiguities, ungrammatical constructions, conflicting acoustic and language model scores). Thus we experience diminishing returns as we improve our model by training on more data. Finally, the BLLIP 1987 corpus is smaller than the 40 million words used for the lattice trigram model, so the comparison is not completely appropriate.

The current best performing models, in terms of WER, for the HUB-1 corpus, are the parser of Charniak (2000) and of Roark (2001), applied to N-best lists by Hall

Model	N-best List/Lattice	Training Size	WER (%)	SER (%)
Current Model	List	20M	13.1	71.0
	Lattice	20M	13.1	70.4
	List	1M	14.8	74.3
	Lattice	1M	14.9	74.0
Oracle (50-best lattice)	Lattice		7.8	
Charniak (2000)	List	40M	11.9	
Roark (2001)	List	1M	12.7	
Hall (2003)	Lattice	30M	13.0	
Chelba (2000)($\lambda = 0.4$)	Lattice	20M	13.0	
Lattice Trigram	Lattice	40M	13.7	69.0
Roark (2001)	Lattice	1M	15.1	73.2
Treebank Trigram	Lattice	1M	16.5	79.8
No language model	Lattice		16.8	84.0

Table 5.5: Comparison of WER with for parsing HUB-1 words lattices with other works. SER = sentence error rate. WER = word error rate. "Speech-like" transformations were applied to all training corpora. Current model settings: $\hat{b} = b/log((w+2)/2)$, $\alpha = 1/16$, $\beta = 1$, with overparsing.

and Johnson (2003). N-best list parsing, as seen in our evaluation, requires repeated analysis of common subsequences, a less efficient process than directly parsing the word lattice. The WERs of Roark (2001), a top-down probabilistic parsing model, are notable because they are achieved by training on only 1 million words of the Penn Treebank. The difference in WER between our parser and those of Charniak (2000) and Roark (2001) applied to word lists may be due to the lower PARSEVAL scores of our system. Xu *et al.* (2002) report inverse correlation between labelled precision/recall and WER. We achieve 73.2/76.5% LP/LR on section 23 of the Penn Treebank, compared to 82.9/82.4% LP/LR of Roark (2001) and 90.1/90.1% LP/LR of Charniak (2000).

5.3.3 Time and Space Requirements

The implementation of the parsing model, using Java, is costly in terms of computational resources. The algorithms and data structures have been optimized using a memory and processor profiler. Parsing section 00 of the Penn Treebank on average takes approximately 20 hours, on an Intel Pentium 4 (1.6GHz) Linux system using 1GB memory. This is partially due to the need to manually invoke additional Java garbage collection cycles to free memory which is no longer used (for example, edges which have been deleted from the chart). We suggest that the implementation may be faster using another programming language which allows direct access to memory resources, such as C++.

5.4 Summary of Results

The parser has been evaluated using both strings and word lattices. Evaluation shows disappointing results for the PARSEVAL suite of measures. When compared to the original implementation of the parsing model (Collins, 1999), our implementation performs poorly. There are several differences between the systems which may contribute to the differing scores, including the POS tagging, and the dynamic search and pruning heuris-

tics. Our system uses a large amount of computational resources, necessitating the use of a variable beam function which reduces parse quality. Although the parsing scores are not competitive with the state-of-the-art systems for strings, the parser implemented in this work builds parse trees from word lattices directly, at a significant computational savings over parsing N-best word lists. The accuracy of the parser at selecting the true utterance, given a word lattice or list, is comparable to other recent implementations of syntactic language modelling. The WER achieved by a mixture of the trigram and head-driven probabilistic parsing model probabilities is lower than that of the trigram model alone.

Chapter 6

Conclusions

In this work, we set out to apply the parsing model of Collins (1999), one of the current state-of-the-art parsers, to word lattices. Successful integration of high-level language models and acoustic models for speech recognition was long thought a computationally intractable problem, due to the large size of word lattices. Recently, through increased computing resources, optimization of algorithms, and development of sequential coupling, such integration has become possible. Success, in terms of word error rate (WER), has been seen using probabilistic parsing systems as language models for speech recognition, but the head-driven model had not been previously applied.

Incorporation of high-level language models (including parsing models) into automated speech recognition has been accomplished previously, using three main coupling paradigms — tight, incremental, and sequential. Our review of the current literature showed that sequential coupling is the best suited for integration of high-level language models, as integration using the other coupling techniques leads to a large increase in complexity.

We reimplemented the parsing model II of Collins (1999), using Java, adapting it to the domain of word lattices. The parsing task is more complex than parsing strings, as the search encompasses finding the best parse given many possible word sequences.

The word sequences can be stored in a compressed format — a word lattice — or the N best paths can be parsed individually. As we were interested in extracting data from word lattices for use in automated speech understanding, we expanded upon traditional measures of success, and evaluated the performance of the model using both parsing scores (the PARSEVAL measures), and WER.

The system was evaluated over two sets of data: strings and word lattices. As PARSEVAL measures are not applicable to word lattices, we measured the parsing accuracy using string input. The resulting scores were lower than the original implementation of the model (Collins, 1999), applied to strings. Despite this, the model was successful as a language model for speech recognition as measured by WER. Here, the system performs better than a simple N-gram model trained on the same data, while simultaneously providing syntactic information in the form of parse trees.

6.1 Summary of Contributions

6.1.1 Review of the State-of-the-Art

We present the first thorough review of syntactic language modelling applied to word lattices and N-best lists, including an analysis of the various coupling paradigms used to combine acoustic and language models. We suggest expanding the widely used measures of success to include attempts to measure the ability of a language model to extract high-level information, such as syntactic structure and semantic relationships, from speech.

6.1.2 Implementation of Probabilistic Parsing for Word Lattices

We implement, using Java, a probabilistic parser which takes strings or HTK Standard Lattice Format word lattices as input. The modular implementation allows for different

grammars (parameterizations of probabilistic parsing) to be easily incorporated. Various tunable dynamic programming heuristics speed up the parsing process with varying impact on parsing scores.

6.1.3 Head-Driven Parsing for ASR

The parsing model II of Collins (1999) was reimplemented and its parameter set was used as the grammar for the word lattice parser. Evaluation results show that head-driven probabilitic parsing can improve upon WER while extracting parse trees for use in speech understanding.

6.2 Limitations and Future Directions

6.2.1 Limitations

Time, Space, and Accuracy

The current implementation of the parser requires almost one gigabyte of memory, and runs about 100 times slower than original model II parser of Collins (1999). Further optimization, or reimplementation in another programming language, will almost certainly decrease parse times and memory requirements. In order to run the parser with reasonable time and memory resource usage, its dynamic search parameters (beam and thresholds) must be restrictive, decreasing the probability of finding the model-optimal parse. Parsing scores reported for our implementation are lower than those reported in Collins (1999). This may be due to differences in POS tagging accuracy. Future work should focus on improving the parse accuracy of our implementation.

Scaling Up

The speech corpus used in this work consists of rather small lattices, with fewer ambiguities than are often produced by "real word" systems. The HUB-1 corpus can be described as "lab speech." Lab speech — speech corpora obtained from professional speakers reading a script in controlled conditions — is usually easier to interpret than real world speech. Testing of this model on a spontaneous speech corpus would be rather difficult, as we do not account for disfluencies such as filled pauses, corrections, etc. As mentioned above, we also encounter memory and time resource limitations with the sparse lattices of the HUB-1 corpus. The current implementation would likely not be able to parse the larger lattices of spontaneous speech. Adaptations to the parsing model, as described in the following section, would be required.

6.2.2 Future Directions

Parsing Spontaneous Speech

The current model has been evaluated on "lab speech." Adaptations to the parsing model, or use of a preprocessor which filters disfluencies from word lattices, could be made in order to parse spontaneous (*i.e.*, non-lab) speech. Such a system, even with filtering of disfluencies, would need to be domain-specific. General spontaneous speech contains many colloquial expressions (often ungrammatical), which would receive low scores from our parser. A manually-parsed corpus of large-vocabulary speech would be required to train the parser to operate over such data. Additionally, spontaneous speech corpora are often plagued by lower quality acoustic data, which would result in much larger word lattices. Due to the time and space requirements of our implementation, we would likely need to restrict a spontaneous speech lattice to its N-best paths, which can result in bias and increased WER.

High-level Language Modelling for Related Tasks

Automatic speech recognition (ASR) is just one of several related tasks for which a parsing model, such as that implemented in this work, may be useful as a language model. Optical character recognition (OCR) is another area where the true word sequence is often ambiguous. Character-level bigram probabilities are often used in OCR tasks such as recognizing postal addresses, reading data from cheques, etc. For applications of OCR where the text contains full sentences (*i.e.*, document scanning), a language model can be used to find the most likely sentence. The sequential coupling paradigm could also be applied to this problem. A simple model could be used to generate possible sentences in the form of a list or word lattice, and a high-level language model could rescore or filter the hypotheses. Application of a parsing model to written text would likely work better than for ASR, as typed text does not contain silences or disfluencies such as filled pauses and self-corrections. Scaling up to real-world text would not be as problematic as scaling to real-world speech, as written text is more often grammatical, and similar in style to the corpora available to train parsing models.

Unlexicalized Parsing for Speech Recognition

Klein and Manning (2003) reports competitive parsing scores for an unlexicalized probabilistic parser. The parsing framework developed in this work could be adapted to incorporate such new unlexicalized models. A mixture of unlexicalized parsing with N-gram scores would likely reduce WER significantly, as knowledge used in unlexicalized parsing has a degree of independence from that used by an N-gram model. Additionally, unlexicalized parsing is more portable to other domains, as most domain-specific information is found in the lexicon. When compared to lexicalized systems for parsing word lattices (*e.g.*,, the parsers of this work and Hall and Johnson (2003)), such a system has the potential to be simpler, faster and more portable.

Word-Predictive Language Modelling

The parser developed in this work does not predict words in the manner of traditional language-models (*i.e.*, we do not calculate next-word probabilities $\mathcal{P}(w_i|w_1 \ldots w_{i-1})$). Future work could focus on formalizing the parsing approach as a strictly word-predictive language model, which would search for the best word sequence instead of the best parse tree. Chelba (2000) presents a general approach to such formalization. Note that this may result in reduced parse quality. Choice of search strategy (best parse or best word sequence) should be made based on the goals of the task.

Incorporation of Additional Features

The parser developed in this work could be combined with a speech recognizer which uses prosody to detect phrasal boundaries (punctuation). Word lattices annotated with phrasal boundaries could be useful — in text, punctuation is an important factor for accurate parsing.

New Corpora and Evaluation Metrics

The NIST HUB-1 corpus, with 213 samples, is rather small. We encourage development of a corpus of word lattices along with annotated parse trees for the true utterances. This would allow simultaneous testing of WER and PARSEVAL scores on the same data. Such a corpus could be pruned to N-best paths using dynamic selection of N as described in this work. Additionally, an alternative for the PARSEVAL metrics could be developed for speech, where the number of words may differ between the reference and proposed parses.

Appendix A

Details of the Parsing Algorithm

A.1 Algorithms for Chart Closure

```
closure() {
    while (agenda not empty) {
        edge = agenda.pop();
        // chart.add applies dynamic prog. constraints, may reject edge
        if (chart.add(edge)) {
            unaryExtend(edge);
            adjacentEdges = chart.node(edge.right());
            for each (leftEdge in adjacentEdges) {
                binaryAdjoin(leftEdge, edge);
            }
        }
    }
}
```

Figure A.1: Chart closure algorithm.

```
unaryExtend(edge) {
    // newEdge properties and weight set according to table A.2 during
    // creation of newEdge
    switch(edge.type) {
        case STOP {
            possibleParents = parent(edge.H);
            for each (parent in possibleParents) {
                if (parent == NPB)
                    newEdge = new LEFT_NPB(child=edge,P=NPB);
                else {
                    possibleSubcats = subcat(P,H);
                    for each (subcat in possibleSubcats) {
                        newEdge = new LEFT(child=edge,P=parent,S=subcat);
                    }
                }
            }
        } case LEFT {
            possibleSubcats = subcat(P,H);
            for each (subcat in possibleSubcats) {
                newEdge = new RIGHT(child=edge,P=parent,S=subcat);
            }
        } case LEFT_NPB {
            newEdge = new RIGHT_NPB(child=edge,P=NPB);
        } case RIGHT {
            newEdge = STOP(child=edge,H=child.P);
        } case RIGHT_NPB {
            newEdge = STOP(child=edge,H=child.P);
        }
        newEdge.t = newEdge.w + outside(newEdge);
    }
    agenda.add(newEdge);
}
```

Figure A.2: Unary edge extension algorithm.

```
binaryAdjoin(leftEdge,rightEdge) {
    // newEdge type and parameters as listed in binary tables
    // A.3-A.4.  newEdge weight set during edge creation.
    if (leftEdge.type == COORD) {
        if (rightEdge.type == LEFT) {
            // LEFT -> COORD LEFT
            newEdge =
              LEFT(c1=leftEdge,c2=rightEdge,c=leftEdge.W)
        else if (rightEdge.type == LEFT_NPB) {
            // LEFT_NPB -> COORD LEFT_NPB
            newEdge =
              LEFT_NPB(c1=leftEdge,c2=rightEdge,c=leftEdge.W)
    .
    .
    .
    }

    newEdge.t = newEdge.w + outside(newEdge);
    agenda.add(newEdge);
}
```

Figure A.3: Binary edge creation algorithm. A sample of the creation of new edges is shown. New edges are created depending on the types of the `leftEdge` and `rightEdge`, using binary edge creation tables from section A.2.

A.2 Edge Creation Rules and Parameters

New Edge	Edge Category		Comments
STOP	$P = none$	$H =$ POS	a STOP edge is created for each POS supplied by tagger
	$T =$ POS	$W = c_1.W$	
	$V = c_1.verb$	$D = 0$	
	$c = none$	$p = none$	
	$u = 0$	$subcat = \{\}$	
COORD	$P = none$	$H =$ CC	created for all words with POS=CC
	$T =$ CC	$W = c_1.W$	
	$V = no$	$D = 0$	
	$c = none$	$p = none$	
	$u = 0$	$subcat = \{\}$	
PUNCT	$P = none$	$H = (:$ or, $)$	created for all words with POS={:,}
	$T = (:$ or, $)$	$W = c_1.W$	
	$V = no$	$D = 0$	
	$c = none$	$p = none$	
	$u = 0$	$subcat = \{\}$	

Table A.1: Initialization: Creation of FINAL edges from WORD edges. A WORD edge (c_1) is supplied to the parser with a list of POS candidates. Edges of the FINAL group are created and added to the AGENDA. The weight of the new edge is initialized to $\alpha a \cdot \beta lm$, using acoustic and language model (*i.e.*, trigram) scores from the input lattice.

Edges	New Edge Category	Parameters
New=LEFT c_1=STOP	$P = parent(c_1.H)$ $S = subcat(P, H)$	$\mathcal{P}_h(H\|P;T;W) \cdot \mathcal{P}_{lc}(subcat\|H,P;T;W)$
New=LEFT_NPB c_1=STOP	P = NPB $H_{-1} = c_1.H$ $T_{-1} = c_1.T$ $W_{-1} = c_1.W$	$P_h(H\|P;T;W)$
New=RIGHT [a] c_1=LEFT	$S = subcat(P, H)$	$\mathcal{P}_{ls}(STOP\|c_1.D,P,H;T;W) \cdot$ $\mathcal{P}_{cc}(c\|P,null,H;null,T;null,W) \cdot$ $\mathcal{P}_p(p\|P,null,H;null,T;null,W) \cdot$ $\mathcal{P}_{rc}(S\|H,P;T;W)$
New=RIGHT_NPB c_1=LEFT_NPB	P = NPB $H_{-1} = c_1.H$ $T_{-1} = c_1.T$ $W_{-1} = c_1.W$	$\mathcal{P}_{NPBls}(STOP\|H_{-1};T_{-1};W_{-1}) \cdot$ $\mathcal{P}_{cc}(c\|P,null,H;null,T;null,W) \cdot$ $\mathcal{P}_p(p\|P,null,H;null,T;null,W)$
New=STOP [b] c_1=RIGHT	$P = none$ $H = c_1.P$ $u = c_1.u + 1$	$\mathcal{P}_{rs}(STOP\|c_1.D,P,c_1.H;T;W) \cdot$ $\mathcal{P}_{cc}(c\|P,H,null;T,null;W,null) \cdot$ $\mathcal{P}_p(p\|P,H,null;T,null;W,null)$
New=STOP [b] c_1=RIGHT_NPB	$P = none$ $H = c_1.P$ $u = c_1.u + 1$	$\mathcal{P}_{NPBrs}(STOP\|H_{-1};T_{-1};W_{-1}) \cdot$ $\mathcal{P}_{cc}(c\|P,H,null;T,null;W,null) \cdot$ $\mathcal{P}_p(p\|P,H,null;T,null;W,null)$

[a] only created if $c_1.S = \{\}$
[b] only created if $u <$ maximum unary chain length

Table A.2: Unary extension. An edge (c_1) is extended (bottom-up) using the unary rules of the parser grammar. The weight of the new edge is: $w = c_1.w \cdot \mathcal{P}_{Pr}(P,T,W) \cdot parameters$, where $parameters$ are given in the table. V, T, and W are passed up from c_1 in all cases. Fields c and p are reset to $none$ and D is set to 0. Except where specified, P, H, and u are passed up from c_1, and $subcat$ is reset to $\{\}$. $P = parent(c_1.H)$ indicates an edge is created for all possible P for the given H. $S = subcat(P,H)$ indicates an edge is created for all possible frames for the given P, H. Probabilities of incomplete punctuation and coordination (sections 3.2.5 and 3.2.6) are included.

Edges	New Edge Category and Parameters	
New=LEFT c_1=COORD	$c = c_1.W$	
c_2=LEFT	**Parameters:** None	
New=LEFT_NPB c_1=COORD	$c = c_1.W$	
c_2=LEFT_NPB	**Parameters:** None	
New=LEFT c_1=PUNCT	$p = c_1.W$	
c_2=LEFT	**Parameters:** None	
New=LEFT_NPB c_1=PUNCT	$p = c_1.W$	
c_2=LEFT_NPB	**Parameters:** None	
New=LEFT [a] c_1=STOP (complement)	$D = add(c_2.D, c_1.V)$ $S = c_2.S - c_1.H$ $p = none$	$V = (c_1.V$ or $c_2.V)$ $c = none$
c_2=LEFT	**Parameters:**[bc] $\mathcal{P}_{cc} \cdot \mathcal{P}_p \cdot$ $\mathcal{P}_{l1}(c_1.H, c_1.T, c_2.c, c_2.p \vert c_2.S, c_2.D, c_2.P, c_2.H; c_2.T; c_2.W) \cdot$ $\mathcal{P}_{l2}(c_1.W \vert c_1.T; c_1.H, c_2.c, c_2.p, c_2.S, c_2.D, c_2.P, c_2.H, c_2.T; c_2.W)$	
New=LEFT c_1=STOP (adjunct)	$D = add(c_2.D, c_1.V)$ $S = c_2.S$ $p = none$	$V = (c_1.V$ OR $c_2.V)$ $c = none$
c_2=LEFT	**Parameters:** $\mathcal{P}_{cc} \cdot \mathcal{P}_p \cdot$ $\mathcal{P}_{l1}(c_1.H, c_1.T, c_2.c, c_2.p \vert c_2.S, c_2.D, c_2.P, c_2.H; c_2.T; c_2.W) \cdot$ $\mathcal{P}_{l2}(c_1.W \vert c_1.T; c_1.H, c_2.c, c_2.p, c_2.S, c_2.D, c_2.P, c_2.H, c_2.T; c_2.W)$	
New=LEFT_NPB c_1=STOP c_2=LEFT_NPB	$V = (c_1.V$ OR $c_2.V)$ $T_{-1} = c_1.T$ $c = none$	$H_{-1} = c_1.H$ $W_{-1} = c_1.W$ $p = none$
	Parameters: $\mathcal{P}_{cc} \cdot \mathcal{P}_p \cdot$ $\mathcal{P}_{NPBl1}(c_1.H, c_1.T, c_2.c, c_2.p \vert c_2.H_{-1}; c_2.T_{-1}; c_2.W_{-1}) \cdot$ $\mathcal{P}_{NPBl2}(c_1.W \vert c_1.T; c_1.H, c_2.c, c_2.p, c_2.H_{-1}, c_2.T_{-1}; c_2.W_{-1})$	

[a] Only created if $c_1.H \in c_2.S$.
[b] $\mathcal{P}_{cc} = \mathcal{P}_{cc}(c_2.c \vert c_2.P, c_1.H, c_2.H; c_1.T, c_2.T; c_2.W, c_2.W)$
[c] $\mathcal{P}_p = \mathcal{P}_p(c_2.p \vert c_2.P, c_1.H, c_2.H; c_1.T, c_2.T; c_2.W, c_2.W)$

Table A.3: Binary adjunction creating EXTEND[LEFT] edges. Edges (c_1 and c_2) are joined, creating a new edge. The weight of the new edge is: $w = c_1.w \cdot c_2.w \cdot \mathcal{P}_{Pr}(P, T, W) \cdot$ *parameters*. Category fields are passed up from c_2, except where specified. u is reset to 0 for the new edge.

Edges	New Edge Category and Parameters	
New=RIGHT c_1=RIGHT	$c = c_2.W$	
c_2=COORD	**Parameters:** None	
New=RIGHT_NPB c_1=RIGHT_NPB	$c = c_2.W$	
c_2=COORD	**Parameters:** None	
New=RIGHT c_1=RIGHT	$p = c_2.W$	
c_2=PUNCT	**Parameters:** None	
New=RIGHT_NPB c_1=RIGHT_NPB	$p = c_2.W$	
c_2=PUNCT	**Parameters:** None	
New=RIGHT [a] c_1=RIGHT c_2=STOP	$D = add(c_1.D, c_2.V)$	$V = (c_1.V$ or $c_2.V)$
	$S = c_1.S - c_2.H$	$c = none$
(complement)	$p = none$	
	Parameters:[bc] $\mathcal{P}_{cc} \cdot \mathcal{P}_p \cdot$	
	$\mathcal{P}_{r1}(c_2.H, c_2.T, c_1.c, c_1.p \vert c_1.S, c_1.D, c_1.P, c_1.H; c_1.T; c_1.W) \cdot$	
	$\mathcal{P}_{r2}(c_2.W \vert c_2.T; c_2.H, c_1.c, c_1.p, c_1.S, c_1.D, c_1.P, c_1.H, c_1.T; c_1.W)$	
New=RIGHT c_1=RIGHT c_2=STOP	$D = add(c_1.D, c_2.V)$	$V = (c_1.V$ OR $c_2.V)$
	$S = c_1.S$	$c = none$
(adjunct)	$p = none$	
	Parameters: $\mathcal{P}_{cc} \cdot \mathcal{P}_p \cdot$	
	$\mathcal{P}_{r1}(c_2.H, c_2.T, c_1.c, c_1.p \vert c_1.S, c_1.D, c_1.P, c_1.H; c_1.T; c_1.W) \cdot$	
	$\mathcal{P}_{r2}(c_2.W \vert c_2.T; c_2.H, c_1.c, c_1.p, c_1.S, c_1.D, c_1.P, c_1.H, c_1.T; c_1.W)$	
New=RIGHT_NPB c_1=RIGHT_NPB c_2=STOP	$V = (c_1.V$ OR $c_2.V)$	$H_{-1} = c_2.H$
	$T_{-1} = c_2.T$	$W_{-1} = c_2.W$
	$c = none$	$p = none$
	Parameters: $\mathcal{P}_{cc} \cdot \mathcal{P}_p \cdot$	
	$\mathcal{P}_{NPBr1}(c_2.H, c_2.T, c_1.c, c_1.p \vert c_1.H_{-1}; c_1.T_{-1}; c_1.W_{-1}) \cdot$	
	$\mathcal{P}_{NPBr2}(c_2.W \vert c_2.T; c_2.H, c_1.c, c_1.p, c_1.H_{-1}, c_1.T_{-1}; c_1.W_{-1})$	

[a] Only created if $c_2.H \in c_1.S$.
[b] $\mathcal{P}_{cc} = \mathcal{P}_{cc}(c_1.c \vert c_1.P, c_1.H, c_2.H; c_1.T, c_2.T; c_2.W, c_2.W)$
[c] $\mathcal{P}_p = \mathcal{P}_p(c_1.p \vert c_1.P, c_1.H, c_2.H; c_1.T, c_2.T; c_2.W, c_2.W)$

Table A.4: Binary adjunction creating EXTEND[RIGHT] edges. Edges (c_1 and c_2) are joined, creating a new edge. The weight of the new edge is: $w = c_1.w \cdot c_2.w \cdot \mathcal{P}_{Pr}(P, T, W) \cdot$ *parameters*. Category fields are passed up from c_1, except where specified. u is reset to 0 for the new edge.

Bibliography

L. R. Bahl, F. Jelinek, and R. L. Mercer. 1983. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5:179–190.

Mary E. Beckman. 1997. *A Typology of Spontaneous Speech*, chapter 2. Springer-Verlag, New York, U.S.A.

E. Black, S. Abney, D. Flickenger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of english grammars. In *Proceedings of Fourth DARPA Speech and Natural Language Workshop*, pages 306–311.

E. Black, F. Jelinek, J. Lafferty, D. Magerman, R. Mercer, and S. Roukos. 1992. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 5th DARPA Speech and Natural Language Workshop*, Harriman, U.S.A.

Hervé Bourlard, Hynek Hermansky, and Nelson Morgan. 1996. Towards increasing speech recognition error rates. *Speech Communication*, 18:205–231.

Eric Brill, Radu Florian, John C. Henderson, and Lidia Mangu. 1998. Beyond N-grams: Can linguistic sophistication improve language modeling? In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics*, pages 186–190, San Francisco, U.S.A.

Sharon Caraballo and Eugene Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298, June.

Bob Carpenter. 2000. Probabilistic graph parsing: A framework for natural language syntactic analysis. Documentation with supplied Java source code.

J.-C. Chappelier and M. Rajman. 1998. A practical bottom-up algorithm for on-line parsing with stochastic context-free grammars. Technical Report 98-284, Swiss Federal Institute of Technology, July.

J.-C. Chappelier, M. Rajman, R. Aragües, and A. Rozenknop. 1999. Lattice parsing for speech recognition. In *6eme Conference sur le Traitement Automatique du Langage Naturel*, pages 95–104, July.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 2000 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 132–129, New Brunswick, U.S.A.

Eugene Charniak. 2001. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics.*

Eugene Charniak, Don Blaheta, Niyu Ge, Keith Hall, John Hale, and Mark Johnson. 1999. *BLLIP 1987-89 WSJ Corpus Release 1.* Linguistic Data Consortium.

Ciprian Chelba. 2000. *Exploiting Syntactic Structure for Natural Language Modeling.* Ph.D. thesis, Johns Hopkins University.

Ciprian Chelba and Frederick Jelinek. 2000. Structured language modeling. *Computer Speech and Language*, 14:283–332.

Michael Collins. 1997. Three generative, lexicalized models for statistical parsing. In Philip R. Cohen and Wolfgang Wahlster, editors, *Proceedings of the 35th Annual Meet-*

ing of the Association for Computational Linguistics, pages 16–23, Somerset, U.S.A. Association for Computational Linguistics.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing.* Ph.D. thesis, University of Pennsylvania.

David Goddeau. 1992. Using probabilistic shift-reduce parsing in speech recognition systems. In *Proceedings of ICSLP-92*, volume I, pages 321–324.

Joshua Goodman. 1997. Global thresholding and multiple-pass parsing. In *Proceedings of the 2nd Conference on Empirical Methods in Natural Language Processing.*

Keith Hall and Mark Johnson. 2003. Language modeling using efficient best-first bottom-up parsing. In *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop.*

M. Harper, L. Jamieson, C. Mitchell, G. Ying, S. Potisuk, P. Srinivasan, R. Chen, C. Zoltowski, L. McPheters, B. Pellom, and R. Helzerman. 1994. Integrating language models with speech recognition. In *Proceedings of the American Association for Artificial Intelligence Workshop on Integration of Natural Language and Speech Processing.*

F. Jelinek. 1990. *Readings in Speech Recognition*, chapter Self-organized Language Modeling for Speech Recognition. Morgan Kaufmann.

Frederick Jelinek. 1969. A fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, 13:675–685.

Frederick Jelinek. 1997. *Information Extraction From Speech And Text.* MIT Press.

Daniel Jurafsky and James H. Martin. 2000. *Speech and Language Processing.* Prentice Hall, New Jersey, U.S.A., 1 edition.

Daniel Jurafsky, Chuck Wooters, Jonathon Segal, Andreas Stolcke, Eric Fosler, Gary Tajchman, and Nelson Morgan. 1995. Using a stochastic context-free grammar as a language model for speech recognition. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 189–192.

K. Kita, T. Kawabata, and H. Saito. 1989. HMM continuous speech recognition using predictive LR parsing. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 703–706.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, Sapporo, Japan.

Ralf Kompe. 1997. *Prosody in Speech Understanding Systems*, volume 1307 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, Germany.

D. R. Ladd and A. Cutler, editors. 1983. *Models and measurements in the study of prosody*, chapter 2. Springer-Verlag, Berlin, Germany.

Alon Lavie and Masaru Tomita. 1993. Efficient generalized LR parsing of word lattices. In *Bar-Ilan Symposium on Foundations of Artificial Intelligence*, June.

L. Levin, O. Glickman, Y. Qu, D. Gates, A. Lavie, C. Rose, C. Van Ess-Dykema, and A. Waibel. 1995. Using context in machine translation of spoken language. In *Proceedings of the Theoretical and Methodological Issues in Machine Translation Workshop*, Leuven, Belgium.

David Magerman. 1994. *Natural Language Parsing as Statistical Pattern Recognition*. Ph.D. thesis, Stanford University.

Lidia Mangu, Eric Brill, and Andreas Stolcke. 1999. Finding consensus among words:

Lattice-based word error minimization. In *Proceedings of Eurospeech*, pages 495–498, Budapest, Hungary.

Lidia Mangu, Eric Brill, and Andreas Stolcke. 2000. Finding consensus in speech recognition: Word error minimization and other applications of confusion networks. *Computer Speech and Language*, 14(4):373–400.

Robert Moore, Fernando Pereira, and Hy Murveit. 1989. Integrating speech and natural-language processing. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 243–247.

Hwee Tou Ng and John Zelle. 1997. Corpus-based approaches to semantic interpretation in natural language processing. *AI Magazine*, 18:45–54.

Stefan Ortmanns and Hermann Ney. 1997. A word graph algorithm for large vocabulary continuous speech recognition. *Computer Speech and Language*, 11:43–72.

Thomas S. Polzin and Alex H. Waibel. 1998. Detecting emotions in speech. In *Cooperative Multimodal Communication*, Tilburg, The Netherlands.

A. Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Conference on Empirical Methods in Natural Language Processing*, May.

Mosur K. Ravishankar. 1997. Some results on search complexity vs accuracy. In *DARPA Speech Recognition Workshop*, pages 104–107, February.

Manny Rayner, David Carter, Vassilios Digalakis, and Patti Price. 1994. Combining knowledge sources to reorder N-best speech hypothesis lists. In *Proceedings of the Human Language Technology Workshop*, pages 217–221.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(3):249–276.

Brian Roark. 2002. Markov parsing: Lattice rescoring with a statistical parser. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 287–294.

David Roussel and Ariane Halber. 1997. Filtering errors and repairing linguistic anomalies for spoken dialogue systems. In *Association for Computational Linguistics Workshop on Spoken Dialogue Systems*, Madrid, Spain, July.

Nicholas Roy, Joelle Pineau, and Sebastian Thrun. 2000. Spoken dialogue management using probabilistic reasoning. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, Hong Kong, October.

Stephanie Seneff. 1992. TINA: A natural language system for spoken language applications. *Computational Linguistics*, 18(1):61–86.

Khalil Sima'an. 1996. Computational complexity of probabilistic disambiguation by means of tree-grammars. In *Proceedings of the 16th International Conference on Computational Linguistics*, pages 1175–1180, Copenhagen, Denmark.

Andreas Stolcke and Jonathan Segal. 1994. Precise n-gram probabilities from stochastic context-free grammars. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, pages 74–49, Las Cruces, U.S.A.

Marc Swerts, Diane Litman, and Julia Hirschberg. 2000. Corrections in spoken dialogue systems. In Yadong Lu, editor, *Proceedings of the Sixth International Conference on Spoken Language Processing*, Beijing, China, October.

Ann Taylor, Mitchell Marcus, and Beatrice Santorini. 2003. *The Penn TreeBank: An Overview*, chapter 1. Kluwer, Dordrecht, The Netherlands.

Sven Wachsmuth, Gernot A. Fink, and Gerhard Sagerer. 1998. Integration of parsing

and incremental speech recognition. In *Proceedings of the European Signal Processing Conference*, volume 1, pages 371–375, Rhodes, Greece, September.

Hans Weber. 1994. Time synchronous chart parsing of speech integrating unification grammars with statistics. In L. Boves and A. Nijholt, editors, *Proceedings of the 8th Twente Workshop on Language Technology*, pages 107–119, Twente, December.

Hans Weber, Jörg Spilker, and Günther Görz. 1997. Parsing n best trees from a word lattice. *Kunstliche Intelligenz*, pages 279–288.

Peng Xu, Ciprian Chelba, and Frederick Jelinek. 2002. A study on richer syntactic dependencies in structured language modeling. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 191–198.

Sheryl R. Young, Alexander G. Hauptmann, Wayne H. Ward, Edward T. Smith, and Philip Werner. 1989. High level knowledge sources in usable speech recognition systems. *Communications of the ACM*, 32(2):183–194, February.

GuoDong Zhou and K. T. Lua. 1999. Using PCFG in mandarin continuous speech recognition. *Computer Processing of Oriental Languages*, 12(3):285–307.

Victor Zue, James Glass, David Goodine, Hong Leung, Michael Phillips, Joseph Polifroni, and Stephanie Seneff. 1991. Integration of speech recognition and natural language processing in the MIT VOYAGER system. In *IEEE ICASSP-91*, volume I, pages 713–716.