Incremental Segmentation and Annotation Strategies for Real-time Natural Language Processing Applications

Mahsa Yarmohammadi

Presented to the Center for Spoken Language Understanding
within the Oregon Health & Science University
School of Medicine
in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy
in
Computer Science & Engineering

September 2016

Center for Spoken Language Understanding
School of Medicine
Oregon Health & Science University

CERTIFICATE OF APPROVAL

This is to certify that the Ph.D. dissertation of
Mahsa Yarmohammadi
has been approved.

Dr. Brian Roark,
Research Scientist, Google Inc.

Dr. Steven Bedrick
Assistant Professor, OHSU

Dr. Richard Sproat
Research Scientist, Google Inc.

Dr. Srinivas Bangalore
Lead Inventive Scientist, Interactions Labs

Dr. Peter Heeman
Associate Professor, OHSU

Dedication

I dedicate this dissertation to my family.

Acknowledgments

I would like to express my special appreciation and thanks to my advisor, Brian Roark, for guiding and supporting me over the years, and allowing me to grow as a researcher. I am very grateful for his support, patience, motivation, and immense knowledge in Natural Language Processing, that make him a great advisor. I have learned a lot from him, both technically and personally, and without his guidance I could not have finished this dissertation successfully.

Many thanks to the other members of my committee, Richard Sproat, Srinivas Bangalore, Steven Bedrick, and Peter Heeman. Their invaluable discussions and collaborations during my PhD study have provided me motivation and enlightened my educational pathway. Their thoughtful comments have vastly improved the quality of this dissertation.

I would like to acknowledge the former and current CSLU faculty, in particular Jan van Santen and Izhak Shafran. Thank you to my fellow graduate students at CSLU, especially Aaron Dunlop, Nate Bodenstab, Kristy Hollingshead, Emily Prud'hommeaux, Eric Morley, Andrew Fowler, Russ Beckley, Maider Lehr, Joel Adams, Meg Mitchell, Ethan Selfridge, Rebecca Lunsford, Brian Bush, Brian Snider, Shiran Dudy, Archana Machireddy, Meysam Asgari, Alireza Bayesteh, Mahsa Elyasi, Hamidreza Mohammadi, and Golnar Sheikhshab. Thanks to Patricia Dickerson for her great administrative support. I also would like to appreciate my colleagues at industry, especially Vivek Kumar Rangarajan Sridhar, for co-mentoring me during my summer internship at AT&T Research Labs, and all my colleagues at Intel corporation, especially Michael Deisher, Sylvia Downing, and Alberto Martinez, for being so flexible with me to finalize this dissertation.

This journey would not have been possible without the love, patience, and emotional and technical support of my husband, Masoud Rouhizadeh, and endless support and inspiration from my mom, my sisters Mahshid and Bita, and my brother Behrouz. I am eternally grateful for you. I also thank Masoud's family for their warm and continuos encouragements.

Contents

List of Tables

List of Figures

Abstract

Incremental Segmentation and Annotation Strategies for Real-time Natural Language Processing Applications

Mahsa Yarmohammadi

Doctor of Philosophy
the Center for Spoken Language Understanding within
the Oregon Health & Science University
School of Medicine

September 2016
Thesis advisor: Dr. Brian Roark

The input data to a real-time natural language processing application, such as a simultaneous speech-to-speech translation system, is received as a continuous stream and there is often no boundary between the units that are appropriate to process by the application. To start real-time processing, the application requires pieces or segments of the stream input that are separated at appropriate positions. In this thesis we propose *hedge parsing*, a fast incremental syntactic parsing approach which enables syntax-aware stream input segmentation. As opposed to full syntactic parsing of the input, hedge parsing is able to parse incomplete data, which makes it suitable for real-time scenarios. In contrast to shallow parsing of the input which only provides bracketing information, hedge parsing annotates full hierarchical structure of the input up to a maximum constituent span. The processing application may improve its performance by benefiting from such syntactic

annotation of the input segments. Using a state-of-the-art non-incremental full syntactic parser, we could achieve an order of magnitude speed up in finding about $\frac{3}{4}$ of the original constituents of the input, with around 90% accuracy. We then improved the latency of hedge parsing, with a slight accuracy degradation, using an incremental parsing framework. To investigate the benefit of hedge parsing in real-time applications, we showcased incorporating hedge parsing in machine translation. We demonstrated that hedge segmentation and annotation may achieve an acceptable translation accuracy/latency trade-off compared to alternative methods. In particular, it remarkably outperforms shallow parsing, emphasizing the advantage of applying some degree of syntax, rather than just shallow chunking structures, in real-time natural language processing applications that require fast analysis of the input.

Chapter 1

Introduction

Simultaneous speech-to-speech translation (SST) is the challenging task of listening to source language speech, and at the same time, producing target language speech. A human interpreter starts to produce target language utterances with little delay, while the speaker is still speaking. Of course, the human interpreter should avoid speaking translations that cause mistakes and must be revised later. Similar to simultaneous translation, one of the most identifying features of real-time systems is little delivery latency, that is the output should be produced with little delay after receiving the input. Thus such systems should work *incrementally*, which means they should process parts of the input, even before it has been completed, and produce the output. The complete output is built gradually by processing all parts of the input. Another important feature of real-time systems is that the output should not be revised at a later time. Simply stated, incrementality means that input data is not entirely known, but in a piece by piece fashion, and the processing starts with the pieces, before input is completely known.

One of the biggest challenges in SST, and incremental processing in general, is input *segmentation*. The input data to an SST system is received as a continuous audio stream and there is often no boundary between utterances. The audio stream is then transcribed to text in an automatic speech recognition module. To start real-time translation, the translation module requires pieces or segments of the text input that are separated at appropriate positions. (For more detail, refer to the schematic overview of an SST system presented in Figure 2.17 of Chapter 2.) Since the system is not allowed to revise or replace the output translation, the input units should be non-overlapping and should be

processed sequentially, one after the other. Segmentation is particularly important due to the fact that the granularity of segments has a great impact on the trade-off between latency and translation accuracy. *Latency* is the difference in time or number of words between when the source utterance is spoken and when the translated target utterance is produced. Shorter segments have lower latencies since they can be delivered more quickly to the translation module, and also they can be more quickly translated, but they will likely result in inferior translation accuracy. On the other hand, longer segments will yield better translation quality at the expense of more delay in a real-time scenario.

1.1 Problem Statement

Most of the previous work on input segmentation for SST mainly focused on phrasing of the input based on *heuristics* such as pauses in the speech (Fügen et al., 2007; Bangalore et al., 2012), the location of comma or period in the transcribed text (Rangarajan Sridhar et al., 2013; Matusov et al., 2007), or combined punctuation-based and length-based methods (Cettolo and Federico, 2006). Studies on human interpreters show that they depend on information of a structural nature before they can start translation. Units of meaning upon which the interpreter can start translation is beyond lexical entities, and the input segmentation follows mainly syntactical principles (Pöchhacker, 2002). Thus determining segment boundaries based on syntactic structure of the language can be potentially helpful in SST systems, and real-time NLP processing in general. However, syntax-based input segmentation has been under explored in the current literature of SST and incremental NLP processing.

In addition to determining segment boundaries, syntax could potentially improve the performance of the SST task, or other incremental tasks, by incorporating syntactic *annotations* into the input segments. Several studies (Mi et al., 2008; Liu et al., 2011; Zhang et al., 2011; Tamura et al., 2013) showed that incorporating syntactic annotations into the input of a regular (non-incremental) translation system is helpful in translation performance for many language pairs particularly those with different word orders. However,

applying syntactic information in SST has been less addressed in current literature. Probably the main challenge in incorporating syntax in SST, as well as other incremental tasks, is that conventional phrase-structure parsing is not directly applicable to sub-sentential segments, since parsing methods build fully connected structures over the entire string. Current studies that address using syntax in SST use methods to predict future syntactic constituents and then how to apply this syntactic prediction to machine translation (Oda et al., 2015; Ryu et al., 2006).

In this thesis, we propose a novel partial parsing method for fast and incremental syntactic analysis of the input sentence that 1) is less computationally demanding than a full parser but more effective than a shallow parser in finding recursive syntactic structures, 2) allows for syntax-based segmentation of the input, and 3) incorporates some degree of syntax into the input segments without requiring the entire sentence.

1.2 Thesis Contributions

The contributions of this thesis include: introducing "*hedge parsing*" as a new partial parsing method suitable for incremental segmentation and annotation of the input to real-time NLP applications; providing a detailed framework for incremental hedge parsing; and exploring hedge parsing in machine translation.

Hedge parsing

We introduce "hedge parsing" as an approach to recovering constituents of length up to some maximum span. Hedge parsing provides local internal hierarchical structure of phrases without requiring fully connected parses. Hedge parsing is helpful as an approximation to full parsing when fast, high-precision recovery of syntactic information is needed. We follow the XML community (Brüggemann-Klein and Wood, 2004) in naming structures of this type *hedges* (not to be confused with the rhetorical device of the same name), due to the fact that they are like smaller versions of trees which occur in sequences. A hedge parse tree is a constrained sequence of sub-trees that are connected to the top-most node of the tree. This property of hedge parse trees is unique compared to the

original hierarchically embedded parse trees. Thus, instead of parsing the entire sentence to recover the hedge constituents, we are able to chunk the sentence into segments that correspond to hedges, and parse the segments independently (and in parallel). Similar constraints have been used in dependency parsing (Eisner and Smith, 2005; Dreyer et al., 2006), where the use of hard constraints on the distance between heads and dependents is known as *vine* parsing. It is also reminiscent of so-called Semi-Markov models (Sarawagi and Cohen, 2004), which allow finite-state models to reason about segments rather than just tags by imposing segment length limits. A hedge parser can be used as a standalone partial syntactic parser, or in full parsing for pruning the search space hence increasing the efficiency of the parsing pipeline.

Real-time hedge parsing

To achieve low-latency real-time hedge parsing, we propose an incremental framework based on an incremental parser to return hedge-parsed segments of the input stream simultaneously as the input is being parsed. Our method works by adding a simple and effective algorithm to a commonly-used incremental shift-reduce parser with beam search decoding, without changing the parser or its grammar. The algorithm incrementally extracts stable partial parse trees, which are available where all the beams agree on the partial outputs, and sends them to the processing application while the sentence is still being parsed. In a related work, Selfridge et al. (2011) investigated stability and accuracy of partial phrases in word lattices of an incremental speech recognition task. Kato et al. (2004) proposed a method to decide the stability of a partial parse tree using a probabilistic incremental parser based on tree adjoining grammar. Similar ideas can be used to produce syntactic segments of other types (such as shallow bracketing) for incremental applications.

Hedge parsing in simultaneous translation

We then try to answer two research questions regarding the impact of hedge parsing in machine translation (MT) performance: 1) How does augmenting a translation model with

such partial syntactic annotations affect a regular (non-incremental) translation performance? Previous research showed better translation accuracy in models augmented with full syntax compared to non-syntactic MT models for many language pairs (Galley et al., 2006; Zollmann et al., 2008). Now we investigate augmenting hedge syntax in MT, and compare translation accuracy/efficiency with non-syntactic and full-syntactic MT models. 2) How does hedge segmentation and annotation of the input affect the trade-off between incremental translation latency and accuracy? We compare hedge-based segmentation and annotation with several heuristic based segmentation methods, such as length-based method, on various syntactic and non-syntactic MT models.

We perform hedge parsing with straight-forward changes to the CYK parsing algorithm and grammar. We show that hedge parsing is orders of magnitude faster than full parsing and it can find hedges with high accuracy. Then we show that our real-time hedge parsing approach greatly decreases the delivery latency of hedge-parsed segments with a slight loss in accuracy. Finally, we demonstrate that augmenting a translation model with hedge syntax significantly outperforms translation performance compared to no or shallow syntax in translation model. Also a hedge-syntax augmented translation model combined with hedge parsed input segments, achieves a good accuracy/latency tradeoff in incremental translation.

1.3 Organization of the Dissertation

In the next chapter, we provide an overview of the technical preliminaries of finite-state methods, context-free methods, parsing, and statistical machine translation. We also present some current approaches to fast partial analysis including shallow parsing, Supertagging, and dependency parsing in applications such as simultaneous speech-to-speech translation. We summarize current input segmentation strategies, which are mostly heuristic-based, and our previously proposed segmentation strategy, which takes into account the translation model.

In the following chapters we present our contributions in this dissertation. Chapter 3 describes the hedge parsing approach and different inference (search or decoding) strategies and examines their accuracy/efficiency tradeoffs. In Chapter 4 we introduce our incremental hedge parsing framework and show how it improves hedge parsing latency. In Chapter 5, we explore applying hedge syntax in machine translation and compare it with heuristic-based methods for input segmentation and annotation. Finally, in Chapter 6, we conclude with our contributions and summarize our findings. We outline future work, including using our suggested approaches in a full pipeline of simultaneous speech-to-speech translation.

Chapter 2

Preliminaries and Background

This chapter begins with technical preliminaries that discuss the models and algorithms referenced throughout this dissertation. These sections will define automata and formal grammars (specifically finite-state machines and context-free languages), part-of-speech tagging, log-linear models, parsing, as well as defining algorithms such as the Viterbi, CYK, shift-reduce, and perceptron algorithms.

Following these sections, we will present background information for current approaches to fast partial (syntactic) analysis of the input. We describe shallow parsing (chunking), which is of linear time complexity, and Supertagging, which extends part-of-speech tagging to parsing by using rich-structure tags. We explain dependency parsing, a way of annotating sentence structure using the functional dependencies between words, and a variant of dependency parsing called vine parsing, which pursues a similar idea to hedge parsing, but in dependency parsing rather than constituency parsing. We then summarize research on incremental parsing algorithms, which are potentially suitable for real-time applications, however most of the existing research only provide the algorithm and do not evaluate their approach on an online task. Next, we provide an overview of statistical machine translation. In particular, we describe simultaneous speech-to-speech translation as an example of a real-time NLP application, followed by a summary of current approaches to input segmentation in this task.

2.1 Knowledge of Language

Language processing applications are distinguished from other data processing systems due to their use of *knowledge of language* (Jurafsky and Martin, 2009). Phonetics and Phonology, Morphology, and Syntax are kinds of knowledge that are required to answer how words are pronounced, how words break down into component parts, and how words are ordered and grouped. Moving beyond the structure of language requires Semantics, Pragmatics, and Discourse knowledge to answer what the meaning of words and sentences are, what kind of actions speakers intend by their use of sentences, and how the context beyond the sentence affects the meaning of the sentence. Various kinds of knowledge can be captured using a small number of formal models and theories. In this thesis, we focus on NLP applications which benefit from the structure of language and do not involve knowledge beyond that.

State machines (automata) and their equivalent languages and grammars are the main models for representing phonology, morphology, and syntax. In the next section we explain formal languages and automata and their corresponding grammars. In this thesis we use finite-state equivalent and context-free models as well as our main contribution, hedge parsing, which is a model with expressiveness and complexity in between these two models.

2.1.1 The Chomsky Hierarchy

The Chomsky hierarchy is a theoretical tool that allows us to compare the expressive power or complexity of different formal mechanisms like finite automata, transducers, and context-free grammars. The Chomsky hierarchy is a collection of four classes of formal languages: regular, context-free, context-sensitive and recursively enumerable languages. Figure 2.1 shows a Venn diagram of the four languages on the Chomsky hierarchy.

Each level corresponds to a generating grammar, which produces a language in the associated class, as well as a recognizing machine, which recognizes a language in the associated class. For example, the class of regular languages corresponds to regular grammars and

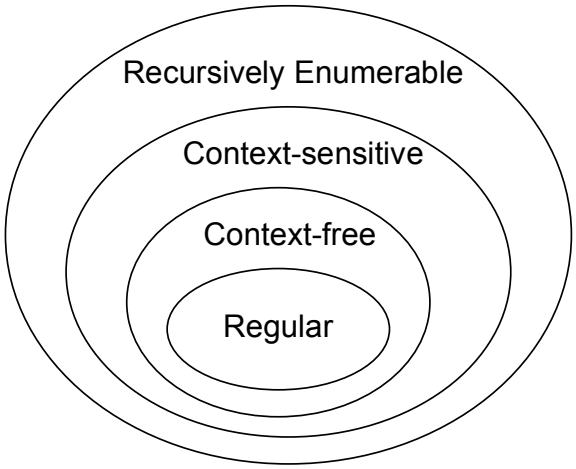


Figure 2.1: The Chomsky hierarchy.

finite-state automata, and the class of context-free grammars can be described by context-free grammars and push-down automata. Each level of this hierarchy is a proper subset of the classes above it, i.e., it has weaker generative power or *complexity* and the rules of the generating grammar are more restrictive than the classes above it. So for example, a context-free grammar can describe formal languages that cannot be described using a finite-state automata.

2.1.2 Finite-state Machines

Finite-state machines define the class of regular languages which can describe a large number of phenomena in natural language including morphological analysis and shallow syntactic structures. Finite-state machines can identify a set of non-overlapping units in a sentence. Sequence tagging problems such as part-of-speech tagging, NP chunking, shallow parsing, Named Entity Recognition, and other related tagging models can be solved using finite-state approaches. There are several dynamic programming algorithms, including Viterbi and Forward-backward algorithms, for efficient inference with finite-state models. In this thesis we mainly use finite-state machines for hedge segmentation in Chapter 3 and shallow parsing in Chapter 4. Phrase-based machine translation in Chapter 4 is also finite-state equivalent.

A finite-state automaton (FSA) is defined by the quintuple $M=(Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of internal states, Σ is a finite set of input alphabet, $\delta{:}Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is a set of final states. The FSA can recognize strings in the following manner. The acceptor starts in the initial state q_0 and checks the first letter of the input string. If the letter matches a transition leaving the state, then the acceptor moves to the next state and also advances one symbol in the input. If the acceptor reaches a final state when it runs out of input, the FSA has successfully recognized a string in the language of automata. If the acceptor never reaches a final state, either due to running out of input or getting some input symbols that does not match a transition, then it rejects the input.

The graph in Figure 2.2 represents the FSA $M_1=(\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, q_1)$ where δ is given by $\delta(q_0, a)=q_0$, $\delta(q_0, b)=q_1$, $\delta(q_1, a)=q_2$, $\delta(q_1, b)=q_2$, $\delta(q_2, a)=q_2$, $\delta(q_2, b)=q_2$. This FSA accepts the string ab. Starting in state q_0 the symbol a is read and the automaton remains in state q_0, then b is read and the automaton goes into state q_1. We are now at the end of the string and in a final state, so the string ab is accepted.

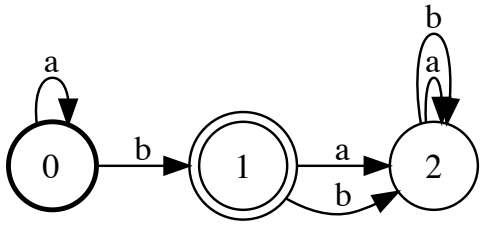


Figure 2.2: An example of a finite-state automaton.

The associated language to a finite-state automaton is the set of all the strings on Σ accepted by the automaton: $L(M_1)=\{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$. The family of languages that is accepted by finite-state acceptors is called regular languages. The regular language accepted by the automaton in Figure 2.2 is $L_1=\{a^n b \mid N \geq 0\}$.

Another equivalent method of characterizing the regular languages is regular grammar. A regular grammar is one that is either right-linear or left-linear. A grammar $G=(V, T, S, P)$, where V is a set of non-terminal symbols (or variables), T is a set of terminal symbols (disjoint from V), S is a designated start symbol, and P is a set of production rules,

is *right-linear* if all production rules are of the form $A \rightarrow xB$, $A \rightarrow x$, where $A, B \in V$ and $x \in T^*$, and it is *left-linear* if all rules are of the form $A \rightarrow Bx$, $A \rightarrow x$. In other words, a rule in a regular grammar has at most one non-terminal on the right-hand side, and that non-terminal must be either the rightmost or leftmost symbol in the string. A right-linear regular grammar for the language in our previous example is $S \rightarrow aS$, $S \rightarrow b$.

2.1.3 Context-free Grammars

Regular languages cannot fully define natural language, e.g. write a grammar for English. We need a greater generative power or complexity than the finite-state methods have. Context-free grammars are used to describe formal languages that cannot be described by FSAs. A CFG consists of a set of production rules and a lexicon of words. Each production rule expresses the ways that words of the language can be grouped together. A grammar $G=(V, T, S, P)$ is *context-free* if all productions in P are of the form $A \rightarrow \alpha$, where $A \in V$ and $\alpha \in (V \cup T)^*$. The item on the left-hand side of each rule is a single non-terminal, and the right-hand side is an ordered list of zero or more terminals and non-terminals. For instance, the grammar $G_2=(\{S\}, \{a, b\}, S, P)$ with productions $S \rightarrow aSb$, $S \rightarrow ab$ is context-free. This grammar generates the context-free language $L(G_2)=\{a^n b^n \mid n \geq 1\}$. This is the simplest grammar to pair up characters (or brackets in programming languages).

Context-free rules can be hierarchically embedded. The rules in a regular grammar are a restricted form of the rules in a context-free grammar. Regular grammar rules cannot express recursive center-embedding rules like $A \rightarrow \alpha A \beta$ where a non-terminal is rewritten as itself, whereas context-free rules can be hierarchically embedded. The context-free languages have a type of automaton that defines them. While this automaton, called a "pushdown automaton", is less commonly used than FSA, it is an extension of FSA with addition of a stack.

The sequence of rule expansions for a given sentence is called a *derivation* of that sentence. A typical derivation in grammar G_2 above is $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$. It is common to represent a derivation by a *parse tree*. Multiple derivations may result in the same parse

$$G_2=(\{S\},\{a,b\},S,P)$$

$$P = \begin{cases} \text{S} \rightarrow aSb \\ \text{S} \rightarrow ab \end{cases}$$

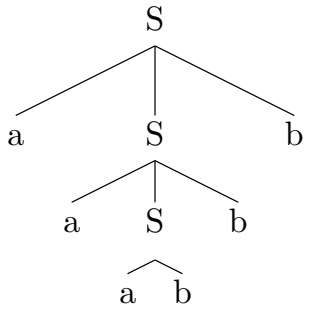


Figure 2.3: (a) A context-free grammar and (b) parse tree for example string *aaabbb*.

tree, differing only in the order of rule expansion. Figure 2.3 shows the parse tree for the string *aaabbb* derivated from grammar G_2.

In this thesis, we use a common variant of CFG called probabilistic CFG, as explained in the next section. The grammars for full parsing and hedge parsing with CYK algorithm, and shallow parsing are instances of probabilistic CFGs.

Probabilistic Context-free Grammars

In practical applications, given a grammar G and string w of terminals, we want to know if w is in $L(G)$, and if so we may want to find a derivation of w. Parsing describes finding a sequence of productions by which $w \in L(G)$ is derived. For any $w \in L(G)$ a number of different parse trees may exist. This situation is referred to as ambiguity. Ambiguity is a common feature of natural languages. One way to solve the problem of syntactic disambiguation, is using a probabilistic CFG (PCFG) for parsing. A PCFG assigns a probability to each parse tree of a sentence. A PCFG differs from a standard CFG by augmenting each rule in P with a conditional probability: $A \rightarrow \alpha$ $[p]$, where p is the probability that the given non-terminal A will be expanded to sequence α. We can represent this probability as $p(A \rightarrow \alpha)$ or as $p(A \rightarrow \alpha | A)$. The sum of probabilities of all the possible expansions of a non-terminal must be 1: $\sum_\alpha p(A \rightarrow \alpha) = 1$. A PCFG assigns a probability to each parse tree of a sentence. This probability is defined as the product of probabilities of all the rules used to derive the parse tree.

Treebanks and PCFG Induction

A PCFG is generally induced from a treebank, a syntactically annotated corpus in which every sentence has a parse tree. By using treebanks as a starting point we can induce grammars with minimal human intervention, and the resulting grammars may have better coverage than the hand-built grammars. One of the simplest ways to learn a PCFG from a treebank is to read the production rules off all the parsed sentences, and assign the probability to each rule by observing how many times it was used in the treebank. The maximum likelihood (relative frequency) estimate for the probability of the rule $A \rightarrow \alpha$ is

$$p(A \rightarrow \alpha) = \frac{count(A \rightarrow \alpha)}{count(A)}$$

In scenarios such as hedge parsing we may transform the original treebank before inducing a grammar, as described in the next section.

Treebank Transformation

There are scenarios where we benefit from changing the treebank or the resulting grammar before parsing. These changes might be intended for using the treebank or the grammar in a specific algorithm (such as binarization to use in the CYK algorithm, as described in the next subsection, or left-corner transform to remove left recursion and use in top-down parsing algorithms) or to improve parsing performance with respect to the parsing efficiency or parsing accuracy.

Improvement in accuracy is achieved through splitting non-terminals into multiple new non-terminals which define a better probability model. One popular treebank transform is parent annotation (Johnson, 1998), which adds the parent label to each non-terminal. Parent annotation refines the probabilities of production rules by adding extra amount of context. More recently Matsuzaki et al. (2005), Petrov et al. (2006), and Petrov and Klein (2007a) proposed latent variable grammars. Each non-terminal in this grammar is annotated with a set of latent variables (e.g., VP_0, VP_1, VP_2), which can be fixed

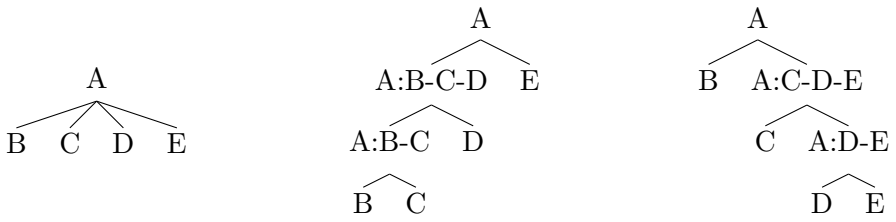


Figure 2.4: (a) Original tree; (b) Left-factoring binarization; (c) Right-factoring binarization

number of latent variables for all non-terminals (Matsuzaki et al., 2005), or determined by a split-merge technique for each non-terminal (Petrov and Klein, 2007a). However, creating new non-terminals, increases the size of production rules and grows the size of the grammar which impacts negatively on parsing efficiency. Moreover, such grammars are prone to the sparsity problem. To address efficiency and sparsity issues, some kind of smoothing such as Markovization (Manning and Schütze, 1999; Roark and Sproat, 2007) is used to merge multiple non-terminals into one. The CYK parser used in Chapter 3 of this thesis, uses the latent variable grammar of Petrov et al. that is learned from the original or hedge-transformed treebank.

Binarization

Treebank or grammar binarization is often used to build an equivalent grammar that works for a parsing algorithm such as CYK or shift-reduce. Two simple binarization are left- and right- factoring methods. Left-factoring of a node n creates a new composite non-terminal by factoring the leftmost children of n, and leaves the last child unchanged. The new composite non-terminal is set as the left child of n, and then it is recursively left-factored until there are no more composite non-terminals. Right-factoring is defined equivalently. Figure 2.4 shows an example of a constituent and its left- and right-factoring transforms. In this thesis, this type of binarization is used to convert PCFGs to their equivalent Chomsky normal form (CNF) in order to use them in the CYK parser.

Sagae and Lavie (2005), Wang et al. (2006a), and Zhang and Clark (2011), used an instance of the transformation/detransformation process described in (Johnson, 1998) in shift-reduce incremental framework for phrase-structure or dependency parsing. The

Figure 2.5: (a) Original tree; (b) Binarized tree (with Markov order-0 smoothing) to use in shift-reduce parser training. The lexical head of the A* nodes is the same as the lexical head of node A.

transformation step converts each node with n (>2) child nodes into $n-1$ binary nodes. New non-terminals introduced in this process are marked with asterisks. The binarized treebank is then used for training an incremental shift-reduce parser. Note that this treebank contains also lexical head annotations and this information is used as features for training the parser. The lexical head of each of the asterisked non-terminals is the same as the head of the original non-terminal. Figure 2.5 shows an example of this binarization method. This type of binarization is used in the shift-reduce parser in this thesis.

2.2 Hidden Markov Models for Tagging

In this section, we provide an overview of one of the most common approaches to tagging with particular focus on part-of-speech (POS) tagging as a very clear and well-known tagging task. We explain Hidden Markov Models (HMMs) method for tagging and the Viterbi algorithm for efficient decoding of tag sequences. POS-tagging is frequently used in early stages of NLP pipelines, e.g. as a prerequisite for parsing (Chapters 3, 4, and 5). In addition, HHMs are used in other tagging tasks in NLP such as segmentation (Chapter 3), sentences boundary detection, shallow parsing or chunking (Chapter 5), named entity recognition, textual entailment, and Supertagging.

2.2.1 Part-of-Speech Tagging

POS tagging is the process of assigning a POS tag to each word in a corpus. The input to a POS tagger is a sequence of words and a specified tagset, and the output is the 1-best

(or n-best) tag sequence of the input. POS tagging distinguishes the word class between main classes, such as noun, verb, pronoun, preposition, adverb, conjunction, participle, and article (Jurafsky and Martin, 2009). For each main class, there are subclasses such as different verb tenses, or whether a noun is singular or plural. The size of the tagset depends on the language and the amount of distinction between tags. For example, the English Penn Treebank corpus (Marcus et al., 1993) has a tagset of size 45 POS tags, including 4 nominal classes for singular/plural and proper/common distinctions, and 7 verb classes for modal (MD), base form verb (VB), past tense (VBD), gerund/present participle (VBG), past participle (VBN), 3rd-person singular present (VBZ), and non-3rd-person singular present (VBP). A complete list of tags and tag descriptions in English Penn Treebank can be found in Appendix A.

2.2.2 Hidden Markov Models

Several methods have been applied to POS tagging including rule-based methods and stochastic methods. Rule-based methods generally involve a set of hand-written disambiguation rules to assign a single POS tag to each word. ENGTWOL is a sample rule-based tagger based on the Constraint Grammar architecture of Karlsson et al. (1995). One of the most common stochastic approaches to POS tagging is the Hidden Markov Model, or HMM. In an HMM-based model, POS tagging is a sequence classification task which, given the observation of a sequence of words, aims to assign it the most likely hidden state sequence of POS tags. Given a word sequence $w_1..w_k$ and a tagset T, the task of POS tagging is to find $\hat{t}_1..\hat{t}_k \in T^k$ such that

$$
\begin{aligned}
\hat{t}_1..\hat{t}_k = & \arg\max_{t_1..t_k \in T^k} P(t_1..t_k|w_1..w_k) \\
= & \arg\max_{t_1..t_k \in T^k} P(w_1..w_k|t_1..t_k)P(t_1..t_k) \\
= & \arg\max_{t_1..t_k \in T^k} \prod_{i=1}^{k} P(t_i|t_0..t_{i-1})P(w_i|t_0..t_i, w_1..w_{i-1}) \\
\approx & \arg\max_{t_1..t_k \in T^k} \prod_{i=1}^{k} P(t_i|t_{i-n}..t_{i-1})P(w_i|t_i)
\end{aligned}
\tag{2.1}
$$

This last approximation, limits the history of the tags to n tags by making a Markov order $n+1$ assumption. It also assumes that the probability of a word given its POS tag is independent of the rest of the words or tags. If n=1, then the we will have a *bigram* HMM tagger which contains a tag transition probability $P(t_i|t_{i-1})$ and observation probability $P(w_i|t_i)$

$$
\hat{t}_1..\hat{t}_k \approx \arg\max_{t_1..t_k \in T^k} \prod_{i=1}^{k} P(t_i|t_{i-1})P(w_i|t_i) \tag{2.2}
$$

In the next section we describe the Viterbi algorithm, an efficient algorithm to find the best tag sequence.

The HMM POS tagging model can be extended to a log-linear model, described in Section 2.3, to allow a variety of features to be used in the model. Various information in the context, including n-grams of surrounding words, n-grams of surrounding tags, and additional orthographical features to tag rare and unknown words, can be captured in a log-linear model through linear combination of weighted features. In this thesis we use the HMM bigram POS taggers with a log-linear model implemented in Hollingshead et al. (2005) and Yarmohammadi (2014) for sequence tagging tasks including POS tagging, hedge segmentation, and shallow parsing. Training the model using large amount of data can be cumbersome. In Yarmohammadi (2014) we investigated distributed training strategies for the structured perceptron introduced by McDonald et al. (2010). They used

word sequence: $W=w_1...w_n$, size of tagset $|T|=m$ for $t=1..n$
For $t=1..n$
    For $j=1..m$
        $\alpha_i(t)=\max_i(\alpha_j(t-1)a_{ij})b_j(w_t)$
        $\zeta_i(t)=\arg\max_i(\alpha_j(t-1)a_{ij})$
$\zeta_0(n+1)=\arg\max_i(\alpha_i(n)a_{i0})$
$\rho(n+1)=0$
For $t=n..1$
    $\rho(t)=\zeta_{\rho(t+1)}(t+1)$
    $\hat{\tau}(t)=\tau_{\rho(t)}$

Figure 2.6: The Viterbi algorithm. Notation adapted from Roark and Sproat (2007).

distributed training to reduce training times in the two tasks of named entity recognition and dependency parsing. We extended that work for another structure prediction task, POS tagging.

2.2.3 Viterbi Algorithm

The Viterbi algorithm is the most common *decoding* algorithm for HMM models, which is determining which sequence of hidden variables is the most likely underlying source of some sequence of observations. The Viterbi algorithm is an application of dynamic programming to find globally optimal solutions by solving a sequence of sub-problems. For a word sequence $w_1..w_n$, a set of m tags $T=\{\tau_i : 1\leq i\leq m\}$, let a_{ij} be the transition probability between hidden states (i.e., tags) defined as $a_{ij}=P(\tau_j|\tau_i)$, and $b_j(w_i)$ be the observation likelihood of word w_i given tag τ_j defined as $b_j(w_i)=P(w_i|\tau_j)$. For simplicity, let $a_{0j}=P(\tau_j|<s>)$ and $a_{i0}=P(</s>|\tau_i)$ where $<s>$ and $</s>$ are the special symbols that only occur at the beginning and end of the sentences respectively.

Figure 2.6 shows pseudocode for the Viterbi algorithm. The algorithm builds a probability matrix, with one column for each observation in time t and one row for each state (tag). The matrix is initialized by 1 in the first cell: $\alpha_0(0)=1$. The algorithm moves column by column, and for each tag at column t, it computes the value $\alpha_j(t)$ by taking the maximum over the extensions of all the paths that lead to the current cell. The three factors that

are multiplied in computing $\alpha_j(t)$ are the previous Viterbi path probability from the previous time step, the transition probability from previous state to current state, and the observation probability of the symbol given the current state. $\zeta_j(t)$ is the backpointer to keep track of where the max came from, and it will be used to reconstruct the maximum likelihood path at the end. The Viterbi algorithm is able to determine the exact solution in linear order of the input sequence length due of the Markov assumption.

2.3 Discriminative Log-linear Models

Collins (2002) introduced a discriminative training framework for the general structural prediction problem of mapping an input structure $x \in X$ onto an output structure $y \in Y$, where X is the set of possible inputs, and Y is the set of possible outputs. For example, for the problem of POS tagging, X is the set of all input sentences and Y is the set of all possible POS tag sequences. For the problem of parsing, X is the set of all sentences and Y is the set of all possible parse trees.

We assume: a set of training examples (x_i, y_i) for $i=1..n$; a function $\mathbf{GEN}(\mathbf{x})$ that enumerates a set of possible outputs for an input x ; $\bar{\alpha} \in \mathbb{R}^d$ a parameter vector; and representation Φ that maps each output structure $y \in Y$ to a global feature vector $\Phi(y)$. There is a mapping from an input x to an output $F(x)$ defined by the formula:

$$F(x) = \arg\max_{y \in \mathbf{GEN}(x)} Score(y) \tag{2.3}$$

where

$$Score(y) = \Phi(y) \cdot \bar{\alpha} \tag{2.4}$$

The model learns the parameter values $\bar{\alpha}$ during the training and the decoding algorithm searches for the y that maximizes 2.3. There are various parameter estimation methods that provide weights α_i for each feature function $\Phi_i(y)$. The sequence tagging models in Chapter 3 and the incremental shift-reduce parser in Chapter 4 of this thesis use the

Inputs: Training examples (x_i, y_i)
Output: Parameters $\bar{\alpha}$
Set $\bar{\alpha}$=0
    For t=1..T
        For i=1..n
            Calculate z_i=arg max$_{z \in \mathbf{GEN}(x_i)}$ $\Phi(z).\bar{\alpha}$
            If $(z_i \neq y_i)$ then $\bar{\alpha}=\bar{\alpha} + \Phi(y_i) - \Phi(z_i)$

Figure 2.7: The perceptron algorithm.

perceptron algorithm, which is described in the next section, to estimate the model parameters. One problem with log-linear models is that the models overfit the training data and are unable to generalize to unseen data. To avoid this problem, some additional techniques are used while training. Averaged perceptron (Collins, 2002) is a standard way to avoid overfitting in the perceptron algorithm.

2.3.1 Perceptron Algorithm

To estimate the parameter values $\bar{\alpha}$ of the model we use the perceptron algorithm. Figure 2.7 shows the perceptron algorithm of Collins (2002). The algorithm initializes the parameter vector as all zeros. For each example, the best-scoring hypothesis z_i is compared with the true hypothesis y_i. If they are different, the algorithm updates the parameter vector $\bar{\alpha}$ by subtracting the features values of z_i to it and adding the feature values of y_i from it. Intuitively, this update is effectively forcing the decoder to produce the correct hypothesis for each training example. The algorithm then moves to the next example. This procedure is repeated for some number of iterations T over the training examples. T is usually determined by choosing the number that gives the highest accuracy on a development set. The regular perceptron algorithm suffers from over-fitting problem. One solution to this problem is the averaged perceptron, which has been shown to generally give improved accuracy over the regular perceptron. In the averaged perceptron, the averaged parameter vector $\bar{\gamma}$ is used instead of $\bar{\alpha}$ as the model parameters. The averaged parameter vector is defined as

$$\bar{\gamma}=\frac{\sum_{i=1..n,t=1..T} \bar{\alpha}^{i,t}}{nT} \tag{2.5}$$

where $\bar{\alpha}^{i,t}$ is the parameter vector immediately after the ith sentence in the tth iteration. The perceptron algorithms used in this thesis apply the averaging technique.

2.3.2 Incremental Structural Prediction

Zhang and Clark (2011) extended the above framework to incremental processing scenarios. The same as the general discriminative log-linear model, they used the averaged perceptron to train the model parameters, however, instead of a dynamic programming decoding algorithm, they used beam-search decoding. The ZPAR shift-reduce parser in Chapters 4 and 5 of this thesis uses the incremental structural prediction with beam-search decoding and the perceptron algorithm described in this section. In a similar work, Collins and Roark (2004) used a global discriminative model and incremental processing for phrase-structure parsing. The major difference is that Collins and Roark, followed a top-down parsing strategy, whereas Zhang and Clark followed a shift-reduce process. In addition, Zhang and Clark's framework did not include a generative baseline model in the discriminative model, as did Collins and Roark.

In Zhang and Clark framework, the structural prediction task (e.g., phrase-structure parsing, dependency parsing, segmentation, POS tagging) is broken into a sequence of decisions. Thus, the output y is built through incremental steps. Suppose that k incremental steps are taken to build y and the incremental change at the ith step ($0<i\leq k$) is $\delta(y,i)$. For shift-reduce parsing, $\delta(y,i)$ can be a shift or reduce action (see Section 2.4.2 for more detail on shift-reduce parsing). The global feature vector at the ith incremental step will be $\Phi(\delta(y,i))$, and the global feature vector $\Phi(y)$ changes to $\Phi(y)=\sum_{i=1}^{k}\Phi(\delta(y,i))$. Therefore, $Score(y)$ can be computed by $Score(y)=\sum_{i=1}^{k}\Phi(\delta(y,i))\cdot\bar{\alpha}$.

At each step of incrementally building the final output, an incremental sub-structure is added to the partially built output. Due to structural ambiguity, different sub-structures can be built. At each step, beam-search decoding keeps a predetermined number of best partial solutions as candidates. To do this, it generates all possible partial solutions and

orders them according to their scores and keeps only the top B of them for next expansions. The greater the beam width, the fewer states are pruned.

2.4 Phrase-structure Parsing

We categorize syntactic phrase-structure (or constituency) parsing methods into two groups: (a) exact inference based on dynamic programming and (b) greedy inference based on a classifier. For the first category, we describe the CYK algorithm and for the second category we explain shift-reduce parsing, which are the two main parsing algorithms we used in this thesis.

2.4.1 CYK Parsing

The CYK algorithm (Cocke and Schwartz, 1970; Kasami, 1965; Younger, 1967) is one of the most widely used methods of syntactic parsing. It employs bottom-up parsing and dynamic programming. The dynamic programming class of algorithms (including the Viterbi algorithm) apply a table-driven method to solve problems by combining solutions to sub-problems. CYK uses a table, referred to as chart, to store intermediate parses as it builds up the parse tree of the given sentence. The standard CYK algorithm operates only on a context-free grammar in Chomsky normal form (CNF). Grammars in CNF are restricted to rules of the form $A \rightarrow B\ C$ or $A \rightarrow w$, i.e., the right-hand side of each rule expands to either two non-terminals or one non-terminal. Any context-free grammar can be converted into a weakly-equivalent (i.e., a grammar that generates the same set of strings) CNF grammar. One of the main steps in CNF conversion is to binarize the rules, which can be performed using the binarization techniques in Section 2.1.3.

By having the rules in CNF form, each non-terminal above the part-of-speech level in a parse tree will have two children. A triangular chart of the upper-triangular portion of a two dimensional $n \times n$ matrix can be used to encode the entire parse structure of an input sentence of length n, as shown in Figure 2.8. The words $w_1, w_2, ..., w_n$ of the input sentence form the base of the chart. Each cell in the chart represents a set of all possible

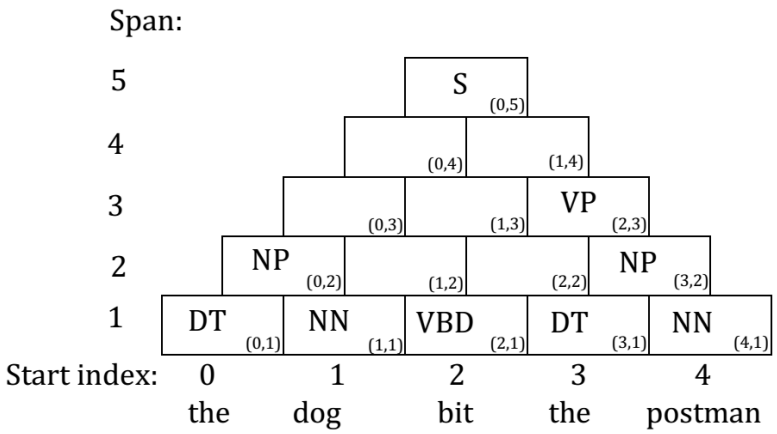


Figure 2.8: A parse represented in a CYK chart.

constituents covering a specific substring (or span). The y-axis of the chart encodes the span of constituents, and the x-axis encodes the start position of constituents. Thus, the cell identified with (s, x) will contain possible constituents covering s words with start index x. In the figure, the cell $(2, 3)$ contains a VP of span three at start position 2, which covers the words "bit the postman" with indices 2, 3, and 4. Filling each cell depends on the cells covering smaller spans within the same substring. The results of each cell are saved so that they can be re-used in building higher cells (longer spans).

Figure 2.9 shows pseudocode of the CYK parsing algorithm. The algorithm begins by initializing span-1 cells with constituents spanning a single word. It then iterates over the rows from the bottom up, typically in a left-to-right manner. At each cell, the algorithm iterates over all the midpoint places m where an input substring with start index x and span s might be split in two. At each split, the algorithm checks if the contents of the two cells can be combined to a new non-terminal as allowed by the grammar. For non-terminals $A_i A_j A_k \in V$, $a_{ijk} = P(A_i \rightarrow A_j A_k)$ is analogous to the transition probabilities in the Viterbi algorithm in Figure 2.6, and $b_j(w_t) = P(A_j \rightarrow w_t)$ is analogous to the observation probabilities. Overall, the algorithm processes $O(n^2)$ cells, and checks $O(n)$ midpoints for each cell, so the complexity of the algorithm is $O(n^3|G|)$, where $|G|$ is the size of the grammar. For each non-terminal in each cell, the algorithm stores $\alpha_i(x, s)$, the probability for non-terminal i with start index x and span s, and $\zeta_i(x, s)$, a backpointer to the children

word sequence: $w_1 ... w_n$, CNF context-free grammar $G=(V, T, S, P)$
s=1
For t=1..n
    x=$t-1$
    For j=1..$|V|$
        $\alpha_j(x, s)$=$b_j(w_t)$
For s=2..n
    For x=0..$n-s$
        For i=1..$|V|$
            $\zeta_i(x, s)$=$\arg\max_{m,j,k} a_{ijk}\alpha_j(x, m-x)a_k(m, s-m+x)$
            $\alpha_i(x, s)$=$\max_{m,j,k} a_{ijk}\alpha_j(x, m-x)a_k(m, s-m+x)$

Figure 2.9: The CYK parsing algorithm. Notation adapted from Roark and Sproat (2007).

of the non-terminal. In Chapter 3 of this thesis, we slightly modify the chart structure and the grammar of the CYK algorithm, and use it for hedge parsing.

2.4.2 Shift-reduce Parsing

Shift-reduce parsing is a bottom-up derivation strategy that can be used to efficiently parse context-free languages in linear time. The main data structures are an input queue of incoming words and a stack. The parser works by doing a series of shifting or reducing operations. Shifting pushes the next word in the queue to the top of the stack and the shifted word becomes a new single-node parse tree (constituent). Reducing pops the top k parse trees from the stack, joins them as one new node, and pushes the new node to the stack. These steps continue until the input queue is empty and the stack contains only a single node which is the top-most non-terminal. Figure 2.10 shows the steps of a shift-reduce parser on our example sentence.

Predicting which action to pursue at each step of parsing is traditionally accomplished with the use of a grammar. If a prefix of the nodes on top of the stack matches the right-hand side of a grammar rule which is the correct rule to use within the current context, the prefix is reduced to the left-hand side of the rule, before the next word is shifted. The grammar may allow more than one valid action at each step of parsing. Recently, classifier-based parsing has been proposed, which makes the action decision by a classifier that chooses

Figure 2.10: Shift-reduce parsing steps.

Step	Stack	Input queue	Action
0		the dog bit the postman	
1	the	dog bit the postman	shift "the"
2	DT	dog bit the postman	reduce (DT→ the)
3	DT dog	bit the postman	shift "dog"
4	DT NN	bit the postman	reduce (NN→ dog)
5	NP	bit the postman	reduce (NP→ DT NN)
6	NP bit	the postman	shift "bit"
7	NP VBD	the postman	reduce (VBD→ bit)
8	NP VBD the	postman	shift "the"
9	NP VBD DT	postman	reduce (DT→ the)
10	NP VBD DT postman		shift "postman"
11	NP VBD DT NN		reduce (NN→ postman)
12	NP VBD NP		reduce (NP→ DT NN)
13	NP VP		reduce (VP→ VBD NP)
14	S		reduce (S→ NP VP)

the action at every step, based on the local parser state with no explicit grammar. In the greedy (or deterministic) variant of the classifier-based parsing only a unique action is chosen at every step, thus only one path is pursued with no backtracking. While the accuracy of a deterministic classifier-based parser is below that of dynamic programming parsing, it is surprisingly good for a greedy parser that runs in linear time (Sagae and Lavie, 2005), and it offers a good alternative for when fast parsing is needed.

The greedy classifier-based approach is sensitive to search errors and subsequent error propagation. To alleviate the effect of search errors and error propagation, Zhang and Clark (2008) proposed using beam search in combination with structured learning. They also found that accuracy could be further improved by using the early-update strategy (Collins and Roark, 2004) during training. The ZPAR incremental shift-reduce parser used in this thesis is an advanced variant of a shift-reduce parser.

2.4.3 Parsing Evaluation

The most widely used parse evaluation metrics are *labeled precision* and *labeled recall*. These are part of the PARSEVAL metrics proposed by Abney et al. (1991), along with

crossing bracket scores. The PARSEVAL metric measures how much the constituents in the hypothesis parse tree look like the constituents in a reference parse which are typically drawn from a treebank like the Penn Treebank.

A given constituent in the hypothesis parse of a sentence is "correct" if there is a constituent in the reference parse with the same span and same non-terminal symbol. Labeled precision (P) and recall (R) of parsing the sentence s is measured as:

$$P = \frac{\text{\# of correct constituents in hypothesis parse of } s}{\text{\# of total constituents in hypothesis parse of } s} \tag{2.6}$$

$$R = \frac{\text{\# of correct constituents in hypothesis parse of } s}{\text{\# of correct constituents in reference parse of } s} \tag{2.7}$$

As with other uses of precision and recall, we combine these scores into a composite score, the F-measure. If precision and recall are equally balanced in the composite F-measure, this is sometime called F_1:

$$F_1 = \frac{2PR}{P+R} \tag{2.8}$$

In the next chapters of this thesis, we use precision, recall, and F1 metrics to evaluate the accuracy of various hedge parsing methods in finding hedge parse trees compared to the reference hedgebank.

2.5 Fast/ Partial Syntactic Parsing

In this section we review existing approaches for fast and/or partial (as opposed to full phrase-structure) parsing with applications in real-time scenarios. These methods can serve as alternatives to hedge parsing, which we proposed and implemented in this thesis. We also review some of the current approaches to incremental parsing using context-free grammars, dependency grammars, and tree-adjoining grammars.

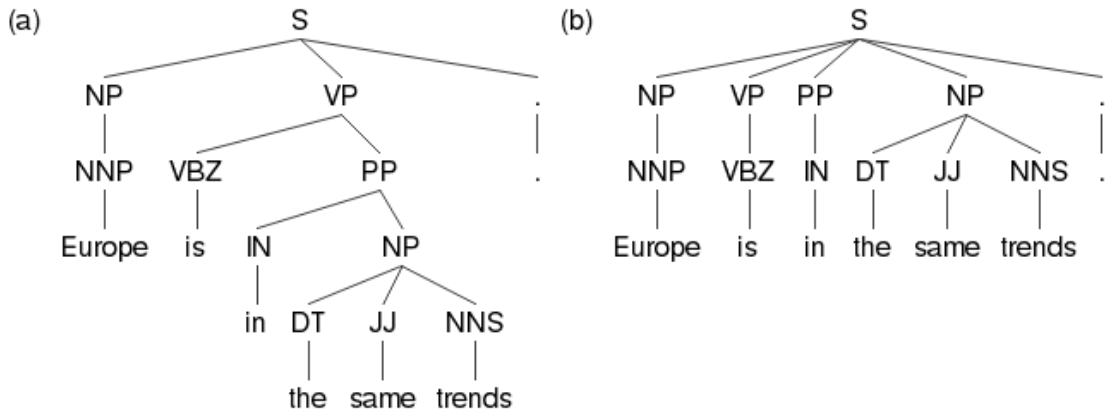
2.5.1 Shallow Parsing (Chunking)

For many language processing tasks, a partial parse of the input may be sufficient. A common style of partial parsing is known as shallow parsing, also known as chunking. Shallow parsing is an analysis of a sentence which identifies the constituents, but does not specify their internal structure or relation with other constituents in the sentence. The set of identified constituents typically includes the phrases that correspond to the content-bearing parts-of-speech such as noun phrases, verb phrases, adjective phrases, and prepositional phrases.[1] Figure 2.11 shows the difference between (a) the full parse tree and (b) the shallow parse trees of an example sentence. Note that shallow parse trees can be represented as bracketed and tagged notations, as shown in Figure 2.11(c). Some applications focus on finding specific categories; such as finding all the base noun phrases in a text, which is called NP-Chunking (Ramshaw and Marcus, 1995). Another common shallow parsing task is the CoNLL-2000 Chunking task (Sang and Buchholz, 2000). This task extends the NP-Chunking task to label eleven different base phrase constituents annotated in the Penn Treebank, including: ADJP, ADVP, CONJP, INTJ, LST, NP, PP, PRT, SBAR, UCP and VP.

Since chunked texts lack a hierarchical structure, a sequence tagging model is directly applicable and sufficient to denote the location and the type of the chunks in a sentence. A tag can be associated with beginning, inside, or outside positions in a chunk, and the standard way to do this is known as IOB tagging, which identifies the beginning (B) and internal (I) parts of each chunk, as well as those tokens that are outside (O) any chunk. Refer to Figure 2.11(c) to see an example. With such an assumption, a shallow parser can be viewed as a classifier that labels each token of a sentence with an IOB tag. Reference shallow parses to train this classifier can be derived from a treebank. Chunking exploits POS tags produced by previous processing steps.

Shallow parsing expressiveness power and complexity falls in the finite-state level in the Chomsky hierarchy. Since shallow parsing can be performed accurately and efficiently

[1] A complete list of POS and phrase-level tags in English Penn Treebank can be found in Appendix A.



c) [NP Europe] [VP is] [PP in] [NP the same trends]
Europe/B is/B in/B the/B same/I trends/I ./O

Figure 2.11: (a) The full parse tree, (b) shallow parse tree, (c) flat bracketing notation and IOB tagged notation of the chunks, for an example sentence.

in linear time, it can be used in a pipeline system to prune the more computationally intensive downstream PCFG parsing. Making use of chunking constraints in a full parsing pipeline have shown an efficiency and/or accuracy gain. Glaysher and Moldovan (2006) achieved efficiency gains by modifying the CYK parsing algorithm to avoid combinations that conflicted with the output of a chunker, i.e., they disallowed constituents in chart cells that crossed chunk boundaries. Hollingshead and Roark (2007) demonstrated significant accuracy improvements in Charniak and Johnson parsing pipeline by using base phrase constraints derived from a shallow parser or from later stages of the same pipeline.

2.5.2 Supertagging

Supertagging extends the part-of-speech tagging to parsing by using very complex tags that are in fact fragments of lexicalized parse trees. Supertagging was originally introduced within the context of Lexicalized Tree-adjoining Grammars (Joshi and Srinivas, 1994), but it has been used for other formalisms like Combinatory Categorial Grammar (CCG, Clark, 2002; Clark and Curran, 2004), Constraint Dependency Grammars (Wang and Harper, 2002; Wang et al., 2003), and Head-driven Phrase Structure Grammar (HPSG,

```
( ("S"
   ("NP-SBJ" ("NNP" "Mr.") ("NNP" "Vinken") )
   ("VP" ("VBZ" "is")
    ("NP-PRD"
     ("NP" ("NN" "chairman") )
     ("PP" ("IN" "of")
      ("NP"
       ("NP" ("NNP" "Elsevier") ("NNP" "N.V.") )
       ("," ",")
       ("NP" ("DT" "the") ("NNP" "Dutch") ("VBG" "publishing") ("NN" "group
") )))))
   ("." ".") ))
```

```
Mr.//NNP//B_Nn              (noun modifier)
Vinken//NNP//A_NXN          (head noun)
is//VBZ//B_Vvx              (auxiliary verb)
chairman//NN//A_nx0N1       (predicative noun)
of//IN//B_nxPnx             (noun-attached preposition)
Elsevier//NNP//B_Nn         (noun modifier)
N.V.//NNP//A_NXN            (head noun)
,//,//B_nxPUnxpu           (appositive comma)
the//DT//B_Dnx              (determiner)
Dutch//NNP//B_Nn            (noun modifier)
publishing//VBG//B_Vn       (participle verb, nominal modifier)
group//NN//A_NXN            (head noun)
.//.//B_sPU                 (sentence punctuation)
```

Figure 2.12: phrase structure tree and obtained supertags for an example sentence. (from Bangalore and Joshi (1999))

Matsuzaki et al., 2007; Blunsom and Baldwin, 2006). Grammars in these frameworks typically associate linguistically motivated rich descriptions (Supertags) with words. By imposing complex constraints in a local context through Supertags, the computation of linguistic structure can be localized. Even when a word has a unique part-of-speech, there will usually be more than one Supertag for this word. Each Supertag corresponds to a different syntactic context in which the lexical item can appear. Figure 2.12 shows the phrase structure tree and obtained Supertags from the phrase structure tree for an example sentence..

Similar to how POS-tagging constrains the down-stream parser, Supertagging is a pre-processing step for (context-sensitive) parsing to filter out inappropriate elementary trees given the context of the sentence. Supertagging has been used as a pruning method to decrease the typical-case runtime of LTAG parsing (Bangalore and Joshi, 1999; Sarkar,

2007) and CCG parsing (Clark, 2002; Clark and Curran, 2004). Supertagging has also been used to incorporate structural information into language modeling (Srinivas, 1996) and showed that it serves as a better language model compared to part-of-speech based language models, since supertag-based classes are more fine-grained than part-of-speech based classes in a class-based language model.

2.5.3 Dependency Parsing

There has been an increase in the use of dependency representations in NLP tasks in recent years due to its high speed and accuracy in full-parsing. In contrast to constituency parsing which describes a sentence by breaking up it into constituents (phrases) which are internally broken into smaller constituents, dependency parsing is a way of describing sentence structure by drawing links connecting the words. Dependency parsing can be performed in linear-time, hence it is much faster than polynomial phrase-structure parsing and more appropriate for real-time NLP applications. Dependency parsing attaches each word to its syntactic head or governor. In most cases, the main verb is the overall head of the sentence. Figure 2.13 shows an example of the phrase-structure tree and the dependency tree of a sentence.

We can create a dependency structure based on the constituent structure by recursively deriving head categories via head percolation rules. Collins (1999) gives a practical set of hand-written rules for Penn Treebank grammars. One notation for a head percolation rule is in the form of "X (r A B C) (l D E F)", which means that for category X, first look for the rightmost A, B, or C, if found, that's the head of X, otherwise look for the leftmost D, E, F. This notation has been used in the ZPAR parser we used in Chapters 4 and 6. For a complete list of English and Chinese constituent head percolation rules in this parser refer to Appendix B. ZPAR uses lexical head annotations as features for training the dependency or phrase-structure parser.

Dependency trees are typically minimal compared to phrase-structure trees, thus a dependency parse for a given sentence can be produced with much less complexity than

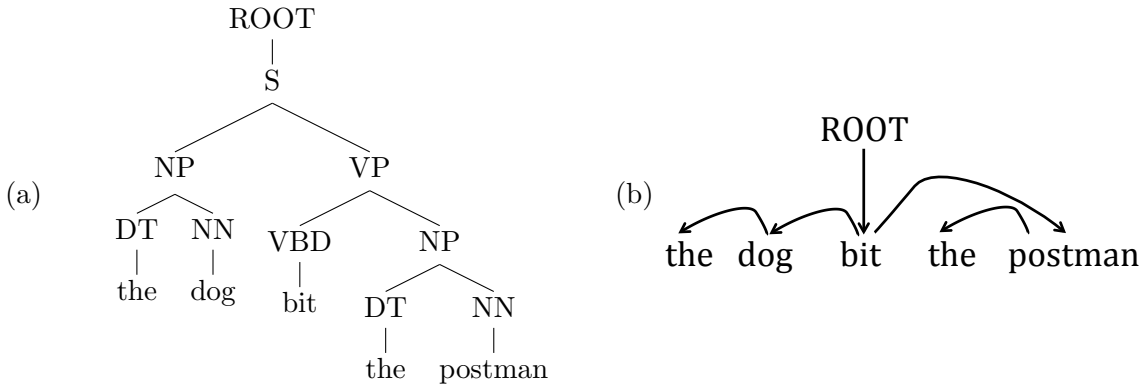


Figure 2.13: (a) Phrase-structure tree and (b) dependency tree of an example sentence.

producing the phrase-structure tree for the same sentence. Although dependencies have traditionally been extracted from constituent parses, there exist deterministic algorithms for dependency parsing that can directly extract syntactic dependency structure very quickly. These algorithms can be described as either *graph-based*, which perform parsing by searching for the highest-scoring dependency graph among all possible outputs, or *transition-based*, which perform parsing by greedily taking the highest-scoring transition based on the parser state. Eisner (1996); Covington (2001); McDonald et al. (2005); Nivre (2003) presented approaches for dependency parsing with lower algorithmic time complexity than constituency parsing. This speed comes with a drop in accuracy compared to dependency structure extracted from context-free parses with PCFGs (Cer et al., 2010). Cer et al. (2010) reported about 13% absolute difference in labeled attachment F1-score, and about 9 seconds difference in speed, between the most accurate and the fastest methods for generating dependencies. The most accurate parser was a reranking phrase-structure parser with about 89% F1-score and 10 seconds speed on an Intel Xeon E5520 for parsing the entire Penn Treebank development set, and the fastest one was a parser with a linear classifier to make local parsing decisions with about 76% F1-score and 1 seconds parsing speed.

2.5.4 Vine Parsing

Our proposed idea of constraining span length in hedge parsing as a type of phrase-structure parsing is reminiscent of constraining dependency length in *vine parsing* for dependency parsing. A word's dependents tend to fall near it in the string. Eisner and Smith (2005) showed that using dependency length as a parsing feature can improve parsing speed and accuracy. Their parser, called a *vine parser*, imposes a bound on the string distance between every child, except the nodes attaching to the root of the tree, and its parent. Such *soft* constraints can improve both speed and accuracy of a simple baseline dependency parser. They reported similar improvements by imposing *hard* constraints on dependency length. This type of constraint, completely prohibits long dependencies in the parser. Such a partial parser does less work than a full parser in practice, and allows for improved precision and speed with some loss in recall. Figure 2.14 shows an example of a full dependency tree, and a vine dependency tree of a sentence retaining only dependencies of length ≤ 3. Four child-to-parent dependencies are broken due to the length constraint, and the root of the four resulting parse fragments are connected to the root of the tree. As we will see in Chapter 3, the hedge transform similarly results in parse fragments that are then connected to the top-most node of the tree as its sequence of children.

Dreyer et al. (2006) and Rush and Petrov (2012) used vine parsing in a coarse-to-fine inference to improve efficiency of dependency parsing. Dreyer et al. (2006) designed a dependency parser for fast training and decoding and for high precision. At the first pass, a probabilistic vine parser produces an n-best list of the most likely parse trees. A discriminative minimum risk reranker then chooses among trees in this list. Rush and Petrov (2012) proposed a multi-pass coarse-to-fine approach for efficient dependency parsing. They started with a linear-time vine pruning pass and build up to higher-order models. To reduce pruning errors, the parameters of the vine parser were trained such that they optimize for pruning efficiency, and not for 1-best prediction. While maintaining state-of-the-art dependency parsing accuracy, this approach achieved two orders of magnitude speed-up.

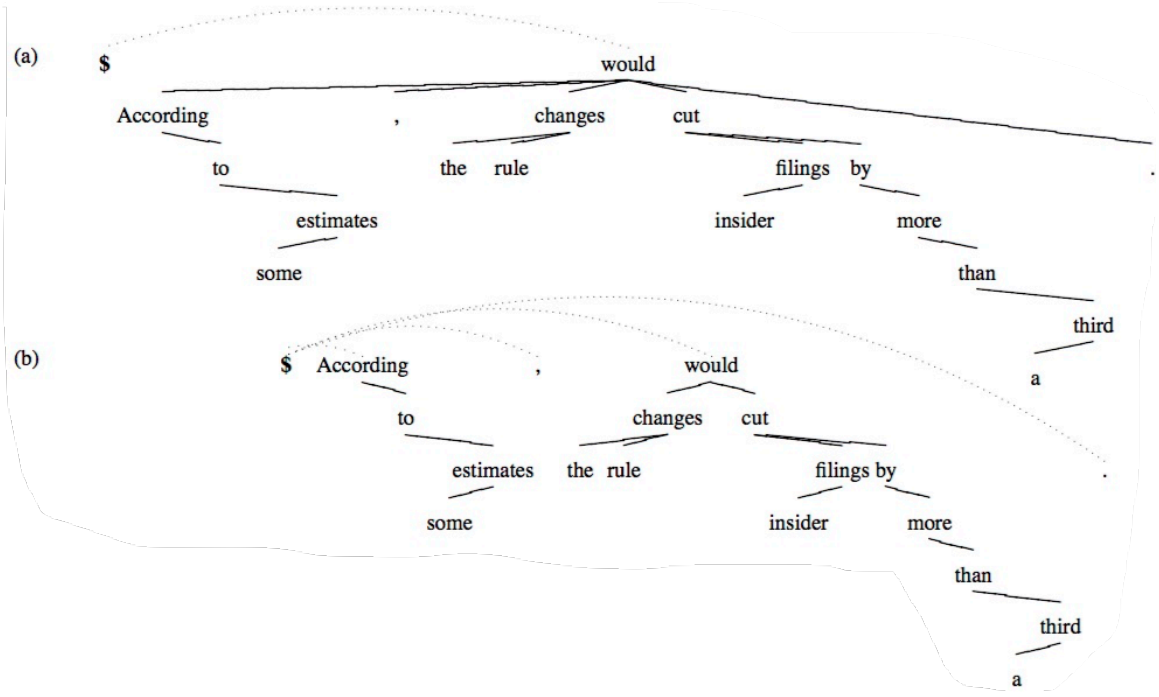


Figure 2.14: (a) A full dependency tree and (b) a vine dependency tree retaining only dependencies of length ≤ 3. (from Eisner and Smith (2005))

2.5.5 Incremental Parsing Algorithms

Previous work on incrementality in NLP have mainly focused on incremental algorithms such as incremental part-of-speech tagging, incremental syntactic parsing with probabilistic context-free grammars (PCFG), dependency grammars, and tree-adjoining grammars (TAG), or incremental semantic parsing, etc. Roark (2001) introduced an incremental probabilistic top-down parsing approach and its application to the problem of language modeling for speech recognition. Later Roark (2004) made the parser more robust by modifying the standard PCFG to smooth the probabilities in such a way that unseen productions are given some probability mass. Collins and Roark (2004) used a very similar parser but they estimated the parameters of the parser using a variant of the perceptron algorithm. They got competitive results on parsing the Penn treebank as that of the generative model using the same feature set.

Shift-reduce parsing approach has been successfully used for phrase-based parsing (Briscoe and Carroll, 1993) and dependency parsing (Sagae and Lavie, 2005; Wang et al., 2006a). These parsers compute parse trees from bottom up in one pass, and use classifiers to make shift-reduce decisions. Rather than making a single decision at each processing step, some shift-reduce parsers use beam-search decoding to resolve structural ambiguity (Zhang and Clark, 2008; Huang et al., 2009; Huang and Sagae, 2010; Zhang and Nivre, 2011; Goldberg et al., 2013). Zhang and Clark (2011) proposed a general framework of a global linear model, trained by the perceptron algorithm and decoded with beam-search, for incremental part-of-speech tagging, word segmentation, dependency parsing, and phrase-structure parsing.

Some incremental parsing studies use tree-adjoining grammars (Joshi and Schabes, 1997). Instead of production rules in CFGs, TAGs have two sets of elementary tree structures (Supertags), and two operations, substitution and adjunction, can then combine trees. Because of the introduction of the adjunction operation, the TAG formalism is provably stronger than CFG. TAG is a member of the family of mildly context-sensitive languages, which is a proper superset of the context-free languages. Shen and Joshi (2005) use a variant of the incremental shift-reduce with beam-search parsing to dynamically incorporate a Supertagger, which attempts to assign the correct elementary trees to each word, and a Lightweight Dependency Analyzer (Bangalore, 2000), which composes the whole parse tree with these elementary trees. Demberg and Keller (2008) and Demberg et al. (2013) proposed a psycholinguistically motivated version of TAG which is designed to model key properties of human sentence processing, namely incrementality, connectedness, and prediction. Hefny et al. (2011) introduced incremental combinatory categorial grammar (CCG) (Steedman, 1986), another lexicalized grammar closely related to TAG, to enable fully incremental left-to-right parsing.

2.6 Statistical Machine Translation

Machine translation is the automatic translation from one natural language into another. Statistical machine translation (SMT) treats translation as a machine learning problem. SMT algorithms automatically learn translation model parameters from the analysis of previously translated text, known as a parallel corpus or bitext. The learned model then can be used to translate new sentences. Generally, SMT algorithms are not tailored to any specific language pair, and an SMT model can be learned for an arbitrary language pair using an SMT toolkit and enough parallel text for that language pair. However, in some types of SMT systems, specific linguistic resources of the language may be needed, such as a high-quality syntactic parser for a syntax-based SMT system.

Two major SMT formalisms have been developed in the SMT literature: phrase-based and syntax-based. The main difference between these two formalisms is that translation rules in phrase-based SMT are bilingual phrases extracted from a parallel corpus of the source and target languages, whereas a syntax-based SMT uses some form of a synchronous context-free grammar (SCFG) to generate hierarchical mapping between the two languages. In Chapter 5, we use phrase-based as well as syntax-based SMT systems provided in the Moses SMT toolkit (Koehn et al., 2007).

2.6.1 Phrase-based SMT

Decoding

The fundamental equation of SMT is based on the source-channel approach in which the translation of a source sentence $\mathbf{f}$ into a target sentence $\mathbf{e}$ is modeled as (Jurafsky and Martin, 2009):

$$\hat{\mathbf{e}}(\mathbf{f}) = \arg\max_{\mathbf{e}} (P(\mathbf{e})P(\mathbf{f}|\mathbf{e})) \tag{2.9}$$

$P(\mathbf{e})$ is the language model of the target language and $P(\mathbf{f}|\mathbf{e})$ is the translation model. In a source-channel model of translation, we have to think of 'source' and 'target' backwards. To translate the source sentence, three stochastic channel operations are employed: first, the target string is segmented into phrases, second, these phrases are translated to source phrases, and finally, the translated phrases are re-ordered according to a distortion model.

Decoding in phrase-based methods is equivalent to finite-state transducers (Lopez, 2008). The core algorithm typically employs a general framework described by Wang and Waibel (1997) and Koehn (2004). The target string is built from left to right. To extend the translation hypotheses at each step, a subsequence in the source string which matches a source-side phrase in the phrase table is selected, and its target-side phrase is added to the target string. Each partial hypothesis is scored based on the translation model, the distortion model, and the target language model. To handle the huge search space, a beam-search is used to keep only the n-best partial hypotheses at each step.

Phrase Extraction

Phrases are typically extracted from a parallel corpus of the source and target sentences that are aligned at the word level. Word alignment identifies translation relationship among the words in a bitext, and represents the result in a bipartite graph or two-dimensional matrix between the two sentences. The most commonly used word alignment techniques in large-scale SMT systems are unsupervised techniques of the IBM models (Brown et al., 1993) and HMM models (Vogel et al., 1996). We refer the reader to Prud'hommeaux (2012) for further details about word alignment methods. Figure 2.15 shows an example word alignment. The basic idea in alignment-based phrase extraction is to enumerate all possible phrases in one language and check whether the aligned words in the other language are consecutive, with the possible exception of words that are not aligned (Och and Ney, 2004; Vogel et al., 2000; Venugopal et al., 2003). The Moses toolkit uses alignment-based phrase extraction techniques. There are also phrase extraction methods that work without the need of building an initial word alignment, such as integrated

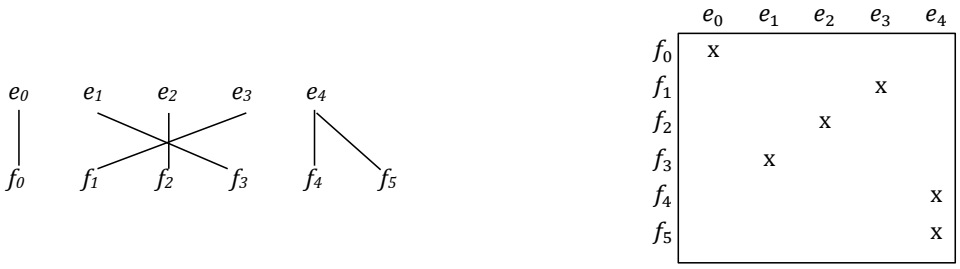


Figure 2.15: An example of two translationally equivalent sentences **e** and **f** and a possible word alignment in graph and matrix representations.

phrase segmentation/alignment algorithm in Zhang et al. (2003), or joint phrase alignment and extraction from a parallel corpus that is not word-aligned (Neubig et al., 2011). In contrast to the conditional alignment-based models, Marcu and Wong (2002) introduced a joint probability model that assumes that lexical correspondences can be established not only at the word level, but at the phrase level as well.

2.6.2 Syntax-based SMT

Decoding

Syntax-base SMT systems often use a log-linear modeling approach which is a generalization of the source-channel approach. This model was first introduced by Och and Ney (2002) for SMT. While the basic operations of phrase segmentation, phrase reordering, and phrase translation remain the same as in the source-channel model, a log-linear model allows for easier extension by adding new features and discriminative training of the model parameters. In contrast to the source-channel model, a log-linear approach directly models the posterior probability:

$$\hat{\mathbf{e}}(\mathbf{f}) = \arg\max_{\mathbf{e}} P(\mathbf{e}|\mathbf{f}) \tag{2.10}$$

The Moses toolkit uses CYK+ algorithm (Chappelier and Rajman, 1998), an improved CYK algorithm adopted to decode non-Chomsky Normal Form translation rules. CYK+ uses a two-dimensional chart to record both completed and partial results. Each cell

of the chart contains two lists of items: (1) a list of non-terminals that can parse the corresponding substring, and (2) a list of incomplete hypothesis (dotted rule) of the form $\alpha\bullet$, with α a string of non-terminals that can derive the corresponding substring, and for which there are rules in the grammar whose right-hand side starts with the string α. The procedure for filling the chart follows the same pattern as the standard CYK parsing.

Rule Extraction

Instead of phrases in a phrase-based model, a syntax-based model extracts SCFG rules. An SCFG is a generalization of a CFG to the case of two output strings. An SCFG is a way to simultaneously generate source and target sentences and the correspondence between them. Chiang (2006) provides an overview of SCFGs and several variants. As shown in Section 2.1.3, a CFG consists of a set of terminal and non-terminal symbols, and a set of production rules. An SCFG uses a source terminal set T_s, a target terminal set T_t, and a shared non-terminal set V, and the production rules of the form $X \rightarrow <\gamma, \alpha, \sim>$ where $X \in V$ is a non-terminal, $\gamma \in (N \cup T_s)^*$ is a list of zero or more non-terminals and source terminals, $\alpha \in (V \cup T_t)^*$ is a list of zero or more non-terminals and target terminals, and $\sim$ is a one-to-one mapping from non-terminal occurrences in γ to non-terminal occurrences in α (Venugopal et al., 2007). The mapping $\sim$ can be implicitly defined by co-indexing non-terminals in γ and their corresponding non-terminals in α. Figure 2.16 shows a fragment of an SCFG and the parse tree pair it generates in the two languages.

An SCFG can be automatically extracted from a bitext, with or without syntactic annotations, on top of the output of a phrase-based model. Hierarchical phrase-based models introduced by Chiang (2005, 2007) induce SCFGs without relying on any linguistic annotations. Hierarchical phrase-based models are commonly used and they typically improve translation accuracy significantly compared to a phrase-based system. A single non-terminal X is used in the production rules, and a maximum of two non-terminals are allowed in the right-hand side of any rule.

(a)
$S \rightarrow <NP_1$ misses NP_2, NP_2 manque à $NP_1>$
$NP \rightarrow <$John,John$>$
$NP \rightarrow <$Mary,Mary$>$

(b)

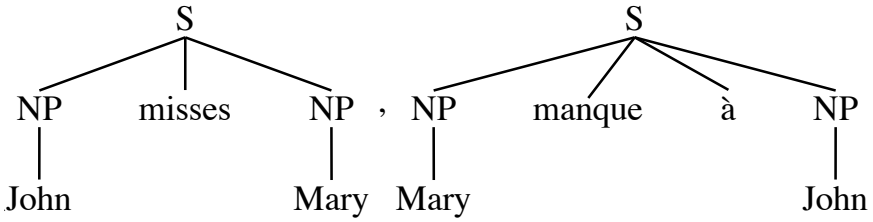


Figure 2.16: (a) An example SCFG for English and French, and (b) SCFG derivations represented as a pair of trees. (from Chiang (2006))

An SCFG can be induced from a bitext whose source, or target, or both languages are informed by the linguistic syntax of the language. Often only one language, which is usually the target side, has meaningful linguistic annotation, and the SCFG rules mirror the known syntax to the other language. Many of the studies in the literature are designed to translate into well-structured language, such as English, for which there are high-quality syntactic parsers available (Lopez, 2008). In general, we call these models tree-based in this thesis. If only the target language of a tree-based model has linguistic syntax, the model is called string-to-tree, if only the source language has linguistic syntax, the model is called tree-to-string, and if both languages have linguistic syntax, the model is called tree-to-tree. Several studies have shown improvements of string-to-tree models (Zollmann and Venugopal, 2006; Marcu et al., 2006; DeNeefe et al., 2007), tree-to-string models (Huang et al., 2006; Liu et al., 2006; Zhang et al., 2007a), and tree-to-tree models (Nesson et al., 2006; Zhang et al., 2007b) over phrase-based approaches. In this thesis we train and decode hierarchical phrase-based and tree-based models using syntax-based tools provided in the Moses toolkit.

2.6.3 MT Evaluation

BLEU (Papineni et al., 2002) is the most popular automated method for evaluation of machine translation. BLEU automatically scores a machine-generated candidate translation by measuring a weighted average of the number of overlapping n-grams of different length between the candidate and one or more human-generated references.

To deal with comparing the candidate translation against multiple references, BLEU extends the familiar precision metric to a modified n-gram precision. It first counts the maximum number of times each word in the candidate (unigram) is used in any single reference. The count of each candidate word is then clipped by (attached to) this maximum reference count. The modified precision is similarly computed for higher order N-grams, often up to 4-grams, as well. The sentence-level modified n-gram precision is then extended over the entire test set. BLEU adds the clipped n-gram counts over all the candidate sentences (referred to as *candidates* in the following formula), and divides by the total number of candidate n-grams in the test set. The modified precision score is thus:

$$p_n = \frac{\sum_{c \in \{candidates\}} \sum_{n\text{-}gram \in c} \text{Count}_{\text{clip}}(n\text{-}gram)}{\sum_{c' \in \{candidates\}} \sum_{n\text{-}gram' \in c'} \text{Count}(n\text{-}gram')} \tag{2.11}$$

In addition, BLEU adds a penalty to penalize candidates that are too short. Normally, precision is combined with recall to deal with these problems. However, recall over multiple references is not a good measure, because a good candidate translation only recalls one of the references, not all. Instead, BLEU includes a multiplicative factor called brevity penalty over the entire corpus.

$$BP = \begin{cases} 1 & c > r \\ e^{(1-r/c)} & c \leq r \end{cases} \tag{2.12}$$

where c is the total length of the candidate translation corpus, and r is the effective reference length computed by summing the lengths of the best matches for each candidate. Finally, the BLEU score is calculated as the geometric mean of the modified n-gram precisions, multiplied by the brevity penalty:

$$BLEU{=}BP \times \exp(\frac{1}{N}\sum_{n=1}^{N} logp_n) \tag{2.13}$$

While BLEU score is the most commonly used metric in MT evaluation, it has some drawbacks. It is based on n-gram precision overlap that might not be suitable for free word-order languages. It also assigns equal weight to any n-gram sequences no matter how informative they are. Some BLEU alternatives such as NIST (Doddington, 2002), assign higher weights to more informative n-grams by taking their relative frequency into account. That is, the more infrequent an n-gram, the more informative it is. Another drawback of BLEU is that it only considers exact word and n-gram matching. METEOR (Metric for Evaluation of Translation with Explicit ORdering) (Banerjee and Lavie, 2005) is another BLEU alternative that is based on the harmonic mean of unigram precision and recall and performs stemming and synonymy matching. METEOR is then less sensitive to n-gram matches and exact word repeats.

In Chapter 5, we use the multi-bleu perl script available in the Moses toolkit to measure MT BLEU scores. This script takes tokenized and sentence-aligned reference file and MT output file. It also accepts multiple reference translations.

2.7 Simultaneous Speech-to-Speech Translation

Some previous work has focused on how incremental NLP algorithms are utilized in real-time applications. Schlangen and Skantze (2009) and Skantze and Hjalmarsson (2010) described a general model and conceptual framework for incremental processing in dialogue systems. They presented the topology of the network of modules, how information flows in this network, and how incremental units of information are processed. Schuler

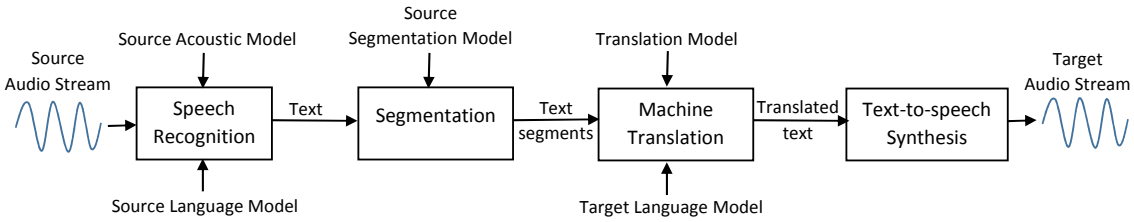


Figure 2.17: Simultaneous speech-to-speech translation pipeline.

et al. (2009) described a framework for incremental interpretation during speech decoding. They incorporate referential semantic information into the probabilistic language model of the system. Hassan et al. (2009) incorporates dependency parsing to statistical machine translation by adding CCG-based incremental dependency parser features to the phrase-based translation model. Atterer and Schlangen (2009) presented an incremental semantic segmentation method that performs incremental slot filing as it receives a stream of words in a dialogue system. Chunks correspond to sense units and segmentation is based on a notion of semantic/pragmatic completeness. An example usage of incremental NLP algorithms is Speech-to-Speech Translation (SST) that involves incremental processing of the input stream and simultaneously generating the output translation.

SST technology enables speakers of different languages to communicate real-time without knowing each others' languages. In recent years, SST has achieved acceptable quality through combining three key pieces of technology (Bangalore et al., 2012): automatic speech recognition (ASR), statistical machine translation, and, text-to-speech synthesis (TTS). Typically a segmentation component is required for preparing the ASR's output for the MT system in order to achieve high translation quality. Figure 2.17 presents a schematic overview of the simultaneous translation pipeline. Given the input audio stream, the ASR component produces a continuous stream of partial transcripts, which is segmented by the Segmentation component into appropriate chunks for the MT component. The MT component translates each of these chunks into the target language, and the translation output speech is delivered by the TTS component. The focus of this thesis is on segmentation and MT components. The goal of the segmentation component is to

provide segments that balance translation accuracy and latency. In the next section, we discuss some of the current input segmentation strategies in SST.

2.7.1 Input Segmentation Strategies

Some related work investigated input segmentation strategies in simultaneous speech translation. Appropriate segmentation of the source audio stream and the ASR output that is fed into machine translation is critical for a low-latency real-time translation while maximizing the overall translation quality. One category of segmentation methods is to use linguistic or non-linguistic heuristics, such as length of segments, time of audios, or predicted punctuations in the ASR output. Another category, directly considers the impact of the segmentation strategy on translation performance, thus tries to jointly optimize segmentation and translation. In this section, we summarize some of the studies in these two categories.

Heuristics, Non-linguistic, and Linguistic Approaches

Fügen et al. (2007) compared several segmentation strategies. As a baseline, they automatically segmented ASR hypotheses as well as reference transcripts at sentence boundaries. Sentence boundaries are automatically predicted using a log-linear model that combines language model and prosodic features. They obtained 36.6% BLEU score by translating ASR reference transcripts and 33.4% by translating ASR hypotheses for Spanish-English translation. In spite of relatively good translation quality, the average sentence lengths (30 words) are too high to be applicable to simultaneous translation. By taking all automatically predicted punctuation marks as split points, they reduced the average segment length to an acceptable 9 words with almost no decrease in the BLEU score. An alternative approach of simply cutting the merged utterances every n words can destroy semantic context, hence the translation scores are affected significantly. For example for n=7 they obtained 30.1% BLEU score for ASR reference transcripts and 27.5% for ASR hypotheses translations.

By defining segment boundaries based on non-speech regions in the ASR hypotheses, including recognized silences and nonhuman noises, Fügen et al. (2007) achieved better results than those obtained with length-based methods. For example, the non-speech duration thresholds of 0.3 seconds achieved 32.6% BLEU score for ASR hypotheses translation. Finally, adding lexical features provided by a language model to acoustic features outperforms all chunking strategies using acoustic features only (32.9% BLEU score). This semantic approach, which incorporates some context information, achieved the acceptable average segment length of 9 (and a small standard deviation of 6).

Bangalore et al. (2012) defined the segments by silence frames with a threshold of 8-10 frames (100 ms) in the ASR output. They did not use any lexical features for segmenting the utterances. Later, Rangarajan Sridhar et al. (2013) compared several linguistic and non-linguistic segmentation strategies for speech translation, with the goal of achieving a system that balances translation accuracy and latency. For non-linguistic segmentation, segmenting the text after every n words performs well, particularly for the larger n values, however, longer segments increase the latency and fixed length segmenting typically destroys the semantic context. Another approach, which they call hold-output model, segments the input sentence into minimally sized chunks such that crossing links in a optimal word alignment occurs only within individual chunks. To develop this model, a kernel-based SVM is applied at each word position. This strategy yields poor translation performance due to short segments (2-4 words) generated by the model, which do not provide sufficient context for translation.

For linguistic segmentation, Rangarajan Sridhar et al. (2013) tried segmenting the source text into sentences, or comma-separated segments which are predicted using a kernel-based SVM classifier, or conjunction-word based segments in which the segments are separated at either conjunction (e.g. “and”, “or”) or sentence ending boundaries. These linguistic methods work the best. Interestingly the use of gold-standard segments as input to MT versus the use of predicted segments does not make a great difference in MT accuracies in spite of errors in the set of predicted segments. In another linguistic method, they performed chunking within each sentence to segment the sentence into four chunk types

of noun, verb, particle, and adverbial. This type of segmentation yielded quite poor translation performance, mainly due to short chunk sizes. Although in general, the kind of error it makes is different from that in the hold-output method which also generates short segments.

To assess latency, Rangarajan Sridhar et al. (2013) compared two scenarios: one time-based segmentation in which the partial ASR hypotheses after a pre-defined timeout are sent directly to MT, and another where the best segmentation strategy (comma-separated chunks) is used to segment the partial ASR hypotheses before sending it to MT. The latter scenario shows a better translation accuracy particularly for timeout values less than five or six seconds, however, addition of the segmenter into the pipeline introduces a significant delay of about 1 second.

Several other similar studies address input segmentation strategies based on non-linguistic, linguistic, and combined criteria, for text or speech translation. Among these studies, Matusov et al. (2007) investigated the impact of automatic sentence boundary and sub-sentence punctuation prediction on translation of automatically recognized speech. The best translation results they achieved were when boundary detection algorithms were directly optimized for translation quality. Cettolo and Federico (2006) investigated punctuation-based, length-based, and combined text segmentation criteria and verified their impact on a Chinese-English statistical phrase-based translation system. They found that the best performance can be achieved by combining both linguistic and input length constraints. Wolfel et al. (2008) described five particular research challenges, including input segmentation, in simultaneous translation of German lectures to English. Baumann et al. (2014) investigated the timing aspects of incremental translation and speech synthesis by taking into account speech delivery timings for both input and output. They found that overall latency resulted from both the source utterance timing, its translation, and the target utterance delivery.

Joint Segmentation and Translation Optimization

Another important aspect of SST translation that has been less addressed in previous studies is joint segmentation and optimization of translation performance. Ideally any segmentation method should also take into account the way the specific system uses these segments works. However, segmentation strategies are mainly on the basis of heuristics and the impact of segmentation on translation performance is not directly considered.

In a recent work, Fujita et al. (2013) proposed a source segmentation method that considers the relationship between the source and target languages. To decide segmenting points, they use lexicalized information from the phrase table and reordering probabilities available in phrase-based translation systems. In another study, Oda et al. (2014) proposed algorithms for learning segmentation strategies for simultaneous speech translation that optimizes translation performance according to an evaluation measure calculated by a greedy search and dynamic programming. Shavarani et al. (2015) later extended this work by proposing a new algorithm that improves latency by up to 12% without BLEU score degradation for the same average segment length.

Along the lines of joint segmentation and translation optimization, in Yarmohammadi et al. (2013)[2] we proposed a novel approach for segmenting the incoming stream by exploiting the alignment structure between words (phrases) across a language pair. We explored an optimal segmentation such that segments could be translated to the target language *monotonically*. In other words, our goal was to split the sentence into segments such that phrase reordering would occur inside each segment and not across segments. Figure 2.18 shows an example of a word alignment matrix and all possible monotonic phrase alignments (4 alignments) for two parallel sentences, shown with different line styles.

We compared three incremental decoding and two different input segmentation strategies for simultaneous translation in terms of accuracy and latency. The three decoding strategies we used are: 1) phrase-based SMT using the Moses toolkit, 2) an FST implementation

[2]http://www.aclweb.org/anthology/I13-1141

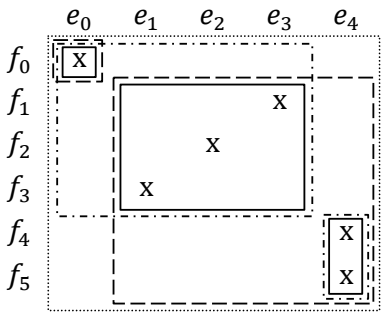


Figure 2.18: Word alignment matrix for two parallel sentences. Monotonic phrase alignments are shown with different line styles.

of the translation model, equivalent to the phrase-based translation in the Moses toolkit without reordering, and 3) an incremental beam search decoder introduced by Sankaran et al. (2010), which modifies the beam-search decoding algorithm for phrase-based MT aiming at efficient computation of future costs and avoiding search errors. The two segmentation strategies we tried for these three decoders were silence-based and monotonic phrase-based. To decide segmenting points, we applied a perceptron classifier, which used word form and length features, at each word position. Our segmentation model was trained on the same parallel data we used for machine translation.

Our experiments on English-French translation of TED talks released as part of the IWSLT evaluation (Federico et al., 2011) showed that incremental translation of monotone-based segments could get higher accuracy than the silence-based segments for all the three decoders. This accuracy improvement would cost a slight increase in latency in the first and third decoders, but interestingly the FST decoder showed reduction in latency, in spite of lacking the reordering knowledge—available in the first decoder uses—as well as history of translation—available in both other decoders use.

In a recent study, Siahbani et al. (2014) extended this work by integrating our proposed monotone-based segmentation with an incremental MT decoder. They used a variant of left-to-right decoding for a hierarchical phrase-based MT model (LR-Hiero) (Siahbani et al., 2013), which is suitable for incremental scenarios. A hierarchical phrase-based model typically uses the CYK decoding algorithm which requires the entire input before decoding begins, whereas LR-Hiero uses a beam-search decoder generating the translation

incrementally in a left-to-right manner. They also improved monotone-based segmentation compared to our work by using a richer feature set, including decoder feedback features. Overall, they achieved a very fast simultaneous translation system (23 times faster than non-incremental translation system) with reasonable translation quality (only 1.24 BLEU score loss). These results imply that even though the monotone-based segmentation is trying to optimize a phrase-based translation system, it could perform very well in a syntax-based system as well.

Chapter 3

Fast Syntactic Annotation and Segmentation using Hedge Parsing

3.1 Introduction

As we described in Chapter 2, there is a trade-off between efficiency and complexity of inference (search) using finite-state and context-free algorithms. Finite-state models are very fast and efficient for inference, but they are not powerful enough to define recursive syntactic structures. In contrast, context-free mechanisms have greater expressive power of the syntax and allow for recursive phrase construction, but they are generally computationally expensive. Finding a less computationally demanding syntactic parser than a full parser but more expressive than a shallow parser is potentially very useful for use in real-time applications for fast syntactic analysis of the input. Finding such a partial parser is the focus of this chapter.

Full hierarchical analysis of a sentence gives a complex complete parse tree of that sentence, as shown in Figure 3.1. This parse tree represents hierarchically embedded structures consisting of leaves of the tree or terminal symbols (i.e., words) and non-terminals labeling constituents (i.e., phrases such as NP, VP, PP), connected to the top-most non-terminal S.[1] To assign full parse trees to sentences, typically a parsing algorithm (e.g., CYK) employs a full context-free grammar (CFG) to produce trees (see Section 2.4 in Chapter

[1]See Section 2.1.3 of Chapter 2 for more details on syntactic trees. Note that for simplicity we do not represent the ROOT symbol in our examples.

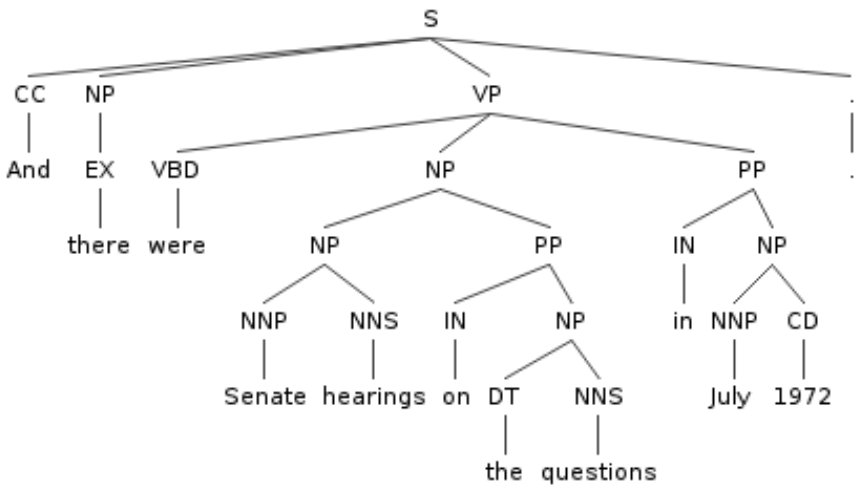
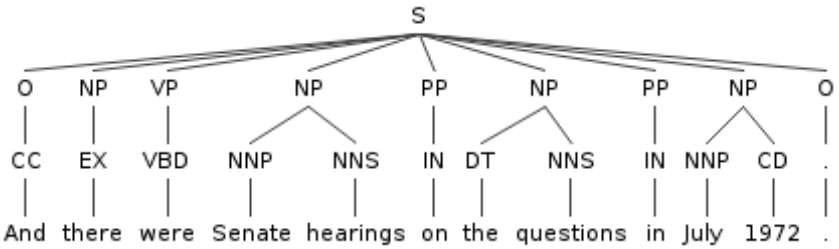


Figure 3.1: Full syntactic parse tree for an example sentence.

2). Shallow parsing (also known as chunking) identifies flat, non-overlapping constituents of the sentence, as shown in Figure 3.2. Shallow constituents typically correspond to major parts-of-speech, including noun phrases, verb phrases, propositional phrases, and adjective phrases. Since chunked segments lack hierarchical structures, a flat bracketing notation, shown in Figure 3.2(b), is sufficient to represent them. Chunking can be characterized by finite-state models and the chunking task can be viewed as a sequence tagging problem. For more detail about shallow parsing refer to Section 2.5.1 in Chapter 2.

a)



b)

And [NP there] [VP were] [NP Senate hearings] [PP on] [NP the questions] [PP in] [NP July 1972] .

Figure 3.2: (a) Shallow parse tree and (b) flat bracketing notation of the chunks.

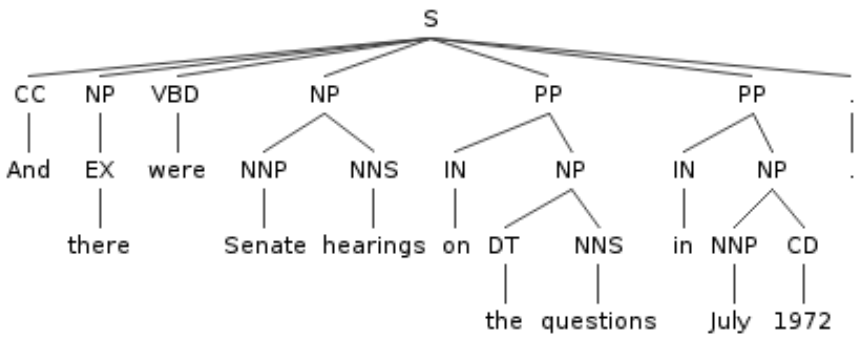


Figure 3.3: Hedge parse tree for the example sentence in Figure 3.1 with maximum constituent span of 4 (L=4).

A partial parser beyond a shallow parser could provide portions of recursive syntactic structures, such as the parts that are limited to a local context. Local recursive structure—beyond basic syntactic segmentation in shallow parsing—could be of utility even in the absence of fully connected structures. For example, in incremental/simultaneous machine translation (Bangalore et al., 2012; Yarmohammadi et al., 2013; Oda et al., 2014; Siahbani et al., 2014), sub-sentential segments are translated independently and sequentially, hence the fully-connected syntactic structure is not generally available. Even so, locally-connected source language parse structures can inform both segmentation and translation of each segment in such a translation scenario.

One way to provide local hierarchical syntactic structures without fully connected trees is to focus on providing full hierarchical annotations for structures within a local window, ignoring global constituents outside that window. We can obtain hedges via tree transformation, whereby non-terminals labeling constituents of span $>L$ in the tree, i.e., the nodes that cover more than L words, are recursively elided and their children promoted to attach to their parent. In such a way, hedges are sequentially connected to the top-most non-terminal in the tree, as demonstrated in Figure 3.3. After applying such a transform to a treebank, we can induce grammars and modify parsing to search as needed to recover just these constituents. We call this approach *hedge parsing*, i.e., discovering every constituent of length up to some span L.

Hedge parsing, as a type of partial syntactic parsing, may be of utility to various parsing tasks, as well as within a full parsing pipeline to prune the search space and increase the efficiency of full CFG parsing. As we saw in Chapter 2, similar methods have been used in a full parsing pipeline to quickly constrain subsequent inference, such as Supertagging (Bangalore and Joshi, 1999) which assigns a rich tag to each word of a sentence and can be used to prune the (context-sensitive) parser (see Section 2.5.2 of Chapter 2), or finite-state constraints which are used to constrain portions of the dynamic programming parsing chart (Glaysher and Moldovan, 2006; Roark and Hollingshead, 2008; Roark et al., 2012). Similar constraints have been used in vine parsing to constrain the distance between heads and dependents in dependency parsing (see Section 2.5.4 of Chapter 2). To the best of our knowledge, our 2014 paper (Yarmohammadi et al., 2014) was the first work to consider this type of partial constituency parsing approach for natural language.

The most basic parsing algorithm for probabilistic context-free grammars, nevertheless widely used, is the dynamic programming CYK algorithm. We explained this algorithm in detail in Section 2.4.1 of Chapter 2. The CYK algorithm builds longer-span constituents by combining smaller-span constituents as allowed in a probabilistic CFG (PCFG). The idea is applicable to the hedge transform which works by constraining the span of constituents. In this chapter, we explain hedge parsing using the CYK algorithm and constraining the chart cells. We propose several methods to parse hedge constituents and examine their accuracy/efficiency tradeoffs. First, we review finite-state and context-free parsing complexity, which we described in Chapter 2, and briefly describe some of the pruning and prioritization methods for the CYK algorithm to improve efficiency of PCFG parsing. Then, we discuss modifications to inference and the resulting computational complexity gains for hedge parsing, and how such hedge parsing behaves when combined with CYK pruning and prioritization methods. Finally, we investigate pre-segmenting the sentences with a finite-state model prior to hedge parsing, and achieve large speedups relative to hedge parsing the whole string, though at a loss in accuracy due to cascading segmentation errors.

We demonstrate that:

1. Hedge parsing is a partial syntactic parsing approach that has a complexity and expressive power between shallow and full parsing.

2. With straight-forward modifications to the CYK parser and the grammar we can perform high-accuracy hedge parsing with orders of magnitude faster speed than the traditional parsing. For example, we can find constituents with the maximum span of 7 words at the parsing speed of about 26 words per second and accuracy of 87% for our English data set using the BUBS parser (see Section 3.7.2). Traditional parsing which finds every constituent of the sentence takes about 3 words per second, on average, at the state-of-the-art accuracy of about 90%.

3. Hedge parsing can find hedges of a sentence without requiring the entire sentence, instead the segments of the sentence which correspond to complete hedges can be parsed. This will generally result in more efficient parsing due to processing smaller segments, and also will increase incrementality due to being able to syntactically analyze the incomplete input. Using the BUBS parser, we can hedge parse sentences with over 200 words per seconds speed and about 83% accuracy by pre-segmenting the sentences prior to parsing.

4. The ideas and algorithms in this chapter establish the baseline results for hedge parsing. Leveraging non-incremental dynamic programming CYK parsing does not really address our ultimate goal of applying partial hedge parsing to incremental NLP applications. The idea of pre-segmenting the sentence prior to parsing as an attempt to increase incrementality has the disadvantage of cascading segmentation errors. We notice significant potential in the greedy incremental parsers with linear complexity, which has been recently widely used for both constituency and dependency parsing with great success. Results for this approach will be provided in chapter 4.

3.2 Finite-state vs Context-free Parsing

Finite-state parsing (also called shallow parsing or chunking) is equivalent to parsing with a regular right-linear (or left-linear) grammar of the form $A \rightarrow w^*$ or $A \rightarrow w^*B$. (See

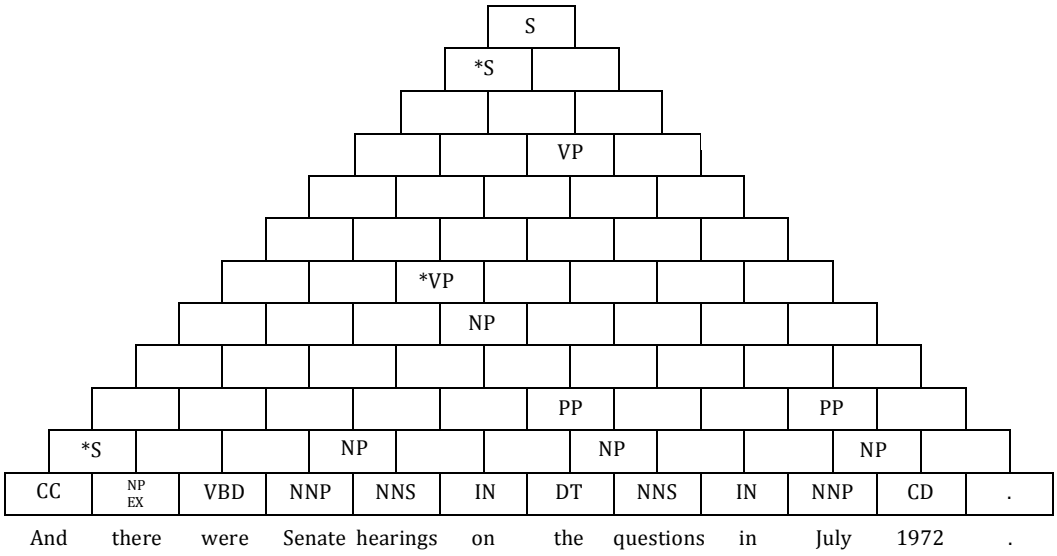
Section 2.1.2 of Chapter 2 for more details about regular grammars.) Shallow parsing cannot express recursive center-embedding rules like $A \Rightarrow^* \alpha A \beta$ where a non-terminal A is written to itself, surrounded by non-empty strings α and β. Finite-state inference is $O(n)$. In fact, finite-state inference can be viewed as a sequence classification problem and the standard approaches to training classifiers apply to this problem. The standard way to do shallow parsing is IOB tagging, which represents the beginning and internal words of a chunk by B and I tags and the words outside any chunk by O tag (see Section 2.5.1 of Chapter 2). In Section 3.6 of this chapter, we use a type of sequence tagging model to hedge segment the input sentence as a pre-processing step of hedge parsing.

The CYK algorithm for CFG parsing combines bottom-up parsing with dynamic programming to perform exact inference of the input's syntactic structure based on a CFG. In Chapter 2 we saw how dynamic programming solves a problem by dividing it into sub-problems and combining solutions to sub-problems. In the case of parsing, dynamic programming uses a table, referred to as the chart, of partial parses to solve the repeated parsing of subtrees problems. Figure 3.4(a) shows the CYK chart for our example sentence. Each cell in this chart contains a set of non-terminals that represent the constituents covering a particular substring (or span) of the input. The first row, contains the constituents with a span of 1, and higher levels represent larger spans. The result of parsing with such binary grammar is shown in Figure 3.4(b). Non-terminals starting with the symbol '*' are temporary non-terminals created through Chomsky normal form (CNF) binarization. Complexity of parsing with a full CFG is $O(n^3|G|)$ where n is the length of input and $|G|$ is the grammar size constant.In Section 3.5.2 we modify the CYK chart to match the hedge constraint, and describe how this will decrease the complexity of hedge parsing. For more detail about the CYK algorithm refer to Chapter 2.

3.3 CYK Pruning and Prioritization Methods

Several methods for increasing efficiency of CYK parsing have been proposed recently. The CYK algorithm is an *exact* inference which finds the globally optimal solution given

a)



b)

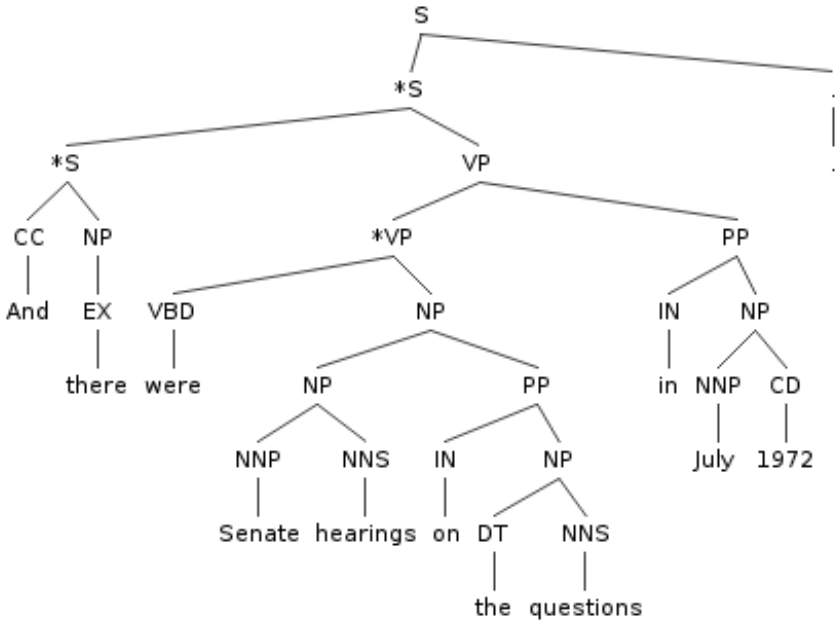


Figure 3.4: (a) CYK chart with binarized non-terminals (b) left-binarized parse tree.

a grammar. To improve efficiency, *approximate* inference methods have been used which make hard decisions typically based on local information to constrain the search space. One of the widely used approximate inference methods is *pruning*. Pruning methods exclude portions of the search space by directly eliminating paths of the search space as they are expanded (i.e., beam-search), or constraining the final search space via earlier stages of a pipeline (i.e., coarse-to-fine parsing or POS tagging) (Bodenstab, 2012). In addition to pruning there is another class of methods for efficient parsing called *prioritization*. Such methods prioritize the order of search so that promising candidates are pursued first. Prioritization does not make hard decisions to restrict the search space, so the inference may be exact instead of approximate.

The CYK parser we use in this chapter (Bodenstab et al., 2011) makes use of several recent pruning and prioritization approaches in learning and inference. These approaches are introduced in (Bodenstab et al., 2011, 2010; Bodenstab, 2012; Dunlop, 2014). We examine the effect of some of these approaches on hedge parsing performance. The pruning approaches we apply are: 1) beam-search, which limits the number of candidates explored at each step, and 2) complete closure, a type of chart cell constrain which individually classifies each chart cell as being open or closed to all constituents. In the case of beam-search, a prioritization function (also called figure-of-merit or FOM) sorts all candidate non-terminals based on some heuristic and only the k-best non-terminals are retained in the chart cells at each iteration. The FOMs we apply here are based on inside and outside probabilities in the inside-outside algorithm that computes the posterior probabilities of constituents in a chart. The FOMS are: 1) inside FOM, which is the simple inside probability of each non-terminal, and 2) lexical boundary FOM, which improves the inside probability by combining it with a heuristic estimate of the outside probability. The outside probability in this case is learned from words surrounding the left and right boundaries of the non-terminal.

3.4 Hedge Tree Transform

The hedge tree transform converts the original parse tree into a hedge parse tree. In the resulting hedge parse tree, every child of the top-most node spans at most L words. To transform an original tree to a hedge tree, we remove every non-terminal with span larger than L and attach its children to its parent. First, in a bottom-up pass over the nodes, We label span length on each node by recursively summing the span lengths of each node's children, with terminal items by definition having span 1. A second top-down pass evaluates each node before evaluating its children, and removes nodes spanning $>L$ words. For example, assuming L=4, the node VP and its child NP in the original parse tree in Figure 3.1 have spans $>L$ words. The spans of these two nodes are 9 and 5 respectively. Thus they are removed in the hedge transform, resulting in the hedge parse tree in Figure 3.3.

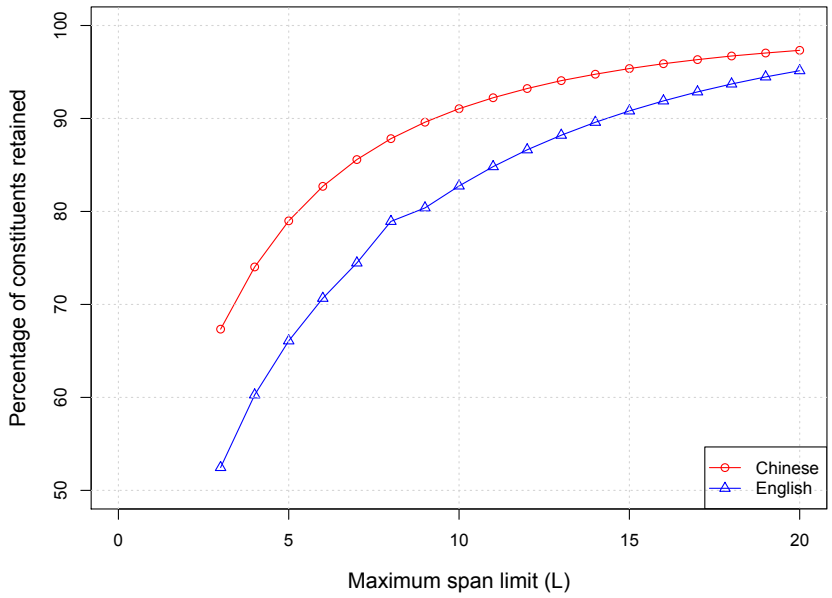


Figure 3.5: Percentage of constituents retained at various span length parameters L=3–20 for English and Chinese training data.

If we apply this transform to an entire treebank, we can use the transformed trees to induce a PCFG for parsing as described in Section 2.1.3 of Chapter 2. In Section 3.5.1 we introduce such a grammar which we call a hedgebank grammar. Figure 3.5 plots the percentage of constituents from the original English and Chinese Penn treebanks (training sections) retained in the transformed version, as we vary the maximum span length parameter L. For English, over half of constituents have span 3 or less (which includes frequent base noun phrases); L=7 covers approximately three quarters of the original constituents, and L=15 over 90%. Most experiments in this paper will focus on L=7, which is short enough to provide a large speedup yet still cover a large fraction of constituents. For Chinese, more constituents are retained at various Ls compared to English. The reason is smaller average span length of constituents in Chinese (4.4 words) compared to English (6.5 words) as we will show in Table 3.2 (see §3.7.1). Another justification is the lower branching factor in the Chinese treebank compared to the English (Levy and Manning, 2003), while the average height of the trees and average number of words per sentence are similar in both treebanks. The average height of the trees is 9.5 in Chinese and 10.0 in English, and the average number of words per sentence is about 27 in Chinese and about 24 in English. This causes less pruning of the constituents in Chinese hedge transform compared to English.

3.5 Hedge Parsing

Hedge parsing is the approach of finding the hedge parse tree of a given sentence. Since hedge structures can be recursive structures, finite-state approaches such as sequence tagging generally cannot find hedges efficiently, thus context-free approaches to syntactic parsing seems more appropriate for hedge parsing. In Section 2.1.3 of Chapter 2 we described PCFGs, PCFG induction, and treebank transform. We saw that by starting with a treebank, PCFG rules can be read off from the parse trees in the treebank. To either improve parsing performance or use the grammar in a specific algorithm such as CYK parsing or hedge parsing, we transform the treebank to the desired form before extracting the grammar rules. The hedge tree transform introduced in the previous section is an

instance of treebank transform before PCFG induction. We will describe the resulting grammar later in this section.

In Chapter 2, we categorized syntactic parsing methods into two groups of exact inference based on dynamic programming and greedy inference based on a classifier. In this chapter, we focus on dynamic programming and in particular its most popular algorithm, the CYK, which allows us to straightforwardly apply the hedge condition on parsing. In this section, we explain hedgebank grammars and then modifying the CYK chart according to the hedge condition. Hedge parsing with a classifier-based parser will be the focus of the next chapter. Such a parser will alleviate some of the issues we face in CYK hedge parsing.

3.5.1 Hedgebank Grammar

After applying the hedge transform to an entire treebank, we can learn a PCFG from the transformed treebank which we call a *hedgebank grammar*. A hedgebank grammar is a fully functional PCFG and can be used with any standard parsing algorithm, i.e., these are not generally finite-state equivalent models. However, using the Berkeley grammar learner (see Section 3.7.2), we find that hedgebank grammars are typically smaller than treebank grammars, reducing the grammar constant and contributing to faster inference. Complexity of parsing with a hedgebank grammar and a hedge-constrained CYK chart is in between the complexity of shallow parsing and full CYK parsing.

Recall from Section 2.1.3 of Chapter 2 that for use by a CYK parsing algorithm, trees are binarized prior to grammar induction, resulting in temporary non-terminals created by binarization. Figure 3.6(b) shows an example of a binarized hedge tree. These temporary non-terminal are marked with asterisk, such as *S in the figure. Other than the symbol at the root of the tree, the only constituents with span length greater than L in the binarized hedge tree will be labeled with these special binarization non-terminals. Further, in the systematic binarization through left- or right- factoring (see Section 2.1.3), the constituents with span greater than L will either begin at the first word (leftmost grouping) or end at

the last word (rightmost). These properties enables constraining the number of cells in the CYK chart requiring work. In the next subsection we show how a hedge-constrained CYK chart works.

3.5.2 Hedge-constrained CYK Chart

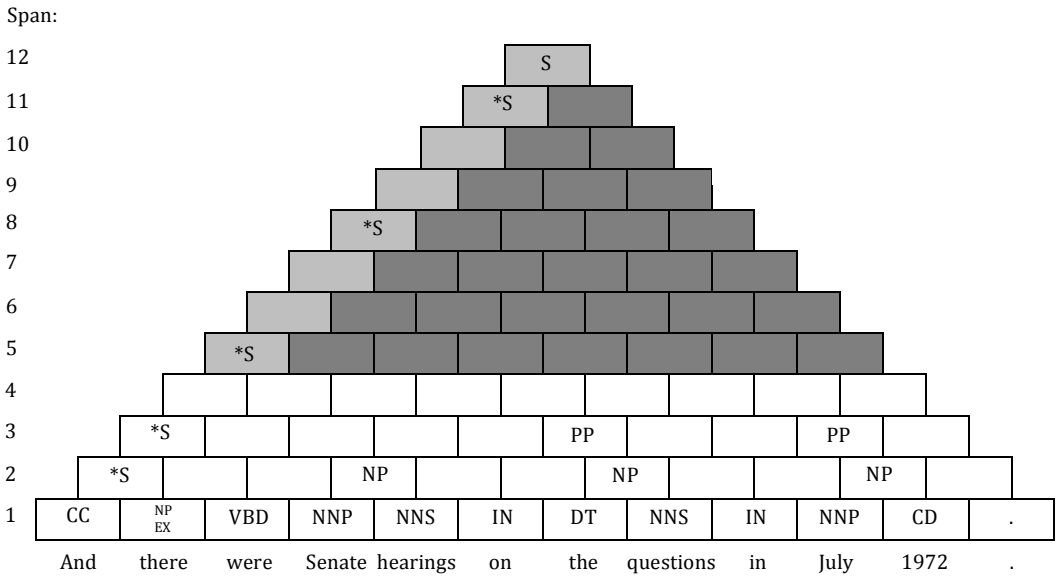
Since we limit the span of non-terminal labels in a hedgebank grammar, we can constrain the search performed by the parser, and thus greatly reduce the CYK processing time. In essence, we perform no work in chart cells spanning more than L words, except for the cells along the periphery of the chart, which are just used to connect the hedges to the root. In fact, these cells contain the temporary non-terminals created by binarization prior to grammar induction. Figure 3.6(a) shows an example of a hedge constrained CYK chart. Chart cells spanning more than L words are in dark gray, meaning that they are closed to all constituents, and the periphery cells are in light gray, meaning that they can only contain special non-terminals created by binarization.

As we saw earlier in this chapter, complexity of parsing with a full CYK parser is $O(n^3|G|)$ where n is the length of input and $|G|$ is the grammar size constant. In contrast, complexity of parsing with a hedge constrained CYK is reduced to $O((nL^2+n^2)|G|)$. To see that this is the case, consider that there are $O(nL)$ cells of span L or less, and each has a maximum of L midpoints, which accounts for the first term. Beyond these, there are $O(n)$ remaining active cells with $O(n)$ possible midpoints, which accounts for the second term. Note also that these latter cells (spanning $>L$ words) may be less expensive, as the set of possible non-terminals is reduced to only those introduced by binarization.

3.6 Hedge Segmentation

A unique property of hedge constituents compared to constituents in the original parse trees is that they are sequentially connected to the top-most node. To clarify, in the example of hedge parse tree in Figure 3.3, the hedge constituents CC, NP, VBD, NP, PP, PP, and "." do not share any sub-tree, but they share the same parent S, which is

a)



b)

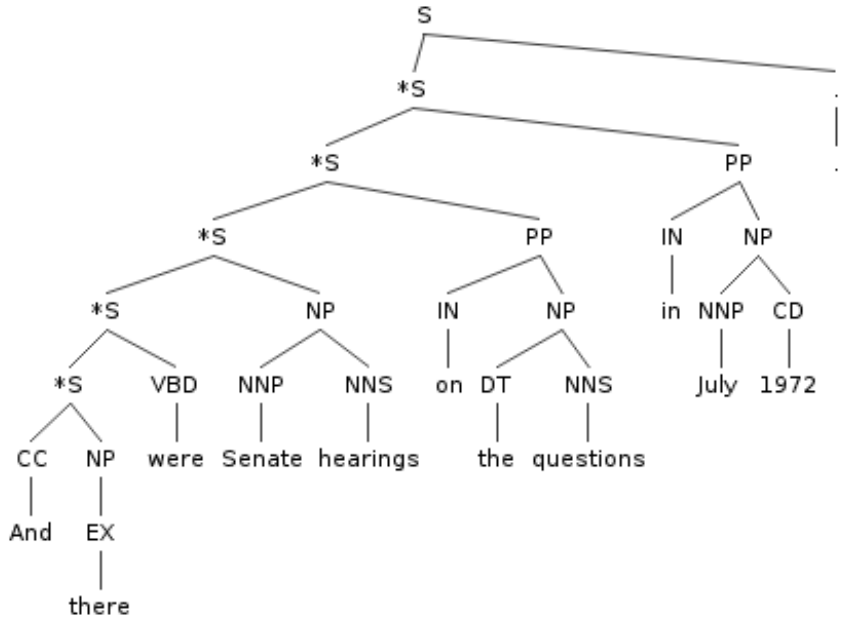


Figure 3.6: (a) Example chart demonstrating cell closures when performing inference with a span-4 hedgebank grammar. (b) Left-binarized hedge parse tree.

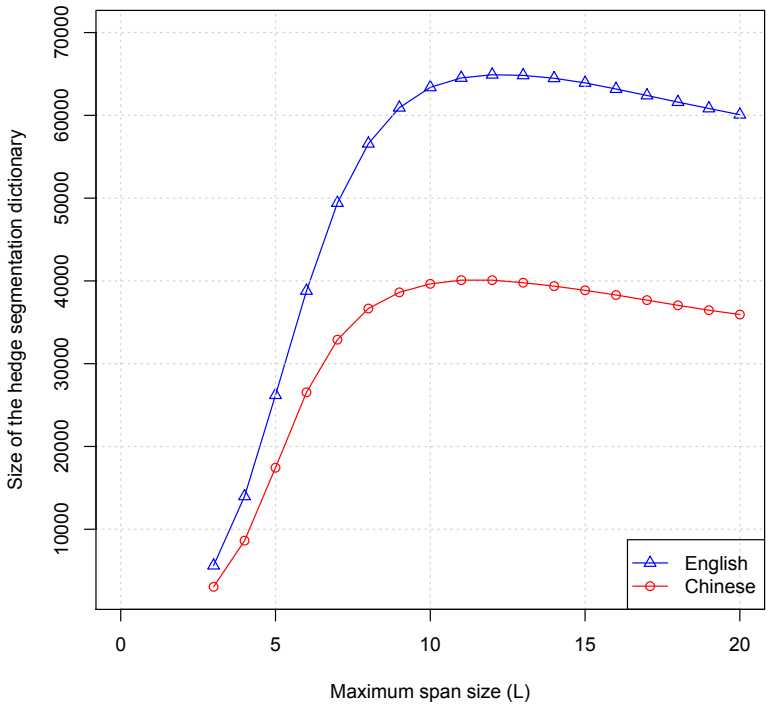


Figure 3.7: Size of the hedge segmentation dictionary at various span length parameters L=3–20 for English and Chinese training data.

the top-most node in tree. This property enables us to chunk the sentence into segments that correspond to complete hedges, and parse the segments independently (and simultaneously) instead of parsing the entire sentence. To realize the frequency of the hedge segments that are directly connected to the top-most node, we counted the number of unique hedge segment structures – which we refer to as the *hedge segmentation dictionary* – in the entire hedgebank. In the example hedge parse tree in Figure 3.3, the unique hedge segment structures are "(CC)", "(NP (EX))", "(VBD)", "(NP (NNP) (NNS))", "(PP (IN) (NP (DT) (NNS)))", "(PP (IN) (NP (NNP) (CD))", and "(.)", so the size of the hedge segmentation dictionary is 7. Figure 3.7 plots the size of the hedge segmentation dictionary for the English and Chinese hedgebanks (training sections) as a function of the maximum span parameter L. The size of the dictionary remarkably increases as L increases, although the size is roughly consistent between L=10 to L=15, and is slightly decreasing between L=15 to L=20. This is due to the larger hedge structures in larger L values and thus the lower chance of similarity between the structures.

In the next subsections we present our segmentation model which takes the input sentence and chunks it into appropriate segments for hedge parsing. We treat this as a binary classification task which decides if a word can begin a new hedge. We use hedge segmentation as a finite-state pre-processing step for hedge parsing.

3.6.1 Segmentation Model

Our task is to learn which words can begin (B) a hedge constituent. This task is a form of sequence tagging model such as chunking described in Section 2.5.1 of Chapter 2. Given a set of labeled pairs (S, H) where S is a sentence of n words $w_1 \dots w_n$ and H is its hedge parse tree, word w_b belongs to B if there is a hedge constituent in H spanning $w_b \dots w_e$ for some $e \geq b$ and w_b belongs to $\bar{B}$ otherwise. Many of the hedge segments are unary constituents, or POS tags whose parents are removed due to the hedge transform. To predict the hedge boundaries more accurately, we grouped consecutive unary or POS-tag hedges together under a new parent non-terminal labeled G. For example, the POS-tag hedge "(CC AND)", the unary hedge "(NP (EX there))", and the POS-tag hedge "(VBD were)" are grouped under a parent G to form the hedge "(G (CC AND) (EX there) (VBD were))". Unlabeled segmentation tags for the words in the example sentence in Figure 3.3 are:

> "And/B there/$\bar{B}$ were/$\bar{B}$ Senate/B hearings/$\bar{B}$ on/B the/$\bar{B}$ questions/$\bar{B}$ in/B July/$\bar{B}$ 1972/$\bar{B}$ ./B"

In addition to the simple unlabeled segmentation with B and $\bar{B}$ tags as the above, we tried a labeled segmentation with B_C and $\bar{B}_C$ tags where C is hedge constituent type. For English, we restrict the types to the most important types – following the 11 chunk types annotated in the CoNLL-2000 chunking task (Sang and Buchholz, 2000) – by replacing all other types with a new type OUT. Labeled segmentation tags for the words in the example sentence in Figure 3.3 are:

> "And/B_G there/$\bar{B}_G$ were/$\bar{B}_G$ Senate/B_{NP} hearings/$\bar{B}_{NP}$ on/B_{PP} the/$\bar{B}_{PP}$ questions/$\bar{B}_{PP}$ in/B_{PP} July/$\bar{B}_{PP}$ 1972/$\bar{B}_{PP}$ ./B_G"

3.6.2 The Classifier and Feature Set

To automatically predict the class of each word position, we train a multi-class classifier from labeled training data using a discriminative linear model, learning the model parameters with the averaged perceptron algorithm (Collins, 2002). For more details about this general framework refer to Section 2.3 of Chapter 2. We follow Roark et al. (2012) in the features they used to label words as beginning or ending constituents. Table 3.1 summarizes the feature set. The segmenter extracts features from word and POS-tag input sequences and hedge-boundary tag output sequences. The feature set includes trigrams of surrounding words, trigrams of surrounding POS tags, and hedge-boundary tags of the previous words. An additional orthographical feature set is used to tag rare[2] and unknown words. This feature set includes prefixes and suffixes of the words (up to 4 characters), and presence of a hyphen, digit, or an upper-case character. Reported results are for a Markov order-2 segmenter, which includes features with the output classes of the previous two words.

	Lexical	Orthographical	POS
t_i	t_i, w_i	$t_i, w_i[0]$	t_i, p_i
t_{i-1}, t_i	t_i, w_{i-1}	$t_i, w_i[0..1]$	t_i, p_{i-1}
t_{i-2}, t_i	t_i, w_{i+1}	$t_i, w_i[0..2]$	t_i, p_{i-1}, p_i
t_{i-2}, t_{i-1}, t_i	t_i, w_{i-2}	$t_i, w_i[0..3]$	t_i, p_{i+1}
	t_i, w_{i+2}	$t_i, w_i[n]$	t_i, p_i, p_{i+1}
	t_i, w_{i-1}, w_i	$t_i, w_i[n-1..n]$	$t_i, p_{i-1}, p_i, p_{i+1}$
	t_i, w_i, w_{i+1}	$t_i, w_i[n-2..n]$	t_i, p_{i-2}
		$t_i, w_i[n-3..n]$	t_i, p_{i-2}, p_{i-1}
		t_i, w_i has digit	$t_i, p_{i-2}, p_{i-1}, p_i$
		t_i, w_i has upper-case	t_i, p_{i+2}
		t_i, w_i has hyphen	t_i, p_{i+1}, p_{i+2}
			$t_i, p_i, p_{i+1}, p_{i+2}$

Table 3.1: Classifier features for tagging hedge segment boundaries. w is word form, t is hedge-boundary tag, and p is POS tag. All lexical, orthographical, and POS features also occur with t_{i-1}.

[2]Rare words occur less than 5 times in the training data.

3.7 Methods

So far, we have introduced the hedge transform, hedgebank grammars, and the hedge-constrained CYK parser. In the next sections, we will use these techniques to parse hedges. The brute-force baseline approach is to parse the sentence using a full PCFG and then hedge-transform the result. This method should yield a ceiling on hedge-parsing accuracy, as it has access to rich contextual information (as compared to grammars trained on transformed trees). Naturally, inference will be slower; we aim to improve efficiency upon this baseline while minimizing accuracy loss.

Improving the efficiency of the baseline method is achieved at the expense of accuracy in all the approaches we present in this chapter. We propose three hedge parsing approaches:

1. Using hedge-constrained CYK parser with a hedgebank grammar. First, the treebank is hedge transformed and a hedgebank grammar is induced from it. Then, the CYK parser whose cells above the span L are closed for efficiency uses this grammar to parse the input sentence. We see the results of this case in Section 3.8.1;

2. Applying CYK prioritization and pruning methods on the baseline as well as the approach 1. We try combinations of the prioritization and pruning methods available in our parser to restrict the search space of a full or constrained CYK chart. The results are presented in Section 3.8.2;

3. Segmenting the sentences using the hedge segmentation model presented in Section 3.6, and then parse the segments using a hedgebank grammar. In this case, the input sentence is segmented into smaller chunks using a classifier which is trained on the hedge segment boundaries of the training hedgebank. Since hedge parsing complexity is correlated with input length, parsing smaller inputs could be more efficient than parsing the entire sentence. The chunks are then parsed using a hedge-constrained parser and a hedgebank grammar. This case is described in Section 3.8.3.

Treebank		Partitions	Sentences	Words	Avg Sent Len	Avg Span Len
English	train	Sections 2-21	39,832	950,028	23.8	6.5
	dev	Section 24	1,346	32,853	24.4	6.7
	test	Section 23	2,416	56,684	23.5	6.5
Chinese	train	Articles 1-270, Articles 400-1151	18,086	493,708	27.3	4.4
	dev	Articles 301-325	350	6,801	19.4	5.1
	test	Articles 271-300	348	8,008	23.0	4.0

Table 3.2: Corpora statistics.

3.7.1 Data

We run experiments on English and Chinese. For English we use the WSJ Penn Treebank corpus (Marcus et al., 1999) consisting of approximately one million words from the Wall Street Journal. We use section 2-21 for training, section 24 for development, and section 23 for testing. For Chinese we use the Penn Chinese Treebank (Xue et al., 2005) consisting of about 540k words of Chinese newswire text. Our data set is articles 1-270 and 400-1151 for training, articles 301-325 for development, and articles 271-300 for testing. Treebank trees are pre-processed to remove empty nodes, node indices, and function tags. We use these data sets for English and Chinese experiments throughout this dissertation. In this Chapter, as well as Chapters 4 and 5, we first present the results on the development set, and then we report the results of the best configurations on the test set.

Table 3.2 shows the statistics of each corpus, including total number of words, total number of sentences, the average span length of all constituents in the trees (excluding the root and its immediate child), and the average length of sentences for training, development, and testing divisions. The Chinese training set is approximately half of the size of the English training set in terms of the number of sentences and word count. Chinese development and test sets are also quite small, thus accuracy results on Chinese treebank are likely to be noisy. The average sentence length, i.e., number of words per sentence, is slightly larger in Chinese training data, although the average span length of constituents in the parse trees is remarkably smaller in Chinese. That is the reason why the majority of constituents are retained in Chinese hedge transform.

3.7.2 Experimental Setup

For all experiments in this chapter, we performed CYK parsing using the BUBS parser[3] with Berkeley SM6 latent-variable grammars (Petrov and Klein, 2007b) learned by the Berkeley grammar trainer with default settings. Except for the prioritization and pruning models in section 3.8.2, we performed exhaustive parsing in all experiments. We compute accuracy from the 1-best Viterbi tree extracted from the chart using the standard EVALB script (see Section 2.4.3 of Chapter 2 for parsing evaluation). Accuracy results are reported as precision, recall and F1-score, the harmonic mean between the two. In contrast to traditional parsing accuracy evaluation which evaluates finding every constituent of the sentence, here we focus on finding the hedge constituents. Thus, instead of comparing the results against the original reference treebank, we evaluate accuracy with respect to the hedge transformed reference treebank, i.e., we do not penalize the parser for not discovering constituents longer than the maximum length. Segmentation accuracy is reported as an F1-score of unlabeled segment bracketing. The gold standard segmentation tags are unlabeled or labeled $B/\bar{B}$ tags from the hedge transformed original treebank.

We ran timing tests on an Intel 2.66GHz processor with 3MB of cache and 2GB of memory. Note that segmentation time is negligible compared to the parsing time, hence is omitted in reported time. Efficiency results are reported as number of words parsed per second (w/s).

3.8 Results

In Sections 3.8.1, 3.8.2, and 3.8.3 we present hedge parsing results on the development set, and in Section 3.8.4 we present the results on the test set.

[3] https://code.google.com/p/bubs-parser

Parser	Hedge Parsing Acc/Eff			
	P	R	F1	w/s
Full w/full CYK	88.8	89.2	89.0	2.4
Hedgebank	87.6	84.4	86.0	25.7

Table 3.3: English hedge parsing results on section 24 for L=7.

3.8.1 Hedge Parsing Results

Table 3.3 presents hedge parsing accuracy on the English development set for the full parsing baseline, where the output of the regular PCFG parsing is transformed to hedges and evaluated, versus parsing with a hedgebank grammar, with no segmentation of the strings. We find an order of magnitude speedup of parsing, but at the cost of 3 percent F-measure absolute. Note that most of that loss is in recall, indicating that on average, the constituents predicted in that condition are nearly as reliable as in full parsing. Further comparison of the full parser and hedgebank parser results reveals that the majority of differences are due to under-predicting the constituents by the hedgebank parser. On average, 1.9 (out of 13.8) of the constituents in the hedge parse tree of a sentence found by the full parser are not predicted in the hedgebank parser. The latter also over-predicts 1.2 constituents in a sentence and mis-predicts the label of 0.3 constituents on average, compared to the former.

3.8.2 Prioritization and Pruning Results

For efficient decoding, BUBS parser allows for approximate inference using some prioritization and pruning methods. The parser implements a beam-search variant of the CYK algorithm which at each chart cell, all possible non-terminals are sorted by a prioritization function (FOM). BUBS also provides complete closure pruning technique to predict whether a chart cell participates in a correct parse, and closes the cell if it does not. For each cell, a binary classifier decides to open or close the cell using lexical and syntactic features of the input sentence. Complete closure technique, prunes 56% of all chart cells in full parsing of the development set, as reported in Bodenstab et al. (2011).

Parser	Hedge Parsing Acc/Eff			
	P	R	F1	w/s
Full w/full CYK	88.8	89.2	89.0	2.4
Inside FOM	87.7	88.2	87.9	285.8
Inside FOM + Complete Closure	88.0	88.3	88.1	716.2
Boundary FOM	88.6	89.0	88.8	204.7
Boundary FOM + Complete Closure	88.9	89.1	89.0	544.7
Hedgebank	87.6	84.4	86.0	25.7
Inside FOM	86.7	83.8	85.2	1083
Inside FOM + Complete Closure	86.7	83.8	85.2	1405
Boundary FOM	87.1	84.0	85.5	871.2
Boundary FOM + Complete Closure	87.0	83.9	85.5	1281

Table 3.4: English hedge parsing results for pruning and prioritization models on section 24 for L=7.

In order to allow for FOM and complete closure in decoding, we should limit the beam-width of the chart cells to a fraction of the total possible constituents. We use the default beam limit in the parser, which is the maximum beam-width of 30 for cells of span >1, and 60 for cells of span 1 plus 20 for unary rules (those with only a single child). Complete closure pruning model and lexical boundary FOM should be trained for a specific grammar. This grammar is a full PCFG or a hedgebank grammar depending on the grammar we use to parse the data. To train the models, we use BUBS' default parameter values and WSJ section 22 as the development data.

Table 3.4 shows the results of combining hedge parsing with prioritization and pruning models in full parsing and hedge transforming the results, and also hedge parsing using a hedgebank grammar. In both cases, we present the accuracy and efficiency of exhaustive parsing, beam-search parsing with Inside and Lexical boundary FOMs, and beam-search parsing with the complete closure pruning. As expected, beam-search parsing is much faster than the exhaustive parsing but accuracy slightly decreases. This accuracy loss is more in Inside FOM, which is a trivial prioritization function compared to the lexical boundary FOM. Combining both prioritization models with the complete closure pruning model improves parsing both accuracy-wise and speed-wise. The prioritization and pruning methods we investigated here, show similar patterns when they are applied to (a)

the full chart in parsing with a full PCFG, and (b) the hedge-constrained CYK chart in parsing with a hedgebank grammar. However, we see more accuracy degradation in the pruned parsing compared to the exhaustive parsing in condition *b*. In both conditions *a* and *b*, the best-performing prioritization and pruning method (i.e, "Boundary FOM + Complete Closure") results in huge speedups compared to exhaustive parsing: 544.7 w/s as opposed to 2.4 w/s in condition *a*, and 1281 w/s compared to 25.7 w/s in condition *b*. These speedups come with no cost in accuracy in condition *a* (89.0% F1-score), but with slight accuracy loss (85.5% F1-score compared to 86.0% F1-score) in condition *b*.

3.8.3 Hedge Segmentation and Parsing Results

Table 3.5 shows the results on the English development set when segmenting prior to hedge parsing. The first row shows the result with no segmentation, the same as the last row in Table 3.3 for ease of reference. The next row shows behavior with perfect segmentation. The final two rows show performance with automatic segmentation, using a model that includes either unlabeled or labeled segmentation tags, as described in Section 3.6.1. Segmentation accuracy is better for the model with labels, although overall that accuracy is rather low. We achieve nearly another order of magnitude speedup over hedge parsing without segmentation, but again at the cost of nearly 5 percent F1-score.

In addition to combining the pruning and prioritization approaches with hedge parsing in no segmentation scenario, which the results were presented in Section 3.8.2, we looked into combining these approaches with hedge segmentation. Our primary investigation shows that in contrast to no segmentation scenario, hedge segmentation does not combine

Segmentation	Seg F1	Hedge Parsing Acc/Eff			
		P	R	F1	w/s
None	n/a	87.6	84.4	86.0	25.7
Oracle	100	91.3	88.9	90.1	188.6
Unlabeled	80.6	77.2	75.3	76.2	159.1
Labeled	83.8	83.1	79.5	81.3	195.8

Table 3.5: English hedge segmentation and parsing results on section 24 for L=7.

well with the pruning and prioritization approaches introduced in Section 3.3. Although up to an order of magnitude speedups are achieved compared to the exhaustive case, we noticed more than 10% absolute accuracy degradation. More investigation on this topic is of interest as future work.

In all scenarios where the chart is constrained to search for hedges, we should learn a matched grammar. In the no-segmentation scenario, we learn a hedgebank grammar, which is matched to the maximum length allowed by the parser. In the pre-segmentation scenario, we first decompose the hedge transformed treebank into its hedge segments and then learn a hedgebank grammar from the new corpus. We noticed severe hedge parsing accuracy degradation in unmatched conditions: we observed about 15% accuracy reduction where a full CFG is used with a hedge-constrained chart compared to the case where a full CFG is used with a full CYK chart. Another case is about 15% accuracy reduction in the pre-segmentation scenario, where the hedgebank grammar is trained from a non-decomposed hedge transformed treebank.

3.8.4 Test Set Results

We present results of our best configurations on the English test set, section 23, in Table 3.6, and the Chinese test set, articles 271-300, in Table 3.7. The results show the same patterns as on the development set. With no segmentation of the input, parsing with hedgebank grammar is significantly faster, but less accurate, than the baseline full parsing and then hedge-transform the result. With pre-segmentation of the input, more speedups at the expense of accuracy are achieved. The general behaviors are similar in Chinese and English, however, accuracy reduction in moving from no-segmenting to pre-segmenting the input is more noticeable in Chinese (for L=7), presumably due to our lower hedge segmentation accuracy for this language.

Finally, Figure 3.8 shows the speed of inference, labeled precision, labeled recall, and F-measure of annotating hedge constituents on the English and Chinese test sets as a function of the maximum span parameter L, versus the baseline parser. Overall, English

Segmentation	Grammar	Segmentation Acc			Hedge Parsing Acc/Eff			
		P	R	F1	P	R	F1	w/s
None	Full w/full CYK	n/a			90.3	90.3	90.3	2.7
None	Hedgebank	n/a			88.3	85.3	86.8	26.2
Labeled	Hedgebank	84.0	86.6	85.3	85.1	81.1	83.0	203.0

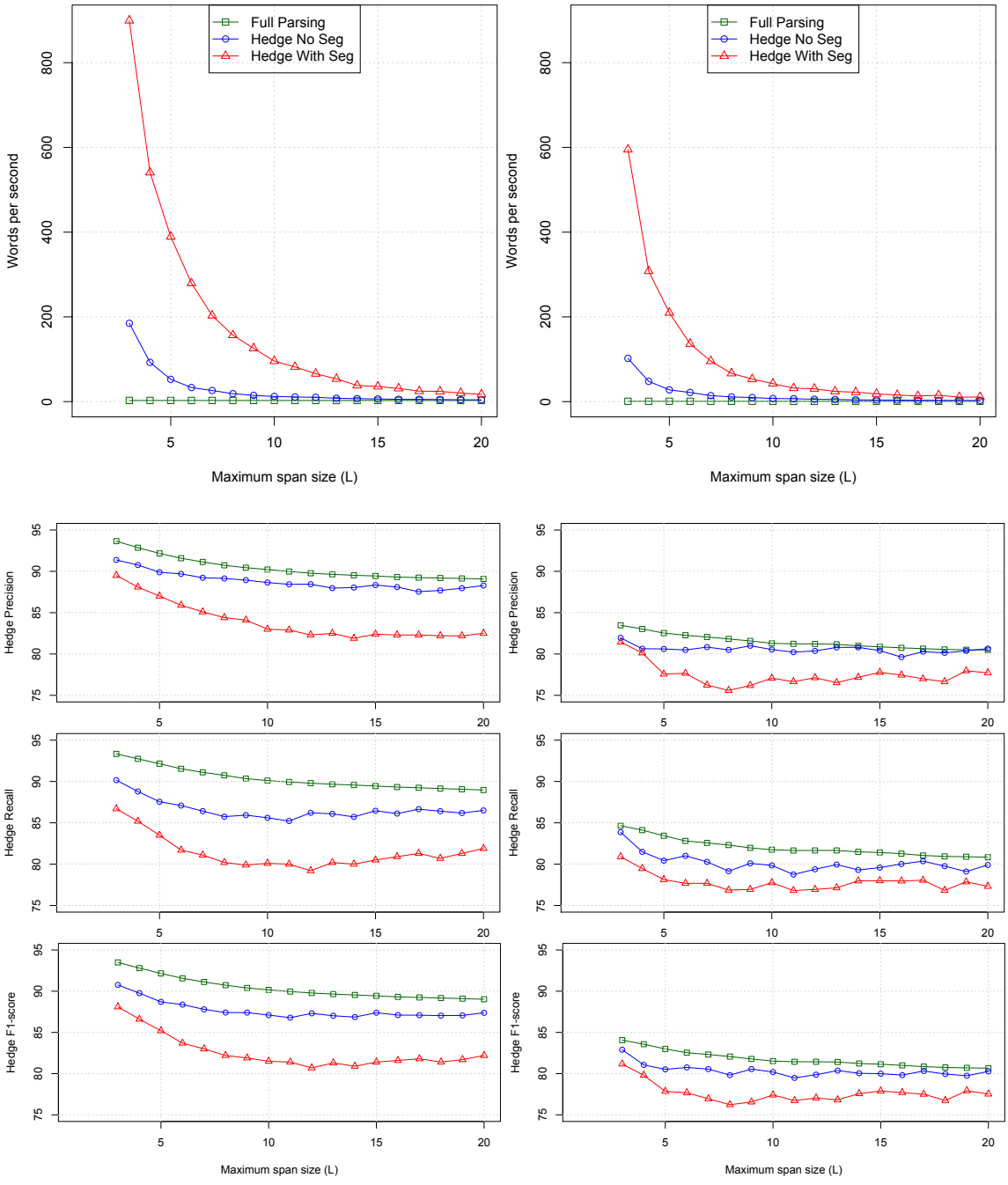
Table 3.6: English hedge segmentation and parsing results on test data, section 23, for L=7.

Segmentation	Grammar	Segmentation Acc			Hedge Parsing Acc/Eff			
		P	R	F1	P	R	F1	w/s
None	Full w/full CYK	n/a			82.1	82.6	82.3	0.7
None	Hedgebank	n/a			80.8	80.3	80.6	14.0
Labeled	Hedgebank	78.9	80.6	79.8	76.2	77.7	76.9	95.5

Table 3.7: Chinese hedge segmentation and parsing results on test data, for L=7.

hedge parsing is faster and the speedups are larger compared to Chinese. Likewise, English hedge parsing is generally more accurate than Chinese. This is consistent with the previous findings in the literature that PCFG parsing is typically more accurate for English than Chinese. Keep in mind that the number of reference constituents increases as L increases, hence both precision and recall can decrease as the parameter grows.

For both English and Chinese, using a hedgebank grammar achieves speedups, at the cost of accuracy, over using a full grammar, and segmentation achieves large speedups for smaller L values. In the pre-segmentation scenario, we observe consistent accuracy degradation in English, pointing to the need for improved segmentation. Similarly, we observe accuracy reduction for Chinese, but the curve is flatter than that for English. The reason could be the difference in hedge segmentation accuracy range for various L values in these languages. For Chinese, hedge segmentation accuracy varies only about 6% absolute F-measure from 86% F-measure at L=3 to 80% F-measure at L=20, whereas hedge segmentation accuracy variation is 11% absolute F-measure in English, from 94% at L=3 to 83% at L=20. This pattern again emphasizes the impact of segmentation performance on hedge parsing performance. An online joint parsing and segmentation approach through an incremental parser with linear complexity is the focus of the next chapter. That approach segments and parses hedges simultaneously as the input is being parsed, preventing the cascading segmentation errors problem.



(a) (b)

Figure 3.8: (a) English and (b) Chinese hedge parsing efficiency and accuracy results on test data, for L=3–20.

3.9 Summary

We proposed hedge parsing approach for applications that require a fast syntactic analysis of the input beyond shallow bracketing. We introduced the approach and some initial experiments on different inference strategies. Hedge parsing combines well with the pruning and prioritization approaches for efficient CYK parsing. We found that hedge parsing can achieve a significant speedup by pre-segmenting the string, although with the problem of cascading segmentation errors to parsing step, particularly for English. In Chapter 4, we improve hedge parsing by employing a classifier-based shift-reduce parser. Although accuracy is generally lower with classifier-based parsers as opposed to dynamic programming, the framework allows for very efficient $O(N)$ performance. More importantly, such a parser allows for highly-accurate real-time hedge parsing, without requiring pre-segmentation of the sentence offline or knowing the entire sentence in advance.

Chapter 4

Real-time Hedge Annotation and Segmentation via Incremental Parsing

4.1 Introduction

In some NLP applications, language processing is performed incrementally based on the generation of partial outputs (segments) from a parser. To achieve *low-latency* real-time processing, these partial segments must be returned during parsing. Latency is the difference in time or number of tokens, between the point that an output is produced and the point that it is released to the processing application.

In the previous chapter, we introduced shallow syntactic annotations, called hedges, as local hierarchical structure of phrases within a limited span L. Hedges are independent and sequential segments that could be retrieved even without knowing the entire sentence. To produce hedges in Chapter 3, we parsed the sentence using a transformed context-free grammar (CFG) and also constrained the parser. We learned the grammar from a hedge-transformed treebank, and constrained the search performed by the CYK parser to further improve efficiency/accuracy. We also demonstrated that pre-segmenting the sentences prior to hedge parsing yields large speedups, though at a substantial loss in accuracy due to cascading segmentation errors.

In this chapter, we propose a method for joint hedge annotation and segmentation, incrementally as the sentence is being parsed, without requiring any grammar or treebank

transform or constraining the parser. We demonstrate that our approach has the following advantages over the previous findings: a) since we do not transform the treebank, we benefit from the extra structure built by a full CFG parsing which results in more accurate hedges; b) as a result of simultaneous hedge annotation and segmentation, we avoid the pre-segmentation errors that cascade to the parsing; and c) the latency of producing hedges greatly decreases, because the hedges are returned incrementally during the parse.

We use a shift-reduce parsing approach with beam search decoding which builds up the parse tree incrementally. The parser is trained on the original treebank and performs a full CFG parsing. To return low-latency hedges, we augment the parser with a simple and novel buffering algorithm which returns the hedges incrementally as the sentence is being parsed. The buffering algorithm stores partial constituents of parsing in a buffer, and gradually releases the constituents that it assumes are most likely hedges. Note that the hedges are returned in the left-to-right order as they appear in the full parse tree.

Recall that one of the main characteristics of a real-time system is that no revision of the output is allowed. We should make sure that the released segments are not changed later during the decoding. In an incremental beam-search decoding, there is uncertainty in the partial output due to different structures in the beam. However, there are certain points during decoding where all the beams agree on the same intermediate output. At these points, we have *stable* parse results, which are the constituents that will be unchanged by any subsequent decoding. After the stable results are recognized, the buffering algorithm receives them as the input. This ensures that the released hedges from the buffering algorithm are stable, even if they are not fully correct compared to the true hedges. We show that using an incremental parser augmented with the buffering algorithm is very effective in producing low-latency hedge segments. For example, for L=7, we achieve about 36% improvement in latency with less than 1% absolute accuracy loss in returning 100% stable hedges compared to offline production of hedges, in both English end Chinese.

In a related work, Kato et al. (2004) proposed a method to delay the decision of which partial parse tree should be returned, until the validity of the partial parse tree becomes

greater than a threshold. They used a probabilistic incremental parser based on tree adjoining grammar (TAG) to parse the input, and incrementally calculated the validity for each partial parse tree on a word-by-word basis. They demonstrated a trade-off between the delay and the percentage of valid partial parse trees in the output. To recognize stable parses, they used TAG formalism, which is more complex than context-free. For more detail about TAG formalism refer to Section 2.5.5 of Chapter 2. Instead, we will use context-free parsing and recognize stable parses only based on beam agreement which requires minor extra computation. In another study, Selfridge et al. (2011) investigated stability and accuracy of partial phrases in word lattices of an incremental speech recognition task. They compared the stability/accuracy trade-off between three incremental speech recognition methods. In other related work, Saraclar et al. (2002) improved the latency of a real-time broadcast news transcription system by improving the front end and the acoustic model of their baseline speech recognition system. These studies focus on automatic speech recognition finite-state lattices and do not address the task of stability in context-free syntactic parsing.

We leverage partial syntactic parsing stability for real-time syntax-based input segmentation. Current segmentation methods we discussed in detail in Section 2.7.1 of Chapter 2 segment the input stream offline and as a pre-processing step for decoding (Cettolo and Federico, 2006; Matusov et al., 2007; Fügen et al., 2007; Rangarajan Sridhar et al., 2013; Oda et al., 2014). Some of these methods could be applied real-time using a classifier to decide segments boundaries, such as classifying punctuation marks in Rangarajan Sridhar et al. (2013), however the classifier may be a significant overhead on segmentation. Moreover, these methods produce raw word sequences of the input, whereas we use syntactic knowledge in segmentation as well as annotation of the input, which we demonstrate to be useful for simultaneous translation in the next chapter.

4.2 Latency

In a real-time incremental processing system, it is important to assess latency. A segment (particularly defined as hedges in this chapter) might not be released immediately after its last word is parsed, instead it might be released after several following words are parsed. We use a word-based measure to define segment latency in this chapter. Latency for each word is the word (token) difference between the positions of the sentence where (1) the word is parsed, and (2) the segment containing the word is returned from the parser.

The ideal scenario, which yields minimum delay, is when each segment is returned right after its last word is parsed. The largest delay is when the sentence is completely parsed and then the segments are returned. In the case of hedge segments, this is performed by returning the hedge segments after the entire sentence is hedge parsed either by using a hedgebank grammar or using a full grammar followed by hedge transforming the result. In the next section, we explain the framework and algorithms to reduce the average hedge annotation and segmentation latency, while minimizing hedge parsing accuracy loss.

4.3 Shift-reduce Parsing

In this chapter, we use ZPAR (Zhang and Clark, 2011), a classifier-based shift-reduce parsing framework for incremental phrase-structure parsing. ZPAR combines a global discriminative model for incremental structured prediction, with a beam-search algorithm for decoding. In this section, we explain the training and decoding procedure of the parser, and how the parser incorporates beam-search decoding to handle the structured ambiguity in natural language parsing.

4.3.1 Classifier-based Parser

A basic shift-reduce parser works by doing a series of shifting or reducing actions, as described in Section 2.4.2 of Chapter 2. The decision whether to shift or reduce at each

parsing step, is traditionally made using a context-free grammar. In state-of-the-art statistical parsers, this decision is made by a *classifier* that chooses an action based on a set of features derived from the local parser state, with no explicit grammar (Sagae and Lavie, 2005; Wang et al., 2006b). The decoding and training framework for the classifier-based parser in this chapter is the global incremental structured prediction model we described previously in Chapter 2. In this model, the output y corresponding to an input x (x is an input sentence and y is its parse result in this case) is built through a number of incremental steps. After the last step, the global feature vector of each possible output is calculated by summing up the local feature vectors at each incremental step, and then the score of that possible output is calculated by the inner product of its global feature vector and the parameter vector of the model.

The parameter vector is trained using the averaged perceptron algorithm (described in Section 2.3.1 of Chapter 2). The perceptron algorithm initializes the parameter vector to zero, and updates it after decoding the training examples. The output parse for each input example is produced by the decoder, which is a beam-search decoder in the parser we use, and works in the way we explain in section 4.3.3. If the decoder output is different from the correct parse, then the parameter vector is updated by adding the global feature vector of the correct output, and subtracting the global feature vector of the decoder output. During training, ZPAR uses the early-update method (Collins and Roark, 2004): at any incremental step, if there is no possibility of the correct parse being in the agenda, decoding is stopped and the best decoder partial output and the correct partial output are passed for parameter estimation. Early-update improves the accuracy and efficiency of the original perceptron for beam-search decoding. Feature templates in a classifier-based parser typically include n-grams of word-forms, POS tags, and lexical heads of top nodes in the parser stack and queue. More details about the feature templates used for phrase-structure parsing in ZPAR are provided in Zhang and Clark (2011).

4.3.2 Parsing Actions

The parsing algorithm in ZPAR, considers only trees with unary and binary branching. It uses an instance of the binarization process described in Chapter 2 Section 2.1.3. This binarization converts each node with n (>2) into $n-1$ binary nodes, and new temporary non-terminals introduced in this process are marked with asterisks. The binarized treebank is then used for training the parser. This treebank contains also lexical head annotations and uses them as features for training the parser. The lexical head of each of the asterisked non-terminals is the same as the head of the original non-terminal. In Appendix B, we present the English and Chinese constituent head-finding rules we used to annotate the lexical heads. For more details about how these rules work, refer to Section 2.5.3 of Chapter 2.

According to this binary transformation, the parser extends the actions of a basic shift-reduce parser to the following types:

- SHIFT, which pushes the next node in the queue onto the stack.
- REDUCE-unary-X, which makes a new unary node with label X by popping the top node of the stack, making it the child of the new node, and pushing the new node onto the stack.
- REDUCE-binary-{L/R}-X, which makes a new binary node with label X by popping the two top nodes of the stack, making them the left and right children of the new node, and pushing the new node onto the stack. The left (L) and right (R) indicate the dependency direction of the head of the new node.

Label X is the constituent type, e.g., NP, VP, PP, etc. The parser's output is a binary parse tree which is de-binarized to the original n-ary format at the end of parsing.

4.3.3 Beam-search Decoding

The basic shift-reduce parser can be extended to enable handling of structural ambiguity in natural language parsing. A common approach is to explore multiple derivations in

parallel. To make the amount of memory tractable to store multiple derivations as they are constructed, usually a fixed *beam* of derivations is maintained and the others are not pursued. Incremental beam search strategies have been successfully used for constituency or dependency parsing (Roark, 2001; Collins and Roark, 2004; Zhang and Clark, 2008; Huang et al., 2009; Huang and Sagae, 2010; Zhang and Nivre, 2011; Goldberg et al., 2013).

Beam-search is used for decoding in the incremental structured prediction model of the parser, and it is one of the two major components of this model besides the averaged perceptron. At each incremental step of parsing, the parser keeps the B-best partial parse results in an agenda. The partially built structures are represented as a set of state items. At each incremental step, every state item from the agenda is extended in all possible ways (actions), generating new state items. The new state items are ranked by their scores and the B-best are put back to the agenda. Note that the score of a partial output is computed using the parameter vector trained by the averaged perceptron algorithm, as described in section 4.3.1. This process iterates until the stopping criteria is met, and then the 1-best item from the last agenda is the final output.

Figure 4.1 presents an example, in form of a graph, of beam-search decoding for B=4 for a sentence in the development set. Nodes in this graph indicate state items and arcs indicate actions. Each level of this graph corresponds to an incremental step of parsing, which is indexed by numbers. It takes 23 steps (idx=0–22) to fully parse this example sentence. At the first step, the only possible action is SHIFT. At the other steps, every state item is extended by all possible actions, but only top B (4 in this case) of those are kept in the agenda and could possibly be extended later on. To save space, in the figure we only show these 4 actions. Action S_*word* means that the word and its POS tag are shifted to the stack, and R_X means a unary or binary reduction to constituent X. Double-circle nodes represent stable partial outputs, which we explain in the next section.

By following the actions in the path that starts from the first step and continues to the final step, the final parse result of the sentence is built, as shown in Figure 4.2(a). Figure 4.2(a) also shows the non-temporary constituent numbers in the order they are built, starting

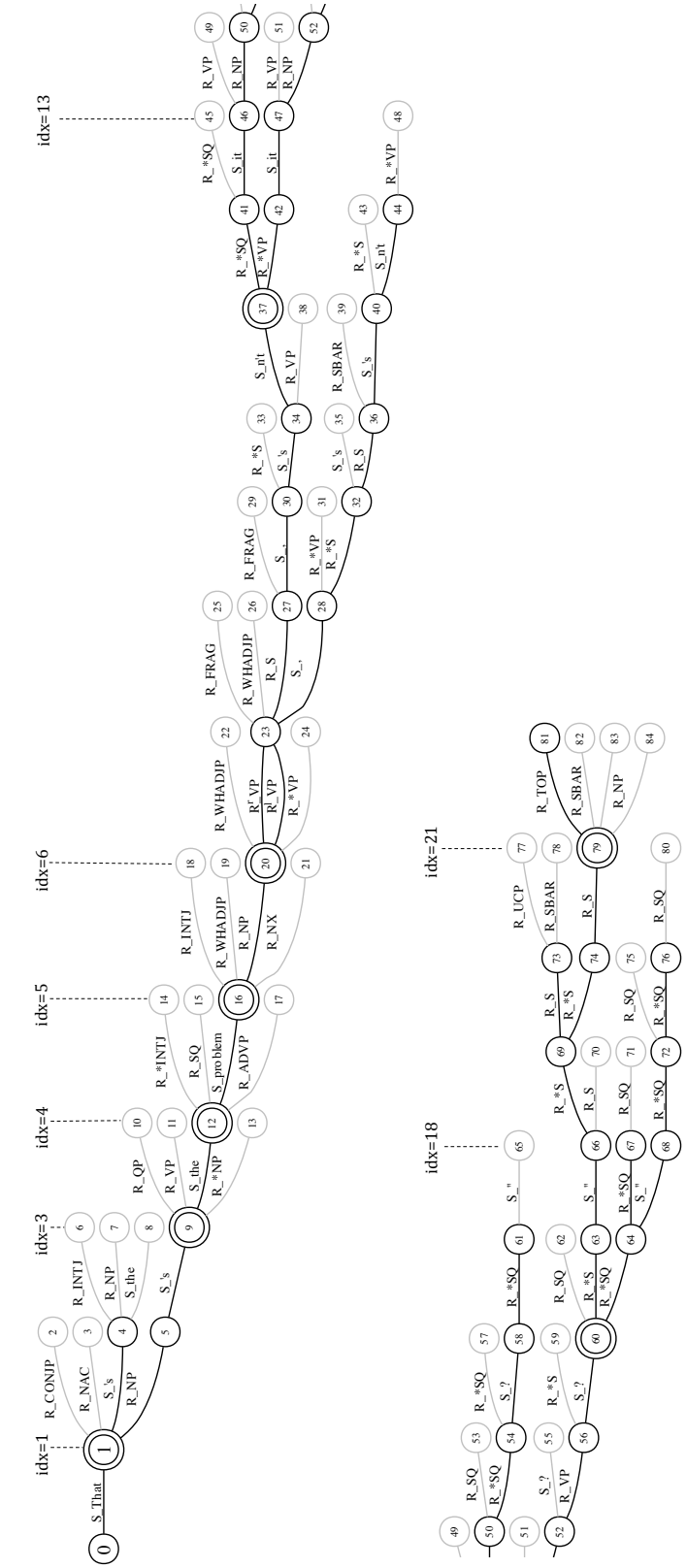


Figure 4.1: Beam-search decoding example.

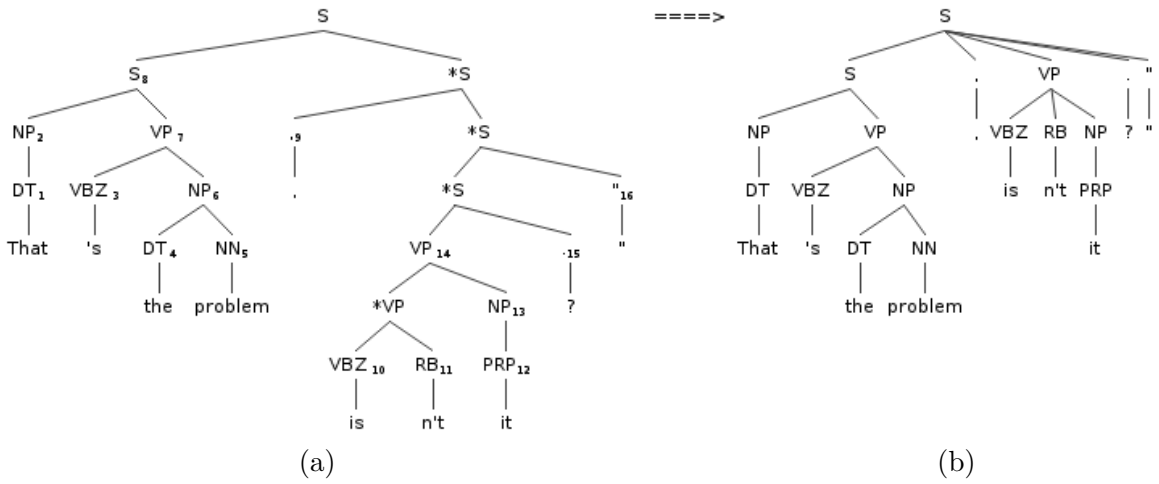


Figure 4.2: (a) Binary, and (b) n-ary trees for parse result of the example sentence in Figure 4.1.

from 1 for the left-corner of the tree, to 16 for the right-most constituent excluding TOP. This binary tree is then transformed to the original n-ary format and the temporary nodes are removed, shown in Figure 4.2(b). Note that the constituent numbers are equivalent to in-order traversal of this tree. The parenthetic representation of this tree is "(S (S (NP (DT That)) (VP (VBZ 's) (NP (DT the) (NN problem)))) (, ,) (VP (VBZ is) (RB n't) (NP (PRP it))) (. ?) (" "))".

In the next section, we analyze the stability of partial parse trees produced by the ZPAR shift-reduce phrase-structure parser which uses a full CFG.

4.4 Stable Partial Outputs

The parser's output structure is built incrementally. At each step of parsing, an incremental structure is added to the partially built parse tree. A partial output is *stable* if it remains unchanged by any subsequent decoding. In a deterministic parser, every parsing step produces a stable partial. In a non-deterministic parser, which keeps different derivations in a beam, the most probable partial at any step may not be the best scoring at the end of the derivation. The reason for this is that as more input words are received and the states are extended, a different path may become the most probable. For example, in

Figure 4.1, at idx=6, the partial result "(NP (DT That)) (VBZ 's) (INTJ (DT the) (NN problem))" at the first beam has the highest score, but the partial "(NP (DT That)) (VBZ 's) (NP (DT the) (NN problem))" at the third beam remains at the final output.

A stable partial output in a shift-reduce parser with beam search decoding becomes available where all the B-best beams agree on the same partial output. Figure 4.1, marks the incremental steps (indices) where a stable partial output is built. At the marked indices, all top 4 beams agree on the entire or a sub-part of the partial parse results. If all the beams are extended from a single node at a marked index—indices 1, 3, 4, 5, 6, 21—then the entire partial parse up to that index is stable. Otherwise, the stable sub-part is built at the closest common-branching index in the past—indices 11 and 16 for indices 13 and 18 respectively. The double-circle nodes, indicate the state items where the stable result is built. In total, after parsing 8 constituents out of the total 16 constituents in the final parse result (excluding TOP), the partial parse result is stable and could be released. We call such constituents "stable constituents". The stable constituents in our example are constituent numbers 1, 3, 4, 5, 6, 7, 11, 15, according to the constituent numbers in Figure 4.2(a). Alternatively, stable constituent numbers can be interpreted as the total number of constituents (including POS tags) in the stable partial parses at that point. These stable partial parses are:

(DT That)

(NP (DT That)) (VBZ 's)

(NP (DT That)) (VBZ 's) (DT the)

(NP (DT That)) (VBZ 's) (DT the) (NN problem)

(NP (DT That)) (VBZ 's) (NP (DT the) (NN problem))

(NP (DT That)) (VP (VBZ 's) (NP (DT the) (NN problem)))

(S (NP (DT That)) (VP (VBZ 's) (NP (DT the) (NN problem))))) (, ,) (VBZ is) (RB n't)

(S (NP (DT That)) (VP (VBZ 's) (NP (DT the) (NN problem))))) (, ,) (VP (VBZ is) (RB n't) (NP (PRP it))) (. ?)

Tables 4.1 and 4.2 present frequency and distribution of stable constituents compared to total constituents in the full CFG parse trees of sentences for various beam sizes (B) for English and Chinese development data. The "Total", "Stable", and "Pct Stable" columns show the average number of total constituents, average number and percentage

Table 4.1: Average frequency and distribution of stable constituents in English development set.

Beam	Total	Stable	Pct Stable	Pct Stable in First Half	Pct Stable in Second Half
1	42.5	42.5	100%	100%	100%
2	42.4	25.2	59.6%	70%	55%
4	42.3	16.2	38.3%	57%	30%
8	42.3	13.3	31.4%	52%	22%
16	42.3	11.7	27.6%	48%	17%

Table 4.2: Average frequency and distribution of stable constituents in Chinese development set.

Beam	Total	Stable	Pct Stable	Pct Stable in First Half	Pct Stable in Second Half
1	39.0	39.0	100%	100%	100%
2	39.3	19.3	49.0%	62%	52%
4	39.2	11.3	28.8%	47%	31%
8	39.2	8.4	21.4%	42%	23%
16	39.2	7.6	19.3%	39%	19%

of stable constituents in the parse trees respectively. Frequency of total constituents is similar in all beam sizes, but frequency of stable constituents decreases remarkably for larger beam sizes. As the beam size exponentially increases, the percentage of stable constituents exponentially decreases. This means that as the beam size increases, it is less likely to achieve a beam agreement during parsing. The last two columns of the tables demonstrate the distribution of stable constituents in in-order traversal of the parse trees, which is the order of parsing the constituents. The "Pct Stable in First Half" column shows what percentage of the total constituents in the first half of the in-order list are stable constituents. The next column, shows this percentage in the second half of the list. For a deterministic parse (which is the case of B=1) the stable constituents are distributed evenly across the halves. This is due to the fact that every constituent in a deterministic parse is stable as soon as it is available. As the beam size increases, the stable constituents are more often among the constituents in the first half rather than the second half, particularly for larger beam sizes. This means that it is more likely to achieve a beam agreement during the first steps of incremental parsing compared to the last steps, particularly for larger beam size values. Frequency and distribution patterns

are quite similar in English and Chinese, although the percentage of stable constituents during parsing of Chinese is about 10 percent absolute less than that in English.

The general trends we find in studying the stable partial parser's outputs are as follows:
(a) In incremental shift-reduce parsing with beam-search decoding using different beam sizes, after parsing some of the constituents of the syntactic structure of a sentence—which we call them stable constituents—the partial parse structure up to that point becomes stable.
(b) Stable constituents are more frequent in parsing with smaller beam sizes.
(c) Stable constituents are distributed mostly in the first half of the final constituents rather than the second half. This difference is more noticeable in larger beam sizes.
(d) The patterns are the same in English and Chinese, however Chinese parsing has generally fewer number of stable constitutes.

By knowing the stable partial results, we could release intermediate segments to the real-time application, without requiring future revisions, before parsing the entire input is finished. In particular, these stable partial results may contain hedge segments, which can be release intermediately. It may seem that such hedge segments can be released by hedge transforming every stable partial parse tree. However, this will release a lot of constituents before building complete hedges, as we will explain in section 4.7. In the following section, we introduce an algorithm that solves this problem.

4.5 Buffering Algorithm for Stable Low-latency Hedge Segmentation

Considering that hedges are disjoint subtrees, it is possible to recognize and return these subtrees as the sentence is being parsed using an incremental parser. To recognize these subtrees it is not enough to find the stable partial results, since they do not necessarily match or contain the hedge subtrees. Thus, we need a complement method for bookkeeping the hedges and releasing them as soon as they are recognized. In this section, we present

a simple and low-overhead bookkeeping method to identify hedge subtrees in the stable parse results with a high degree of confidence. This method, which we refer to as the buffering algorithm, works by adding a new queue structure to the parser framework and using some simple rules to check if the buffer contains any hedges.

An alternative method to our rule-based procedure is to use statistical models to identify hedges, such as applying a classifier at each stable partial parse to predict hedge boundaries. A classifier can be trained on the partial subtrees of the full parse trees in the original treebank, as well as the gold-standard hedges available in these partial subtrees. However, using such a classifier requires training a separate statistical model in addition to the parser. As opposed to the simple rule-based buffering algorithm, applying the classifier could be a significant overhead to the incremental framework, modulo the errors the classifier may make.

A possible modification to our proposed approach could be a *risky* buffering strategy. In this strategy, we keep the buffering algorithm unchanged. However, instead of sending the stable partial parse results to the buffering algorithm, we send the partial results that are not necessarily stable and may change during later full parsing. These partial results could, for example, be the outputs that the majority of beams agree on, or the outputs on the top-most beam with high score margin. A risky buffering strategy could improve latency of segmentation compared to the stable buffering method we present in this chapter. However, there is risk of future revisions in the released segments from the buffering algorithm, due to mismatches between the current partial parse result and the future results. These mismatches might be the result of 'extra shift' action(s), 'extra reduce' action(s), and/or 'different reduce' action(s) in the current parse compared to the future parse. Thus, the entire or part of the released segments maybe parsed differently in future. Moreover, the released segments might contain extra input tokens compared to the segments that can be released in future, due to 'extra shift' actions in the current partial output, or they might be missing some of the input tokens due to 'extra reduce' actions. The risky buffering strategy should have a recovery method in case revisions are required.

In this section we introduce the buffering algorithm. As described earlier in Section 4.3, the shift-reduce parser builds the output parse tree by performing an action at each step, and storing the resulting node in the parser's stack. The buffering algorithm replicates the same procedure in a limited-span buffer: the action at each step and the resulting node are duplicated in the buffer. Every constituent in a hedge tree has a limited span, i.e., it spans at most L words. We apply the same constraint to the buffer: sum of the spans of the items in the buffer must not exceed L. When the buffer reaches the limit L, the algorithm releases all or part of the buffer that are most likely hedges. Note that since the buffer is constrained, the released segments will meet the hedge criterion of spanning at most L words. Figure 4.3 shows our proposed algorithm in detail, for a deterministic parser. This algorithm is straightforwardly extendable to a non-deterministic parser with beam search. To guarantee releasing of stable hedges in such a parser, buffering and releasing of the constituents is delayed until the parser reaches a stable constituent.

The BufferAndRelease function is called every time an action A is performed in the stack and a new constituent N is built. If N is the top-most non-terminal in the tree, the buffering process ends by completely emptying the buffer BF (lines 4-6 of the BufferAndRelease function in Figure 4.3). Otherwise, if BF spans more than L words before inserting N, it is released partially and then N is inserted (lines 7-9). If BF has not reached its limit, N is buffered without partially releasing the buffer (line 9).

The EmptyBuffer function releases buffer items according to the parameter k. Every item from the beginning of B to the last item spanning more than k words is released and the other items are saved in BF. Note that if there is no item to release (for example, if k=1 and all items are unary constituents or POS tags) then the first item is released and all others are saved (lines 7-8 of the EmptyBuffer function). A larger value for k results in more conservative buffer releasing, hence better accuracy and more latency in producing segments.

The DoAction function inserts N into the buffer. If the action A is SHIFT then N (which is a POS tag in this case) is inserted into the buffer (lines 2-3 of the DoAction function). If A is any type of unary or binary REDUCE, then N is a parent node which its children

```
1: function BufferAndRelease(L,k)
2:     N ←stack.top()                                          ▷ node
3:     A ←getAction(N)                                         ▷ action
4:     if N == TOP then
5:         EmptyBuffer(0)                                      ▷ empty all in buffer
6:         return
7:     else if buffer span == L + 1 then
8:         EmptyBuffer(k)                                      ▷ empty buffer according to k
9:     DoAction(N,A)
```

```
1: function EmptyBuffer(k)
2:     nb ← number of nodes in buffer BF                       ▷ nb: position of the last item with span < k
3:     nb_span ← span of BF[nb]                                ▷ span of the item in nb
4:     while nb ≥ 0 and nb_span ≤ k do                         ▷ find nb
5:         nb ← nb − 1
6:         nb_span ← span of BF[nb]
7:     if nb == 0 then                                         ▷ if no item to release
8:         release only 1st node in BF
9:     else
10:        release BF from beginning to nb
```

```
1: function DoAction(N,A)
2:     if A == SHIFT then
3:         shift N into buffer
4:     else if A == REDUCE then
5:         if N span ≤ buffer span then                        ▷ reduce is possible
6:             reduce N into buffer
7:         else
8:             EmptyBuffer(0)                                  ▷ empty all in buffer
```

Figure 4.3: The buffering algorithm for a deterministic parser.

have been already parsed. If the children are still in *BF*, the children are replaced with N (lines 5-6). Otherwise, either N spans more than L words or the hedge constituent has been over-segmented in error at an earlier stage; in either case, *BF* is completely emptied and N is not buffered (lines 7-8).

Figure 4.4 shows an example of running the buffering algorithm during deterministic parsing of the example sentence in Figure 4.2 for the span limit L=3 and the buffer parameter k=3. At the initial step, the algorithm receives the node $N = \mathrm{DT}_1$ which is produced by the action A = SHIFT in the parser. The buffer is initially empty. Since N is not the TOP node and the buffer span has not exceeded the limit L, DoAction runs

	node (N)	action (A)	buffer span	buffer (BF)	Release
1:	DT_1	Shift	0		
2:	NP_2	Reduce	1	DT_1	
3:	VBZ_3	Shift	1	NP_2	
4:	DT_4	Shift	2	NP_2 VBZ_3	
5:	NN_5	Shift	3	NP_2 VBZ_3 DT_4	
6:	NP_6	Reduce	4	NP_2 VBZ_3 DT_4 NN_5	
7:	VP_7	Reduce	3	VBZ_3 NP_6	NP_2
8:	S_8	Reduce	3	VP_7	
9:	$,_9$	Shift	0		VP_7
10:	VBZ_{10}	Shift	1	$,_9$	
11:	RB_{11}	Shift	2	$,_9$ VBZ_{10}	
12:	PRP_{12}	Shift	3	$,_9$ VBZ_{10} RB_{11}	
13:	NP_{13}	Reduce	4	$,_9$ VBZ_{10} RB_{11} PRP_{12}	
14:	VP_{14}	Reduce	3	VBZ_{10} RB_{11} NP_{13}	$,_9$
15:	$._{15}$	Shift	3	VP_{14}	
16:	$“_{16}$	Shift	4	$VP_{14}._{15}$	
17:	S	Reduce	2	$._{15}“_{16}$	VP_{14}
18:	TOP				$._{15}“_{16}$

Figure 4.4: Buffering algorithm run on the example in Figure 4.2 for L=3 and k=3.

without releasing the buffer and shifts DT_1 into the buffer. At step 2, the buffer span is 1 and the node $N = NP_2$ resulting from a reduce action in the parser is received. Since the span of N is equal to the buffer span, reduction is possible, hence NP_2 replaces DT_1 in the buffer. The algorithm continues buffering the inputs coming from the parser at steps 3, 4, and 5. At step 6, the buffer span becomes greater than the allowed limit L after the node NN_5 is inserted to the buffer at the end of step 5, so part of the buffer should be released by the EmptyBuffer function before reducing $N = NP_6$ into the buffer. The value of nb which is the number of nodes in the buffer is 4, and nb_span which is the span of the last item in the buffer (NN_5) is 1. The while loop in line 4, iterates over the buffer items from the end to the beginning of the buffer until it finds an item with span greater than k=3. At step 6, the while loop reaches the beginning of the buffer and nb becomes zero, meaning that there is no item spanning more than k=3 words, so all items have the chance to contribute in complete hedges later on. Thus the algorithm keeps all the items in the buffer and releases only the first item so that the buffer meets the criterion of spanning

up to L words. The next release occurs at step 8 where $N = \mathrm{S}_8$ and A = REDUCE is received. Since the span of N (4) is greater than the span of the buffer (3), reduction is not possible. Thus the buffer is completely released and S_8, which has a span greater than the allowed limit, is not inserted into the buffer. The process continues until the TOP node is received and the algorithm terminates by releasing the buffer items.

4.6 Data and Experiments

We run the experiments on the English WSJ Penn Treebank corpus (Marcus et al., 1999) using section 2-21 for training, section 24 for development, and section 23 for testing. For Chinese we use the Penn Chinese Treebank (Xue et al., 2005), articles 1-270 and 400-1151 for training, articles 301-325 for development, and articles 271-300 for testing. We perform shift-reduce parsing using the ZPAR parser version 0.6 with 15 training iterations for all beam sizes. Hedge parsing accuracy results are measured with precision, recall, and F1-score using the standard EVALB script. We evaluate accuracy with respect to the hedge transformed reference treebank. Latency results are hedge segmentation latency, measured as the averaged word-based latency across all the words in the data set. The latency for each word is the delay, in terms of number of words (tokens), between the position in the sentence that the word is parsed and the position the hedge segment containing the word is released.

4.7 Results

Figure 4.5 presents accuracy versus latency of real-time hedge production on the English development set for L=7, the buffer parameter k=1, 2, 3, and parser beam sizes B= 1, 2, 4, 8, 16. We choose L=7 for experiments, for comparability with Chapter 3. Keep in mind that hedges are only released at stable states, so they are of 100% stability. As the beam size increases from 1 to 16 at a certain k, hedge accuracies increase—due to better overall performance of parsing with a larger beam—although at the cost of latency. The

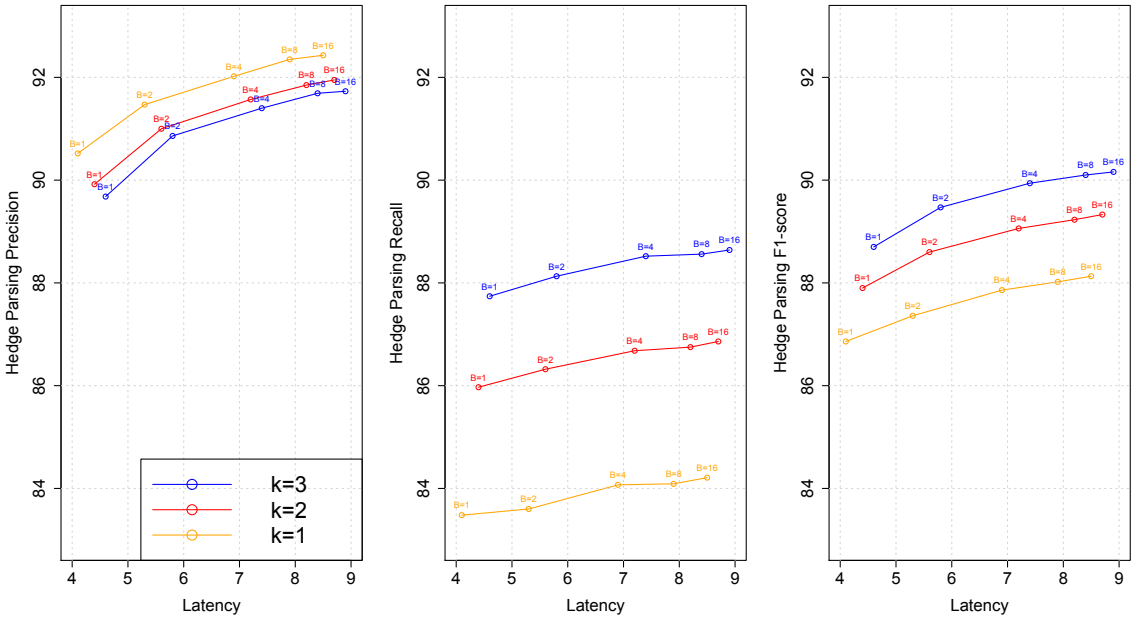


Figure 4.5: Real-time hedge parsing accuracy vs latency (words) on section 24, for L=7 and buffering parameters k=1, 2, 3.

reason for latency loss, as we analyzed in Section 4.4, is that when the beam size increases, number of stable partial constituents decreases and they tend to appear in the initial constituents. This figure also shows that with a more conservative buffer (k=3), hedges are more accurate, in terms of F1-score, but they are available with a greater latency. Changes of precision and recall for various k values at a certain beam, show that a more aggressive buffer, releases more precise hedges at a high cost of recall of correct hedges. The reason for this is that a more aggressive buffer releases flatter constituents.

Tables 4.3 and 4.4 show hedge parsing accuracy and latency of incremental shift-reduce parsing (B=16 and L=7) and its combination with on-line hedge segmentation methods, along with non-incremental CYK parsing with offline hedge segmentation, for English and Chinese development data. The first two rows are the results of using the CYK parser, first with a full CFG, and second with a hedgebank CFG. For details about parsing with a hedgebank grammar refer to Chapter 3 Section 3.5.1. The last four rows are the results of

Table 4.3: English hedge parsing accuracy and latency results on section 24 for L=7. Beam size for Shift-Reduce parsers is 16. Buffer parameter k for "Shift-Reduce + Buffering" is 3.

Parser	Hedge Release	Hedge Parsing Acc P	R	F1	Latency w
Full CYK	Offline	88.8	89.2	89.0	14.7
Hedgebank CYK	Offline	87.6	84.4	86.0	14.7
Shift-Reduce	Offline	91.6	90.2	90.9	14.7
Shift-Reduce + Oracle release	Real-time	91.6	90.2	90.9	5.9
Shift-Reduce + Every-stable release	Real-time	93.3	52.8	67.5	4.3
Shift-Reduce + Buffering	Real-time	91.7	88.6	90.2	8.9

Table 4.4: Chinese hedge parsing accuracy and latency results on dev set for L=7. Beam size for Shift-Reduce parsers is 16. Buffer parameter k for "Shift-Reduce + Buffering" is 3.

Parser	Hedge Release	Hedge Parsing Acc P	R	F1	Latency w
Full CYK	Offline	83.0	84.8	83.9	14.0
Hedgebank CYK	Offline	76.9	79.3	78.1	14.0
Shift-Reduce	Offline	85.3	86.5	85.9	14.1
Shift-Reduce + Oracle release	Real-time	85.3	86.5	85.9	8.3
Shift-Reduce + Every-stable release	Real-time	86.6	57.1	68.8	6.0
Shift-Reduce + Buffering	Real-time	85.4	83.7	84.6	10.0

using the shift-reduce parser in this chapter, first in the offline mode, and then in the real-time mode using three hedge release methods: "Shift-Reduce+Buffering" proposed in this chapter (k=3), "Shift-Reduce+Oracle release", which performs a perfect release of hedges once they become stable, and "Shift-Reduce+Every-stable release", which releases hedges by performing a hedge transform on the partial parse tree at every stable step. Offline release methods do not release hedges until the end of parsing, so they provide a ceiling on latency. Note that the latency results for offline CYK and shift-reduce parsers (the first three rows) are by definition similar. However, they are not identical, since the outputs of the parsers are not necessarily the same. In the oracle release method there is no hedge parsing accuracy degradation by definition compared to the offline mode, while latency is improved significantly. The every-stable method could be imagined as the extreme case of an aggressive buffer. Although it improves latency compared to offline as well as oracle release modes, it significantly decreases recall of correct hedges. About 48% of the correct hedges in English and 43% in Chinese are not recalled if we release hedges at every stable

Table 4.5: English hedge parsing accuracy and latency results on section 23 for L=7, B=16, k=3.

Parser	Hedge Release	Hedge Parsing Acc P	R	F1	Latency w
Full CYK	Offline	90.3	90.3	90.3	14.0
Hedgebank CYK	Offline	88.3	85.3	86.8	14.0
Shift-Reduce	Offline	92.5	91.0	91.7	14.1
Shift-Reduce + Buffering	Real-time	92.7	89.3	91.0	9.0

step without buffering. Combining the shift-reduce parser with buffering, increases the recall at the cost of some latency. This combination achieves a very close hedge parsing F1-score compared to the offline hedge segmentation (90.2% versus 90.9% for English and 84.6% versus 85.9% for Chinese) with a latency in between the offline and oracle hedge segmentation methods.

Table 4.5 presents results of our best configuration (B=16 and k=3) on English evaluation set, section 23, along with the best hedge parsing results reported in Chapter 3. Table 4.6 presents the same information for Chinese evaluation set. We find about 36% improvement in latency—at the cost of only less than 1% hedge parsing F-measure absolute—when using the buffering algorithm to produce low-latency hedges in shift-reduce parsing. The same pattern holds for Chinese, although accuracy/latency improvement in Chinese is slightly weaker than that in English. The reason is probably the fact that Chinese has fewer percentage of stable constituents, as we saw in section 4.4. Even though the shift-reduce and CYK parsers are not directly comparable due to different statistical models, our results suggest that using an incremental parser could be of utility for real-time applications. The two parsers have a close difference in their standard performance on full parsing than their performance on hedge parsing. Full parsing F-measure for the ZPAR shift-reduce parser is 90.4% and for BUBS CYK parser is 88.7%.

Figures 4.6 and 4.7 show the best configuration (B=16 and k=3) on English and Chinese evaluation sets for various L values. Again the behaviors are comparable in both languages. By increasing L, the latency in real-time parsing increases logarithmically towards the latency in offline parsing. Note that the latency in offline parsing is the same for all L

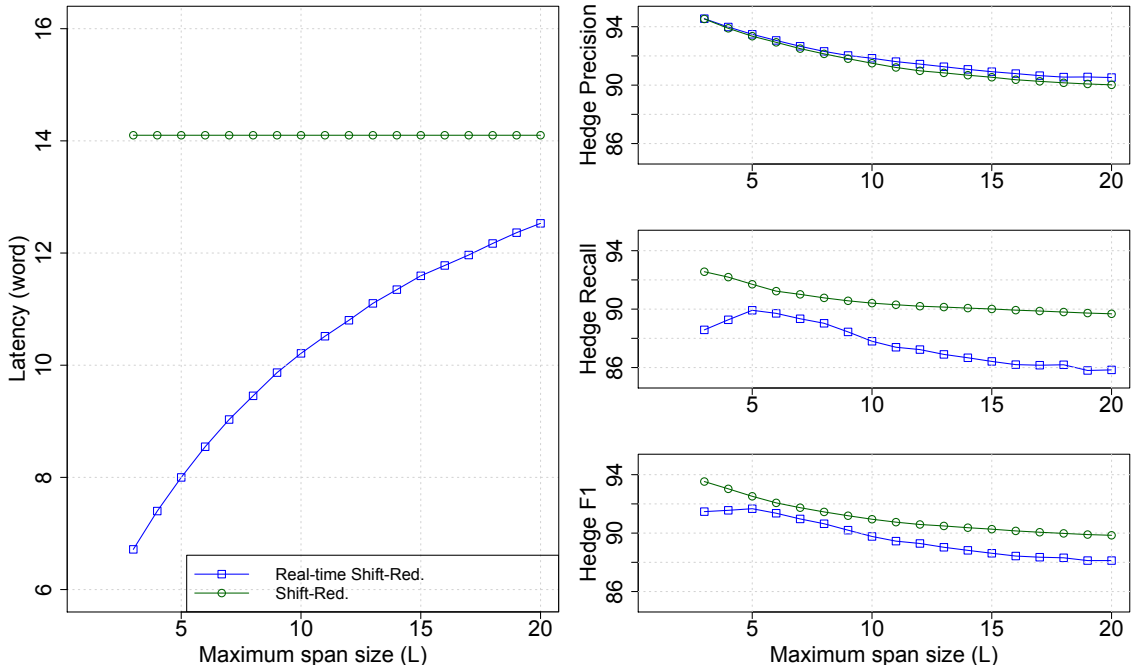


Figure 4.6: English hedge segmentation latency and hedge parsing accuracy on test data, section 23, for L=3–20, B=16, k=3.

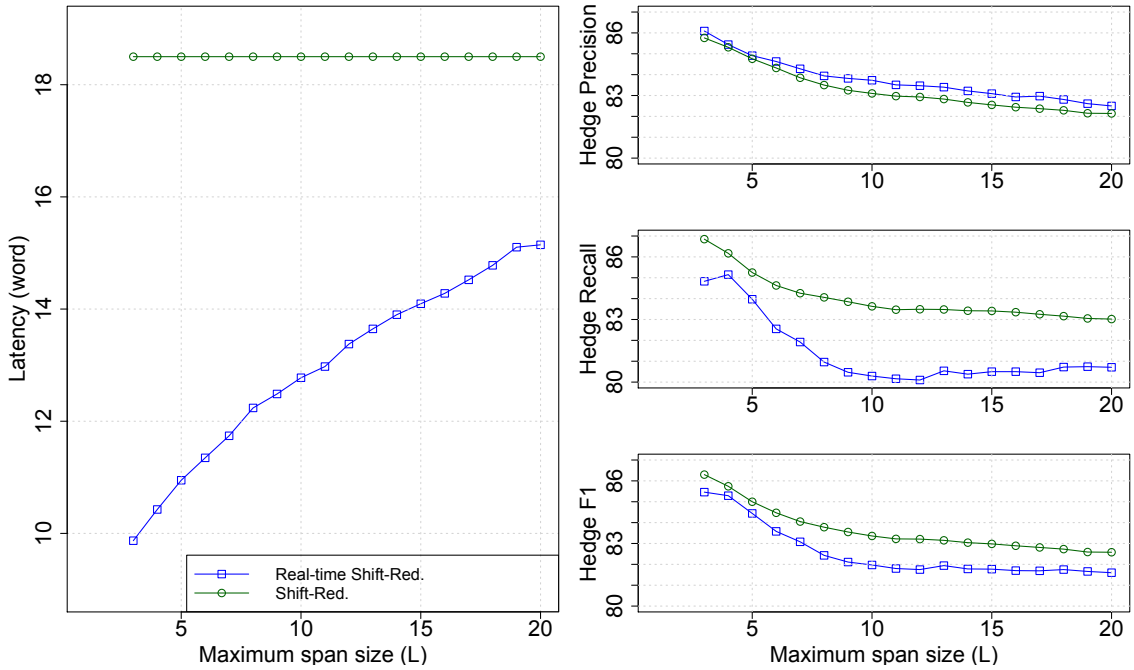


Figure 4.7: Chinese hedge segmentation latency and hedge parsing accuracy on test data, articles 271-300, for L=3–20, B=16, k=3.

Table 4.6: Chinese hedge parsing accuracy and latency results on test set for L=7, B=16, k=3.

Parser	Hedge Release	Hedge Parsing Acc P	R	F1	Latency w
Full CYK	Offline	82.7	82.8	82.7	18.4
Hedgebank CYK	Offline	77.2	78.4	77.8	18.4
Shift-Reduce	Offline	83.8	84.3	84.0	18.4
Shift-Reduce + Buffering	Real-time	84.3	81.9	83.1	11.7

values. The reason is that all of the hedges are returned after the full parsing is finished. Since the buffering algorithm might over-segment hedges for all Ls, some of the true hedges are not recalled in the real-time scenario, although the released hedges are more precise. The mean of precision and recall, F1-score, indicates that the accuracy in the real-time parsing scenario is very close to that in the offline parsing for every L value.

Our general observations of incremental shift-reduce parsing for low-latency hedge segmentation are as the following:

(a) In contrast to non-incremental CYK parsing, incremental shift-reduce parsing could be more suitable for real-time processing such as input stream segmentation and particularly hedge segmentation.

(b) Releasing hedge segments at every stable state decreases hedge parsing accuracy significantly in spite of decreasing the latency compared to offline segmentation.

(c) Our proposed solution to improve accuracy is to augment the parser with a buffering procedure to buffer the stable constituents until they probably form a complete hedge. Shift-reduce parsing plus buffering achieves a slightly worse accuracy than offline segmentation accuracy for all L values in English and Chinese while latency is significantly lower in real-time scenario, particularly for smaller L values.

(d) A more conservative buffer releases more accurate hedges with slightly higher latency compared to a more aggressive buffer.

4.8 Summary

In this chapter, we proposed a method to simultaneously parse sentences and produce hedge segments. Our method could incrementally extract 100% stable hedges and send them to the processing application while the sentence is still being parsed. Similar ideas can be used to produce syntactic segments of other types (such as shallow bracketing) for incremental applications. The proposed approach also replaces pre-segmenting the input prior to hedge parsing, which causes cascading errors, as we saw in Chapter 3. We showed that our approach greatly decreases the latency of producing segments with a slight loss in accuracy for English and Chinese. In Chapter 5, we will combine some of the techniques in this chapter and Chapter 3 to incorporate hedge segmentation and annotation, in the input as well as the engine of the online task of simultaneous translation.

Chapter 5

Evaluation of Annotation and Segmentation Strategies in Machine Translation

5.1 Introduction

In Chapter 3, we introduced hedge parsing that parses the input sentence in non-overlapping segments each of which have rich internal syntactic analyses. Then in Chapter 4 we proposed an incremental framework to return hedge segments from the parser incrementally and with low latency, without requiring the entire sentence to begin the parse. These properties make hedge parsing potentially useful for real-time NLP applications such as simultaneous speech-to-speech translation. In this chapter, we investigate the impact of hedge annotation and segmentation in the on-line task of translation. We focus on the MT component of the speech-to-speech translation system, which receives text input and produces text output. We consider two main aspects of applying hedge syntax into MT: first, how such partial syntactic annotations, on either the target or the source side, affect a regular (non-incremental) machine translation task, and second, how such segmentation of the input affects accuracy/latency trade-off of a simultaneous (incremental) machine translation.

Input segmentation strategies in simultaneous translation are mainly studied in FST- or phrase-based translation systems that do not use syntactic information in either the translation model or the input. Bangalore et al. (2012) framed the speech-to-speech translation as first segmenting the input stream, and then applying the conventional MT approaches to translate the segments. They defined segment boundaries as the long pauses in the output of an automatic speech recognition (ASR) system. Recent studies have evaluated other segmentation strategies based on linguistic, non-linguistic, or joint-optimization criteria (Cettolo and Federico, 2006; Matusov et al., 2007; Fügen et al., 2007; Rangarajan Sridhar et al., 2013; Oda et al., 2014; Wolfel et al., 2008; Kolss et al., 2008), as described in detail in Chapter 2 Section 2.7.1. All of the above segmentation methods produce raw word sequences of the input which do not contain structural annotation of the source language. Likewise, the translation model does not contain such information. Instead, in this chapter, we leverage the syntactic knowledge of the source language to both segment the input, and improve the translation quality.

Applying syntactic information in MT can improve translation quality compared to phrase-based translation (Zollmann et al., 2008). However, incorporating syntax into the input segments to a simultaneous translation system is challenging. The reason is that traditional syntactic parsing is not directly applicable to sub-sentential segments, since parsing methods typically require the entire sentence. Current studies that address the use of syntax in simultaneous translation (Ryu et al., 2006; Oda et al., 2015) use methods to predict future syntactic constituents which form a complete syntactic phrase, when parsing sub-sentential segments. The standard CFG grammar is not particularly defined for incomplete sentences, so the parser may generate an incorrect parse. Moreover, applying this syntactic prediction in translation models is not trivial and requires addressing the problems such as reordering and language model probabilities. In contrast, the grammar we use in our approach is appropriate for sub-sentential segments. In addition, our approach is straightforwardly applicable to conventional MT systems.

Hedge parsing allows for syntax-based segmentation of the input, and incorporating local hierarchical syntax into the input segments without requiring the entire sentence. Partial hedge syntactic annotation could straightforwardly replace full syntactic annotation, either in the source or target language, in a conventional MT system. First, we compare the translation performance of a regular (non-incremental) translation model that incorporates hedge syntax with translation models that either do not use any syntactic information (so called phrase-based models), or incorporate non-linguistic syntax, or shallow linguistic syntax in the form of chunking, to the source or target side of a translation model. Second, we examine the translation performance and segmentation latency of an incremental translation system that incorporates hedge syntax in input segmentation and annotation as well as in the translation model. We compare this system with segmentation methods that produce raw segments and do not use syntactic information in the translation model, as well as segmentation and translation systems that use non-linguistic, or shallow syntactic information.

We demonstrate that:

1. Adding knowledge of local hierarchical syntactic structures within a local context to the *target* side of a translation model, significantly improves translation quality at the cost of some speed compared to the translation models without syntactic knowledge, with non-linguistic syntactic annotation, or non-hierarchical shallow syntactic annotation.

2. Hedge-syntactically informed MT on the *target* side, achieves a close quality compared to that in a full-syntactically informed MT for the language pairs we tried.

3. When incorporating syntactic knowledge to the *source* side of a translation model, MT performance is highly influenced by the richness level of syntactic annotations of the model and the input. Full syntactic information is considerably effective in translation quality, while hedge syntactic information falls behind that particularly for smaller span limits.

4. Incremental translation in a system that combines hedge segmentation and annotation of the input segments with a hedge-syntax MT model, achieves a good accuracy/latency trade-off compared to other segmentation strategies. More importantly, this system is significantly superior to the shallow-syntax segmentation and translation system, in terms of translation performance as well as latency of segmentation. This emphasizes the advantage of using some degree of syntactic hierarchy, as opposed to non-hierarchical shallow chunking structures, in real-time NLP applications that require fast analysis of the input.

5.2 Phrase-based and Syntax-based Machine Translation

Two major statistical MT formalisms developed in the literature are phrase-based and syntax-based models. In Section 2.6 of Chapter 2 we defined the fundamentals of these formalisms from a theoretical perspective. The main difference between the two is that translation rules in phrase-based MT are bilingual phrases extracted from a parallel corpus of the source and target languages, whereas a syntax-based MT uses some form of a synchronous context-free grammar (SCFG) to generate hierarchical mapping between the two languages. In this chapter, we use phrase-based and variations of syntax-based models in practice. We first show the differences of these MT models and how they are learned from a parallel training data, from a more practical perspective by providing examples.

5.2.1 Phrase-based MT

In a phrase-based system, translation rules are bilingual phrases extracted from raw (unannotated) parallel corpora. Possible phrases are extracted from word-aligned parallel sentences. The phrase table is built by accumulating all of the phrases from the entire corpus and calculating their probabilities using relative frequency. Figure 5.1(a) shows an example of the word alignment of two Japanese-English parallel sentences from the real data set we used. Figure 5.1(b) shows possible phrases consistent with that word alignment, as extracted by the MT toolkit that we used (Moses, Koehn et al., 2007). The

Figure 5.1: An example of (a) a word alignment of two parallel sentences and (b) phrase pairs consistent with this word alignment.

```
ヨーロッパ でも ,  同様の 動き が | Europe is in the same trends
が | is
,  | the
,  同様の | the same
,  同様の 動き | the same trends
同様の | same
同様の 動き | same trends
動き | trends
ある 。 | .
```

Figure 5.2: Phrase table.

phrase extraction procedure in the Moses toolkit follows alignment-based phrase extraction heuristics proposed by Och et al. (1999) (for more detail about this method, refer to Section 2.6.1 of Chapter 2). The extracted phrases in the Moses phrase table format are presented in Figure 5.2. Each entry consists of the source phrase and target phrase. For example, the Japanese phrase **同様の 動き**, which covers three consecutive words, is translated to the English phrase *the same trends* covering three consecutive words. Each entry also contains the alignment points, but for simplicity we have not shown them in the figure.

5.2.2 Syntax-based MT

Instead of phrases in a phrase-based MT model, a syntax-based model extracts SCFG rules. An SCFG can be automatically extracted from a parallel corpora, with or without syntactic annotations, on top of the output of a phrase-based model.

Hierarchical Phrase-based Model

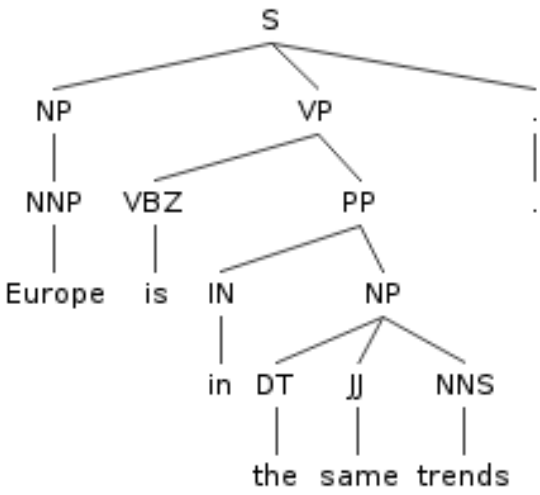
A variant of syntax-based MT, known as hierarchical phrase-based (Chiang, 2005, 2007), is a natural hierarchical extension to phrase-based models. A hierarchical phrase-based model extends the phrases by adding new rules in which the sub-phrases are replaced with a non-terminal X. There is no independent syntactic parsing on either side of the parallel data, instead, the hierarchical rules are inferred from the data, with no direct association to linguistic syntax. The rules are in the form of SCFG, with a single non-terminal X. Figure 5.3 shows the hierarchical phrase-based SCFG rules extracted from the phrase pairs in Figure 5.2. For example, the phrase '*the same trends*' contains the sub-phrase '*the same*', and replacing this sub-phrase with X creates the new rule 'X *trends*'. A simple phrase-based model is not able to represent such phrase hierarchy.

Tree-based Models

Another type of syntax-based models, called tree-based models, use linguistic syntactic annotations of the language in training data. Either the target side, source side, or both sides of the training data could contain syntactic annotations, and syntactic translation rules are inferred from aligned string-tree, tree-string, or tree-tree pairs respectively. Tree-based models constrain the rules in the hierarchical phrase based models to those that correspond to valid syntactic constituents in the parse trees. Figure 5.4 shows the parse tree of the English sentence in our example and how the hierarchical phrase-based rules are constrained to actual constituents instead of X. The sixth rule in the figure for instance, shows that the phrase '*the same trends*' corresponds to a noun phrase (NP). Here we see fewer phrases compared to the hierarchical phrase-based – for instance '*same trends*' which does not correspond to any linguistic structure is removed from the rule set – although the grammar is much richer. In Section 5.2.4 we explain corpus preparation and rule extraction in tree-based models in more detail.

```
が [X] | is [X] | 1
, [X] | the [X] | 1
同様の [X] | same [X] | 1
動き [X] | trends [X] | 1
ある 。 [X] | . [X] | 1
, 同様の [X] | the same [X] | 1
同様の 動き [X] | same trends [X] | 1
, 同様の 動き [X] | the same trends [X] | 0.33
[X][X] 動き [X] | [X][X] trends [X] | 0.33
, [X][X] [X] | the [X][X] [X] | 0.33
ヨーロッパ でも [X][X] 動き が [X] | Europe is in [X][X] trends [X] | 0.33
ヨーロッパ でも [X][X] が [X] | Europe is in [X][X] [X] | 0.33
ヨーロッパ でも , [X][X] が [X] | Europe is in the [X][X] [X] | 0.33
[X][X] ある 。 [X] | [X][X] . [X] | 0.5
ヨーロッパ でも [X][X] が [X][X] [X] | Europe is in [X][X] [X][X] [X] | 0.5
```

Figure 5.3: Hierarchical phrase-based SCFG rules.




```
が [X] | is [VBZ] | 1
, [X] | the [DT] | 1
同様の [X] | same [JJ] | 1
動き [X] | trends [NNS] | 1
ある 。 [X] | . [.] | 1
, 同様の 動き [X] | the same trends [NP] | 1
ヨーロッパ でも [X][NP] が [X][.] [X] | Europe is in [X][NP] [X][.] [S] | 0.5
ヨーロッパ でも [X][NP] が [X][.] [X] | Europe is in [X][NP] [X][.] [TOP] | 0.5
```

Figure 5.4: Basic syntax SCFG rules.

5.2.3 Word Re-ordering

Words with equivalent meanings do not always appear in the same order in both sentences. For example, verb '*is*' appears immediately following subject '*Europe*' in the English side of our example, but its equivalent word in Japanese 'か' comes after the object of the sentence. Therefore, some mechanism of word reordering is needed. In phrase-based models, phrases can capture some *local* reordering, limited to the phrases seen in the training data. Syntax-based models attempt to generalize reordering beyond the lexical knowledge represented in phrase-based model. Reordering is encoded in SCFG rules and *long-distance* reordering can be captured.

Zollmann et al. (2008) performed a systematic comparison of phrase-based and syntax-based (hierarchical and string-to-tree) MT. They isolated the impact of several important design decisions including training data size, language model size, and reordering methods on translation quality. Their experiments show that syntax-based approaches can result in significant improvement compared to phrase-based model for language pairs that are adequately non-monotonic. In particular, they observe consistent improvements for language pairs with long-distance reordering such as Urdu-to-English and Chinese-to-English.

5.2.4 Training Tree-based Models

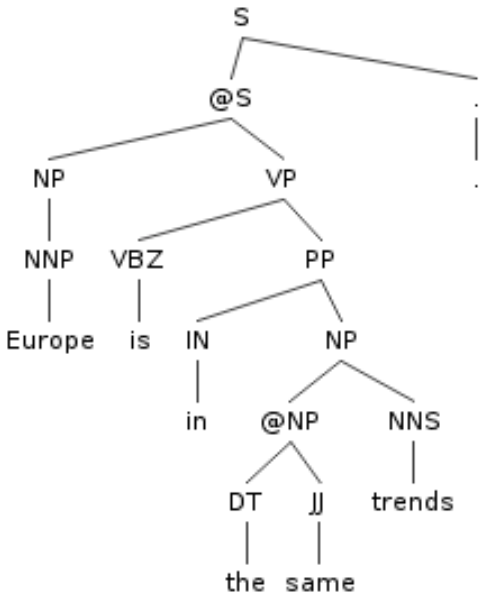
Syntactic annotation in a tree-based model is provided by parsing the training data, either on the source, target, or both sides, with a syntactic parser. Typically the English side, for which there are high-quality syntactic parsers, is parsed and then the syntax is mirrored onto the other side (Lopez, 2008). After parsing is done, basic SCFG rules for tree-based models are automatically extracted from the annotated parallel text using the MT toolkit we describe in Section 5.4.2. In the next subsection, we describe two types of SCFG rule extraction methods supported by this MT toolkit that we used in our experiments.

Relaxing Parses

Each non-terminal in a basic SCFG has to correspond to an actual non-terminal in the parse tree. If the target side of a phrase in a string-to-tree model (or the source side in a tree-to-string model) matches a constituent span in the parse tree, then the constituent's label is assigned as the left-hand-side non-terminal of the SCFG rule, otherwise no rule is extracted. This will severely constrain the number of rules that can be extracted. There are a number of methods to relax this constraint and increase the number of extracted rules. In this chapter, we use two kinds of parse relaxation supported by the MT toolkit we use: left-factoring the parse trees; and using heuristics to combine neighboring nodes into tags as described in syntax-augmented machine translation (SAMT) by Zollmann and Venugopal (2006).

The idea of left-factoring is to add more internal non-terminals to the parse trees so that additional rules can be extracted. Figure 5.5 shows the result of left-factoring our example parse tree in Figure 5.4. After the transform, two new non-terminals "@S" and "@NP" are added, which yield to adding six new translation rules, and modifying the probabilities of some of the existing rules. These added nodes define new rules for their children nodes, and are also combined with their siblings or parents to create new rules. This idea is similar to left- or right- factoring binarization of a grammar to use it in a CYK parser (see Section 2.1.3 of Chapter 2).

Figure 5.6 shows the SAMT transform of the example parse tree in Figure 5.4. In addition to the word spans of the sentence that correspond to a POS tag, unary, or n-ary non-terminals, all other pairs of neighboring nodes are labeled with composite tags, and they can contribute in creating new translation rules. If nodes are siblings, they are combined using the "+" operation. For example "IN" which covers span 2 is combined with its sibling "NP" which covers spans 3 to 5 and create a new composite tag "IN+NP". If the nodes are not siblings but they are still neighbors, then the "++" operation is used. An instance is the "VBZ" covering span 1 and "IN" covering span 2 and their combination "VBZ++IN". If applicable, the spans are combined using the division operation in categorial grammar.




```
が [X] | is [VBZ] | 1
, [X] | the [DT] | 1
同様の [X] | same [JJ] | 1
動き [X] | trends [NNS] | 1
ある 。 [X] | . [.] | 1
, 同様の [X] | the same [@NP] | 1
, 同様の 動き [X] | the same trends [NP] | 0.5
[X][@NP] 動き [X] | [X][@NP] trends [NP] | 0.5
ヨーロッパ でも [X][@NP] 動き が [X] | Europe is in [X][@NP] trends [@S] | 0.5
ヨーロッパ でも [X][NP] が [X] | Europe is in [X][NP] [@S] | 0.5
[X][@S] ある 。 [X] | [X][@S] . [S] | 0.25
[X][@S] ある 。 [X] | [X][@S] . [TOP] | 0.25
ヨーロッパ でも [X][NP] が [X][.] [X] | Europe is in [X][NP] [X][.] [S] | 0.25
ヨーロッパ でも [X][NP] が [X][.] [X] | Europe is in [X][NP] [X][.] [TOP] | 0.25
```

Figure 5.5: Left-factoring syntax SCFG rules.

For example, “NP/NNS” for span 3 to 4 indicates a partial NP missing an NNS to the right.

```
      span:3-4 label="DT+JJ", "NP/NNS"      span:0-1 label="NNP++VBZ"
      span:4-5 label="JJ+NNS", "NP\DT"      span:1-2 label="VBZ++IN"
      span:2-5 label="IN+NP"                span:2-3 label="IN++DT"
      span"1-5 label="VBZ+PP"               span:5-6 label="NNS++."
      span:0-5 label="NNP+VP", "S/."        span:2-4 label="PP//NNS"
      span:1-6 label="VP+.", "S\NNP"        span:1-4 label="VP//NNS"
                                            span:3-6 label="NP++."
                                            span:2-6 label="PP++."
が [X] | is [VBZ] | 1
, [X] | the [DT] | 1
同様の [X] | same [JJ] | 1
動き [X] | trends [NNS] | 1
ある 。 [X] | . [.] | 1
, 同様の [X] | the same [DT+JJ] | 1
同様の 動き [X] | same trends [JJ+NNS] | 1
, 同様の 動き [X] | the same trends [NP] | 0.2
[X][DT+JJ] 動き [X] | [X][DT+JJ] trends [NP] | 0.2
[X][NP/NNS] 動き [X] | [X][NP/NNS] trends [NP] | 0.2
, [X][JJ+NNS] [X] | the [X][JJ+NNS] [NP] | 0.2
, [X][NP\DT] [X] | the [X][NP\DT] [NP] | 0.2
ヨーロッパ でも [X][DT+JJ] 動き が [X] | Europe is in [X][DT+JJ] trends [NNP+VP] | 0.1
ヨーロッパ でも [X][DT+JJ] 動き が [X] | Europe is in [X][DT+JJ] trends [S/.] | 0.1
ヨーロッパ でも [X][NP/NNS] 動き が [X] | Europe is in [X][NP/NNS] trends [NNP+VP] | 0.
ヨーロッパ でも [X][NP/NNS] 動き が [X] | Europe is in [X][NP/NNS] trends [S/.] | 0.1
ヨーロッパ でも [X][NP] が [X] | Europe is in [X][NP] [NNP+VP] | 0.1
ヨーロッパ でも [X][NP] が [X] | Europe is in [X][NP] [S/.] | 0.1
ヨーロッパ でも , [X][JJ+NNS] が [X] | Europe is in the [X][JJ+NNS] [NNP+VP] | 0.1
ヨーロッパ でも , [X][JJ+NNS] が [X] | Europe is in the [X][JJ+NNS] [S/.] | 0.1
ヨーロッパ でも , [X][NP\DT] が [X] | Europe is in the [X][NP\DT] [NNP+VP] | 0.1
ヨーロッパ でも , [X][NP\DT] が [X] | Europe is in the [X][NP\DT] [S/.] | 0.1
[X][NNP+VP] ある 。 [X] | [X][NNP+VP] . [S] | 0.166667
[X][NNP+VP] ある 。 [X] | [X][NNP+VP] . [TOP] | 0.166667
[X][S/.] ある 。 [X] | [X][S/.] . [S] | 0.166667
[X][S/.] ある 。 [X] | [X][S/.] . [TOP] | 0.166667
ヨーロッパ でも [X][NP] が [X][.] [X] | Europe is in [X][NP] [X][.] [S] | 0.166667
ヨーロッパ でも [X][NP] が [X][.] [X] | Europe is in [X][NP] [X][.] [TOP] | 0.166667
```

Figure 5.6: SAMT syntax SCFG rules.

5.3 Methods

In this section, we first explain the MT models we used, and then the types of the inputs to the models in terms of segmentation and annotation, and finally we describe what combinations of the models and inputs we applied in different scenarios to measure the impact of segmentation and annotation in regular and simultaneous MT performance.

5.3.1 MT Models

Our baseline translation model is a phrase-based model which we refer to as PBMT. PBMT is trained from the raw parallel data without any syntactic information, hence it supposedly provides a floor on translation accuracy. We build syntax-based models that incorporate different levels of syntax into MT. We try a hierarchical phrase-based model (HIERO) with no linguistic syntax, and tree-based models with full-syntax, shallow-syntax, and hedge-syntax annotations, forming string-to-tree models S2T-full, S2T-hdge, and S2T-shal, and tree-to-string models T2S-full, T2S-hdge, and T2S-shal.

For tree-based models we annotate one side of the training data with syntactic labels, which is the English side in all of our experiments. We use two state-of-the-art phrase-structure parsers: Berkeley parser (Petrov and Klein, 2007a), which is a variant of non-incremental chart parsing, and the ZPAR incremental shift-reduce parser (Zhang and Clark, 2011). ZPAR is also integrated with the real-time hedge segmentation framework introduced in Chapter 4, to improve latency in simultaneous translation. Both parsers perform full parsing. For hedge-syntax models, we hedge transform the full parse results, as described in Section 3.4 of Chapter 3. For shallow-syntax models we transform the full parse results into a sequence of shallow constituents via a well-known conversion script known as *chunklink*[1] used originally in the CoNLL-2000 chunking task (Sang and Buchholz, 2000).

5.3.2 Inputs

We study inputs from two points of view: annotation type, and segmentation type. Annotation type divides the inputs into two types of raw (un-annotated) or syntactically annotated inputs. From the point of view of segmentation, inputs are either complete non-segmented sentences or segments of a complete sentence.

[1] http://ilk.uvt.nl/team/sabine/homepage/software.html

Annotation Types

Annotation type of the input depends on the translation model. The input to the MT system in a phrase-based, hierarchical phrase-based, or string-to-tree syntax-based model is a conventional unannotated string. The input to a tree-to-string model is a parse tree, not a string, since the MT model uses source-side parses to drive the translation process. The input sentence can be parsed using the same parser that annotated the training data, and it typically has the same syntactic richness level as that of the training data. To the best of our knowledge, existing work on tree-based MT has mainly focused on *full* parsing of the training data. In this work, we propose and evaluate the impact of *partial* parsing of the training data on the performance of an MT system. Such partial parsing could be produced by a parser which uses a grammar trained on a transformed treebank, or they could be the transformed output of fully parsed data. In our methods, we first fully parse the input sentence and then transform the result to partial (shallow or hedge) parse trees. This is the same procedure we followed for annotating the training data.

Segmentation Types

In regular –non-incremental– translation, the input is a complete sentence, whereas in incremental translation, the input is in the form of segments, one at a time. The translated segments are then combined to form the translation of the original –prior to segmentation– sentence. The number of segments, their lengths, and their types are identified according to the segmentation methods. In this work, we apply four segmentation methods: (a) according to length in number of tokens, (b) according to the punctuation, (c) shallow parsing boundaries, and (d) hedge parsing boundaries. Methods (a) and (b) use orthographic features and can be applied on raw inputs, whereas methods (c) and (d) rely on syntactic features which come from the parse tree of the input. To segment based on methods (c) and (d), the input is fully parsed and then the shallow or hedge parses are extracted. In addition, for method (d) we could extract the hedge parses simultaneously as we fully parse the input, using the techniques we proposed in Chapter 4.

5.3.3 Combining Inputs and MT Models

In this study, the two aspects under discussion are, first, the impact of syntactic annotation on regular MT performance, detached from input segmentation, and second, the impact of input segmentation on incremental MT performance. To investigate these two aspects, we define three experimental scenarios to examine (1) impact of *target* syntactic annotation on translation, (2) impact of *source* syntactic annotation on translation, and (3) impact of input segmentation on simultaneous translation. Scenarios (1) and (2) address the first aspect under discussion and (3) centers on the second aspect. For each scenario we integrate phrase-based and syntax-based MT models with complete/segmented raw/syntactically-annotated input types in different ways.

For (1) we apply PBMT, HIERO, and all variants of string-to-tree models (S2T-full, S2T-hdge, and S2T-shal) on complete raw input sentences. String-to-tree models are widely used in standard syntax-based MT and they can result in translation quality improvement compared to phrase-based model for language pairs with different word orders, such as Urdu-English and Japanese-English (Zollmann et al., 2008). In Section 5.5.1 we evaluate this configuration on accuracy and efficiency of Urdu to English and Japanese to English translation.

For the scenario (2) we apply PBMT, HIERO, and all variants of tree-to-string models (T2S-full, T2S-hdge, and T2S-shal) on complete input sentences which are raw for PBMT and HIERO, but syntactically annotated for T2S models. T2S models are combined with inputs that have the same syntactic richness of the model, as well as inputs with unmatched richnesses with the model. For example, T2S-hdge model is combined with hedge-syntax annotated inputs as well as shallow- and full-syntax annotated inputs. This aims to examine the effect of matched input and MT model versus unmatched conditions on translation performance. In Section 5.5.2 we demonstrate MT accuracy and efficiency of this configuration on English to Japanese translation. T2S models are not as widely used as S2T models in the literature, however, they are convenient for syntax-based input segmentation, which will be the focus of the scenario (3).

After exploring phrase-based, hierarchical phrase-based, and source-syntax MT models on complete (non-segmented) inputs in the scenario (2), in (3) we examine the MT accuracy and efficiency of the same models on *segmented* inputs. The pattern of combining MT models and inputs is the same as the previous scenario, i.e., the models receive either their matched or unmatched inputs in term of syntactic richness. In particular, we are interested in evaluating MT performance gain using syntax-based segmentation (methods (c) and (d) above) with orthographic segmentation (methods (a) and (b)). In addition, we explore real-time segmentation for method (d) to examine its effect on improving simultaneous translation latency. In Section 5.5.3 we compare and contrast translation accuracy, efficiency, and latency for different input types for English to Japanese translation.

5.3.4 MT Performance Evaluation Measures

We evaluate our systems based on the performance of machine translation. The three metrics we measure are *accuracy*, *efficiency*, and *latency*.

Accuracy

Accuracy is the quality of translation which is measured using BLEU scores. BLEU intends to approximate human judgment of a machine-translated output. It works at the corpus level: scores of individual sentences are calculated by comparing them against one or more reference translations, and then these scores are averaged over the whole corpus. In case of segmented inputs, all the translated segments of a sentence are concatenated to form the complete translation before measuring the BLEU score. We conducted case-insensitive BLEU in our experiments. For more detail about BLEU refer to Section 2.6.3 of Chapter 2.

Efficiency

Efficiency is the speed of translation and shows the average time taken to translate each sentence. In case of non-segmented inputs, efficiency is measured as seconds per sentence

(sec/sen), which is the total time taken to translate a corpus divided by number of its sentences. Sentence translation time includes the time required to read a sentence, initialize the search space, decode the best translation, and deliver the result to the output. In case of segmented inputs, efficiency for each sentence is calculated by summing up the time taken to translate all its segments. The speed reported for the entire corpus is then calculated by taking the average of efficiencies of all sentences in the corpus.

Latency

Latency is an important measure in real-time translation. Generally, assuming the sentence to be the unit of translation, latency is the delay between the point that a sentence is available and the point its translation is delivered. In this chapter we measure latency for variable-length sub-sentential units. Similar to Chapter 4, here we use a word-based latency measure: seconds per word (sec/w), which is the average delay for translating each word of the source text after the word is received. Latency for each word is the time (or token) difference between the point that the word is received and the point that it is translated.

In a full-fledged speech-to-speech translation pipeline, latency includes the delay of ASR, MT, and TTS components. In the offline translation, which the system waits until the end of sentence to start translation, latency is the aggregated time of these three components, however, in the real-time scenario, the latency decreases because ASR, segmentation, MT, and TTS modules start processing sub-sentential segments before the entire sentence is received. Note that in this case, a segmentation component is added to the pipeline, thus its performance could greatly influence accuracy/latency trade-off of real-time translation. In this chapter, we take apart ASR and TTS components from the pipeline, and evaluate translation latency focusing on segmentation and MT components. A full pipeline latency evaluation could be a future research direction.

5.4 Experimental Setup

5.4.1 Data

We run our experiments on Urdu-English and Japanese-English language pairs. Sentences in Urdu and Japanese have subject-object-verb (SOV) word order pattern, as opposed to subject-verb-object (SVO) in English. Thus translating between these languages to/from English has the potential to achieve higher accuracy for syntax-based MT against phrase-based MT, compared to translating between English and languages with similar word order such as European languages.

For Urdu we use manually constructed parallel corpora built by Post et al. (2012) for machine translation between English and six Indian languages. There are four different English translations for each source Urdu sentence of the corpus. To correctly pair source and target sentences in training data, each source sentence is repeated four times. With these repetitions, there are 33k sentence pairs (1198k words) for training. Translation evaluation of the development and test sets, is performed with respect to four translations as alternate references. Each reference for the development set has 736 sentence pairs, and the four references have 67k words altogether. These number are 605 sentence pairs and 42k words for the test set.

For Japanese, we use ASPEC[2], Asian Scientific Paper Excerpt Corpus, provided by the 2nd Workshop on Asian Translation (WAT, Nakazawa et al., 2015). It consists of a Japanese-English scientific paper abstracts corpus of 3M parallel sentence pairs (78M words) for training, 1790 sentence pairs (44k words) for development, and 1812 sentence pairs (44k words) for test. Following the guidelines provided by WAT for a robust baseline system, we used the first part out of the three parts of the training data (1M sentences) for training the translation models, and all of the three parts for training the language models. For comparability with the Urdu translation task, we also consider a subset of the Japanese-English data of the same size of the Urdu-English data, randomly selected

[2]http://lotus.kuee.kyoto-u.ac.jp/ASPEC

from the large corpus, which we refer to as *small Japanese data set*. In spite of their equal number of sentences, these corpora differ significantly in average length of sentences (22.1 words in small Japanese data set as opposed to 14.4 words in Urdu). This will result in larger parse trees in Japanese with higher number of constituents and we see about 40% higher number of original constituents in Japanese training data compared to Urdu training data. Therefore, a lower percentage of the original constituents are retained after hedge transform in Japanese (79%) as opposed to Urdu (86%).

To increase the number of rules that can be extracted in tree-based models, we apply parse relaxing methods we described in Section 5.2.4. For the large data set of Japanese-English, we left-binarize the parse trees before extracting the rules in tree-based models. For the small data sets of Japanese-English and Urdu-English, we transform the parse trees to SAMT format using the *relax-parse* script provided in the MT toolkit.

5.4.2 Data Preparation and MT Toolkit

We prepare data for MT training by tokenizing and lowercasing the sentences, and then removing parallel sentences with length more than 50 tokens. For tree-based models, sentences are tokenized in Penn Treebank tokenization style. To parse the data for tree-based models, the Berkeley parser is trained on the WSJ corpus using Berkeley latent variable grammar with 6 split-merge cycles (Petrov and Klein, 2007b), and ZPAR is trained on the same data with 15 training iterations and beam size of 16.

We use the Moses SMT toolkit (Koehn et al., 2007) to train the models and decode the test sets. Word alignment is performed using multi-threaded GIZA++, MGIZA (Gao and Vogel, 2008), with "grow-diag-final-and" method. The language model is built from the target side training data using the KenLM tool (Heafield, 2011) with order of 5 and modified Kneser-Ney smoothing method. For remaining settings we used the Moses defaults. This includes the default reordering model of word-based extraction ("wbe"), which gives a cost linear to the reordering distance. To find the optimal weights for the linear model of translation we used minimum error rate training (MERT) tuning algorithm (Och, 2003).

For the large Japanese-English data, the phrase and rule tables become too large to fit into memory, so we build binary phrase tables with on-demand loading using the "CreateOnDiskPt" command in the Moses toolkit. For decoding a sentence, only the part of the phrase or rule table that is required to translate the sentence is loaded into memory.

5.5 Results

5.5.1 Impact of Target Syntactic Annotation on Translation

We first evaluate the overall performance of MT in Urdu to English and Japanese to English translation when applying target syntax to the translation model. The inputs are complete non-segmented sentences, since we intend to isolate the impact of syntax from the impact of input granularity.

Translation Accuracy

Figures 5.7 and 5.8 show the impact of augmenting hedge-syntax information (S2T-hdge) with various maximum span parameter L into machine translation accuracy compared to S2T-full model, for Urdu and Japanese development sets, respectively. Note that we used the small Japanese data set to make the evaluation comparable with Urdu. The S2T-hdge model slightly outperforms S2T-full model for $L \geq 8$ in Urdu and $L \geq 9$ in Japanese. Slight improvement of S2T-hdge over S2T-full in larger Ls suggests that, in our data set, augmenting full hierarchical syntactic annotation into translation may not be more helpful than augmenting such annotation for a limited span.

Table 5.1 shows the translation accuracy of phrase-based model and syntax-based models at different levels of target syntactic annotation for translating from (a) Urdu and (b) Japanese to English for development data. For hedge-syntax models, we choose $L=8$, as the optimal comparable operating point for translation accuracy and original constituents preservation in both languages. First, we observe better performance of all syntax-based models compared to PBMT for both language pairs. Second, within the syntax-based

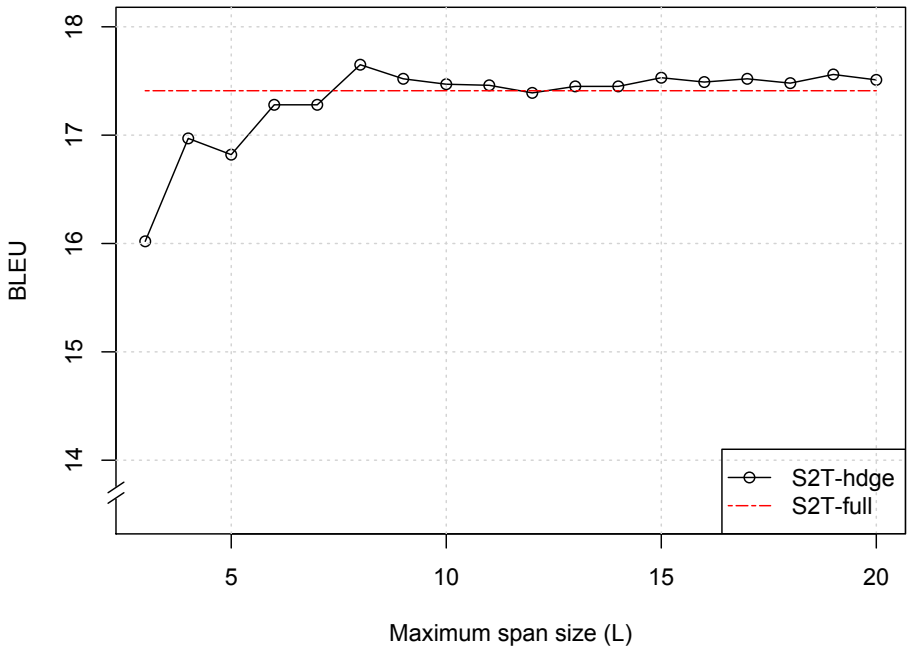


Figure 5.7: Urdu to English syntax-based translation accuracy versus L on development data.

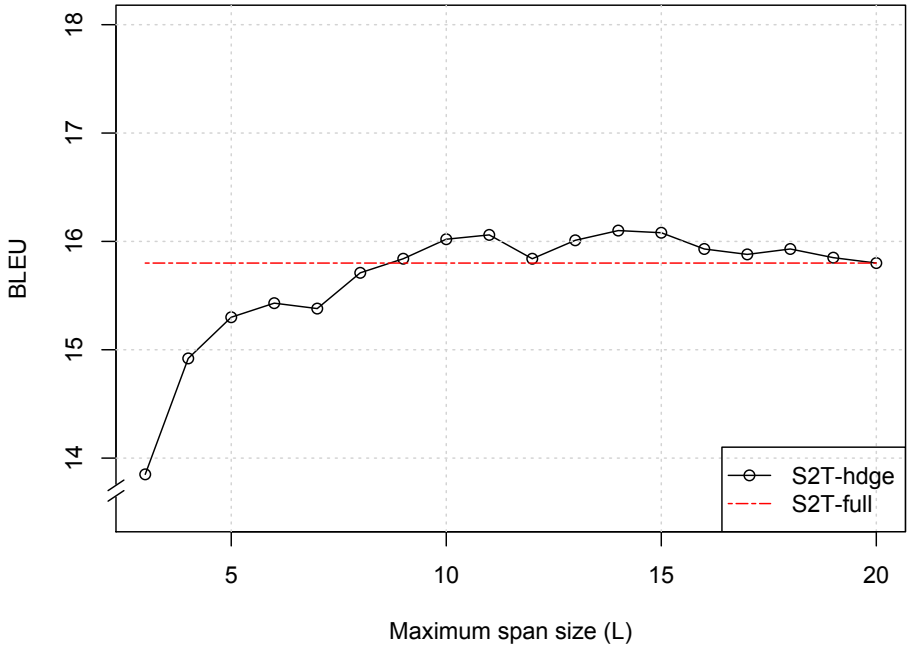


Figure 5.8: Japanese to English (small data set) syntax-based translation accuracy versus L on development data.

Translation Model	Accuracy (BLEU)
PBMT	14.01
HIERO	16.18
S2T-shal	15.15
S2T-hdge	17.65
S2T-full	17.41

(a)

Translation Model	Accuracy (BLEU)
PBMT	11.33
HIERO	13.68
S2T-shal	12.91
S2T-hdge	15.71
S2T-full	15.80

(b)

Table 5.1: Translation accuracy on development data for (a) Urdu to English, and (b) Japanese to English (small data set).

models we observe better performance of tree-based models compared to HIERO except for S2T-shal. This observation is consistent with our expectation that using some degree of syntactic hierarchy is beneficial in MT as opposed to non-hierarchical shallow structures. The differences between S2T-hdge and S2T-full systems in this table are not statistically significant for both Urdu to English ($p\text{-}value$=0.2188) and Japanese to English ($p\text{-}value$=0.8515) translations. To estimate the significant difference, we used the stratified approximate randomization test provided in MultEval (Clark et al., 2011).

Translation Accuracy/Efficiency Trade-off with Beam Setting

One of the most important considerations in decoding is accuracy/efficiency trade-off. SCFG decoding has several parameters that impact this trade-off. In this section we examine the effect of beam setting in the CYK algorithm implemented in the Moses chart decoder. The beam setting parameter in the Moses toolkit is called cube-pruning-pop-limit (or *cbp*), that restricts the number of hypotheses generated for each cell. Higher *cbp* numbers slow down the decoder, but may result in better accuracy. Other settings in the Moses toolkit that effect decoding speed are reordering limit, handling of unknown words, and some additional technical settings.

Figures 5.9 and 5.10 show accuracy, in terms of BLEU, and decoding speed, in terms of seconds per sentence (sec/sen), for syntax-based translations using a variety of beam settings, on the development sets. We investigate exponentially increasing *cbp* values of $\{10, 20, 30, 45, 70, 100, 150, 230, 400\}$. The speed is measured as the total time in seconds taken to translate all sentences, divided by the total number of sentences. Same as the

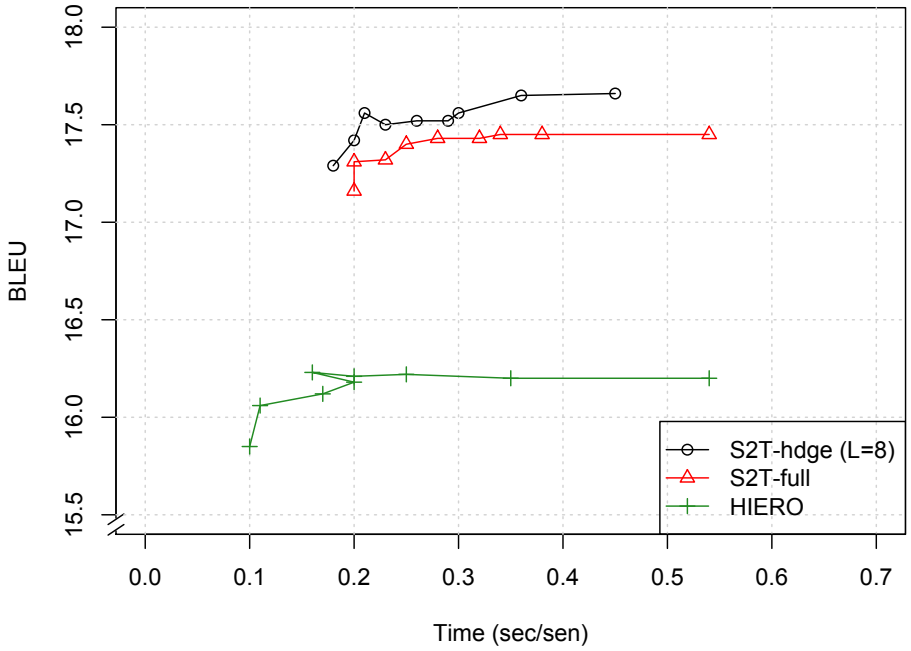


Figure 5.9: Urdu to English syntax-based translation accuracy versus speed on development data.

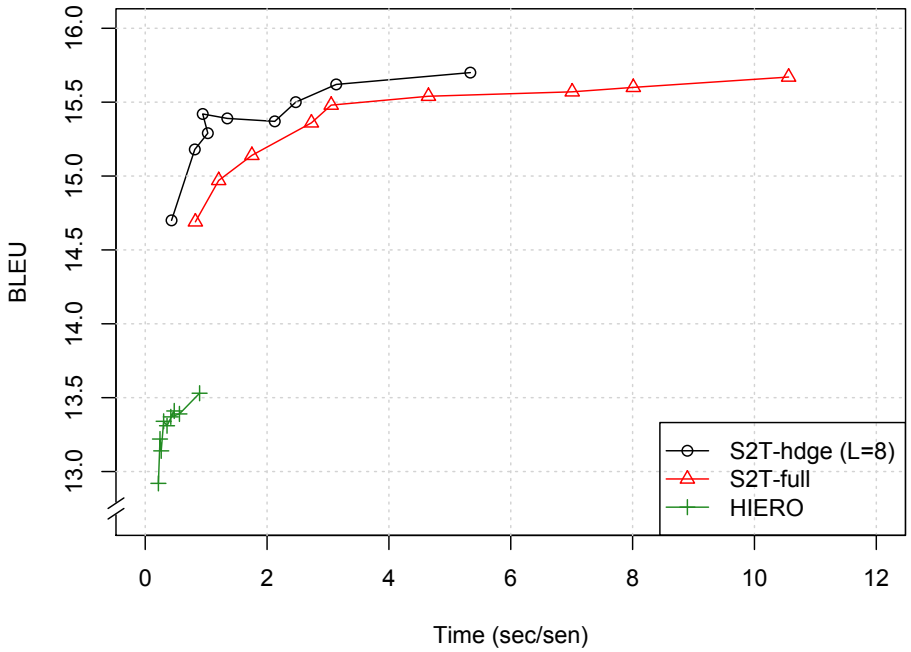


Figure 5.10: Japanese to English (small data set) syntax-based translation accuracy versus speed on development data.

above, we choose L=8 for hedge-syntax models. The results show that in all models, for polynomial increment of *cbp* by 1.5×, the BLEU score increases logarithmically such that for *cbp* values greater than 230, BLEU is almost fixed. As expected, speed decreases when *cbp* increases, with a few exceptions for small *cbp*s. S2T-hdge consistently outperforms S2T-full in terms of translation accuracy and efficiency, in all *cbp* values for both languages. Compared to HIERO, in both languages, tree-based models provide higher translation accuracy in all *cbp*s at the expense of lower efficiency. This difference is noticeably significant in Japanese, possibly due to the larger parse trees with higher number of constituents in Japanese (see Section 5.4.1). As we have seen in Section 5.2.2, HIERO utilizes hierarchical phrases inferred from unannotated data, with no direct association to syntactic information from parse-trees.

Test Set Results

Table 5.2 presents results of our best configurations—as determined by results on the development set—on the *test* set, choosing the optimal operating points for syntax-based models: *cbp*=230, L=8 for S2T-hdge. We observe the same pattern in *accuracy* as we saw in the development set of both languages (see Table 5.1). In terms of efficiency the general pattern is that efficiency has a trade-off with accuracy; the more efficient model used the lower accuracy achieved. The only observed exception is that we gain the best accuracy and efficiency at the same time using S2T-hdge in Urdu. Post et al. (2012) reported the accuracy of 19.53 and 20.99 BLEU scores for HIERO and SAMT test set baseline translations. We observe around 2.8 BLEU score improvement in each model compared to this baseline.

We performed the same evaluation on the *large* Japanese-English data, including all training and test sets (see Section 5.4.1). Results are presented in Table 5.3 for the default *cbp*=1000. We see the same pattern as we observed in the small development set and the small test set in terms of accuracy (Tables 5.1 and 5.2-b), and efficiency (Table 5.2-b). However, as expected, we see overall higher BLEU and lower efficiency, due to using larger phrase/rule tables extracted from more training data. Table 5.3 also demonstrates that

Translation Model	Accuracy (BLEU)	Efficiency (sec/sen)
PBMT	19.28	0.23
HIERO	22.46	0.23
S2T-hdge	24.00	0.22
S2T-full	23.73	0.24

(a)

Translation Model	Accuracy (BLEU)	Efficiency (sec/sen)
PBMT	13.16	0.52
HIERO	14.61	0.77
S2T-hdge	15.98	2.64
S2T-full	16.72	5.56

(b)

Table 5.2: Accuracy and efficiency of translating (a) Urdu to English and (b) Japanese to English (small data set) test sets using phrase-based and syntax-based models.

translation performance is only slightly different when using the incremental shift-reduce ZPAR parser instead of Berkeley parser to annotate our data. Nakazawa et al. (2015) reported the accuracy of 18.45, 18.72, and 20.36 case-sensitive BLEU scores for PBMT, HIERO and S2T test set baseline translations. Their PBMT uses a superior reordering model to the Moses default reordering model we used. Under the same condition of reordering model for PBMT and case-insensitive evaluation, we observe BLEU score improvements in our models, with about the same pattern in differences.

The overall directions we find in incorporating syntactic annotation on target language for Urdu to English and Japanese to English translation are as follows:

(a) Syntax-based models outperform phrase-based model in translation accuracy at the cost of translation efficiency.

(b) Within the syntax-based models, those who use linguistically informed syntactic annotation at a higher level than shallow chunking, i.e. full or hedge parsing, outperform

Translation Model	Accuracy (BLEU)	Efficiency (sec/sen)
PBMT	18.22	1.02
HIERO	20.35	4.85
S2T-hdge (Berkeley)	20.98	7.77
S2T-full (Berkeley)	21.25	10.15
S2T-hdge (ZPAR)	20.98	7.62
S2T-full (ZPAR)	21.23	11.84

Table 5.3: Accuracy and efficiency of translating Japanese to English (large data set) test sets using phrase-based and syntax-based models.

non-linguistically informed hierarchical phrase-based model in terms of translation accuracy at the expense of translation efficiency.
(c) For both languages we could achieve an optimal operating point at maximum span limit L, where target-side hedge-syntax MT performs just as accurate or even slightly better than full-syntax MT.

5.5.2 Impact of Source Syntactic Annotation on Translation

So far we have evaluated the impact of syntactic annotation of the target side of an MT model on translation performance in Urdu to English and Japanese to English. In this section we explore the impact of incorporating syntactic annotation of the *source* side on translation accuracy and efficiency and the trade-off between the two in different syntactic richness levels. Same as Section 5.5.1, the inputs are complete non-segmented sentences, since we still focus on separating the impact of syntax from the impact of input granularity. Note that inputs are raw, unannotated senesces for PBMT and HIERO, but they are syntactically annotated for tree-based T2S-shal, T2S-hdge, and T2S-full models. In the following section (Section 5.5.3), we will use the same MT models on *segmented* inputs to evaluate the effect of input segmentation on simultaneous translation.

In the current section as well as Section 5.5.3, we consider English to Japanese translation and we do not report Urdu translation. The Urdu-English data set is originally collected for translation *into* English. For each Urdu document, four reference translations are collected. Generally by increasing the number of references BLEU is improved, particularly for short sentence in the Urdu-English data set, since the opportunities to match the reference increases. If we change the direction of translation, we should choose one of the references as the source sentence and there will be only one target sentence as the reference for the source, and as a result, BLEU significantly decreases. Therefore we disregard providing the results on Urdu, although the ideas are general and applicable to arbitrary language pairs.

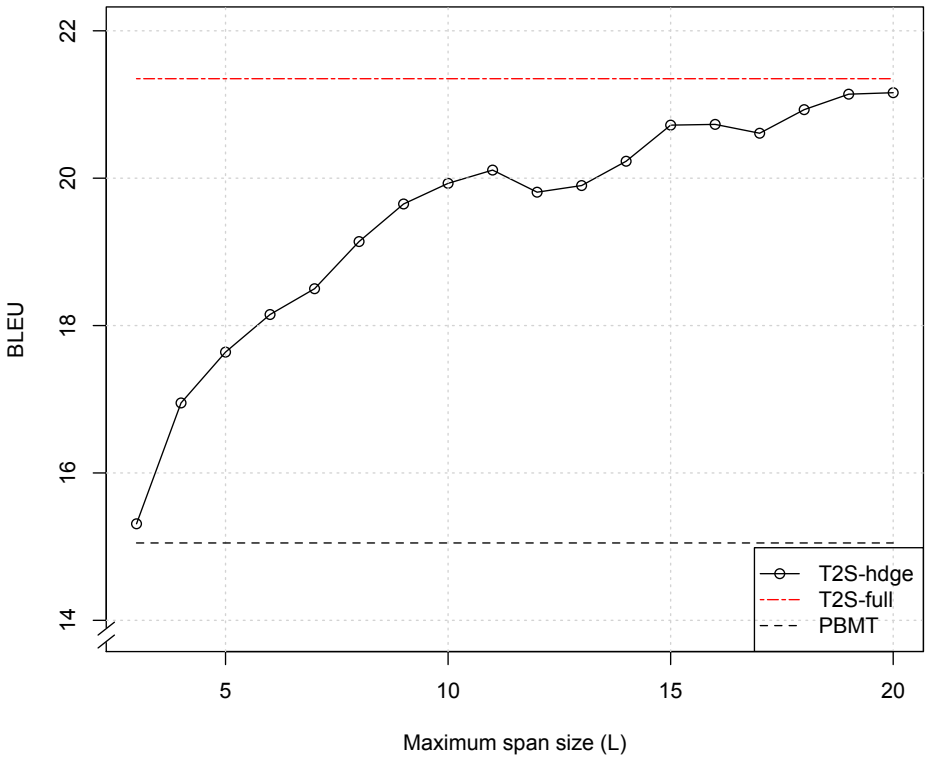


Figure 5.11: English to Japanese (small data set) syntax-based translation accuracy versus L on development data.

Translation Accuracy

Figure 5.11 demonstrates the effect of incorporating source-side hedge-syntax information (T2S-hdge) at different maximum span parameter L on MT accuracy against full-syntax model and phrase-bases model, on the development set. As opposed to target side syntactic annotation, we observe better performance of T2S-full compared to T2S-hdge, regardless of L value. Although as we increase L value, T2S-hdge accuracy becomes closer to T2S-full. T2S-hdge consistently outperforms the PBMT baseline.

Table 5.4 reveals MT accuracy of PBMT and syntax-based models at various degrees of source syntactic annotation, on development set. For consistent comparison with presented results in previous section, we choose maximum span limit L=8. Here we notice significant impact of syntactic annotation richness on translation accuracy. This is due to the fact that

Table 5.4: Translation accuracy on development data for English to Japanese (small data).

Translation Model	Accuracy (BLEU)
PBMT	15.05
HIERO	19.45
T2S-shal	14.58
T2S-hdge	19.14
T2S-full	21.35

T2S models rely remarkably on source side annotation to derive translation. Lower levels of syntactic richness may actually hurt the translation accuracy as we see that T2S-shal falls behind HIERO and even PBMT, and T2S-hdge falls behind HIERO. We achieve the best performance by incorporating full parse trees into MT model and input. We performed an analysis on the kind of errors often occur in a T2S-hdge model compared to a T2S-full model. We found that translation accuracy is noticeably lower in a T2S-hdge system when translating compound/complex sentences. These sentences include coordinating sentences (connected with coordinating conjunctions such as ‘and and ‘but), or subordinate clauses (connected with subordinating conjunctions such as ‘which, ‘that, and ‘while). The reason is that the annotations distinguishing the coordinating or embedded sentences (e.g., non-terminals S or SBAR) are removed from the input due to the hedge transform. Thus the whole input appears as a single sentence to the translation model, although they are actually separate (embedded) sentences.

Translation Acc/Eff Trade-off with Syntactic Richness of Input Annotation

As explained before, a critical aspect of decoding is the trade-off between translation accuracy/efficiency. In Section 5.5.1 we explored this trade-off at different beam settings. Tree-to-string models are huge and the system runs out of memory during decoding. Therefore, we binarize the rule table to help on-demand memory usage so that only the part of the rule table that is required to translate a sentence is loaded into memory. We noticed that beam setting does not impact efficiency of translation using the binarized models, i.e., efficiencies are similar when beam changes. Therefore, instead of changing

beam size to examine translation accuracy/efficiency, in this section, we modified syntactic richness of input annotation to various tree-to-string models.

Table 5.5 presents translation performance of various MT models for English to Japanese translation of the development set in large data set. To study the impact of syntactic richness of input annotation in tree-based models, we tried each model with inputs with matched or unmatched annotation levels. For example, the hedge-syntax model is combined with its matched hedge-annotated input as well as unmatched shallow- and full-annotated inputs. Note that in all conditions, inputs are entire sentences. Also the results for parsing the training and input data with Berkeley and ZPAR parsers are provided.

Regarding accuracy, the best performance in tree-based systems is when a translation model is combined with its matched input: T2S-shal model has by far higher BLEU when it receives shallow-annotated inputs as opposed to hedge- or full-annotated inputs; Likewise, T2S-full model has its best accuracy if it receives fully syntactically annotated inputs. However, a T2S-hdge model performs best with a fully syntactic input, about one BLEU score higher than combining this model with hedge-syntactic input. We observe the same trend using both parsers, i.e., relative accuracy changes in comparing different model/input combinations is independent of the parser type. The impact of model/input combination in translation efficiency is although visible but not significant. This is partially due to using binarized models in decoding.

The phrase-based model shows the worst performance accuracy- and efficiency-wise. HIERO outperforms T2S-shal and T2S-hdge models in terms of accuracy at the cost of efficiency, and performs close to T2S-full model, as it was the case in the small development set. Within best-performing tree-based systems, the type of parser makes a difference in translation performance. In general, Berkeley-parsed data provide rather better efficiency whereas ZPAR-parsed data have slightly higher accuracy. The best overall accuracy/efficiency trade-off is when a T2S-full model is combined with a fully annotated input.

Table 5.5: Accuracy and efficiency of translating English to Japanese development set (large data) using phrase-based and syntax-based models.

Translation Model	Input Annotation	Accuracy (BLEU)	Efficiency (sec/sen)	Acc-ZPAR (BLEU)	Eff-ZPAR (sec/sen)
PBMT	raw	24.84	2.03	24.84	2.03
HIERO	raw	29.14	1.77	29.14	1.77
T2S-shal	shallow	26.91	1.41	27.18	1.46
	hedge	16.32	1.27	16.74	1.27
	full	16.32	1.32	16.80	1.28
T2S-hdge	shallow	18.59	1.97	18.32	1.78
	hedge	28.58	1.71	28.53	2.01
	full	29.48	1.85	29.37	1.98
T2S-full	shallow	17.99	1.97	18.73	1.85
	hedge	26.71	1.88	27.05	2.06
	full	29.98	1.70	30.02	2.08

Test Set Results

Same as what we did for Japanese to English translation in Section 5.5.1, in Table 5.6 we report the translation performance on the English to Japanese test set using the best configuration on the development set. We see the same pattern in accuracy as we saw in the development set (see Table 5.4). In terms of efficiency, PBMT has the best efficiency, next is T2S-hdge with good improvement compared to HIERO at the cost of slight accuracy loss. T2S-full has similar efficiency to HIERO with the best accuracy than all other models.

Results for MT performance evaluation on large English-Japanese are presented in Table 5.7. The same pattern in the small development set in terms of accuracy and efficiency (Table 5.5) holds for the large test set. Translation accuracy is slightly higher, at the expense of lower efficiency, when the Berkeley parser is used to annotate the data as opposed to the ZPAR parser.

The general trends we find in incorporating syntactic annotation on source language for English to Japanese translation are as follows:
(a) In tree-to-string MT systems, syntactic richness of the input and the model has a significant impact on MT performance. Partial syntactic annotated T2S systems, which

Translation Model	Accuracy (BLEU)	Efficiency (sec/sen)
PBMT	14.70	0.40
HIERO	19.05	1.40
T2S-hdge	18.97	0.91
T2S-full	21.22	1.39

Table 5.6: Accuracy and efficiency of translating English to Japanese test set (small data set) using phrase-based and syntax-based models.

include shallow and hedge syntax, fall behind full syntactic T2S systems and also hierarchical phrase-based systems. In some cases, shallow (chunking) syntactic annotation T2S model performs even worse than a simple phrase-based model.

(b) Source-side hedge-syntax MT is consistently less accurate than source-side full-syntax MT for all maximum span limits $L{=}3-20$. For the trials of $L{=}8$, efficiency of a hedge-syntax model is often slightly better than the efficiency of a full-syntax model.

(c) In tree-to-string MT systems, the best-performing configuration is when the MT model and the input sentence are consistent on the level of syntactic richness. (Although a hedge-syntax model performs slightly better when it receives fully annotated inputs.) This trend was independent of the type of parser in our experiments.

(d) In contrast to string-to-tree models, the type of parser could noticeably impact the MT performance in tree-to-string models.

Translation Model	Accuracy (BLEU)	Efficiency (sec/sen)
PBMT	24.40	2.28
HIERO	29.15	2.09
T2S-hdge (Berkeley)	28.98	1.89
T2S-full (Berkeley)	30.73	1.84
T2S-hdge (ZPAR)	28.65	1.55
T2S-full (ZPAR)	30.58	1.66

Table 5.7: Accuracy and efficiency of translating English to Japanese test set (large data set) using phrase-based and syntax-based models.

5.5.3 Impact of Input Segmentation on Simultaneous Translation

In simultaneous translation, the unit of translation is usually smaller than a full sentence. Full sentences can be relatively long and this can cause a significant delay between presenting the translation results and the speaker's utterance. Thus, the input is segmented into sub-sentential units before submission to the MT module. Latency and quality of real-time translation is a function of the segmentation strategy. An optimal segmentation minimizes the segment length while maximizing the speed and quality of translation. Hypothetically an optimal method should consider linguistic syntax or semantics of the language. We evaluate and compare accuracy/latency trade-off of several syntax-based and orthographic segmentation methods for both offline and real-time segmentation scenarios.

We combine all the translation models from the previous section with the segmented inputs produced by four segmentation strategies. The translation models include PBMT, HIERO, and T2S. Note that the same as the previous section, S2T is not considered since we apply syntax on the input (source) side. Our input segmentation strategies are: (a) trivial every n token segments (n-token), (b) punctuation-based segments where a punctuation mark is any of [. , : ; ? ! -], (c) shallow segments in which segment boundaries are determined by shallow chunks, and (d) hedge segments in which segment boundaries are defined by hedges. Punctuation and n-token segments in (a) and (b) are raw, whereas shallow and hedge segments in (c) and (d) are syntactically annotated with shallow-syntax and hedge-syntax respectively. Note that methods (a) and (b) are not applicable to T2S models, since these models require source-side parse tree to guide the translation. Syntactically motivated segments (c) and (d) however, are applicable to PBMT and HIERO models after the annotation are removed.

In the following subsections, we explore two scenarios of input segmentation and translation. In the first scenario we assume that the segmentation is performed offline, i.e., after receiving the complete sentence output from ASR. Hence, we use non-incremental Berkeley parser for input annotation and segmentation. In this condition, we neglect segmentation time and only consider translation time. In the second scenario, we report

real-time hedge segmentation using the incremental framework we proposed in Chapter 4. Here, hedge segmentation process starts as soon as the first input word is received. Once a hedge segment is confidently determined, it is provided to the MT module. We use the incremental ZPAR parser for real-time input annotation and segmentation. In contrast to the first scenario, we ignore translation time to be able to study hedge segmentation time. The reason is that translation is order of magnitude slower than parsing, hence hedge segmentation time is negligible compared to the translation.

Offline Segmentation

Table 5.8 presents the MT accuracy, efficiency, and latency of different combinations of input segmentation and MT models on the development set. (The performance of the same MT models on non-segmented inputs were previously reported in Table 5.7.) The table also shows the average number of tokens per segment (*Len*) for each method. For n-token segmentation with n=8, *Len* is 6.98 token. Note that *Len* is not 8 since there might be segments shorter than 8 tokens at the end of the sentences. Punctuation-based segmentation has the longest segments hence smallest number of segments, while shallow-based segmentation has the shortest, but largest number of segments. Hedge-based segments with L=8 are about 1 token longer than shallow-based segments on average, and they are around 1.5 tokens shorter than hedge-based segments with L=16. Recall from Section 3.4 of Chapter 3 that the average span length in English parse trees in 6.5 tokens. This means that with L=∞ and perfect hedge segmentation, the average segment length would be 6.5 tokens.

To report efficiency and latency results, we define two conditions. First condition is "sequential translation", which assumes that translation of each segment is started after the translation of its previous segment is finished. If efficiency were defined as sum of the time taken to translate all the segments, this condition favors segmentation methods that produce longer (hence fewer) segments. The reason is that MT has search initialization and I/O overhead for translating each input unit. This overhead is lower with longer units (hence smaller number of units) such as a full sentence or entire paragraph, but

Input Segmentation		Translation Model	Len (tok)	Accuracy (BLEU)	Sequential Translation		Parallel Translation	
					Efficiency (sec/sen)	Latency (sec/w)	Efficiency (sec/sen)	Latency (sec/w)
8-token	raw	PBMT	6.98	21.23	5.8	4.1	1.7	1.7
	raw	HIERO	6.98	21.71	5.3	3.7	1.6	1.5
Punct	raw	PBMT	11.75	23.72	3.3	3.3	1.7	1.8
	raw	HIERO	11.75	27.74	3.8	4.1	2.2	2.4
Shallow	raw	PBMT	2.25	14.74	22.4	9.4	4.6	2.2
	raw	HIERO	2.25	14.86	12.3	7.9	1.3	1.2
	annot.	T2S-shal	2.25	14.80	12.2	7.8	1.2	1.1
	annot.	T2S-hdge (L=8)	2.25	15.06	12.8	8.2	1.3	1.2
	annot.	T2S-hdge (L=16)	2.25	15.08	12.3	7.9	1.2	1.1
	annot.	T2S-full	2.25	15.10	12.4	8.0	1.2	1.1
Hedge (L=8)	raw	PBMT	3.28	19.24	10.1	7.0	1.9	1.5
	raw	HIERO	3.28	21.19	8.7	5.8	1.3	1.2
	annot.	T2S-shal	3.28	15.05	8.0	5.3	1.1	1.1
	annot.	T2S-hdge (L=8)	3.28	21.45	8.1	5.4	1.2	1.1
	annot.	T2S-full	3.28	21.57	8.3	5.5	1.2	1.1
Hedge (L=16)	raw	PBMT	4.76	20.81	6.6	4.6	1.6	1.5
	raw	HIERO	4.76	23.84	6.6	4.6	1.6	1.5
	annot.	T2S-shal	4.76	15.61	5.5	3.7	1.1	1.1
	annot	T2S-hdge (L=16)	4.76	25.31	6.1	4.2	1.3	1.3
	annot.	T2S-full	4.76	25.51	6.0	4.1	1.3	1.3

Table 5.8: Accuracy and efficiency of translating segmented English to Japanese development set (large data set).

the overhead increases for shorter units such as sub-sentential segments, as the number of segments increases and translation overheads of the segments are aggregated. As can be seen in the table, accuracy as well as efficiency and latency results are generally determined by length of segments in this condition.

A more fair condition could be "parallel translation" that starts translating the sentence segments at the same time. It is a valid assumption since we are doing offline segmentation and we have the entire sentence beforehand. In this condition, efficiency will be the *maximum* (as opposed to *sum* in sequential translation) time taken to translate the segments of a sentence, averaged over all the sentences in the source text. Latency is the time taken to finish translating a word (i.e., the segment which contains the word) after it's been received, averaged over all the words in the source text. Note that in parallel translation, translating a segment will not be delayed by translating its previous segments, thus latency improves as opposed to sequential translation.

With "parallel translation", the best performance in terms of accuracy, at the cost of having the lowest efficiency and highest latency, is achieved by Punct segmentation combined with the HIERO model (27.74 BLEU, 2.2 sec/sen, 2.4 sec/w). This is because of long segments which results in a very accurate, though slow, translation due to having more context at each unit of translation. Accuracy/efficiency, and similarly accuracy/latency, trade-off of hedge segmentation combined with the hedge-syntax MT is acceptable for L=8 (21.45 BLEU, 1.2 sec/sen, 1.1 sec/w) and quite good for L=16 (25.31 BLEU, 1.3 sec/sen, 1.3 sec/w). Although the length of hedge segments is around half of that in 8-token segmentation, accuracy of translating hedge segments is very similar to the accuracy of 8-token translation for L=8 (21.45 BLEU), and much higher than 8-token translation for L=16 (25.31 BLEU). In fact, hedge-based segmentation model combined with T2S-hdge for L=16 is the second best-performing combination among all. Compared to the best-performing configuration – Punct segmentation with HIERO – the length of segments in hedge segmentation with T2S-hdge L=16 is about 60% (7 tokens) shorter, however, translation of such hedge segments is about 38% faster and only about 9% less accurate. These

results suggest that hedge segmentation and annotation could be an informative decision and the segments appear to have appropriate semantic content for translation.

While punctuation-based segmentation yields to high quality translation, it relies on a fully punctuated input string. The output of ASR has no punctuation mark and it is often punctuated using a classifier which decides punctuation boundaries after observing each token. In our experiments we assumed perfect punctuation of the input, but the classifier could have a tedious overhead on segmentation. Rangarajan Sridhar et al. (2013) reported the processing time of about *1 second per token* for punctuation-based segmentation to the ASR output. Our syntax-based segmentation methods also assume punctuated inputs, but the parser can be trained on a non-punctuated text. Nevertheless the possible disadvantage of parsing without punctuations should be taken into account. Jones (1994) studied the role of punctuation in parsing. He stated that for simple sentences use of punctuation has little or no advantage over not using them, but for longer sentences it has significant advantage.

Shallow-based segmentation combined with every MT model shows quite poor performance in terms of accuracy. The main reason is that such segments are very short and contain only 2.25 tokens on average. Moreover, the non-recursive shallow annotation of these segments does not seem to be helpful in improving translation quality. This result is consistent with *chunk* segmentation strategy combined with phrase-based MT models reported in Rangarajan Sridhar et al. (2013). To alleviate the problem of shallow segments being too short, they concatenated neighboring chunks to form longer segments and provide enough context for translation. They achieved remarkable accuracy improvement with increasingly larger chunk sizes. Similar to their procedure, we concatenated neighboring chunks of types NP, VP, and PP. As a result, the average length of segments increased to 2.6 tokens per segment. The translation accuracy for shallow-based input segmentation combined with T2S-shal for L=8 improved to 15.45 BLEU score, which is a statically significant improvement compared to 14.80 BLEU score reported for the simple shallow-based segmentation in Table 5.8. The efficiency and latency were improved to

10.0 sec/sen and 6.4 sec/w respectively in "sequential translation", and 1.1 sec/sen and 1.0 sec/w in "parallel translation".

Real-time Hedge Segmentation

In this section, we only focus on hedge segmentation method which is the main contribution of this thesis, and we examine the effect of hedge segmentation on real-time translation latency. In practice, parsing the input in order to syntactically annotate it, is significantly faster than the translation module such that parsing time is negligible compared to MT. Thus, here we assume perfect MT efficiency (0 sec/sen) so that the simultaneous translation latency is limited to the segmentation latency. We use the incremental ZPAR parser as opposed to Berkeley parser in the previous section. Consistent with the previous section, we report a word-based latency, which for each word is defined as the token (or time) difference between the word position in the sentence and the position where the partial parse containing the word is returned from the parser. We report latency in terms of tokens (w/w) as well as seconds (msec/w).

Table 5.9 presents MT accuracy and latency for offline and real-time hedge segmentation with L=8 on the development data. Note that the input segments are raw for PBMT and HIERO models, but they are annotated with the matched syntax for tree-based models. The first half of the table, shows offline segmentation results where hedge segmentation is performed after fully parsing the sentence via hedge transforming the result. This maximizes hedge parsing accuracy while also maximizing segmentation latency. The second half of the table presents real-time segmentation results where the parser is combined with the buffering algorithm from Chapter 4 to reduce segmentation latency. Buffering parameter k is set to 3 so the buffer is conservative in releasing the segments. In this scenario, hedge segmentation is performed simultaneously as the sentence is being fully parsed. The buffering algorithm gradually releases hedge segments at each stable partial parse result. This decreases segmentation latency at the cost of hedge parsing accuracy loss.

Table 5.9: Accuracy and segmentation latency of translating segmented English to Japanese development set (large data set) in offline and real-time segmentation modes.

Input Segmentation		Translation Model	Accuracy (BLEU)	Latency (w/w)	(msec/w)
Offline Hedge (L=8)	raw	PBMT	19.41	12.4	91.5
	raw	HIERO	21.28	12.4	91.5
	annot.	T2S-shal	15.26	12.4	91.5
	annot.	T2S-hdge (L=8)	21.17	12.4	91.5
	annot.	T2S-full	21.51	12.4	91.5
Real-time Hedge (L=8)	raw	PBMT	19.15	8.2	49.4
	raw	HIERO	20.83	8.2	49.4
	annot.	T2S-shal	15.20	8.2	49.4
	annot.	T2S-hdge (L=8)	20.90	8.2	49.4
	annot.	T2S-full	21.10	8.2	49.4

We observe the same pattern of accuracy compared to the previous section for offline segmentation and Berkeley parser. The best performing model is T2S-full. T2S-hdge slightly falls behind HIERO in offline segmentation, but it performs better than HIERO in real-time segmentation. Latency remarkably improves moving from offline to real-time segmentation, although translation accuracy slightly drops presumably due to higher segmentation accuracy in the offline mode. At the expense of about 1-2% accuracy loss (0.27-0.40 BLEU), we see 30% (3.5 tok) improvement in latency in terms of tokens and 40% (2.5 msec) improvement in latency in terms of seconds. The proposed segmentation strategy performs considerably fast compared to punctuation boundary classification reported in Rangarajan Sridhar et al. (2013), in spite of higher BLEU score in punctuation-based segmentation.

Test Set Results

Table 5.10 shows the results of our best development set configuration for offline segmentation on the test set. We observe the same pattern in MT accuracy, efficiency, and latency as we saw in the development set. In the parallel translation scenario, punctuation-based segmentation combined with HIERO model has the best accuracy, lowest efficiency, and highest latency, while hedge-based segmentation combined with T2S-hdge model (L=16)

Table 5.10: Accuracy and efficiency of translating segmented English to Japanese test set (large data set).

Input Segmentation		Trans. Model	Accuracy (BLEU)	Sequential Trans.		Parallel Trans.	
				Eff. (sec/sen)	Lat. (sec/w)	Eff. (sec/sen)	Lat. (sec/w)
8-token	raw	HIERO	21.87	4.5	3.1	1.3	1.3
Punct	raw	HIERO	28.15	3.6	3.4	2.2	2.3
Hedge (L=8)	annot.	Hedge (L=8)	21.64	8.2	5.5	1.2	1.1
Hedge (L=16)	annot.	Hedge (L=16)	25.79	5.9	4.1	1.3	1.3

has noticeably better efficiency/latency at the cost of accuracy. Hedge-based segmentation combined with T2S-hdge model (L=8) performs similarly to 8-token segmentation combined with HIERO, while Hedge-based segmentation is slightly more efficient, it is slightly less accurate.

Table 5.11 demonstrates the best configuration for real-time hedge segmentation on the test set. The same as in the development set, assuming a perfect MT efficiency, real-time hedge segmentation can improve translation latency significantly at the cost of some accuracy. On our test set, linguistic syntax models perform better than HIERO, the non-linguistic syntax model.

The general trends we observe in segmenting the input to simultaneous translation using syntax-based and non-syntax-based methods we investigated are as the following:
(a) In non-syntax or shallow-syntax segmentation methods, longer segments yield better

Input Segmentation		Translation Model	Accuracy (BLEU)	Latency (w/w)	(msec/w)
Offline	raw	HIERO	21.30	12.2	82.2
Hedge	annot.	T2S-hdge (L=8)	21.58	12.2	82.2
(L=8)	annot.	T2S-full	21.78	12.2	82.2
Real-time	raw	HIERO	20.98	8.0	43.2
Hedge	annot.	T2S-hdge (L=8)	21.19	8.0	43.2
(L=8)	annot.	T2S-full	21.34	8.0	43.2

Table 5.11: Accuracy and segmentation latency of translating segmented English to Japanese test set (large data set) in offline and real-time segmentation modes.

translation accuracy, however, hedge-syntax method (combined with a hedge-syntax MT model) shows pretty high accuracy in spite of producing relatively short segments. This suggests that hedge segmentation could be effective in producing syntactically and semantically informative segments of the input.

(b) Shallow-syntax segmentation performs quite poor in every configuration although we expect significant improvement if we concatenate neighboring segments to form longer segments.

(c) Since translating each segment regardless of its length has MT overhead, sequential translation of the segments biasedly results in high efficiency for segmentation methods producing longer segments. To alleviate this issue we present parallel translation results in addition to sequential translation.

(d) Punctuation-based segmentation combined with hierarchical phrase-based MT model shows the best translation accuracy, while the worst translation efficiency assuming parallel segment translation, among all input and model combinations. Punctuation does not exist in speech and detecting it could be a non-negligible overhead on the speech-to-speech translation pipeline, whereas our hedge-based segmentation using the incremental parser shows quite efficient performance.

(e) Real-time hedge segmentation, yields to comparable translation accuracy as opposed to offline hedge segmentation while segmentation latency is remarkably reduced. Assuming an optimistic translation time, real-time hedge segmentation could improve overall accuracy/latency trade-off of simultaneous translation.

5.6 Summary

In this chapter, we investigated the impact of incorporating hedge parsing, as a partial parsing approach, to machine translation. We compared hedge-syntax MT system with MT systems which use other types of syntax or do not use syntactic information. We also examined the incremental hedge parsing approach we proposed in Chapter 4 in simultaneous translation. We demonstrated that for non-monotonic language pairs we studied, the knowledge of local hierarchical structures in the MT system could significantly improve

regular or simultaneous translation performance compared to the MT systems with non-hierarchical shallow syntax. Our results suggest that hedge syntax could be as effective as full syntax when they are applied to the target side of an MT system.

Chapter 6

Conclusion and Future Work

6.1 Summary

The primary goal of this research was developing a fast incremental syntactic parsing approach that enables incorporating partial syntactic knowledge of language in real-time language processing applications. To accomplish this goal, we proposed 'hedge parsing', real-time hedge parsing, and its application in simultaneous machine translation.

We first introduced a novel partial parsing method for real-time NLP applications that require a fast syntactic analysis of the input beyond shallow syntactic annotation. Hedge parsing provides local internal hierarchical structure of phrases covering up to some maximum span, without requiring fully connected parses. The idea of constraining the span of constituents in hedge parsing is similar to constraining the dependency length in Vine parsing. The maximum span parameter allows tuning the annotation of internal structure as appropriate for the application domain, trading off annotation complexity against inference time. These properties make hedge parsing potentially very useful for incremental text or speech processing, such as streaming text analysis or simultaneous translation. One interesting characteristic of these annotations is that they allow for string segmentation prior to inference, provided that the segment boundaries do not cross any hedge boundaries. We found that baseline segmentation models did provide a significant speedup in parsing, but that cascading errors remain a problem.

We then suggested a real-time hedge parsing framework relying on incremental shift-reduce parsing algorithm for joint low-latency hedge parsing and segmentation. The major advantage of classifier-based shift-reduce parsing over CYK parsing is its incremental nature as well as its linear-time complexity, which makes it suitable for real-time applications. To ensure that the returned hedges will not change in the next stages of parsing we detected stable intermediate parse results in the full parsing beam search lattice. To the best of our knowledge, stability in parsing beam search space has not been studied before. Once we recognized such stable partial parse trees, we released their complete hedges and we delayed releasing yet-to-complete hedges until receiving next stable results. We demonstrated that this framework is very effective in producing fast low-latency hedges.

Finally we explored the impact of hedge parsing on the performance of machine translation. We showed that we can achieve a significant improvement in translation quality by integrating the knowledge of hedge syntax on the target language side of the translation model compared to shallow- or no-syntax integration. This improvement is comparable to the performance of a full-syntactically informed MT model. We also found that applying hedge syntax of the source language side of the translation model falls behind full syntax although again outperforms shallow syntax. Hedge segmentation and annotation of the inputs resulted in an acceptable accuracy/latency trade-off in simultaneous translation as opposed to alternative segmentation methods. In particular, it notably outperforms shallow syntax segmentation and annotation.

6.2 Future Work

There are many directions of future work to pursue here. Improvements to the hedge transform in Chapter 3, such as grouping consecutive unary nodes or POS tags under non-terminals, could improve accuracy without greatly reducing efficiency. Combining hedge transform with other tree transforms not discussed in this thesis, particularly those that change the topology of the tree such as left-corner transform or regular approximation, could positively affect accuracy or efficiency. Depending on the application, hedge

transform could be relaxed by some conditions, for example we may want to save all NP constituents in the transformed tree, regardless of their span lengths.

Improvements to the segmentation model in Chapter 3 could greatly improve accuracy as well as efficiency of hedge parsing in pre-segmenting scenarios. This improvement could be achieved through a better classifier (e.g., by feature engineering or using a neural model), or through an improved hedge transform which allows for more accurate hedge segmentation. To reduce the negative impact of segmentation errors to the parsing performance we could choose fewer number of segments and those which are most likely the correct segments. To do this, we can change the segmentation model such that it makes precision-oriented decision (i.e., increases precision at the expense of recall) in classifying segment boundaries. Although our preliminary results of this approach on current hedge transform does not improve hedge parsing accuracy/efficiency trade-off, more investigation on this area is of interest. Finally, hedgebank grammar could be used as an efficient coarse grammar to prune the subsequent search in a full coarse-to-fine pruning model, for example in the pipeline systems of the sort discussed in Hollingshead (2010).

Similar ideas in Chapter 4 can be used to produce syntactic segments of other types (such as shallow bracketing) for incremental applications. Our method guarantees 100% stability of the released segments and we expect even further improvements in latency at the cost of reduced stability. It might be the case that if the majority of the beams, above some specific threshold, in beam search decoding agree on the intermediate parse result, the released segments at these points will be stable or slightly changed later during parsing. Finding such threshold would be of interest. In addition to measuring stability—likelihood of a partial parse remaining unchanged compared to the final parse—we could also measure confidence—likelihood of partial correctness compared to the true parse—similarly to the study of incremental speech recognition presented in Selfridge et al. (2011). An alternative method to the buffering procedure is to apply a classifier at each stable partial parse to predict hedge boundaries. This classifier can be trained using a rich feature set including the features that are used for full parsing of the sentence.

Pursuing hedge segmentation and annotation in a full speech-to-speech translation pipeline, including machine translation preceded by speech recognition and followed by text-to-speech synthesis, is another avenue for future work. In Chapter 5 we evaluated the machine translation component in isolation. In other words, we assumed that our system receives the transcribed speech, translates it to the target language text, and does not convert the target text to the target speech. Evaluating a full SST pipeline enables measuring the latency of the other two components as well as the impact of cascading speech-recognition errors on translation accuracy. We would like to examine the scalability of our results to translating several languages with different morphological and syntactic properties such as German, which has an interesting noun and verb compounding system, or Persian, which its normal declarative sentences are of subject-object-verb structure and is a pro-drop language, meaning that the subject of a sentence often appears at the end of the verb and thus the end of the sentence. Such properties may also affect the speech synthesizer.

Appendix A

English Penn Treebank Tagset

Table A.1: English Penn Treebank POS tagset.

POS Tag	Description
#	Pound sign
$	Dollar sign
“	Opening quotation mark
”	Closing quotation mark
,	Comma
(	Opening parenthesis
)	Closing parenthesis
.	Sentence terminator
:	Colon or ellipsis
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential ‘there’
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNP	Proper noun, singular
NNPS	Proper noun, plural

POS Tag	Description
NNS	Noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	“to”
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Table A.2: English Penn Treebank phrase tagset.

Phrase Tag	**Description**
ADJP	Adjective Phrase
ADVP	Adverb Phrase
CONJP	Conjunction Phrase
FRAG	Fragment
INTJ	Interjection
LST	List marker
NAC	Not a Constituent
NP	Noun Phrase
NX	Used within certain complex NPs to mark the head of the NP
PP	Prepositional Phrase
PRN	Parenthetical
PRT	Particle
QP	Quantifier Phrase
RRC	Reduced Relative Clause
S	Simple declarative clause
SBAR	Clause introduced by a subordinating conjunction
SBARQ	Direct question introduced by a wh-word or a wh-phrase
SINV	Inverted declarative sentence
SQ	Inverted yes/no question, or main clause of a wh-question, following the wh-phrase in SBARQ
UCP	Unlike Coordinated Phrase
VP	Vereb Phrase
WHADJP	Wh-adjective Phrase
WHAVP	Wh-adverb Phrase
WHNP	Wh-noun Phrase
WHPP	Wh-prepositional Phrase
X	Unknown, uncertain, or unbracketable

Appendix B

English and Chinese Constituent Head Percolation Rules

Table B.1: Head-finding Rules for English (from the ZPAR parser (Zhang and Clark, 2011)).

Constituent	**Rule**
ADJP	l NNS QP NN $ ADVP JJ VBN VBG ADJP JJR NP JJS DT FW RBR RBS SBAR RB
ADVP	r RB RBR RBS FW ADVP TO CD JJR JJ IN NP JJS NN
CONJP	r CC RB IN
FRAG	r
INTJ	l
LST	r LS :
NAC	l NN NNS NNP NNPS NP NAC EX $ CD QP PRP VBG JJ JJS JJR ADJP FW
NP	rd NN NNP NNPS NNS NX POS JJR; l NP; rd $ ADJP PRN; r CD; rd JJ JJS RB QP
NX	l
PP	r IN TO VBG VBN RP FW
PRN	l
PRT	r RP
QP	l $ IN NNS NN JJ RB DT CD NCD QP JJR JJS
RRC	r VP NP ADVP ADJP PP
S	l TO IN VP S SBAR ADJP UCP NP
SBAR	l WHNP WHPP WHADVP WHADJP IN DT S SQ SINV SBAR FRAG
SBARQ	l SQ S SINV SBARQ FRAG
SINV	l VBZ VBD VBP VB MD VP S SINV ADJP NP
SQ	l VBZ VBD VBP VB MD VP SQ
UCP	r
VP	l TO VBD VBN MD VBZ VB VBG VBP AUX AUXG VP ADJP NN NNS NP
WHADJP	l CC WRB JJ ADJP
WHADVP	r CC WRB
WHNP	l WDT WP WP$ WHADJP WHPP WHNP
WHPP	r IN TO FW
X	r

Table B.2: Head-finding Rules for Chinese (from the ZPAR parser (Zhang and Clark, 2011)).

Constituent	**Rule**
ADJP	r ADJP JJ AD; r
ADVP	l CS; r ADVP AD JJ NP PP P VA VV; r
CLP	r CLP M NN NP; r
CP	r DEC CP ADVP IP VP; r
DNP	r DEG DNP DEC QP; r
DP	r QP M CLP; l DP DT OD; l
DVP	r DEV AD VP; r
IP	r VP IP NP; r
LCP	r LCP LC; r
LST	r CD NP QP; r
NP	r NP NN IP NR NT; r
NN	r NP NN IP NR NT; r
PP	l P PP; l
PRN	l PU; l
QP	r M QP CLP CD OD; r
UCP	r IP NP VP; r
VCD	r VV VA VE; r
VP	l VE VC VV VNV VPT VRD VSB VCD VP IP; l
VPT	l VA VV; l
VRD	l VV VA; l
VSB	r VV VE; r
FRAG	r VP VV NP NR NN NT; r

Bibliography

Abney, S., Flickenger, S., Gdaniec, C., Grishman, C., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B. and Strzalkowski, T. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In E. Black (ed.), *Proceedings of a workshop on Speech and natural language*, pages 306–311, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Atterer, M. and Schlangen, D. 2009. RUBISC: a robust unification-based incremental semantic chunker. In *Proceedings of the 2nd Workshop on Semantic Representation of Spoken Language*, pages 66–73.

Banerjee, S. and Lavie, A. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, volume 29, pages 65–72.

Bangalore, S. 2000. Performance evaluation of supertagging for partial parsing. In *Advances in probabilistic and other parsing technologies*, pages 203–220.

Bangalore, S. and Joshi, A. K. 1999. Supertagging: An Approach to Almost Parsing. *Computational Linguistics* 25(2), 237–265.

Bangalore, S., Rangarajan Sridhar, V. K., Kolan, P., Golipour, L. and Jimenez, A. 2012. Real-time incremental speech-to-speech translation of dialogs. In *Proceedings of the 2012 Conference of the NAACL:HLT*, pages 437–445.

Baumann, T., Bangalore, S. and Hirschberg, J. 2014. Towards Simultaneous Interpreting: The Timing of Incremental Machine Translation and Speech Synthesis. In *11th International Workshop on Spoken Language Translation (IWSLT 2014)*.

Blunsom, P. and Baldwin, T. 2006. Multilingual deep lexical acquisition for HPSGs via supertagging. In *Proceedings of the 2006 conference on empirical methods in natural language processing*, pages 164–171, Association for Computational Linguistics.

Bodenstab, N., Dunlop, A., Hall, K. and Roark, B. 2011. Beam-width prediction for efficient context-free parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 440–449.

Bodenstab, N., Dunlop, A., Roark, B. and Hall, K. 2010. Exponential Decay Pruning for Bottom-Up Beam-Search Parsing. In *Pacific Northwest Regional NLP Workshop (NW-NLP)*.

Bodenstab, N. M. 2012. *Prioritization and pruning: efficient inference with weighted context-free grammars*. Ph. D.thesis, Oregon Health & Science University.

Briscoe, T. and Carroll, J. 1993. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational linguistics* 19(1), 25–59.

Brown, P., Della Pietra, V., Della Pietra, S. and Mercer, R. 1993. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics* 19(2), 263–311.

Brüggemann-Klein, A. and Wood, D. 2004. Balanced Context-Free Grammars, Hedge Grammars and Pushdown Caterpillar Automata. In *Extreme Markup Languages*.

Cer, D. M., de Marneffe, M.-C., Jurafsky, D. and Manning, C. D. 2010. Parsing to Stanford Dependencies: Trade-offs between Speed and Accuracy. In Nicoletta Calzolari, Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner and Daniel Tapias (eds.), *LREC*, European Language Resources Association.

Cettolo, M. and Federico, M. 2006. Text segmentation criteria for statistical machine translation. In *Proceedings of the 5th international conference on Advances in Natural Language Processing*, pages 664–673.

Chappelier, J.-C. and Rajman, M. 1998. A Generalized CYK Algorithm for Parsing Stochastic CFG. In *Proceedings of the First Workshop on Tabulation in Parsing and Deduction*, pages 133–137.

Chiang, D. 2005. A Hierarchical Phrase-based Model for Statistical Machine Translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 263–270, Stroudsburg, PA, USA: Association for Computational Linguistics.

Chiang, D. 2006. An introduction to synchronous grammars. In *Tutorial given at ACL 2006*.

Chiang, D. 2007. Hierarchical phrase-based translation. *computational linguistics* 33(2), 201–228.

Clark, J. H., Dyer, C., Lavie, A. and Smith, N. A. 2011. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 176–181, Association for Computational Linguistics.

Clark, S. 2002. Supertagging for Combinatory Categorial Grammar. In *Proceedings of the International Workshop on Tree Adjoining Grammars*, pages 19–24, Venice, Italy.

Clark, S. and Curran, J. R. 2004. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 103, Association for Computational Linguistics.

Cocke, J. and Schwartz, J. 1970. *Programming Languages and Their Compilers: Preliminary Notes*. Courant Institute of Mathematical Sciences, New York University.

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

Collins, M. 2002. Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *Proceedings of the conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–8.

Collins, M. and Roark, B. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111, Association for Computational Linguistics.

Covington, M. A. 2001. A fundamental algorithm for dependency parsing. In *In Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102.

Demberg, V. and Keller, F. 2008. A psycholinguistically motivated version of TAG. In *Proceedings of the ninth international workshop on tree adjoining grammars and related formalisms*, Tbingen.

Demberg, V., Keller, F. and Koller, A. 2013. Incremental, predictive parsing with psycholinguistically motivated tree-adjoining grammar. *Computational Linguistics* 39(4), 1025–1066.

DeNeefe, S., Knight, K., Wang, W. and Marcu, D. 2007. What Can Syntax-Based MT Learn from Phrase-Based MT? In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 755–763.

Doddington, G. 2002. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*, pages 138–145, Morgan Kaufmann Publishers Inc.

Dreyer, M., Smith, D. A. and Smith, N. A. 2006. Vine parsing and minimum risk reranking for speed and precision. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 201–205.

Dunlop, A. J. 2014. *Efficient Latent-Variable Grammars: Learning and Inference*. Ph. D.thesis, Oregon Health & Science University.

Eisner, J. and Smith, N. A. 2005. Parsing with soft and hard constraints on dependency length. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 30–41.

Eisner, J. M. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345, Association for Computational Linguistics.

Federico, M., Bentivogli, L., Paul, M. and Stüker, S. 2011. Overview of the IWSLT 2011 Evaluation Campaign. In *Proceedings of International Workshop on Spoken Language Translation*.

Fügen, C., Waibel, A. and Kolss, M. 2007. Simultaneous translation of lectures and speeches. *Machine Translation* 21(4), 209–252.

Fujita, T., Neubig, G., Sakti, S., Toda, T. and Nakamura, S. 2013. Simple, lexicalized choice of translation timing for simultaneous speech translation. In *Proceedings of Interspeech*, pages 3487–3491.

Galley, M., Graehl, J., Knight, K., Marcu, D., DeNeefe, S., Wang, W. and Thayer, I. 2006. Scalable inference and training of context-rich syntactic translation models. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 961–968, Association for Computational Linguistics.

Gao, Q. and Vogel, S. 2008. Parallel implementations of word alignment tool. In *Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, pages 49–57, Association for Computational Linguistics.

Glaysher, E. and Moldovan, D. 2006. Speeding up full syntactic parsing by leveraging partial parsing decisions. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 295–300, Association for Computational Linguistics.

Goldberg, Y., Zhao, K. and Huang, L. 2013. Efficient Implementation of Beam-Search Incremental Parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics.*, pages 628–633, Association for Computational Linguistics.

Hassan, H., Sima'an, K. and Way, A. 2009. A syntactified direct translation model with linear-time decoding. In *Proceedings of the 2009 Conference on EMNLP*, pages 1182–1191.

Heafield, K. 2011. KenLM: Faster and Smaller Language Model Queries. In *Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, United Kingdom.

Hefny, A., Hassan, H. and Bahgat, M. 2011. Incremental combinatory categorial grammar and its derivations. In *Computational Linguistics and Intelligent Text Processing*, pages 96–108.

Hollingshead, K. 2010. *Formalizing the use and characteristics of constraints in pipeline systems*. Ph. D.thesis, Oregon Health & Science University.

Hollingshead, K., Fisher, S. and Roark, B. 2005. Comparing and combining finite-state and context-free parsers. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 787–794, Association for Computational Linguistics.

Hollingshead, K. and Roark, B. 2007. Pipeline Iteration. In *Proceedings of the 45st Annual Meeting on Association for Computational Linguistics*, pages 952–959, Prague, Czech Republic.

Huang, L., Jiang, W. and Liu, Q. 2009. Bilingually-constrained (monolingual) shift-reduce parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3*, pages 1222–1231, Association for Computational Linguistics.

Huang, L., Knight, K. and Joshi, A. 2006. A Syntax-Directed Translator with Extended Domain of Locality. In *Proceedings of the Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*, pages 1–8, New York City, New York: Association for Computational Linguistics.

Huang, L. and Sagae, K. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086, Association for Computational Linguistics.

Johnson, M. 1998. PCFG models of linguistic tree representations. *Computational Linguistics* 24(4), 613–632.

Jones, B. E. 1994. Exploring the role of punctuation in parsing natural text. In *Proceedings of the 15th conference on Computational linguistics-Volume 1*, pages 421–425, Association for Computational Linguistics.

Joshi, A. K. and Schabes, Y. 1997. Tree-adjoining grammars. In *Handbook of formal languages*, pages 69–123, Springer.

Joshi, A. K. and Srinivas, B. 1994. Disambiguation of super parts of speech (or supertags): Almost parsing. In *Proceedings of the 15th Conference on Computational Linguistics-Volume 1*, pages 154–160.

Jurafsky, D. and Martin, J. H. 2009. *Speech and Language Processing (2nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Karlsson, F., Voutilainen, A., Heikkilae, J. and Anttila, A. 1995. *Constraint Grammar: a language-independent system for parsing unrestricted text*, volume 4. Walter de Gruyter.

Kasami, T. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

Kato, Y., Matsubara, S. and Inagaki, Y. 2004. Stochastically evaluating the validity of partial parse trees in incremental parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 9–15, Association for Computational Linguistics.

Koehn, P. 2004. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *Machine translation: From real users to research*, pages 115–124, Springer.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A. and Herbsts, E. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL Interactive Poster and Demonstration Sessions*, pages 177–180.

Kolss, M., Vogel, S. and Waibel, A. 2008. Stream decoding for simultaneous spoken language translation. In *Interspeech*, pages 2735–2738, ISCA.

Levy, R. and Manning, C. 2003. Is it harder to parse Chinese, or the Chinese Treebank? In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 439–446, Association for Computational Linguistics.

Liu, Y., Liu, Q. and Lin, S. 2006. Tree-to-string alignment template for statistical machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 609–616, Association for Computational Linguistics.

Liu, Y., Liu, Q. and Lü, Y. 2011. Adjoining tree-to-string translation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1278–1287, Association for Computational Linguistics.

Lopez, A. 2008. Statistical Machine Translation. *ACM Computing Surveys* 40(3), 1–49.

Manning, C. D. and Schütze, H. 1999. *Foundations of statistical natural language processing*, volume 999. MIT Press.

Marcu, D., Wang, W., Echihabi, A. and Knight, K. 2006. SPMT: Statistical Machine Translation with Syntactified Target Language Phrases. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 44–52, Sydney, Australia: Association for Computational Linguistics.

Marcu, D. and Wong, W. 2002. A phrase-based, joint probability model for statistical machine translation. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 133–139, Association for Computational Linguistics.

Marcus, M. P., Marcinkiewicz, M. A. and Santorini, B. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics* 19(2), 313–330.

Marcus, M. P., Santorini, B., Marcinkiewicz, M. A. and Taylor, A. 1999. Treebank-3. Linguistic Data Consortium, Philadelphia.

Matsuzaki, T., Miyao, Y. and Tsujii, J. 2007. Efficient HPSG Parsing with Supertagging and CFG-filtering. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 1671–1676.

Matsuzaki, T., Miyao, Y. and Tsujii, J. 2005. Probabilistic CFG with latent annotations. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 75–82, Association for Computational Linguistics.

Matusov, E., Hillard, D., Magimai-Doss, M., Hakkani-Tür, D., Ostendorf, M. and Ney, H. 2007. Improving speech translation with automatic boundary prediction. In *Proceedings of Interspeech*.

McDonald, R., Hall, K. and Mann, G. 2010. Distributed training strategies for the structured perceptron. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 456–464, Association for Computational Linguistics.

McDonald, R., Pereira, F., Ribarov, K. and Hajič, J. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530, Association for Computational Linguistics.

Mi, H., Huang, L. and Liu, Q. 2008. Forest-Based Translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 192–199.

Nakazawa, T., Mino, H., Goto, I., Neubig, G., Kurohashi, S. and Sumita, E. 2015. Overview of the 2nd Workshop on Asian Translation. In *Proceedings of the 2nd Workshop on Asian Translation (WAT2015)*, pages 1–28, Kyoto, Japan.

Nesson, R., Shieber, S. M. and Rush, A. 2006. Induction of Probabilistic Synchronous Tree-Insertion Grammars for Machine Translation. In *5th Conference of the Association for Machine Translation in the Americas (AMTA)*, Boston, Massachusetts.

Neubig, G., Watanabe, T., Sumita, E., Mori, S. and Kawahara, T. 2011. An unsupervised model for joint phrase alignment and extraction. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 632–641, Association for Computational Linguistics.

Nivre, J. 2003. An Efficient Algorithm for Projective Dependency Parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Och, F. J., Tillmann, C. and Ney, H. 1999. Improved Alignment models for Statistical Machine Translation. In *Proc. of the Joint SIGDAT Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP99)*, pages 20–28, University of Maryland, College Park, MD, USA.

Och, F. J. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 160–167, Association for Computational Linguistics.

Och, F. J. and Ney, H. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 295–302, Association for Computational Linguistics.

Och, F. J. and Ney, H. 2004. The alignment template approach to statistical machine translation. *Computational linguistics* 30(4), 417–449.

Oda, Y., Neubig, G., Sakti, S., Toda, T. and Nakamura, S. 2014. Optimizing Segmentation Strategies for Simultaneous Speech Translation. In *Proceedings of the 52nd Annual Meeting of the ACL*, pages 551–556.

Oda, Y., Neubig, G., Sakti, S., Toda, T. and Nakamura, S. 2015. Syntax-based Simultaneous Translation through Prediction of Unseen Syntactic Constituents. In *The 53rd Annual Meeting of the Association for Computational Linguistics (ACL)*, Beijing, China.

Papineni, K., Roukos, S., Ward, T. and jing Zhu, W. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318.

Petrov, S., Barrett, L., Thibaux, R. and Klein, D. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440, Association for Computational Linguistics.

Petrov, S. and Klein, D. 2007a. Improved Inference for Unlexicalized Parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics, Proceedings of the Main Conference*, pages 404–411.

Petrov, S. and Klein, D. 2007b. Learning and inference for hierarchically split PCFGs. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, AAAI'07, pages 1663–1666.

Pöchhacker, F. 2002. *The Interpreting Studies Reader*. New York: Routledge (Taylor and Francis).

Post, M., Callison-Burch, C. and Osborne, M. 2012. Constructing parallel corpora for six indian languages via crowdsourcing. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 401–409, Association for Computational Linguistics.

Prud'hommeaux, E. T. 2012. *Alignment of narrative retellings for automated neuropsychological assessment*. Ph. D.thesis, Oregon Health & Science University.

Ramshaw, L. and Marcus, M. 1995. Text Chunking Using Transformation-Based Learning. In *Proceedings of the Third ACL Workshop on Very Large Corpora*.

Rangarajan Sridhar, K. V., Chen, J., Bangalore, S., Ljolje, A. and Chengalvarayan, R. 2013. Segmentation Strategies for Streaming Speech Translation. In *Proceedings of the 2013 Conference of the NAACL:HLT*, pages 230–238.

Roark, B. and Sproat, R. 2007. *Computational Approaches to Morphology and Syntax*. Oxford Surveys in Syntax & Morphology, OUP Oxford.

Roark, B. 2001. Probabilistic top-down parsing and language modeling. *Computational linguistics* 27(2), 249–276.

Roark, B. 2004. Robust garden path parsing. *Natural language engineering* 10(01), 1–24.

Roark, B. and Hollingshead, K. 2008. Classifying chart cells for quadratic complexity context-free inference. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 745–751, Association for Computational Linguistics.

Roark, B., Hollingshead, K. and Bodenstab, N. 2012. Finite-State Chart Constraints for Reduced Complexity Context-Free Parsing Pipelines. *Computational Linguistics* 38(4), 719–753.

Rush, A. M. and Petrov, S. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 498–507, Association for Computational Linguistics.

Ryu, K., Matsubara, S. and Inagaki, Y. 2006. Simultaneous English-Japanese spoken language translation based on incremental dependency parsing and transfer. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 683–690, Association for Computational Linguistics.

Sagae, K. and Lavie, A. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 125–132, Association for Computational Linguistics.

Sang, E. F. T. K. and Buchholz, S. 2000. Introduction to the CoNLL-2000 Shared Task: Chunking. In *Proceedings of Conference on Computational Natural Language Learning (CoNLL)*, pages 127–132.

Sankaran, B., Grewal, A. and Sarkar, A. 2010. Incremental decoding for phrase-based statistical machine translation. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, WMT '10, pages 216–223, Stroudsburg, PA, USA: Association for Computational Linguistics.

Saraclar, M., Riley, M., Bocchieri, E. and Goffin, V. 2002. Towards automatic closed captioning: low latency real time broadcast news transcription. In *Proceedings of Interspeech*.

Sarawagi, S. and Cohen, W. W. 2004. Semi-Markov Conditional Random Fields for Information Extraction. In *NIPS*, volume 17, pages 1185–1192.

Sarkar, A. 2007. Combining supertagging and lexicalized tree-adjoining grammar parsing. *Complexity of Lexical Descriptions and its Relevance to Natural Language Processing: A Supertagging Approach* .

Schlangen, D. and Skantze, G. 2009. A general, abstract model of incremental dialogue processing. In *Proceedings of the 12th Conference of EACL*, pages 710–718.

Schuler, W., Wu, S. and Schwartz, L. 2009. A framework for fast incremental interpretation during speech decoding. *Computational Linguistics* 35(3), 313–343.

Selfridge, E. O., Arizmendi, I., Heeman, P. A. and Williams, J. D. 2011. Stability and accuracy in incremental speech recognition. In *Proceedings of the SIGDIAL 2011 Conference*, pages 110–119, Association for Computational Linguistics.

Shavarani, H. S., Siahbani, M., Seraj, R. M. and Sarkar, A. 2015. Learning Segmentations that Balance Latency versus Quality in Spoken Language Translation. In *12th International Workshop on Spoken Language Translation (IWSLT 2015)*, pages 217–224, Da Nang, Vietnam.

Shen, L. and Joshi, A. K. 2005. Incremental LTAG parsing. In *Proceedings of the Conference on HLT:EMNLP*, pages 811–818.

Siahbani, M., Sankaran, B. and Sarkar, A. 2013. Efficient Left-to-Right Hierarchical Phrase-Based Translation with Improved Reordering. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP, Seattle, Washington, USA*, pages 1089–1099.

Siahbani, M., Seraj, R. M., Sankaran, B. and Sarkar, A. 2014. Incremental translation using hierarchical phrase-based translation system. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 71–76, IEEE.

Skantze, G. and Hjalmarsson, A. 2010. Towards incremental speech generation in dialogue systems. In *Proceedings of the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 1–8.

Srinivas, B. 1996. "almost parsing" technique for language modeling. In *The 4th International Conference on Spoken Language Processing, Philadelphia, PA, USA, October 3-6, 1996*.

Steedman, M. 1986. *Combinators and grammars*. R. Oehrle, E. Bach, D. Wheeler (Eds.), Categorial Grammars and Natural Language Structures, Foris, Dordrecht.

Tamura, A., Watanabe, T., Sumita, E., Takamura, H. and Okumura, M. 2013. Part-of-Speech Induction in Dependency Trees for Statistical Machine Translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 841–851, Association for Computational Linguistics.

Venugopal, A., Vogel, S. and Waibel, A. 2003. Effective phrase translation extraction from alignment models. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 319–326, Association for Computational Linguistics.

Venugopal, A., Zollmann, A. and Vogel, S. 2007. An Efficient Two-Pass Approach to Synchronous-CFG Driven Statistical MT. In *Proceedings of NAACL HLT*, pages 500–507.

Vogel, S., Ney, H. and Tillmann, C. 1996. HMM-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 836–841.

Vogel, S., Och, F. J., Tillmann, C., Nießen, S., Sawaf, H. and Ney, H. 2000. Statistical methods for machine translation. In *Verbmobil: Foundations of Speech-to-Speech Translation*, pages 377–393, Springer.

Wang, M., Sagae, K. and Mitamura, T. 2006a. A fast, accurate deterministic parser for Chinese. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 425–432, Association for Computational Linguistics.

Wang, W. and Harper, M. P. 2002. The SuperARV language model: Investigating the effectiveness of tightly integrating multiple knowledge sources. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 238–247, Association for Computational Linguistics.

Wang, W., Harper, M. P. and Stolcke, A. 2003. The robustness of an almost-parsing language model given errorful training data. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).*, volume 1, pages I–240, IEEE.

Wang, X., Lin, X., Yu, D., Tian, H. and Wu, X. 2006b. Chinese Word Segmentation with Maximum Entropy and N-gram Language Model. In *Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing*, pages 138–141, Sydney, Australia: Association for Computational Linguistics.

Wang, Y.-Y. and Waibel, A. 1997. Decoding algorithm in statistical machine translation. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, pages 366–372, Association for Computational Linguistics.

Wolfel, M., Kolss, M., Kraft, F., Niehues, J., Paulik, M. and Waibel, A. 2008. Simultaneous machine translation of German lectures into English: Investigating research challenges for the future. In *Spoken Language Technology Workshop, 2008.*, pages 233–236, IEEE.

Xue, N., Xia, F., Chiou, F.-D. and Palmer, M. 2005. The Penn Chinese TreeBank: Phrase structure annotation of a large corpus. *Natural language engineering* 11(02), 207–238.

Yarmohammadi, M. 2014. Discriminative training with perceptron algorithm for pos tagging task. Technical Report CSLU-2014-001.

Yarmohammadi, M., Dunlop, A. and Roark, B. 2014. Transforming trees into hedges and parsing with "hedgebank" grammars. In *Proceedings of the 52nd Annual Meeting of the ACL*, pages 797–802, Baltimore, Maryland: Association for Computational Linguistics.

Yarmohammadi, M., Sridhar, V. K. R., Bangalore, S. and Sankaran, B. 2013. Incremental Segmentation and Decoding Strategies for Simultaneous Translation. In *Proceedings of the 6th International Joint Conference on Natural Language Processing (IJCNLP)*, pages 1032–1036.

Younger, D. H. 1967. Recognition and parsing of context-free languages in time n 3. *Information and control* 10(2), 189–208.

Zhang, D., Li, M., Li, C.-H. and Zhou, M. 2007a. Phrase Reordering Model Integrating Syntactic Knowledge for SMT. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 533–540.

Zhang, H., Fang, L., Xu, P. and Wu, X. 2011. Binarized forest to string translation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 835–845, Association for Computational Linguistics.

Zhang, M., Jiang, H., Aw, A., Sun, J., Li, S. and Tan, C. L. 2007b. A Tree-to-Tree Alignment-based Model for Statistical Machine Translation. In *Proceedings of the MT Summit XI*.

Zhang, Y., Vogel, S. and Waibel, A. 2003. Integrated phrase segmentation and alignment algorithm for statistical machine translation. In *Natural Language Processing and Knowledge Engineering, 2003.*, pages 567–573, IEEE.

Zhang, Y. and Clark, S. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Association for Computational Linguistics.

Zhang, Y. and Clark, S. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics* 37(1), 105–151.

Zhang, Y. and Nivre, J. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 188–193, Association for Computational Linguistics.

Zollmann, A. and Venugopal, A. 2006. Syntax augmented machine translation via chart parsing. In *Proceedings of the Workshop on Statistical Machine Translation*, pages 138–141, Association for Computational Linguistics.

Zollmann, A., Venugopal, A., Och, F. and Ponte, J. 2008. A systematic comparison of phrase-based, hierarchical and syntax-augmented statistical MT. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 1145–1152, Association for Computational Linguistics.