Predictive Dependency Parsing

Dissertation an der Universität Hamburg

Arne Köhn

2019

Fakultät für Mathematik, Informatik und Naturwissenschaften,
Fachbereich Informatik

I want to thank first and foremost Christine Köhn, who not only tolerates me every day but who also read drafts of this thesis and made many detailed and extremely helpful comments. This thesis improved a lot from this. My parents always unconditionally supported my desire to be and work in academia. This thesis would probably not exist without that support. Christopher Habel and Wolfgang Menzel gave me the possibility to work freely on interesting topics. Wolfgang Menzel also gave me helpful feedback on my work and this thesis. Chris Biemann and Timo Baumann provided helpful remarks on drafts.

Gutachter: _____ _____

Abstracts

This dissertation is concerned with analyzing the syntactic structure of dynamically evolving sentences before the sentences are complete. Human processing of both written and spoken language is inherently incremental, but most computational language processing happens under the assumption that all relevant data is available before processing begins. I discuss different approaches to build incremental processors and how to evaluate them.

I introduce two different approaches to incremental parsing. One performs restart-incremental parsing, obtaining very high accuracies. The other uses a novel transition system combined with a discriminative component; while it parses with lower accuracy, it can be trained on arbitrary dependency treebanks without any pre-processing and parses sentences at speeds of 3ms per word. Both approaches can be trained on existing treebanks and are language independent. Also, both try to provide as much information as possible by also predicting structure containing stand-ins for words not yet seen. To show that these structural predictions do provide non-trivial information, I demonstrate that n-gram language models benefit from incorporating these predictions, which is only possible if the predictions encode long-spanning information about the sentence structure.

Diese Dissertation befasst sich mit der Analyse syntaktischer Strukturen von noch unvollständigen Sätzen. Menschliche Sprachverarbeitung sowohl des geschriebenen als auch gesprochenen Wortes is inhärent inkrementell, während bei maschineller Verarbeitung meist davon ausgegangen wird, dass alle relevanten Informationen bereits zugreifbar sind bevor die Verarbeitung beginnt. Ich bespreche verschiedene Ansätze um inkrementelle Prozessoren zu bauen und diese zu evaluieren.

Ich stelle zwei verschiedene Ansätze für inkrementelles Parsing vor, die beide versuchen so viel Information wie möglich zu generieren indem sie Struktur vorhersagen die Platzhalter for noch nicht gesehene Worte enthält. Beide können auf existierenden Baumbanken trainiert werden und sind sprachunabhängig. Ein Ansatz ist restart-inkrementell, wodurch er sehr hohe Genauigkeiten erzielt. Der andere nutzt ein neuartiges Transitionssystem kombiniert mit einer diskriminativen Komponente; er parst mit geringerer Genauigkeit, kann aber ohne Vorverarbeitung auf beliebigen Dependenzbaumbanken trainiert werden und kann Sätze mit einer Geschwindigkeit von 3ms pro Wort parsen. Um zu zeigen, dass

die strukturellen Vorhersagen tatsächlich nicht-triviale Information enthalten, zeige ich, dass n-gram-Sprachmodelle von diesen Informationen profitieren; dies ist nur möglich, da die Vorhersagen Informationen über die Satzstruktur kodieren, die über den begrenzenten Horizont der n-gram-Sprachmodelle hinausgehen.

Contents

Chapter 1.

Introduction

Natural language is ubiquitous in our lives. Humans use natural language to interact with each other, but interaction with computers is also often performed using natural language. Be it searching for information on the Internet, using voice commands to operate technical devices such as smartphones or (somewhat) smart speakers, or when a computer handles requests on telephone hotlines. These interactions can be speech or text-based, and the machine can be an agent engaging in dialogue – e. g. when performing as a digital assistant (Comerford et al. 2001) – or augment an interaction between humans – e. g. by translating between languages (Wahlster 2000) or by listening to a discussion and providing helpful information using different modalities (Milde, Wacker, et al. 2016). This type of interactive augmentation can also take place without a dialogue, when language is only produced and not perceived – e. g.auto-completion, spelling correction, or other suggestions when writing a text.

All interactions described above extend over the time axis: speech is both produced and consumed over time, and text is both read and written in a piece-wise fashion. I did not write this thesis in one step, and my readers have usually only read a few sentences of this thesis up to this point. Nonetheless, a partial understanding of this thesis's content has hopefully already emerged, which will extend until the end of the thesis is reached. Not only does the understanding happen incrementally, but prediction also takes place as well. Readers of scientific literature have an expectation about the forthcoming structure, and I invite you to guess the title of the last chapter. You will probably not be far off. This incrementality of language can be observed on many levels, where parts are composed of smaller parts: words are built from phonemes or morphemes, sentences are built from words, texts are built from sentences. Humans make use of this property in many ways, often without realizing it (see Section 1.1). Incremental processing goes hand in hand with prediction: an incomplete sentence can be completed by another speaker to indicate turn-taking (Lerner 2002), showing that the listener not only tried to understand but also can predict the continuation of the speaker's utterance. For syntactic parsing, processing a determiner includes predicting that a noun will follow to which the determiner will be

attached to,[1] and processing a verb leads to a prediction of what objects this verb takes.

Especially for structured problems such as syntactic and semantic parsing, natural language processing systems often assume that all relevant data is already available before starting to process input and ignore the inherent incrementality. This assumption has consequences for where a non-incremental component may be employed. In interactive scenarios such as human-computer interaction, non-incremental components cannot be used without sacrificing interactivity: spelling corrections would only be shown after a text has been written, an utterance would only start to be translated after it has been completely spoken. Incremental processing exploits the incrementality of language by starting to compute and produce output before all input is available, allowing a system to already act on partial input.

1.1. Psycholinguistic Evidence for Incremental Processing

Humans still pose the gold standard for language processing, especially if the processing does not happen on a large scale but in an interactive setting. When speaking, they perform several tasks incrementally and in parallel, from conceptualization to articulation (Levelt 1989). Machines mimicking human behavior need to be able to perform similar computations as humans do in such settings to pose as a competent partner, and psycholinguistic research gives insight into the processes that humans perform.

Interestingly, psycholinguistic research not only examines whether incremental processing takes place but it is often carried out by timing experiments, i. e. psycholinguistic experiments make use of the incremental processing by humans to gain insights into linguistic processes, mostly by eye-tracking (Tanenhaus et al. 1995; Sturt and Lombardo 2005; Malsburg and Vasishth 2011, inter alia), but also by timing self-paced word-by-word reading of sentences (e. g. Gibson and Warren 2004).

Language is perceived incrementally and this incremental processing is influenced by other modalities even while speech is perceived. Tanenhaus et al. (1995) show that subjects in their experiments used visual information to disambiguate spoken utterances while the utterances were still ongoing and that eye movement was guided by that utterance. Sturt and Lombardo (2005) explored whether humans build structure for sentences incrementally by performing eye-tracking for verb-phrase coordination. They show that participants of the study experienced processing disruptions when a sentence continued differently than expected and that these processing disruptions occurred earlier than can be explained by bottom-up parsing procedures. Their experimental setting consists of sentences where stereotypical male or female nouns such as “pilot” or “nurse” co-occur with reflexive

[1] Try to read the following: I will not continue the

pronouns of the opposite gender. The reflexive pronoun is embedded in such a way that upon reading it, it is not connected to the noun when performing bottom-up parsing and therefore would not result in a disruption if humans only performed bottom-up parsing. Therefore, some kind of connecting structure has to be built while reading the sentences. These findings form the psycholinguistic motivations for researching how to generate connected dependency structures.

While text is usually consumed sequentially, eye-tracking also shows that unexpected continuations of a sentence can lead to re-analysis for which the reader performs different strategies, such as re-reading parts of the sentence, or even restarting from the beginning (Malsburg and Vasishth 2011). These two strategies (reading sequentially and restarting) will re-occur throughout this thesis.

1.2. Differentiation to Other Meanings of Incrementality

The term "incrementality" is used to describe different properties in the literature. In contrast to the meaning described above (and in the remainder of this thesis), sometimes systems that work on a complete input but generate output layer by layer, e.g. shallow to deep syntax, are also called incremental, e.g. by Aït-Mokhtar, Chanod, and Roux (2002).

Anytime algorithms (Dean and Boddy 1988) are incremental in the sense that they iteratively improve their output for a fixed input, given more and more processing time. Anytime algorithms can play an essential role in incremental systems as they allow to perform a trade-off between processing time – i.e., system responsiveness – and quality (Köhn and Menzel 2013). The computation of a result can continue until it is needed, making sure that it is the best that could be computed in a given time without requiring other components to wait for a result. Anytime algorithms optimize the time spent *between* inputs, e. g.while waiting for the continuation of an utterance. In general, the question of whether to use anytime algorithms does not affect the other challenges posed by incremental processing.

1.3. Incremental Processors in Natural Language Processing

In an incremental system, all processors need to work incrementally. A prime example is the Verbmobil project, which set out to develop a portable simultaneous interpreter (Kay, Gawron, and Norvig 1994). Verbmobil was a project spanning eight years and several sub-projects. The Verbmobil system consists of several processors interacting with each other: speech recognition and synthesis components, syntactic and semantic

parsers, self-correction detection, natural language generation, dialogue modeling, and, of course, machine translation. This list shows that incrementality is an aspect that touches many topics of natural language processing. The project also exemplifies that building incremental systems is not easy, even with massive funding[2]: Only one of the many components ended up being incremental,[3] and the final report makes no mention of *simultaneous* interpretation anymore (Wahlster 2000).

As human-computer interaction is often carried out using speech, much research on incremental processing has been carried out on speech-based systems such as dialogue systems. These systems deal with comparatively simple utterances – at least in contrast to newspaper texts, the predominant genre in treebanks – and therefore often get by with a rather superficial treatment of text. The reason for this is two-fold: first, the setting in some experiments invites short and focused utterances, and second, each speech-based system needs to perform automatic speech recognition (ASR). As vocabulary size and sentence complexity correlate with the number of errors speech recognition produces, complex utterances can only be reasonably processed with high-quality speech recognition. For example, Sagae et al. (2009) perform incremental natural language understanding of spoken utterances. Their ASR recognized the utterance *"we are prepared to give you guys generators for electricity downtown"* as *"we up apparently give you guys generators for a letter city don town"*, which is hardly meaning preserving and the NLU quality therefore degrades in comparison to a theoretical gold-standard ASR.

I want to exemplify the limiting impact of ASR on the complexity of incremental dialogue systems using several examples. DeVault, Sagae, and Traum (2009) extend the work by Sagae et al. (2009) to better deal with recognition errors by building a predictor that computes the probability that the NLU component has correctly captured the relevant information from a (still ongoing) sentence. Once the probability reaches a certain threshold, the system tries to complete the utterance. Utterance completion is often used by humans to take a turn in dialogue (Hansen, Novick, and Sutton 1996; Lerner 2002). The relatively simple approach by DeVault, Sagae, and Traum (2009) is based on a maximum entropy classifier. It mainly works because the domain is limited – military personnel talking with village elders in some unnamed country. The restricted domain can be observed in an example: The system e. g. correctly completes the utterance *"I have orders"* to *"I have orders to move you and this clinic"*. This completion is only possible because not only the domain is known and extremely restricted, but also very similar utterances have are in the training data, with little abstraction from specific word forms.

[2] Verbmobil had a funding of 116 million DM (about 60 million €, or 78 million € when adjusted for inflation), with significant additional funding from industry

[3] One of the several machine translation systems developed as part of the project is incremental.

Paetzel, Manuvinakurike, and DeVault (2015) train actor policies for rapid task-based dialogue, where a user describes a target image out of several images. The system has to find the correct image as quickly as possible. The work focuses on the interaction effects, the incremental processing architecture, and policy optimization. The understanding component is again a maximum entropy N-gram based classifier, which is sufficient because the utterances are short descriptions of images. Stoness et al. (2005) experiment with multi-modal interaction between incremental parsing of spoken input and a visual context, which is used to disambiguate the spoken input. This research uses a more in-depth understanding system consisting of a syntactic parser and a knowledge base but imitates the ASR using a manual transcription of the audio signal. Schlangen, Baumann, and Atterer (2009) also used manual transcription in a similar-setup – a visual world combined with ongoing utterances – to examine strategies for incremental reference resolution.

Research into incremental repair detection (Hough and Purver 2014; Honnibal and Johnson 2014), i. e. detecting when a user misspeaks and corrects themselves, is essential for interactive systems because without explicit modeling of repairs a user's intention cannot be modeled correctly. This problem only appears in spoken interaction, as written interaction does not exhibit the same type of correction. But still, both Hough and Purver (2014) and Honnibal and Johnson (2014) perform detection on text only, i. e. both assume that perfect automatic speech recognition has been performed.

These examples are not meant to diminish the research produced. They exemplify that due to the recognition errors by ASR, research has either been performed on real interaction with a system based on utterances with limited coverage, or resorted to simulating the ASR when dealing with more complex interactions.

When using speech-based systems, it is the speech recognition quality that imposes a limit for the linguistic complexity that users can employ. With more advanced recognizers, more complex utterances can be recognized by the system, and more complex processing than e. g. keyword spotting needs to take place. Since the findings discussed above, ASR has drastically improved. However, recognition is still error-prone: Milde and Köhn (2018) show that even with extensive training data and a state of the art model for German, sentences read from Wikipedia articles are only recognized with a word error rate of 14%. On the other hand, recognition of less complex utterances is already possible, with word error rates between 5% and 9%, especially for the best-resourced language, English (Han et al. 2018).

But not all interaction relies on speech: In addition to speech-based systems, text-based interactive systems such as web search or grammatical error detection could (and in some cases already do) make use of incremental processing, e. g. for search-as-you-type or instantaneous error detection.

1.4. Plan of this Dissertation

First, this thesis gives an overview about incremental processing (Chapter 2, then explores how syntax parsing (as an example for structured prediction problems) can be extended to be incremental – i. e. to produce output for incomplete input – by examining the structures that a parser should be able to produce (Chapter 3), how this ability can be evaluated using suitable gold standards (Chapter 4) The next two chapters introduce different algorithms to produce such structures: Restart-incremental (Chapter 5) and transition-based (Chapter 6). The merit of creating such structures is shown in Chapter 7, where language models are augmented with information from incremental structures. Chapter 8 concludes the thesis.

Chapter 2.

Incremental Processing in NLP

Incrementality is not only of interest for syntactic parsers but also for other processors of natural language, which have different characteristics. To gain a better understanding of how predictive parsing fits into this landscape, I will give an overview of such processors. Even though these may perform completely different tasks, the strategies to "incrementalize" them can be quite similar. I will first lay out a typology for incremental processors using properties based on the data the processors consume and produce, as well as the relation between the input and output data. Then, I will describe how speech recognition and machine translation, as relatively well-researched incremental processors, fall into these categories and which algorithms and strategies are employed to optimize and measure their incremental behavior.

2.1. A Typology for Incremental Problems

An incremental processor can be characterized by the kind of data it consumes as input, as well as the kind of data it produces. It also exhibits a specific incremental behavior by posing requirements on subsequent inputs it consumes and giving guarantees about the relations between subsequent outputs it creates. I will discuss these properties in the next section and show how different NLP processors align with them in Section 2.2. Of course, I am neither the only nor the first person to develop a categorization for incremental processors; Chapter 5 of Guhe (2007) discusses properties mostly complementary to the ones discussed here.

2.1.1. Data Types for Input and Output

Data can be *structured* (e. g. syntactic or semantic structures) or *sequential* (e. g. text, label sequences, or audio). Sequential data can be *discrete* (e. g. words) or *continuous* (e. g. speech signals) along the time axis. Structured data is always discrete on the time axis because as long as the structure is finite, it can not be meaningfully subdivided into

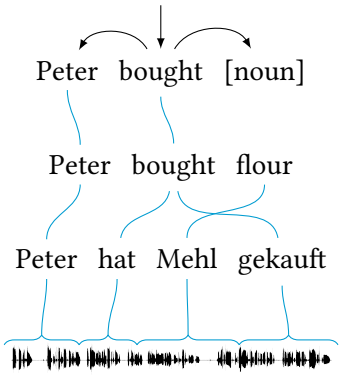
Processor	Data	**Data** / *alignment* properties
		
	Peter bought [noun]	**structured, including prediction**
Parsing		*vertices partially, edges not grounded*
	Peter bought flour	**discrete, sequential**
MT		*reordering, fuzzy mapping*
	Peter hat Mehl gekauft	**discrete, sequential**
ASR		*order-preserving, clear mapping*
		continuous

Figure 2.1.: Different types of data, groundings, and processors. Note that for parsing only the sentence prefix "Peter bought" is processed to exemplify intermediate structure generated for incomplete input. Incremental systems can take many forms, and this example aims to illustrate the common principles of incremtal processing, not a specific task. ASR: Automatic Speech Recognition; MT: Machine Translation

infinitely many parts along the time axis. Data that is discrete along the time axis can, of course, include continuous data such as word embeddings.

A processor can take one type of data as an input and create another one as output; some examples are depicted in Figure 2.1, and I will come back to the properties depicted in the figure throughout this section. Sequential data can usually be subdivided along a time axis, i. e. there exists a total ordering between the elements of the input (or output respectively), see the input and output for the ASR and MT processor in Figure 2.1. On the input side, this tells us in which order the input becomes consumable. On the output side, it tells us in which order (and possibly also at which point in time) the output should be generated. Structured data does not necessarily exhibit this property: Given, for example, the dependency tree produced by parsing in Figure 2.1, the words are ordered, but the dependencies between the words cannot be uniquely attributed to a word: they could be attributed to the head or the dependent.

2.1.2. The Processing Granularity

The *granularity* a processor uses describes the size into which the input and output is subdivided; it is the (temporal) size of the basic units of incremental data. In contrast to the previously discussed properties, granularity can not be deduced from data itself: the same audio stream might be processed in chunks of 10ms or 2s, depending on the processor. Granularity can be either used to describe how a processor consumes and produces data or to describe a sequence of subsequent incremental data (which is, however, usually produced by a processor). A processor can only work incrementally if the input it has to process is composed of several basic units. When considering a larger incremental system, a processor usually seen as non-incremental might be *incremental enough*: a processor is incremental enough if the size of the units it consumes and produces is not larger than required by the overall system (i. e. due to design criteria). Assuming a coarse enough desired granularity, every system can be seen as incremental enough: A normal syntax parser clearly works non-incremental if parsing a single sentence, but incrementally if it is processing a paragraph and the basic units are sentences. A grapheme to phoneme transformation usually converts input word by word[1] and is incremental enough for a language generation system that works on the level of words and does not need the grapheme to phoneme conversion to be able to process sub-word input. On the other hand, if input typed by a user should be vocalized, sub-word granularity might be needed.

The granularity of a pipeline is determined by its most coarse-grained component. The granularity needed can vary by use case. In general, fine-grained processing is harder

[1] This of course depends on the language as e. g. for French the phonemization depends on the surrounding words – for example, the s in "les" is only spoken if the next word starts with a vowel.

than coarse-grained processing; a system can always process data fine-grained internally while having coarse-grained interfaces, whereas the opposite is not possible.

2.1.3. Grounding

Grounding describes the alignment from the generated output to the elements of the input that yielded evidence for this output (Schlangen and Skantze 2009). Grounding allows to reason about which part of the output can be reasonably generated given only partial input. In some cases, this alignment is explicit in both test and training data, e. g. in sequence labeling tasks where each element of the input is assigned a label. In other cases, such as machine translation, there is no gold-standard word alignment,[2] and even a human-generated alignment would not create a one-to-one mapping between input and output. Especially for structured output, some parts may lack grounding altogether, such as the edges and the prediction in the dependency tree in Figure 2.1. Also, the alignments may or may not be *order-preserving*: A tagging task preserves the ordering, whereas in translation, reordering takes place (cmp. "hat Mehl gekauft" → "bought flour" in Figure 2.1).

2.1.4. Monotonicity

A system is *non-monotonic* if it is allowed to retract output it has previously produced. For example, an incremental sequence labeler that is free to re-assign labels can change its mind about every element of a sentence once the sentence is complete. A *monotonic system*, on the other hand, is required to only extend previously generated output without retracting information. Some components are inherently monotonic as their output is not fed to another processor of the system, but to the outside world. E. g., a speech synthesizer is inherently monotonic as it cannot retract sound waves realized through a loudspeaker.

Monotonicity limits the quality a component can produce as it can not revert a decision that turns out to be wrong later on in light of additional available input. In contrast, a non-monotonic component can always achieve the same non-incremental output as a non-incremental component by merely replacing all intermediate output with the output of the non-incremental component once all input is available.

The precise meaning of monotonicity needs to be defined for each component. For sequential output, the most common definition is to only allow appending to the output. For structured output, the structure of an increment could be required to be a super-set of its predecessor.

[2] At least not on the word level, but alignments can be generated automatically, see e. g. Och and Ney (2003)

Non-monotonic output can only be generated sensibly if the consumers of the output can deal with non-monotonic input. Otherwise, these consumers might ignore the revisions made to previous output and end up with an inconsistent input or have to restart their computation in light of new input.

2.1.5. Timeliness

Given a specific input, each NLP processor has to optimize *what* output to produce. An incremental system also needs to decide *when* to provide output. While discrete input provides specific anchors for this decision, continuous input does not, and new output can be generated continuously.[3] Such decisions also need to be made by human interpreters while performing simultaneous translation; they need to buffer input until they can produce additional output. The characteristics of this (human) process vary by language, e. g. the delay is relatively long when translating from German to English because the verb in the input tends to occur later than it needs to be produced for the English target sentence (Goldman-Eisler 1972).

Processors may have a fixed or variable delay, which for some processors can be prescribed and for others can only be observed. With all else being equal, a more timely processor is usually preferable.

2.1.6. Trade-off Between Properties

Incremental components have to make a trade-off between timeliness (i. e. the amount of delay introduced between input and output), output quality, and the amount of non-monotonicity (Beuck, Köhn, and Menzel 2011a; Baumann, Atterer, and Schlangen 2009). High-quality, monotonic output can be obtained by delaying output. Taking this strategy to the extreme results in a non-incremental system that produces the complete output at once after all input becomes available. Allowing non-monotonicity via output revisions reduces the delay, as well as compromises concerning the quality of the output. Gradual trade-offs can also be made: allowing infrequent revisions and mild delays can lessen the negative impact on accuracy. These trade-offs are universal to all incremental processors; examples for performing such trade-offs will be shown in the next section.

[3] While continuous input is made discrete before reaching a processor, the discretization is usually in the order of milliseconds and can be seen as continuous for all practical purposes.

2.2. Incremental Processors

This section discusses two tasks with existing incremental processors to exemplify strategies to "incrementalize" a processor: The first, incremental speech recognition, is a continuous sequence labeling problem usually implemented as a non-monotonic processor. The second one is incremental machine translation, which is a sequence to sequence task with reordering, usually implemented monotonically. Parsing, the task this thesis is about, is a sequence-to-structure task, which can be implemented both monotonically and non-monotonically.

2.2.1. Speech Recognition

Speech recognition lends itself to incremental processing because the decoding happens incrementally even for non-incremental use-cases. Speech recognizers such as Sphinx 4 (Walker et al. 2004) use the token passing algorithm (Young, Russell, and Thornton 1989), which keeps a set of currently possible states (the tokens) at each point in time and moves these tokens forward through the search space as more audio input becomes available, i. e.every few milliseconds. It is, therefore, possible to look into a speech recognizer to obtain the most probable hypothesis at each point in time and use that as output without modifying the recognizer. As the speech recognizer is still the same as the one with a non-incremental interface, the only optimization point is when to let new output through, i. e. to implement a *gatekeeper*. Without a gatekeeper, the output would be too frequent, forcing subsequent processors either to perform gatekeeping themselves or to process extremely high amounts of data.

The gatekeeper's policy can be guided by observing the time that a hypothesis survived as the most probable one without being discarded (Baumann, Atterer, and Schlangen 2009) and producing output once the most probable hypothesis has been stable longer than a fixed threshold. Alternatively, it can be based on the internal state of the recognizer (Selfridge et al. 2011). McGraw and Gruenstein (2012) show that even sophisticated stability estimation based on the internal data only slightly improves upon the simple age-based estimation proposed by Baumann, Atterer, and Schlangen (2009). Given that the speech recognizer is the same for the incremental case as for the non-incremental one, there is no trade-off against accuracy. Some states of the decoder can also be used to trigger a new output reliably: Selfridge et al. (2011) noted that if during decoding, all beams in the beam search pass through the same state, the prefix up to that state can be declared stable because future decoding will not change the most probable path up to that state. This observation essentially makes use of the Markov property and is primarily useful when performing grammar-based recognition.

2.2.2. Machine Translation

Machine translation is the second topic for which incrementality has been studied. While it can be performed using batch processing (e. g.for websites), it has obvious use cases in interactive systems, e. g.for simultaneous translation for speeches or dialogues. To successfully facilitate human-human interaction, it needs to work incrementally.

Modern approaches to machine translation, i. e. neural machine translation, employ a sequence to sequence model where the input sequence is encoded into a representation and then decoded again. One approach uses recurrent networks, which represent the input as a single fixed-length vector and optionally use attention to the input when generating the output sequence (Bahdanau, Cho, and Bengio 2014). It is also possible to not employ a recurrent network, using only attention to model the influences of the input sequence to the generation of the output sequence (Vaswani et al. 2017). In all these cases, the complete input is consumed before translation happens, i. e. the translation is inherently non-incremental.

Incremental Neural Machine Translation (NMT) by Using a Gatekeeper

Machine translation is a task with sequential input and output where the output ordering does not conform to the input ordering, and the ground truth for the grounding is often missing. As the incremental machine translation systems found in the literature are monotonic, the systems need to decide at which point they have enough information from the input to generate the next output token with high confidence. That is, they also need to implement a gatekeeper, but in contrast to the gatekeeper used for speech recognition, it will influence the final result due to the processor being monotonic.

Gu et al. (2017) propose a system where an NMT processor repeatedly proposes an output token based on the currently available input and the already generated output to a gatekeeper. The gatekeeper either accepts this output, resulting in a write operation to the output, or rejects it, resulting in a read operation on the input. The underlying NMT model is trained on complete sentences and therefore not adapted to incremental processing. The gatekeeper can use different policies, yielding different trade-offs between timeliness and quality. Rejecting all output until the input is complete means falling back to a non-incremental behavior; performing alternating read and write actions eliminates delay but results in bad translations. Gu et al. (2017) train the policy using reinforcement learning with the reward based on the resulting BLEU score and the delay incurred; weighting them differently results in different trade-offs. Humans have different preferences regarding this trade-off, depending on whether the output is speech or subtitles (Mieno et al. 2015). An example of an incremental translation can be seen in Figure 2.2, where the translation

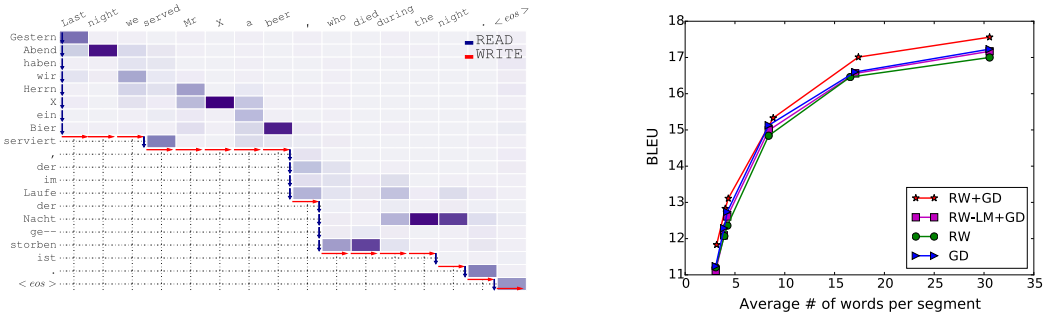


Figure 2.2.: Left: Incremental NMT with attention, from Gu et al. (2017). Y-axis: input, X-axis: output. Blue arrows: read operations, red arrows: write operations. Colors denote the attention given to each input token when generating an output token. Note the delay induced by the reordering for "serviert" (*served*) and "gestorben" (*died*). Right: visualizing trade-off decisions between delay (x-axis, measured in words translated at once) and accuracy (y-axis, measured in BLEU), from He, Grissom II, et al. (2015).

process is depicted using arrows for read and write operations. The system first reads the input up to "Bier" (beer), then generates three tokens, reads "serviert" (served), generates the verb "served", and so on.

Reordering Output

Reordering phenomena constitute a significant hindrance to timely translation, as exemplified in Figure 2.2. Grissom II et al. (2014) deal with this by training a classifier to predict verbs needed for the output but not yet seen in the input, allowing the MT system to produce a verb without having seen its counterpart on the input. The MT system is enhanced by a transition system (similar to the gatekeeper in Gu et al. (2017)), which reads more words, predicts a verb, predicts the next word, or commits to the currently generated translation. Again, the underlying MT system is unchanged, and a policy for the actions is trained using reinforcement learning.

Instead of predicting the verb, He, Grissom II, et al. (2015) propose to edit the gold-standard translations to better fit the source ordering for training the MT system by trying to imitate the transformations a human translator would perform. This approach tackles the problem that the training data available is only composed of translations that

were performed non-incrementally, which is not optimal for simultaneous translations. Human simultaneous translators produce sentences that systematically deviate from the "normal" target language, but such data is not readily available for training (He, Boyd-Graber, and Daumé III 2016). The training data is adapted by generating phrase-structure trees for the target sentences and applying (manually written) syntax-based reordering rules. The system then checks whether the reordering has reduced the delay based on an automatically computed alignment and, if so, uses the reordered version instead of the original one. As in the approach by Gu et al. (2017), the average delay induced by the system can be tuned, see Figure 2.2. Because the processor was not trained on gold-standard data, it might yield sub-optimal results on gold-standard test data. He, Grissom II, et al. (2015) evaluate their system on both gold-standard and transformed data and show that it, in fact, performs better on both targets.

2.2.3. Natural Language Generation

In contrast to the processors already discussed (speech recognition and machine translation) as well as the processor this thesis is about (parsing), natural language generation is often not trained but modeled by hand. This allows changing the processor in ways not easily achievable in other processors. Natural language generation often sits near the end of a pipeline as a means to make information available to the user. It, therefore, has to deal with all delays induced by previous processors.

Skantze and Hjalmarsson (2013) compare a non-incremental dialogue system and an incremental one, with which language learners interact to buy items at a flea market. The speech recognition component was simulated, i. e. a human manually transcribed the speech. As the manual transcription takes time, the system response is noticeably delayed in a non-incremental system where the dialogue system only starts to plan its response once the transcription is complete. In contrast, the incremental system constructs a response as soon as possible, based on partial input. The response is recomputed on changed input, which can have three effects: 1) If the update is consistent with what has been said already, the continuation of what to say is changed (a *covert change*). 2) If the system has to retract information already uttered, an explicit repair has to be produced (an *overt change*). 3) Fillers are inserted to avoid silence when information to produce a complete utterance is currently missing. The system is faster (as it starts earlier) and preferred by users, although it has to correct itself explicitly, resulting in longer responses. Despite its output being monotonic – as it cannot erase information from the hearer's ears – its explicit repairs enable the system to act with low delay (a similar strategy to the one employed by human speakers (Levelt 1989, Chapter 12)). As the natural language generation component is rule-based, it is not constrained by the (non-)availability of data

↓	non-monotonic (1)	non-monotonic (2)	delayed (3)	erroneous (4)
input: a	a/y	a/y		a/y
b	a/x b/y	a/y b/y	a/x b/y	a/y b/y
c	a/x b/y c/z	a/x b/y c/z	a/x b/y c/z	a/y b/y c/z
inc_acc(i)	2: 1/1; 1: 2/2; 0: 2/3	2: 1/1; 1: 1/2 ; 0: 2/3	2: 1/1; 1: 2/2 ; 0: 2/3	2: 0/1; 1: 1/2 ; 0: 2/3
EO	1/4	1/2	0	0
accuracy	3/3	3/3	3/3	2/3

Table 2.1.: Examples for characteristics of incremental output that need to be captured for evaluation, using a sequence labeling task. Correct output: a/x b/y c/z. (1), (2): output for *a* changed; (3): output for *a* held back until input "b" is available; (4): incorrect assignment to "a" stays in output. inc_acc: incremental accuracy, see Section 2.3.2; EO: edit overhead, see Section 2.3.3; accuracy: ratio of correct labels in the complete output. Dashed line exemplifies which output is used to compute inc_acc.

suitable for learning incremental language generation.

2.3. Evaluating Incremental Systems

An incremental system can be evaluated just like a non-incremental one. Evaluation schemata exist for all established tasks such as speech recognition (quality measured in word error rate), PoS tagging (measured in accuracy), phrase structure parsing (measured in precision/recall), or machine translation (e. g. BLEU). Additional evaluation techniques allow examining the behavior more closely, such as compiling error confusion matrices. These schemata enable a comparison between components in a standardized way. As evaluating (incremental) dependency parsers is discussed in detail in Chapter 4, this section focuses on the evaluation of other systems.

The drawback of using non-incremental evaluation measures is the lack of insight into incremental properties. When building incremental systems, not only the final output but also the intermediate behavior is of importance; using evaluations tailored to non-incremental systems fails to give insight into their incremental properties.

Table 2.1 shows abstract input/output patterns for a sequence labeling task, where the correspondence between input and output is given, and no reordering effects take place. A

standard accuracy-based evaluation on the complete output would yield a perfect score for the first three systems, although their behavior over time is quite different: In addition to the non-incremental evaluation for the complete output, the non-monotonicity in (1) and (2), as well as the delay in (3), need to be considered. It is impossible to express all these differences with a single number. I will, therefore, discuss specific metrics for timeliness, monotonicity, and quality.

2.3.1. Measuring Timeliness

Cho and Esipova (2016) propose to measure timeliness by counting for each output element t of an output sequence Y how many input elements from the input sequence X have been consumed before its production ($s(t)$). $\tau(X, Y)$ then computes the translation delay:

$$0 < \tau(X, Y) = \frac{1}{|X||Y|} \sum_{t=1}^{|Y|} s(t) \leq 1$$

This computation is helpful when there is no gold-standard alignment between output and input that can be used to obtain the output timing that could have been achieved under optimal conditions. $\tau = 0$ means all output was made without consuming input, $\tau = 1$ means all input was read before generating output.

Grissom II et al. (2014) introduce latency-BLEU, a metric that averages the BLEU scores of the outputs corresponding to each input prefix. The complete translation is weighed higher than all other partial translations to penalize incorrect translation. If the resulting translation is the same for two processors, the processor with less delay than the other will obtain a higher score. Due to the averaging, the sentence-initial output has more influence on the score than output created near the end of a sequence. It is also not possible to completely distinguish between the quality and the timeliness because quality and timeliness are measured in a single metric. In the MT system by He, Grissom II, et al. (2015), the timeliness can be (indirectly) tuned by adjusting a threshold at which to translate all yet untranslated words. Figure 2.2 (right) shows a plot of the resulting BLEU score against the average number of words translated at once, i. e. the delay. This way, potential users can see the trade-offs that can be made using a system (RW+GD is the proposed architecture, beating the other approaches at each trade-off point).

If an explicit alignment exists between the input and the output – such as in speech recognition – the difference between when a specific output is made (i. e. which amount of input data has been consumed) and the timing of the corresponding input can be measured to obtain an anchored timeliness measure (Baumann, Buß, and Schlangen 2011). Both the

relation to the first occurrence of an output (FO) and the relation to the last change of an output (final decision, FD) can be measured. E. g. if a word ends at 2.5 seconds of the input audio, was first recognized after consuming 3 seconds and was part of all outputs produced after consuming 4 seconds, its FO would be 0.5 seconds, and its FD would be 1.5 seconds. To separate timeliness from quality, these measures can be computed against the final output of the system instead of the gold standard. However, then a reliable alignment between the input and the generated output is needed. The advantage over other methods is its interpretability: A FO of 100ms for a speech recognizer means that it produces, on average, an output 100ms after it has consumed input that carries evidence for this output. FO and FD only differ for non-monotonic processors; FD measures the average delay that is necessary to obtain a reliable output from the processor.

2.3.2. Measuring Incremental Quality

When dealing with monotonic output, the incremental quality can be assessed using the non-incremental quality metrics as the processor cannot revoke previously made output. If a processor can be tuned to provide more or less timely output, the quality can be plotted against delay, as in Figure 2.2.

If a system is non-monotonic, it makes sense to measure the accuracy not only based on the final output but also based on the intermediate output. Averaging the quality for each increment has two disadvantages: Output for early input is weighed more than that for later input, and it is unclear how non-monotonicity affects the quality.

If the input for a processor is a discrete sequence and for each input token a quality measure based on the output produced can be computed, a quality measure with respect to the age of the input can be used (Beuck, Köhn, and Menzel 2011b): The age can be computed with regards to the *frontier:*

Definition 1. The **frontier** of a sequence is its newest element. For a sentence prefix containing n words, the frontier is the nth word.

For every element of each input, we can compute whether the corresponding output is correct. Then, the accuracy for the *n*-th element to the left of the frontier in every generated output is computed, with n starting at zero (measuring the accuracy on the newest elements for all outputs) up to some predefined number. This way, an accuracy curve relative to the age of the input is generated. Table 2.1 contains incremental accuracy (*inc_acc*) measures relative to the age of the input; examples (1) and (2) yield different incremental accuracies even though their non-incremental accuracy is the same. This measurement only makes sense if the processors' output is non-monotonic as otherwise,

the incremental accuracies would all be the same.This approach will also be used to evaluate the predictive dependency parsers in this thesis (see Chapter 4).

2.3.3. Measuring the Degree of Non-monotonicity

Evaluating non-monotonicity can be viewed from two (similar) standpoints: first, how much intermediate output will later be retracted again? Second: how sure can we be that a particular output is reliable, i. e. will also be part of the final output of a processor?

Baumann, Atterer, and Schlangen (2009) and Baumann (2013) tackle the first question by defining the *edit overhead* generated by a non-monotonic processor producing sequential output. They define three edit operations on an output sequence: *add* (append an element to the output), *revoke* (remove the last element from the output), and *substitute* (revoke and then append). The difference $\mathit{diff}(o_i, o_j)$ between two outputs o_i and o_j is the minimal number of edits needed to change o_i into o_j. Note that $2(n-1)+1$ operations are needed to change the first element of an output of length n, whereas changing the last element yields a difference of only one. This notation allows us to compute the edit overhead: We compute the number of edits necessary to obtain the final output $N_{optimal}$ and the number of edit operations actually performed N_{actual} as follows:

$$N_{optimal} = |\mathit{diff}(o_0, o_{tmax})| \tag{2.1}$$

$$N_{actual} = \sum_{t=1}^{tmax} \mathit{diff}(o_{t-1}, o_t) \tag{2.2}$$

The edit overhead is the proportion of unnecessary edits produced by the processor:

$$EO = (N_{actual} - N_{optimal})/N_{actual} \tag{2.3}$$

Table 2.1 shows that EO can distinguish between different levels of non-monotonicity. The edit operations can be adapted to the problem at hand, e. g. if structured output is produced or edits at the start of the sequence should not be penalized more than edits at the end.

2.4. Combining Multiple Incremental Processors

An NLP system usually consists of several processors. In a non-incremental system, all processors can form a pipeline with each processor working on the output of the previous one. Building such a pipeline is non-trivial in an incremental system because, for a given

input, each processor may produce several outputs, which may even be contradictory due to non-monotonicity. Therefore, a system either needs to use *restart-incrementality* (i. e.completely restarting the computation on each new input without relying on previous computations) throughout the pipeline or track changes to perform partial recomputation. Wirén (1992, ch. 5) introduces the notion of dependencies to track which parts of the input a particular output is based on in a chart parser. Only parts of the chart need to be recomputed given a non-monotonic change because we know which chart items are affected by this change. Schlangen and Skantze (2009) extend this notion to a general computation model in which multiple processors work on data organized in incremental units (IU). An IU stands for a minimal unit of information, such as a recognized word. IUs are grounded in other IUs from a previous level and linked to other IUs on the same level, e. g. to describe a sequential relationship. Processors create new IUs based on their input and may revoke IUs already generated. A processor can commit to an IU to signify that if will never revoke this IU so that other processors can rely on it.

2.5. Summary and Discussion

Building incremental processors poses problems that are similar regardless of the specific task. Decisions have to be made regarding the acceptable amount of delay, whether non-monotonic output is acceptable for downstream processors, and if so, to what extent. For all these decisions, the relation between input and output is essential: Is there a precise mapping between input and output, maybe even a one-to-one mapping, and does reordering happen? Several techniques have been discussed for implementing incremental processors: Training a *gatekeeper* to either perform a trade-off between non-monotonicity and delay (Section 2.2.1) or – for monotonic processors – between delay and quality (Section 2.2.2). In Chapter 5, two additional strategies will be discussed in depth: *Beam search*[4] to create non-monotonic output with monotonic extension, and *restart incrementality* for unrestricted non-monotonicity. To evaluate an incremental system, ideally, three characteristics should be measured: timeliness (Section 2.3.1), quality (Section 2.3.2), and non-monotonicity (Section 2.3.3). If a system can be tuned with respect to these characteristics, different trade-off points between these properties can be measured (Section 2.1.6).

When combining several incremental processors into a larger one, the strategies of the processors have to be compatible. With the rise of end-to-end neural NLP systems, one might assume that combination becomes unnecessary. However, neural systems have shown to actually benefit from structured features provided by other NLP systems. For example, neural machine translation can profit from syntactic structure generated by

[4] A variation of the token-pass algorithm used by most speech recognizers

a parser, both for dependency structures (Zhang, Li, et al. 2019) and phrase structures (Saunders et al. 2018); this benefit is more pronounced with limited training data (Currey and Heafield 2019).

The likelihood of an output being stable could be attached to the output to bridge the gap between certainty (monotonic output) and uncertainty (non-monotonic output); see Selfridge et al. (2011) Section 5. Modern NLP processors heavily rely on sub-symbolic representation and use attention mechanisms to obtain relevant information. With this approach, an explicit grounding for partial recomputation as discussed in Section 2.4 – where each part of the output only depends on a specific part of the input – does not work anymore and would need to be replaced with some notion of soft grounding.

Chapter 3.

Representations for Incremental Syntactic Structure

Syntactic structure can be represented using different formalisms, which can be classified into two groups: dependency structures and phrase structures. In a dependency structure, used e. g. by the Prague Dependency Treebank (Hajič, Hladká, and Pajas 2001), the Universal Dependency Treebanks (McDonald, Nivre, et al. 2013) and the Hamburg Dependency Treebank (Foth, Köhn, et al. 2014), the words of a sentence are the nodes of a tree with directed edges, the edges denote syntactic relations between them and carry a dependency label to signify the type of relation. Phrase structure recursively divides a sentence into typed phrases and sub-phrases, the most prominent example for a treebank using this annotation scheme is the Penn Treebank (Marcus et al. 1994). Both types of annotations are used to represent syntax. Phrase structure annotation can only represent projective structures directly because the phrases have to be nested. In contrast, dependency annotations have no such restriction and can encode arbitrary tree structures over sentences without a need to extend the annotation format. Examples of dependency annotations and phrase-structure annotations are shown in Figure 3.1.

For both representation formats, it is not immediately clear how to annotate a sentence prefix; this chapter discusses different possible approaches. As the structure and the algorithm for its creation are often intertwined, algorithms for parsing with phrase structure will also be briefly addressed in this chapter, while the incremental syntax generation of dependency structures will be discussed in Chapter 5 and Chapter 6 in more detail.

3.1. Incremental Phrase Structure Annotations

If only a part of a sentence is available, its phrases can be divided into three categories: The ones wholly contained in the prefix, the ones entirely outside the prefix, and the ones partially inside and partially outside the prefix. The approaches to incremental phrase structure parsing differ concerning which of these classes can be part of the output when

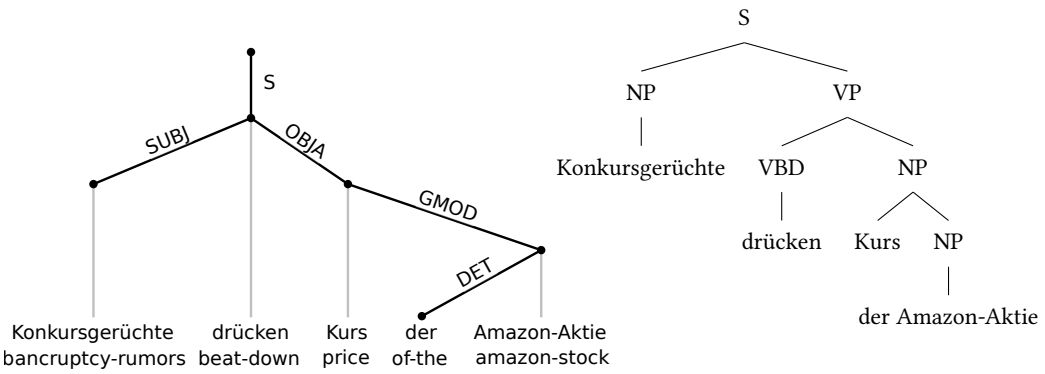


Figure 3.1.: A dependency structure from the Hamburg Dependency Treebank and a corresponding phrase structure.

parsing a sentence prefix.

The *conservative* approach is to only include complete phrases in the incremental syntactic structure. It leads to disconnected structures, as shown in Figure 3.2: incomplete phrases containing phrases that reside entirely inside the prefix are not generated, leaving their already complete sub-phrases disconnected to each other. This approach is closely tied to bottom-up parsing of phrase structure and does not need to add specific constructs for incremental structure as it does not attempt to predict any possible continuations of the sentence.

The *connecting* approach adds all incomplete phrases that are needed to obtain a connected structure. As the incomplete phrases contain either words or other phrases not in the prefix, filler phrases denoting the expectations have to be used as stand-ins for these words and phrases, or they have to be left out. In this case, phrases in the output are not guaranteed to be complete.

One can go one step further and include empty phrases into the structure that denote the expectation of upcoming words filling these phrases. For example, a transitive verb might lead to the prediction of a noun phrase filling the object role. Parsers producing such structures are *predictive*.

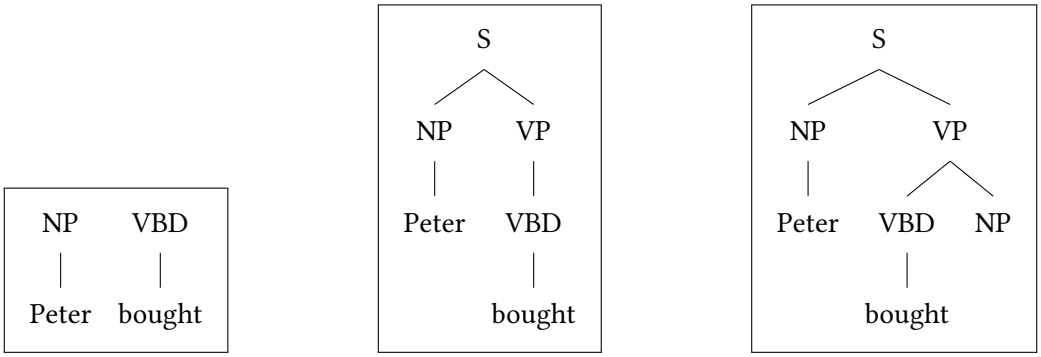


Figure 3.2.: Different approaches to parse the sentence prefix "Peter bought": *Conservative* (left) produces a disconnected structure, *connecting* (middle) also creates the phrases up to the S (*sentence*) phrase, *predictive* (right) adds an expected – currently empty – NP.

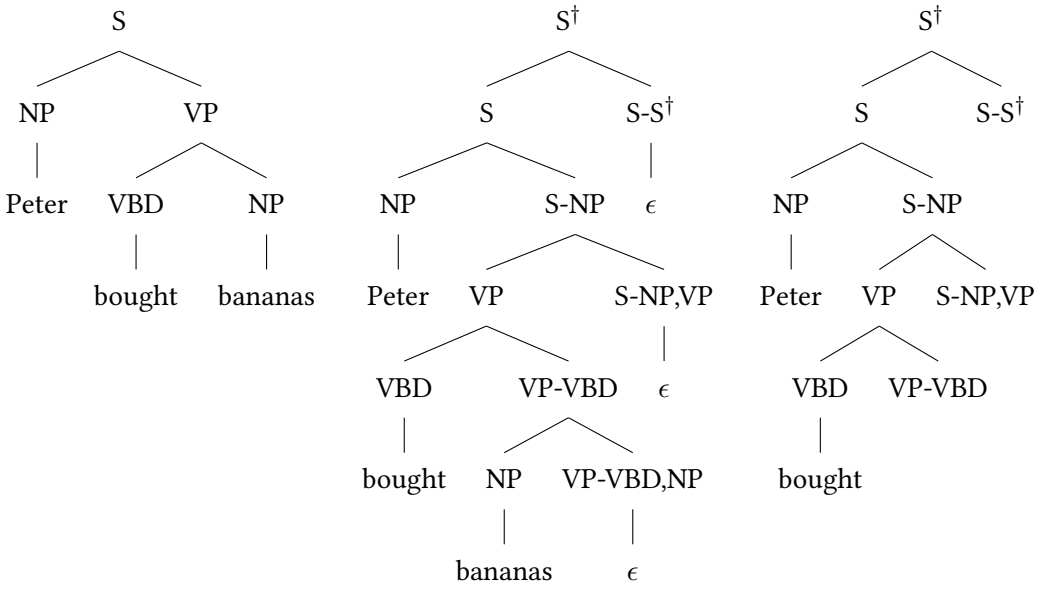


Figure 3.3.: A phrase structure (left) and its binarized counterpart (middle), as used by the Roark parser. Right: Derivation for the prefix "Peter bought".

3.1.1. Incremental Parsing with Probabilistic Context-Free Grammars

One of the parsers creating connected structures is the top-down parser by Roark (2001) based on probabilistic context-free grammars (PCFGs). When parsing with PCFGs, the phrase structure tree is created from the derivation process – a rule derivation creates subordinate structures. Roark's parser works on binary trees (i. e., all nodes have either exactly two children or a single terminal or ϵ as a child). To achieve this, the grammar is right-binarized (Roark and Johnson 1999).[1] This procedure splits longer rules into binary ones with the first non-terminal being the next element to be matched and the second one recursively representing the rest of all rules having that same prefix. Using the example from Roark and Johnson (1999), the rule NP $\rightarrow$ DT JJ NN would be decomposed into four rules:

$$\text{NP} \rightarrow \text{DT NP-DT} \tag{3.1}$$
$$\text{NP-DT} \rightarrow \text{JJ NP-DT,JJ} \tag{3.2}$$
$$\text{NP-DT,JJ} \rightarrow \text{NN NP-DT,JJ,NN} \tag{3.3}$$
$$\text{NP-DT,JJ,NN} \rightarrow \epsilon \tag{3.4}$$

NP-DT,JJ can then be seen as a partially filled NP, which already has a DT and a JJ sub-phrase. All original rules sharing the same prefix on their derivation side now share the same binarized rules; therefore, the parser can decide at a later point in time which of these rules to choose. For example, NP $\rightarrow$ DT JJ NN and NP $\rightarrow$ DT NN share the same prefix. However, without binarization, the parser would already need to decide on whether an adjective follows when consuming the determiner. With binarized rules, this decision can be postponed.

Figure 3.3 shows an example of output for a complete sentence and a sentence prefix. The binarized and non-binarized trees are equivalent, and one can be reconstructed from the other. Each phrase is open for additional content until it is closed by an ϵ transition. Even though the incremental output looks like a predictive one because of the nodes S-S†, S-NP, and VP VP-VBD, it is actually not. These nodes are not representing a phrase but merely enable the parser to potentially add more material to an already existing phrase (S†, S, and VP, respectively). These nodes can be deleted at any time by replacing them with ϵ to denote that they contain no material at all – de-binarizing the tree deletes ϵ nodes. Therefore, the parser does not predict; it only keeps all possibilities open. For the prefix

[1] A binarized grammar may look similar to a Chomsky Normal Form, but in contrast to a CNF, the binarized grammar may have rules with ϵ on the right-hand side when the left-hand side is not the start symbol.

tree in Figure 3.3, the following rules are used:

$$S^{\dagger} \rightarrow S \quad S\text{-}S^{\dagger} \tag{3.5}$$

$$S \rightarrow NP \quad S\text{-}NP \tag{3.6}$$

$$NP \rightarrow Peter \tag{3.7}$$

$$S\text{-}NP \rightarrow VP \quad S\text{-}NP,VP \tag{3.8}$$

$$VP \rightarrow VBD \quad VP\text{-}VBD \tag{3.9}$$

$$VBD \rightarrow bought \tag{3.10}$$

3.1.2. Tree-Adjoining Grammar Approaches

Tree-Adjoining Grammar (TAG) is a formalism introduced by Joshi, Levy, and Takahashi (1975).[2] The previously described parser used a context-free grammar, and the syntax tree is encoded in the derivation tree obtained by recording the rule applications. In contrast, tree-adjoining grammars are context-sensitive, and the grammar explicitly creates the resulting phrase structure tree.[3] Rules in TAG are trees that either have nodes marked for a later *substitution* or they have a node of the same type as their root node and are used for *adjoining*. Figure 3.4 shows an example of adjoining. From an incremental perspective, the nodes marked for substitution are predictions of upcoming structures – phrases in the tree that may not yet be grounded in the prefix.

Tree-adjoining grammars are not necessarily suitable for incremental processing in their original form. Rules can require an out-of-order application in which a word later in the sentence has to be included into a structure before an earlier word can be included – Figure 3.4 shows such an example. The sentence is constructed by first selecting a partial tree for "loved", and the word "has" can only be included into the phrase structure after "loved" because the adjoin operation needs to work on the VP introduced by the partial tree from "loved". The processing ordering is different from a top-down application of a PCFG where the ordering of rule applications is free due to the context-free nature of the formalism. Note, however, that in contrast to the PCFG approaches, the partial result encodes an explicit prediction of an upcoming noun phrase, denoted by the $NP_0 \downarrow$, which is to be substituted by a noun phrase later on.

TAG can be extended to perform incremental processing in different ways. Shen and Joshi (2005) use a variant of TAG – LTAG-spinal – that allows the system to attach treelets

[2] Joshi and Schabes (1997) provide a more recent overview, which was also used for the following description of TAG.

[3] Koller and Kuhlmann (2011) describe a framework with which CFGs can be converted to a formalism that also explicitly creates the tree structure.

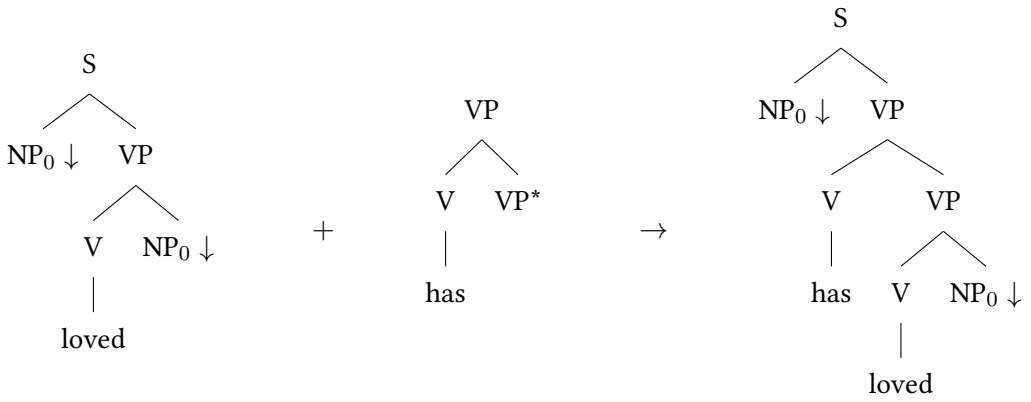


Figure 3.4.: Adjoining operation for TAG. The structure for "has" is adjoined to the structure for "loved", extending the VP. Adapted from Joshi and Schabes (1997).

to a phrase without the need for a substitution node and employ a shift-reduce algorithm over the derivation possibilities. As the shift-reduce algorithm is inherently working on the input from left to right, it is incremental but does not generate connected intermediate structures. Some approaches do produce connected intermediate structures based on TAG: The DVTAG formalism (short for Dynamic Version of TAG) (Mazzei, Lombardo, and Sturt 2007) uses an extended operation schema from plain TAG to enable post-hoc modifications of partial trees. This formalism can model a range of linguistic phenomena (such as adjoining adjectives to a predicted noun). Because DVTAG produces structure earlier in the derivation process than LTAG, its coverage is lower – LTAG can generate 93.5% of the sentence structures in the Penn Treebank test set, DVTAG can only generate 87.1% (Mazzei and Lombardo 2007). To my knowledge, the coverage and accuracy of DVTAG has not been evaluated using a parsing system, where the coverage might be significantly lower than the theoretical optimum due to search errors.

An approach for generating connected structures with TAG that was empirically evaluated is PLTAG, a psycholinguistically motivated version of TAG (Demberg and Keller 2008; Demberg, Keller, and Koller 2013). In contrast to DVTAG, predictions are derived up to the root of the sentence structure, i. e. the complete embedding of the words processed is determined, see the left operation in Figure 3.5. These predictions of structure (denoted by XX↓) are created via treelets without lexical nodes. All predicted structures need to be

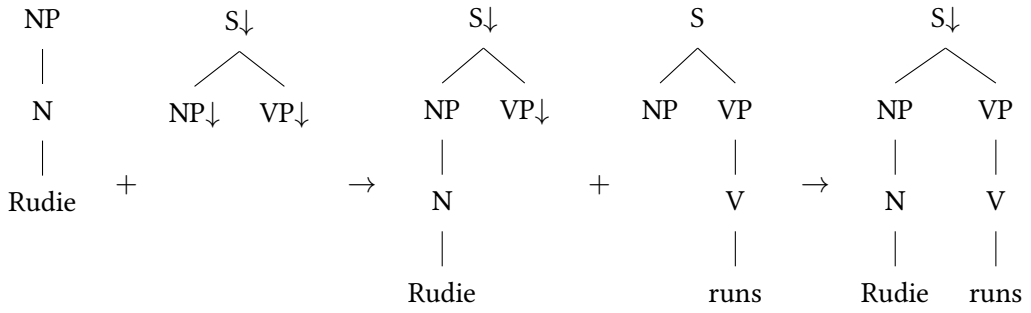


Figure 3.5.: Prediction and verification in PLTAG. The initial tree for "Rudie" is combined with a prediction tree (containing no lexical items). The predictions are then verified by matching the initial tree for "runs" with the tree, resulting the the complete structure.

validated using *verification*: predictions stemming from prediction trees (trees without lexical nodes) need to be verified by trees with lexical nodes. If a complete structure contains a node that has not been verified, the structure is ill-formed and disregarded. Figure 3.5 shows an example of verification in the right operation. The system needs to predict the correct number of phrases to be able to build the correct structure. Because it is not clear which predictions are needed when starting to parse a sentence incrementally, beam search needs to be employed.

The implementation of PLTAG discussed in Demberg, Keller, and Koller (2013) has been evaluated on the Wall-Street Journal part of the Penn Treebank (Marcus et al. 1994), with an automatically extracted lexicon for the elementary, auxiliary, and prediction trees, and conditional probability distributions for the actions (substitution, adjoining, verification). Parsing is performed via beam-search, using supertagging (Bangalore and Joshi 1999) to filter tree candidates. Overall, this system has worse accuracy than its non-incremental counterparts, with an F-score of 78.65 in comparison to 86.7 obtained by Chiang (2000).

3.2. Incremental Dependency Structure Annotations

As this thesis focuses primarily on dependency structures, a definition for incremental structures is needed. The main ideas expressed in this section are the same as in Beuck, Köhn, and Menzel (2011b), Beuck, Köhn, and Menzel (2013), and Köhn and Menzel (2014)

but the naming is adapted,[4] and the formalization is new.

In contrast to phrase structure trees, every part of a dependency structure is anchored in a word – there a no non-terminals in the structure. Figure 3.1 shows an example. A dependency structure consists of words, edges between words, and labels for edges. In data structures, the edges are usually defined using the word's positions. Each annotation scheme has a fixed set of dependency labels that we call L. 0 is the special root node to which a token might be attached to but which is neither attached itself nor contains any other syntactic information.

Definition 2. A *dependency structure* for a sentence is a tuple $\langle W, \pi, l \rangle$, where W is the list of tokens of the sentence, $\pi : W \rightarrow W \cup \{0\}$ the set of directed edges, and $l : W \rightarrow L$ the labeling function for the edges.

An *unlabeled dependency structure* is a corresponding two-tuple $\langle W, \pi, \rangle$

Defining the edges as a function over the tokens enforces the single-head property of dependency structures; each token is mapped to its head, forming a dependent-head pair. Because each token is the dependent of exactly one such relation, the labeling function can be defined over the tokens instead of over edges. The edges of a dependency structure for a complete sentence typically form a tree, i. e. the dependency structure is connected.

Definition 3. A dependency structure is *connected* iff there is a path between each pair of words of the structure (ignoring edge direction), and one word is attached to the root node.

This property can not trivially be maintained for a structure of a sentence prefix, as the word needed for attachment might not be part of the prefix. To deal with this, we have several options: a) wait until all words needed to create a tree are part of a prefix before creating the output, b) lift the connectedness requirement (and leave some words unattached), or c) introduce new nodes into the tree to establish a connection.

Option a) may sound straightforward at first. However, the consequence would be a possibly massive delay of the output: if the root of a sentence is the last word, the output of a parser would degenerate to a non-incremental one as the parser could produce no incremental output at all. Option b) would enable us to define a structure for every sentence prefix. However, the information in this structure would still be less than actually inferable: consider the sentence prefix "She". It is highly likely that "She" will be attached to a verb in the complete sentence structure, and therefore a parser should be encouraged to make this kind of prediction. This leads us to Option c): introducing new nodes. Implementing

[4] Essentially, this thesis uses the term "prediction node" instead of "virtual node" as the latter term seemed to be confusing in discussions.

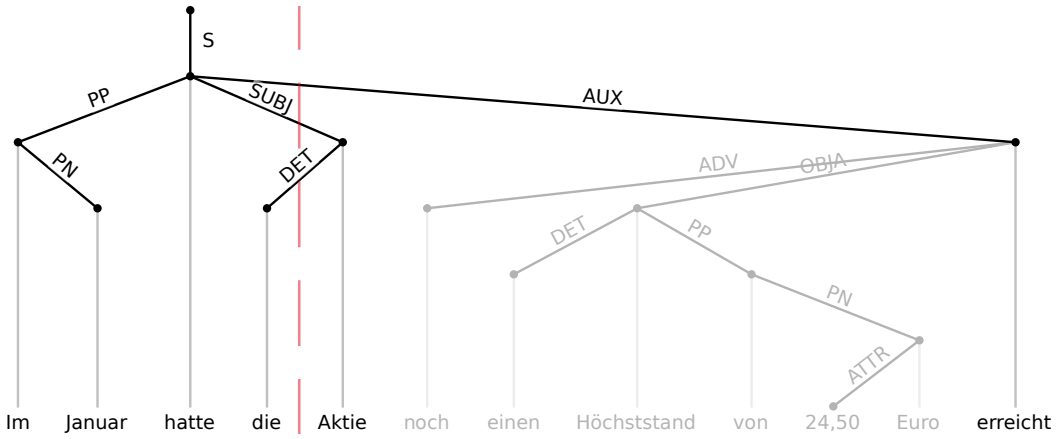


Figure 3.6.: A full dependency structure cut after the word "die" to show how an incremental gold standard is derived. Literally: In January had the share still a peak of 24.50 Euro reached "In January, the share still reached a peak of 24.50 Euro."

Option c) is a central part of this dissertation. Options a) and b) are *conservative* approaches while c) is *predictive*, similar to the different possibilities when using phrase-structure annotations. A conservative approach only creates connections between words of the prefix. Because the correct head of a word in the prefix might be out of the prefix, π and l become partial functions, leaving such words unattached.

To exemplify the different aspects that need to be covered in the predictive approach, we can "cut" a sentence prefix out of a complete sentence and its corresponding syntactic structure. Four different cases can occur for the dependency edges; Figure 3.6 depicts them:

a) edges with both words in the prefix (words "im" to "hatte")

b) edges where the head lies outside the prefix (the determiner "die")

c) edges where the dependent is outside the prefix ("Aktie", "erreicht")

d) edges where both head and dependent are outside the prefix (all greyed-out words)

Point (b) is the most important case to consider for obtaining a connected dependency structure for the sentence prefix. While edges of the category (a) can remain unchanged,

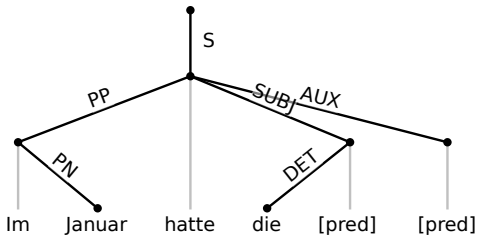


Figure 3.7.: A predictive dependency structure for the prefix shown in Figure 3.6

and (c) and (d) can be omitted, heads outside the prefix need to be replaced somehow for words inside the prefix.

Additional nodes have to be inserted as stand-ins for upcoming words to obtain connected analyses of prefixes. Otherwise, connectedness could only be achieved by incorrectly assigning the head inside the prefix. We will call these additional nodes *prediction nodes*.

Definition 4. A node serving as a stand-in for a word outside the current prefix is a *prediction node*. It represents a single anticipated upcoming word and is included into a dependency structure the same way as a word of the prefix. It can carry morpho-syntactic information and is usually not lexicalized.

Using prediction nodes allows a parser to create structures for sentence prefixes that resemble the structure of complete sentences. In addition to the information they deliver, these structures can be fed to other components expecting "normal" dependency trees, and those components do not need to know anything about prediction nodes – they can be handled just as normal words.

Definition 5. A *predictive* dependency structure for a sentence prefix is a tuple $\langle W, P, \pi, l \rangle$, where W is the list of tokens of the prefix, P a set of prediction nodes, $\pi : W \cup P \rightarrow W \cup P \cup \{0\}$ the set of directed edges, and $l : W \cup P \rightarrow L$ the labeling function for the edges. An *unlabeled* predictive dependency structure is the corresponding three-tuple $\langle W, P, \pi \rangle$.

To ease notation, W^s will refer to the words of a predictive dependency structure s, P^s, π^s, and l^s will refer to the other elements of the tuple, respectively.

Figure 3.7 shows a predictive dependency structure containing two prediction nodes: [noun] and [verb]. Note that only one of them (the noun) is needed to ensure connectedness,

but prediction nodes may be present without being on a path from a word in W to the root.

While the words W are ordered, the prediction nodes P are not; they form a set. As this work is mainly concerned with the syntactic structure and not the prediction of a word sequence, no ordering is given to the prediction nodes other than they are located outside the current sentence prefix. The figures in this dissertation order them for the sake of visualization without asserting that the ordering is encoded in the predictive structure.

For both regular and predictive dependency structures the tokens have a lexicalization and may carry additional morpho-syntactic information such as part of speech or case. While the type of a phrase in incremental phrase structure parsing can often be reasonably well predicted, a word's identity is predictable only in particular circumstances. Therefore, while the prediction nodes may carry morphosyntactic information as well, they are usually not lexicalized because the identity of the word a prediction node stands for can, in most cases, not be predicted.

If additional nodes are not provided by a parser and words are left unattached – i. e. choosing option b – there are two possible interpretations for unattached words: First, simply nothing is known and they could well be attached to a word in the prefix later on. This is the least informative interpretation. Second, all words not attached inside the prefix should actually be attached to an upcoming word outside the prefix. Which interpretation is applicable depends on the guarantees provided by the parser that generated the structure.

3.3. Modeling Uncertainty

A system automatically creating structures for complete sentences usually strives to recreate these gold-standard annotations. However, even for domain experts, it is often hard to predict the correct continuation for incomplete input and, therefore, the correct annotation of the prefix. In these cases, it might be beneficial to explicitly model this uncertainty. Different approaches discussed in this chapter already perform some uncertainty modeling: PLTAG uses substitution nodes to denote expected structure; e. g., a noun phrase might be predicted without specifying how the noun phrase looks like. Similarly, predictive dependency structures add under-specified nodes as stand-ins for upcoming words.

There is another possibility of modeling uncertainty orthogonal to the one just described: A processor may create several alternative structures, which are ordered or even assigned a probability distribution. Producing several ordered alternatives is easily attainable with some approaches (and will be discussed in Chapter 6). Parsers based on PCFG work on probability distributions anyway; an incremental PCFG parser such as the one by

Roark (2001) can be queried for a probability distribution over syntactic structures for a prefix. This property is used e. g. for language modeling, and I will discuss this aspect in Chapter 7. It is next to impossible to carry out an evaluation in a standardized way when using multiple ranked outputs; evaluating incremental structures is already tricky, objectively determining which probability distribution over a broad set of structures is the better one is infeasible. The only way to gauge the merit of multiple outputs is by using them as a part of a larger system that can be meaningfully evaluated – an example for such an integrated evaluation will be discussed in Chapter 7.

Chapter 4.

Gold Standards and Evaluation for Predictive Parsing

Every evaluation of a supervised processor needs a gold standard to compare against. Several treebanks are used to evaluate the parsers in this thesis, which form two groups: treebanks with function-head annotation (where function words are the heads of content words) and treebanks with content-head annotation (where function words are attached to content words). The treebanks in the first category are the Hamburg Dependency Treebank (HDT, German) (Foth, Köhn, et al. 2014), the Penn Treebank converted to dependency structure using the LTH converter (Johansson and Nugues 2007) (PTB, English), and the French Treebank (FTB) (Abeillé, Clément, and Toussenel 2003). The content-head annotated treebanks used in this thesis all use a universal dependencies schema. They are: The HDT converted to UD v2 (Hennig and Köhn 2017; Borges Völker et al. 2019) (UD-HDT), the PTB converted to UD v1 using the Stanford converter (Schuster and Manning 2016) (UD-PTB), and the FTB converted to UD v2 (Seddah et al. 2018) (UD-FTB). In addition to these paired treebanks, the UD-English-EWT treebank (Silveira et al. 2014) for a UDv2 corpus of English and UD-Szeged (Hungarian; a UDv2 conversion of the treebank introduced by Vincze et al. (2010)) are used.

For parsers that are trained, these treebanks also provide the training data. Generally speaking, a parser can be viewed as a function that maps a sentence input to a tree-structure output. This function can be learned and evaluated on treebanks because they contain the same type of data: sentences and their corresponding trees. These pairs of input and output are also needed in an incremental setting. However, there are no pairs of sentence prefixes and their desired corresponding incremental syntactic structures readily available. Based on the findings in Chapter 3, this chapter discusses how to tackle this problem both by automatically creating incremental gold standards and by performing training and evaluation against complete sentence annotations. We will first look at gold standards under the aspect of evaluation and discuss training in Chapter 5.

As the evaluation compares two different structures, the structures will be distinguished

by a subscript: $_s$ for the system output and $_g$ for the gold standard. When several sentences are considered, their structures are distinguished with a superscript.

Non-incremental dependency parsers are evaluated using an accuracy score: For each token w in a given test set we just compare the head of w in the system output – denoted as $\pi_s(w)$ – to the head in the gold standard annotation $\pi_g(w)$. The percentage of correct attachments is the unlabeled attachment score (UAS). As the edges of dependency trees also have labels attached, the labeled attachment score (LAS) represents the fraction of tokens for which both the head and the label agree with the gold standard ($l_s(w) = l_g(w)$).

Definition 6. The unlabeled and labeled attachment scores (UAS/LAS) for a set of sentences T are defined as follows:

$$UAS(T) = \frac{\sum_{t \in T} \sum_{w \in W^t} \mathbb{I}(\pi_s^t(w) = \pi_g^t(w))}{\sum_{t \in T} \sum_{w \in W^t} 1}$$

$$LAS(T) = \frac{\sum_{t \in T} \sum_{w \in W^t} \mathbb{I}(\pi_s^t(w) = \pi_g^t(w) \wedge l_s^t(w) = l_g^{t(w)})}{\sum_{t \in T} \sum_{w \in W^t} 1}$$

Punctuation is handled in two ways for evaluation in the literature: including punctuation and excluding punctuation. Sometimes non-word tokens (i. e. punctuation) are excluded during evaluation (e. g. in Nivre et al. (2007)) and including non-word tokens into a dependency structure can actually be adverse to parsing accuracy (Ma, Zhang, and Zhu 2014). Some dependency treebanks (e. g. the Hamburg Dependency Treebank) do not include the punctuation into the dependency structure but attach them all to the root node. As predicting these attachments is trivial, parsers usually obtain perfect accuracy for punctuation on such treebanks. On the other hand, punctuation is annotated in the majority of the treebanks and in most of the literature, all tokens are included into the evaluation; this is especially needed if the distinction between word versus non-word tokens has to be made by the parser – in that case, excluding the non-word tokens from evaluation would fail to measure that distinction. The evaluations in this paper will include punctuation.

4.1. Mapping Prediction Nodes to Words in Full-sentence Annotations

One approach to evaluating incremental parsers is to use the full-sentence annotations from the non-incremental gold standard as reference (Beuck, Köhn, and Menzel 2011b). Using them allows us to use already existing annotations without modification but requires to reason about the prediction nodes in the system output. When computing the attachment

score for a predictive dependency structure, the same four different cases with regards to the dependency edges as in Section 3.2 have to be distinguished for determining whether a node n is correctly attached (compare also Figure 3.6). Namely, whether the dependent and head are part of the prefix or are prediction nodes.

If both n and $\pi_s(n)$ are part of the prefix, it is easy to decide whether that attachment is correct as we only check whether the same attachment is part of the full-sentence annotation, i. e. whether $\pi_s(n) = \pi_g(n)$. The other three cases require to reason about what a prediction node represents. As each prediction node stands for the expectation of an upcoming word, an attachment of n to a prediction node p should be deemed correct iff p represents the head of n in the full-sentence annotation. Therefore, a mapping from prediction nodes to words outside the prefix is needed. Given a full-sentence annotation and a predictive dependency structure of a prefix, several prediction nodes in the system output might need to be mapped to several out-of-prefix candidate words in the gold standard. The set of prediction nodes to be mapped is P_s, and the set of out-of-prefix words of the gold standard is $O_g = W_g - W_s$.

Formally, we need to select a partial function $m_p : P_s \to O_g$ matching the prediction nodes that adheres to two properties: the function should be an injection, and it should minimize the number of attachment errors. The first property reflects the assumption that a prediction node stands for a specific upcoming word, and therefore two different predictions may not be mapped to the same word. The second property might sound like tailoring the evaluation to the intended results, but consider the example in Figure 4.1: By selecting the matching that minimizes the errors, we measure the potential of the prediction being replaced correctly by a word later on. A matching not minimizing this error would not measure this potential.

To make the following definitions simpler and to avoid having to distinguish between words in the prefix and prediction nodes everywhere, we will extend the matching m_p to a full matching m that also maps all in-prefix words to the corresponding word in the full-sentence annotation. With π denoting the "head of" function to obtain the head of a node, the desired mapping is defined as follows:

Definition 7. The best mapping between two annotations is the one that maximizes the agreement between the edges of the annotations:

$$\phi = \arg\max_{m \in M} \sum_{n \in W_s \cup P_s} \mathbb{I}(\pi_g(m(n)) = m(\pi_s(n))) \tag{4.1}$$

Based on this best mapping, we define a word or prediction n as correctly attached (ignoring the label) iff $\phi(\pi_s(n)) = \pi_g(\phi(n))$. I. e. we check whether two words in the gold standard are the same:

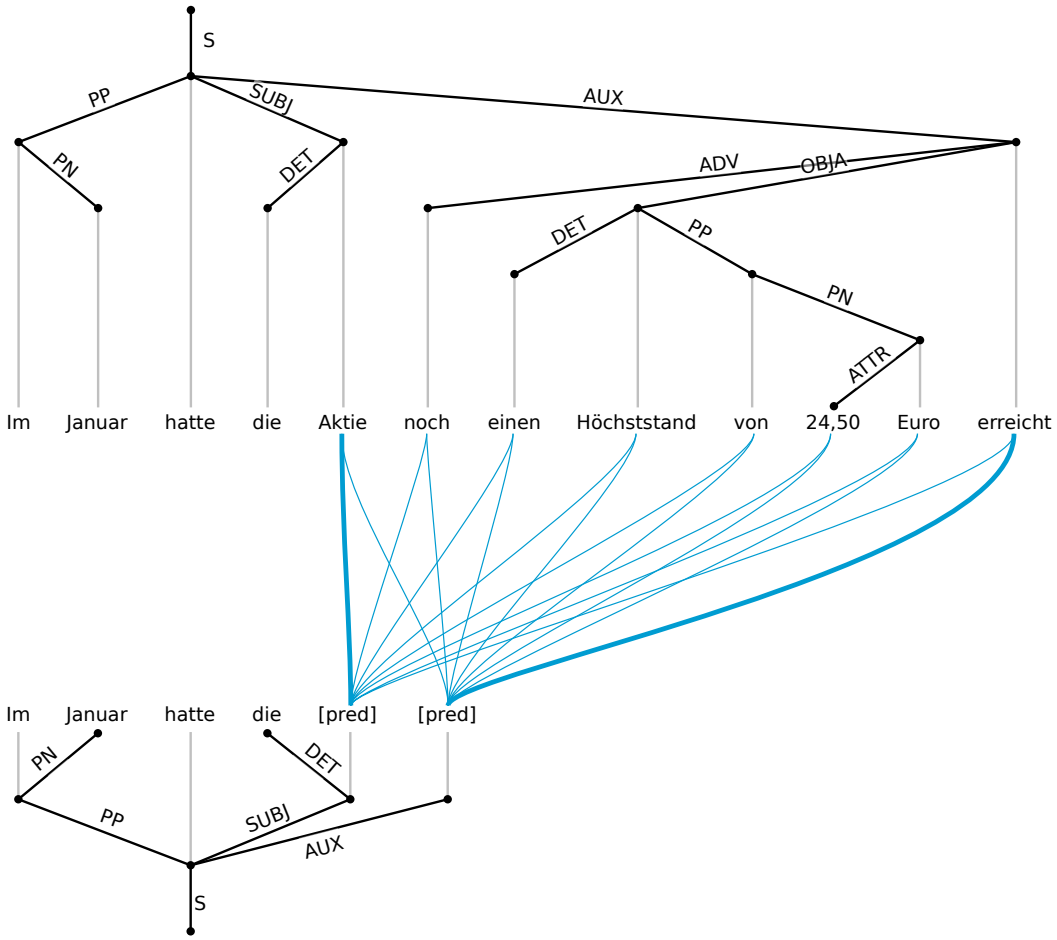


Figure 4.1.: A full-sentence annotation (top) and a predictive dependency structure (bottom). For evaluation, the prediction nodes have to be mapped to the words outside the prefix; lines visualize all possible matches. The mappings maximizing the number of correct attachments during evaluation are highlighted in bold.

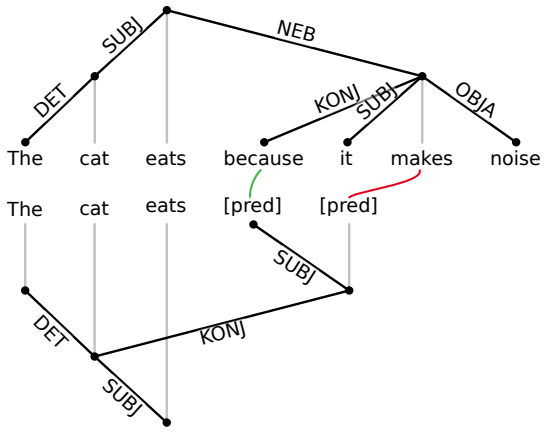


Figure 4.2.: A predictive structure with incorrect predictions. One (green) would be counted as correct by Beuck, Köhn, and Menzel (2011b) because their mapping does not enforce grounding in the prefix.

1. the word we obtain by taking the head of n ($\pi_s(n)$), and mapping it to a word in the gold standard ($\phi(\cdot)$)

2. The word we obtain by mapping n to a word in the gold standard ($\phi(n)$), and obtaining its head ($\pi_g(\cdot)$)

Up to now, this is the approach described in Beuck, Köhn, and Menzel (2011b). However, this approach has one problem: since prediction nodes can be mapped to any word outside the current prefix, a structure consisting of several prediction nodes may fit quite well just because this kind of structure often exists somewhere in an annotation. For example, many sentences will have two sibling nodes attached to a head somewhere. A mapping generated as described above would often contain mappings for prediction nodes that have no valid connection to the prefix (i. e. the edges connecting the prediction nodes to the prefix are incorrect), as depicted in Figure 4.2. A metric based on this type of mapping would over-estimate the prediction capabilities of a parser. Mappings can be constructed to contain only prediction nodes that are directly or indirectly licensed by the prefix to mitigate this effect: We create the mapping iteratively over the prediction nodes. A prediction node n may only be mapped if the head or the dependent is either a word of the prefix (the prefix directly licenses mapping n) or a prediction node that

is already mapped (the prefix indirectly licenses mapping n). Algorithm 1 shows a non-deterministic algorithm that generates mappings that adhere to these constraints. All mappings restricted to licensed node mappings are generated by traversing through all possible paths of the non-deterministic algorithm.

The algorithm iteratively extends a mapping, starting with the mapping only containing the in-prefix words (Line 2). It then chooses a yet unmatched prediction node n and an unmatched word of the gold standard w. If their heads are matched (Line 6) to each other or dependents of w and n are already matched (Line 7), (n, w) is added to the matching, and the next set of candidates is selected. Otherwise, the algorithm terminates and returns the mapping generated so far.

Algorithm 1: Non-deterministic restricted mapping generation

Data: s: predictive structure to be mapped, g: dependency structure mapped against
Result: $m \subseteq W_s \cup P_s \times W_g$

```
1 begin
2     m ← {(w, w) : w ∈ W_s}            // all words in the prefix mapped
3     while m was changed in the last iteration do
4         Choose n ∈ P_s − {e : (e, _) ∈ m}
5         Choose w ∈ W_g − {e : (_, e) ∈ m}
6         if (π_s(n), π_g(w)) ∈ m   ||                    // licensed by head
7         ∃(n', w') ∈ m. π_s(n') = n ∧ π_g(w') = w   // licensed by dependent
8          then
9             m ← m ∪ {(n, w)}
```

All mapped nodes are connected to the prefix with a chain of correct attachments: a mapping to a word outside the prefix is only made when resulting in a correct attachment from or to a word in the prefix or an already mapped prediction node. Note that this does not mean that all attachments are correct. From all the possible mappings generated by the non-deterministic algorithm, a best one still has to be selected by counting the number of correct attachments as defined in Equation 4.1.

4.2. Evaluating Predictive Parses Against Full-sentence Annotations

The quality of a non-incremental dependency parser can be described using a single metric, namely the attachment accuracy (see Definition 6). We count the number of correct attachments and divide it by the number of words. For incremental output, this single metric is not sufficient as it disregards all dynamics over time. Section 2.3 already described evaluation metrics that capture properties of incremental processors, mainly for processors other than parsers. This section introduces an evaluation schema for incremental dependency parsers based on the ones proposed in Beuck, Köhn, and Menzel (2011b) and Beuck and Menzel 2013.

An accuracy score for incremental output could be computed by summing over the attachments for all words in all prefixes. Because words in the prefix can be attached to prediction nodes, mappings would still need to be computed. There are three problems with this approach: First, words early in a sentence are counted more often than words later in the sentence, leading to an artificially increased weight of sentence-initial words. Second, the accuracy score largely disregards the prediction nodes: Whether predicting them is correct and whether they are correctly attached. Third, the metric does not give any insight into the process' behavior over time, e. g. whether the parser is monotonic, or how the accuracy correlates with how recent a word is.

4.2.1. Computing Incremental Accuracy

Because the evaluation should address several aspects, more than a single metric is needed to capture these aspects, as described in Section 2.3 for other processors. To capture the accuracy without the weight shift towards sentence-initial words and at the same time include information about the dynamics of the accuracy over time, we can compute the accuracy relative to the position of the frontier, as introduced in Section 2.3.2.

Let $\mathcal{S}_\int$ be all structures generated by the system to be evaluated based on a non-incremental gold-standard $\mathcal{S}_\}$. To ease the notation, let $\tau^t(n) = W_s^t(|W_s^t| - n)$ be the n-newest word of a prefix $t \in \mathcal{S}$. We can then define the incremental accuracy for predictive dependency parses as follows, with $\mathbb{I}$ being the indicator function:

$$inc\_acc(n) = \frac{\sum_{t \in \mathcal{S}_\int} \mathbb{I}(\pi_g^t(\phi^t(\tau^t(n))) = \phi^t(\pi_s^t(\tau^t(n))))}{\sum_{t \in \mathcal{S}_\}} \max(|W_g^t| - n, 0)} \tag{4.2}$$

That is, we compute how many times the n-newest word of a prefix was attached

correctly and divide it by the number of prefixes that actually had an n-newest word. This yields the attachment accuracy at a specific position relative to the newest word of a prefix.

Reporting $inc\_acc(n)$ for several values for n gives an insight into the dynamics of a system. For a monotonic system, the accuracy never increases with increasing n, i. e. when more of the context to the right becomes available. In contrast, non-monotonic systems can improve the accuracy with increasing n. Performing an additional measurement of the accuracy of full-sentence annotations allows comparing incremental parsers to non-incremental ones.

4.2.2. Subdividing Accuracy With Respect To Predictions

Attachments to prediction nodes behave differently than in-prefix attachments: Because of the way the matching from prediction nodes to the gold standards are generated, an attachment involving a prediction can be counted as correct even though this prediction will be filled with an incorrect word later on. Therefore, the incremental accuracy measurement can be divided into more fine-grained groups than just "correct" and "wrong": an attachment of a word w can be classified into four cases:

correct $\pi_g(\phi(w)) = \phi(\pi_s(w)), \pi_s(w) \in W_s$

correct prediction $\pi_g(\phi(w)) = \phi(\pi_s(w)), \pi_s(w) \in P_s$

wrong prediction $\pi_g(\phi(w)) \neq \phi(\pi(w)), \pi_s(w) \in P_s \wedge \pi_g(w) \in P_g$

wrong $\pi_g(\phi(w)) \neq \phi(\pi_s(w)), \pi_s(w) \in W_s \vee \pi_g(w) \in W_g$

Both correct and wrong attachments are divided into whether they attach to a word outside the prefix or not. An attachment of a word is *correct* if it is correctly attached in-prefix. The attachment is a *correct prediction* if the head is a prediction node that is mapped to a prediction node that is also the head in the gold standard. An attachment is a *wrong prediction* if the parser correctly guessed that the word should be attached out of the prefix, but the head is not mapped to the word's head in the gold standard. In all other cases, the attachment is *wrong*. For a monotonic parser, the "correct" attachments will stay correct, whereas the "correct prediction" attachments might become incorrect later on due to the parser incorrectly filling the prediction. Classifying the attachment of a word w as a wrong prediction means that while the parser correctly assumes that w needs to be attached outside the prefix, it is attached to a prediction node not mapped to the correct word in the gold standard. This class can be seen as "less wrong" than wrong attachments because a different mapping might have resulted in classifying the

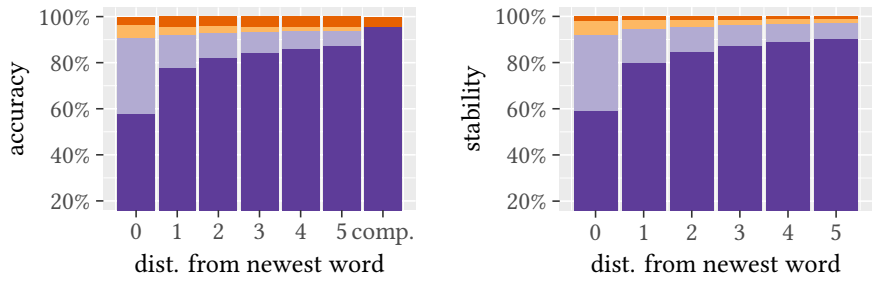


Figure 4.3.: Visualization of incremental parser performance. Left: accuracy, right: stability. Correct: ■, correct prediction: ■, wrong prediction: ■, wrong: ■ X-axis: accuracy of the tokens N left of the newest one; comp: accuracy measured on complete sentences.

attachments as correct.[1] When comparing a non-predictive parser to a predictive one, the *wrong prediction* category should be folded into *correct prediction* as the non-predictive parser does not specify the target of a connection into the future. Therefore, the predictive parser should not be penalized for trying to make these predictions.

As all these metrics sum to a quite high amount of numbers, they can be visualized to illustrate the overall performance of a parser better. Figure 4.3 shows an example for a visualization: Bar charts depict the incremental accuracies with the complete-sentence accuracy on the right serving as a comparison.

4.2.3. Stability Measures

In addition to the accuracy, the stability of the output is of interest (c.f. Section 2.3.3). A stability measure can be obtained by computing $inc\_acc(n)$ against the system output for the complete sentence instead of the gold-standard annotation. An attachment is then considered stable if it is consistent with the final output. Note that this notion of stability is different from the one proposed by Baumann, Buß, and Schlangen 2011, which also requires that all outputs between the current one and the last one generated by the

[1] These definitions of *wrong prediction* and *wrong* differ from the definitions in Beuck, Köhn, and Menzel (2013), who consider an attachment to be an "wrong prediction" also if the parser creates an attachment to a prediction node but the word should be attached in-prefix.

processor are consistent with the complete output. In other words, Baumann et al.'s stability measures the probability that an output will not change anymore. In contrast, our stability measures the probability of a specific output also being in the final output.

4.3. Creating Gold-standard Annotations for Sentence Prefixes

We now have the methods to evaluate predictive dependency parsers, but still, lack the incremental gold standards to compare against. While these could be manually annotated, doing so is prohibitively costly because the number of increments is a multitude of the number of sentences in the corpus. Therefore, the evaluation will be performed against incremental gold standards that are automatically derived from the non-incremental ones. Beuck and Menzel (2013) propose an approach to automatically generate predictive dependency annotations from the annotation of full sentences. Their method tries to generate upper bounds for predictability, which nonetheless over-generate predictions as little as possible. Therefore, not everything that is deemed predictable by the algorithm is predictable in reality, but everything that is predictable should be deemed as predictable. Beuck and Menzel (2013) primarily use these predictive annotations derived from full-sentence annotations to investigate what kind of predictions a parser should be able to make, e. g. how many nouns a parser should be able to predict. However, these annotations can also be used to evaluate parsers (and to train them, see Chapter 5).

To compute a predictive dependency annotation for a prefix of length n from a gold-standard dependency structure $t = \langle W, \pi, l \rangle$, we simulate annotating the words $W^t(1 \ldots n)$ with a structure containing predictions, and reason about which words of $W^t(n+1 \ldots)$ are predictable. The reasoning about predictability primarily relies on the dependency labels of t. To compute the predictive dependency structure for a specific prefix, let $V_i = W^t(1 \ldots n)$ be the tokens in that prefix and $V_o = W^t(n+1 \ldots)$ all tokens that are outside the prefix. The task is to come up with a set $Pr \subseteq V_o$ of words that are deemed predictable based on the words of the prefix (V_i), using the dependency structure as guiding information. The predictive dependency structure is then generated by removing all non-predictable words outside the prefix and delexicalizing the predictable words. Algorithm 2 shows the pseudo-code of the approach proposed by Beuck and Menzel (2013) to determine the set of predictable words.[2] It iteratively adds tokens to the set of predictable tokens (Pr) until no further changes have been made in an iteration. The algorithm makes use of two language-specific pre-defined sets of dependency labels: *lex_predictable* are syntactic roles

[2]The original implementation for German is by Niels Beuck, structuring the algorithm as shown to abstract from a single annotation schema and adaptation to English was done by me.

deemed predictable if the head of the word is known, i. e. part of the prefix. An example of this category is the *obj* label because once a verb is observed, we assume to know the valency of that verb and can therefore predict how many objects it takes. *Predictable* are those roles that also can be predicted if the head of the word is merely predicted. An example would be the *subj* label because a word filling that role can be predicted even if the identity of its governing verb is not known.

Algorithm 2: Determine which words are predictable

Data: $\langle W, \pi, l \rangle$: A dependency structure, n: number of words in the prefix
Result: $Pr \subseteq W$

```
1  begin
2      V_i ← W(1 ... n)
3      V_o ← W(n+1 ...)
4      Pr ← ∅
5      while Pr was changed in the last iteration do
6          for w ∈ V_o do
7              if ∃d ∈ Pr ∪ V_i.  (d, w) ∈ π then
8                  Pr ← Pr ∪ {w}          // predict connection to the root
9              if l(w) ∈ lex_predictable ∧ π(w) ∈ V_i then
10                 Pr ← Pr ∪ {w}      // predict dependents of known words
11             if l(w) ∈ predictable ∧ π(w) ∈ V_i ∪ Pr then
12                 Pr ← Pr ∪ {w}          // predict obligatory dependents
```

A word $w \in V_o$ is assumed to be predictable (and therefore added to Pr) if one of the following three criteria is met:

bottom-up prediction There is another word w' so that w is the head of that word ($w = \pi(w')$), and w' is already known to become part of the predictive dependency structure, either because that word is in the prefix ($w' \in V_i$) or if it is already identified as being predictable($w' \in Pr$). Lines 7 and 8 cover this aspect, and it is needed to create connected structures.

lexical top-down prediction The head of w is in the prefix ($\pi(w) \in V_i$) and w fills a syntactic role – encoded by its dependency label – that is assumed to be required by an already observed word, e. g. the object role: If $\pi(w)$ is a verb for which we know the identity (because it is in V_i) and w is its object, w is assumed to be predictable

because it fills a valency of the verb. In the algorithm, the set of these labels is called *lex_predictable* and this aspect is covered in lines 9 and 10.

top-down prediction The head of w will be in the partial dependency analysis ($\pi(w) \in V_i \cup Pr$) and w fills a syntactic role that is deemed to be always needed, irrespective of the head's lexical identity. That means w can also be predicted if $\pi(w)$ is outside the prefix and its lexical identity is not known ($\pi(w) \in Pr$). An example of this is the subject label: If $\pi(w)$ is in Pr and w is its subject, w is assumed to be predictable. (lines 11 and 12)

Note that this approach highly depends on the annotation schema: The sets *lex_predictable* and *predictable* have to be determined for each annotation schema independently. Determining them requires detailed knowledge of the annotation schema because one has to decide for every single relation whether words attached with this relation should be deemed predictable and, if so, to which of the two groups it should belong.

The label sets *lex_predictable* and *predictable* for German have been adopted from (Beuck and Menzel 2013), while the sets for English have been obtained by manually analyzing the Penn Treebank for predictability. The sets for Universal dependencies were created from the descriptions of the labels in the UDv2 documentation.[3] The relevant sets are listed in Appendix C.

For words marked as predictable, their existence and word class, but not their lexicalization and position, can be predicted. Therefore, the lexical items are substituted with a placeholder, and the part-of-speech tags are generalized to more coarse-grained ones, only differentiating whether the word behaves more like a verb or more like a noun. We will call the set of gold-standard prefix annotations generated this way T_G.

While this procedure is language-independent, some annotation-specific transformations must be applied nonetheless, which are not shown in the pseudo-code. The correct length of right-branching chains such as coordinations or conjunctions can not be predicted, only that there needs to be at least one. However, in both annotation schemata, such chains are sequences of one label, terminated by a different label on the right-most side. In these cases, the intermediate chain is removed from the prediction, and only the right-most part containing the closing label is kept. In addition to this problem with top-down prediction, there is a similar issue for the HDT annotation: Some structures can not reasonably be predicted by a parser (or a human) because the sub-structure to be predicted consists of several words, even though a single word could also reasonably complete the prefix. Figure 4.4 shows such an example: The continuation of the sentence contains several verbs, which form an auxiliary chain. The different dependents in the

[3] `https://universaldependencies.org`

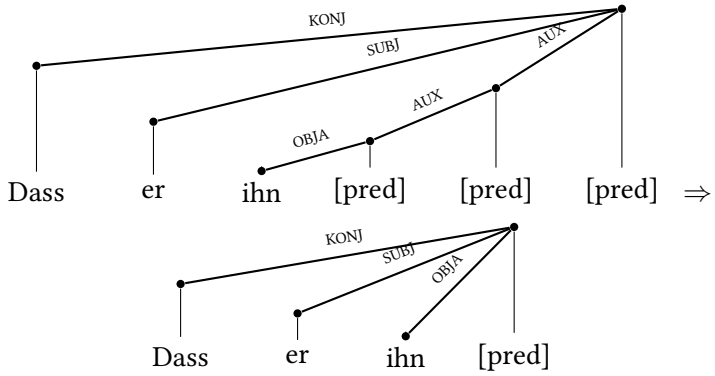


Figure 4.4.: Example for folding: When generating a predictive dependency structure for the first three words of the clause "dass er ihn gesehen haben soll" (literally: "that he him seen has should" meaning: "that he is supposed to have seen him") . All three verbs would be kept for connectedness, but are then folded into a single predicted verb, which is consistent with the possible single verb continuation "dass er ihn sah" (lit. that he him saw).

prefix are attached to different words of this chain, but the sentence could also just have continued with a single verb, in which case all dependents would be attached to the same prediction node. In these cases, Beuck and Menzel (2013) perform *folding*, i. e. multiple prediction nodes generated by the algorithm are merged into a single prediction node. Using folding results in a non-monotonic incremental gold standard: tokens attached to the same prediction node in one increment are supposed to be attached to different tokens in a later increment. A parser producing correct output under these circumstances has to be non-monotonic. This mismatch between the (reasonable) prediction of a single word and realization as several nodes is an artifact of the annotation schema, and its consequences will be discussed in Section 6.4. In a content-head annotation, the auxiliary verbs would be attached to the content-bearing verb as well as the words of the prefix. Therefore, the prediction needed would be the same with a single verb and a verb chain, leading to a more consistent annotation with regards to the predictable structure.

4.4. Evaluating Prediction Nodes

Up to now, prediction nodes have not been the target of evaluation. In Section 4.2, incremental accuracy was introduced to measure the quality of an incremental parser. However, only in-prefix words are captured by this metric, and prediction nodes are only indirectly included as they are needed as heads for in-prefix words. The attachment accuracy of out of prefix nodes is not counted, and prediction nodes that have no child in the prefix are completely disregarded. Also, a parser evaluated under this scheme does not benefit from producing top-down predictions as these are not evaluated at all.

Accuracy can not be computed for the prediction nodes because the number of predictions made by a parser is not predetermined. Instead, precision and recall have to be measured (similar to evaluation of phrase-structure parses, in which the number of phrases is not fixed). The precision is simply the number of correctly attached prediction nodes divided by the number of prediction nodes.

The prediction precision can be computed using set of generated structures $\mathcal{S}_s$, and, for each structure t of $\mathcal{S}_s$, the set of prediction nodes P_s^t of the predicted and gold-standard head of functions π_s^t and π_g^t, and the mapping from predicted structure to the corresponding (non-incremental) gold structure ϕ^t as follows:

$$corr\_pred = \sum_{t \in \mathcal{S}_s} \sum_{p \in P_s^t} \pi_g^t(\phi^t(p)) = \phi^t(\pi_s^t(p)) \tag{4.3}$$

$$pred\_prec = \frac{corr\_pred}{\sum_{t \in \mathcal{S}_s} |P_s^t|} \tag{4.4}$$

$pred\_prec$ is structurally very similar to inc_acc (Equation (4.2)), instead of counting the accuracy at a specified point relative to the frontier, the correct attachments of the prediction nodes are counted.

Computing precision is only a first step to assessing the quality of predictions, but still, a parser will not be rewarded for producing additional predictions. However, recall can not sensibly be computed against whole-sentence annotations: It would pose the complete remaining sentence as a target to be predicted. On the other hand, the incremental gold standards developed in Section 4.3 provide a more reasonable upper bound for predictability.

The parser's output can be mapped against the incremental gold standard in the same way as it is mapped against the non-incremental one, according to the definition in Equation (4.1). Using this mapping, inc_acc and $pred\_prec$ can also still be computed as before. Also, the prediction *recall* can be computed by dividing the number of correctly

attached prediction nodes by the number of prediction nodes in the gold standard prefixes, essentially substituting $\mathcal{S}_g$ for $\mathcal{S}_s$ in the denominator:

$$pred\_recall = \frac{corr\_pred}{\sum_{t \in \mathcal{S}_g} |P_s^t|} \tag{4.5}$$

Without referring to incremental gold standards, only the number of predictions can be compared between different approaches – a number which is much less informative and interpretable.

4.5. Evaluating Non-predictive Incremental Parsers

Some parsers can work incrementally but do not use prediction nodes; words that should be attached to a word outside the current prefix are simply left unattached. Examples are parsers based on a shift-reduce formalism such as the one that will be introduced in Section 5.1.

It might sometimes be desirable to compare parsers performing prediction with prediction nodes to incremental parsers that only produce dependencies between in-prefix tokens and which leave words that should have a head outside of the prefix unattached. The definition of correctness has to be adapted to be applicable for both cases to carry out such a comparison. We assume that a parser will attach currently unattached words to a word currently outside of the prefix once that word becomes available. Therefore, an attachment of a word is deemed correct if it is correctly attached in-prefix or if the correct attachment is outside of the prefix, and the word is either unattached (for non-connecting parsers) or attached to a prediction node (without performing any matching; for predictive parsers).

An output that includes prediction nodes can be interpreted in a compatible way by relaxing the correctness requirements: Instead of matching the prediction nodes to the gold standard, attaching a word to a prediction node is considered correct if that word is attached out of prefix in the gold standard.

This method of evaluation is strictly less informative than evaluating with prediction node matching because all information about prediction is discarded. Also, the accuracies measured are higher than the ones obtained when requiring a match of the prediction nodes. Therefore, this type of evaluation should only be used to compare a predictive processor to a non-predictive one. To compare two predictive parsers, the more informative approach with matching should be used.

4.6. Labeled Versus Unlabeled Evaluation

Even though most parsers generate labeled dependency structures, this thesis will only evaluate unlabeled dependency structures. Edges are often labeled using only the single edge itself as information. Several parsers use first-order labelers, e. g., TurboParser and RBGParser, and already perform reasonably well, even though higher-order features yield slightly higher labeling accuracies (Köhn, Lao, et al. 2014). In this setting, the edge labeling only slightly reduces the accuracy of all edges but has no impact on the degree of non-monotonicity. If the labeling is optimized globally, e. g. as proposed by Shen, Lei, and Barzilay (2016), this introduces additional non-monotonicity and requires reasoning about different kinds of non-monotonicity: what is the difference between a change of attachment, a change of labeling and a change of both? As this question is out of the scope of this thesis, only unlabeled attachments will be evaluated.

4.7. Tying It All Together

In this chapter, several quality metrics have been discussed that cover aspects of incremental parses. The non-incremental accuracy based on complete-sentence output shows how good the output is after consuming all input and whether a degradation in comparison to non-incremental parsers takes place. The incremental accuracy describes how fast the parser can make high-quality decisions. The incremental stability shows how dependable the output is with respect to its age. Prediction precision and recall measure the quality and quantity of forward-looking structure.

Chapter 5.

Training Predictive Dependency Parsers

In Section 3.2, we already examined how the structures that an incremental parser creates should look like, and we discussed approaches to evaluate parsers producing such structures (Chapter 4). All that remains to be done is developing a parser that performs incremental parsing, and this chapter will introduce such parsers. There are two main approaches to syntax parsing: transition-based parsing and graph-based parsing. As transition-based parsers are already incremental in a weak sense, I will discuss them (and their shortcomings for incremental processing) first.

5.1. Transition-based Parsing for Incremental Structure Generation

Using a transition system to build dependency parsers is very popular and has produced state-of-the-art parsers over a relatively long period while keeping the same basic idea (Nivre 2003; Nivre 2007; Nivre 2008; Huang and Sagae 2010; Gómez-Rodríguez and Nivre 2010; Goldberg and Nivre 2013; Rodriguez, Sartorio, and Satta 2014; Honnibal and Johnson 2014; Dyer et al. 2015; Kiperwasser and Goldberg 2016, inter alia).

A transition-based parser employs a shift-reduce automaton to generate a dependency structure for a sentence. The automaton has a defined starting configuration (based on the sentence to parse), a fixed set of possible actions that transform one configuration into another one and stopping criteria. Following the notation by Nivre (2008), the quadruple $< C, T, c_s, C_t >$defines a *transition system*, where

- C is the set of configurations (or states),
- T the set of transitions (partial functions of the type $C \rightarrow C$),
- c_s a function mapping an input sentence to a configuration, and
- $C_t \subset C$ the set of terminal configurations.

Ideally, the set of actions is modeled in a way that each sequence of legal actions leads to a configuration in C_t.

The transition system defines a search space with a given initial state (C_s) and a set of goal states (C_t). Of these goal states, only some represent the correct dependency structure, and a parser's job is to find one of these goal states by preferring them to the ones representing an incorrect structure. These preferences are encoded in scores for the transitions between states; the parser learns a scoring function $s : C \times T \rightarrow \mathbb{R}$ and the score for a state is the sum of scores of the transitions leading to that state. Early transition-based parsers used greedy decoding for trying to find the best goal state, i. e., from the current state, they use the transition with the highest score and repeat this until reaching a goal state. A parser working like this can get trapped in a wrong path because s needs to assign the highest score to the correct transition from each state for the parser to reach a correct terminal configuration, a rather strict requirement for s. This requirement can be relaxed by using beam search (compare sections 2.2.1 and 6.3) as performed by Huang and Sagae (2010), Andor et al. (2016) and many others.

5.1.1. The arc-standard transition system

There is a variety of definitions for configurations and actions in the literature (such as the arc-eager and arc-standard variants (Nivre 2008) or the two-stack variant (Gómez-Rodríguez and Nivre 2010)) that alter the automaton, e. g. to be able to produce non-projective structures.[1] An overview and categorization of such transition systems was compiled by Bohnet, McDonald, et al. (2016), but for this thesis it suffices to discuss two of the structurally simplest ones, the arc-standard transition system (Nivre 2004), and the arc-eager transition system. A configuration of an arc-standard transition system consists of a stack S, an input buffer I, and a set of generated edges O, yielding a triple $< S, I, O >$. The initialization function c_s maps a sentence $s = (w_0 \dots w_n)$ to the configuration $< \perp, s, \varnothing >$. The set of terminal configurations are the ones with empty input and one element on the stack. The set of transitions T for arc-standard are[2]:

shift $< S, w|I, O > \rightarrow < S|w, I, O >$; Moves the leftmost element from the input to the top of the stack.

left-arc $< S|d, h|I, O > \rightarrow < S, h|I, O \cup \{(d, h)\} >$; Attaches the topmost element of the stack to the leftmost of the input and removes the dependent from the input.

[1] Projective structures are the ones where for each dependency relation of two words d and h, the words occuring between d and h in the sentences are children of either d or h (either directly or transitively).

[2] There are different but functionally equivalent descriptions of the arc-standard transition system in the literature. This one follows Nivre (2008).

right-arc $< S|h, d|I, O > \rightarrow < S, h|I, O \cup \{(d, h)\} >$; Attaches the top element of the stack to the leftmost element of the input and removes the dependent from the stack.

If s only uses information from a fixed set of words of the input buffer I, the parser can be used incrementally by performing transitions until a word from I is queried that is not yet available. The parser then yields the current best state as incremental output and waits for the next word. A greedy parser will create monotonic output. In contrast, a parser employing beam-search[3] might create non-monotonic output because the best scoring state might not be a successor of the previously best scoring state.

When restricting the information obtained from I to only the leftmost word of I, the parser can be seen as sufficiently incremental; it can incorporate all the words currently available. However, what kind of structure does a transition-based parser generate in the intermediate states? The arc-standard transition system has a property that limits structure generation to be bottom-up: Once a word w has been attached to another word, no word can be attached to w anymore. This property is enforced by removing w from the stack upon creating an arc with w as the dependent (see definitions of left-arc and right-arc above). This property makes transition-based parsers based on the arc-standard or a similar transition system create structures from bottom to top; attaching a word to its head needs to be delayed until all its dependents have been seen. This effect is most visible for the main verbs: They form the root of the dependency structure and can occur rather early in a sentence but can only be marked to be the root (by being attached to 0) after processing the complete sentence. Bottom-up parsing leads to fragmented output where different parts of the prefix are not connected to each other because either the common head is still missing or a word needs to stay unattached so that other words can be attached to it. Using the intermediate states of a transition-based parser with the two-planar transition system (a transition system that can produce non-projective structures) leads to output containing 2.45 unconnected partial trees on average(Beuck, Köhn, and Menzel 2013).

5.1.2. The arc-eager transition system

The arc-eager transition system is similar to the arc-standard one, but creates right arcs (where the head is on the left of the dependent) in a top-down manner.

[3] Beam-search keeps track of a fixed number n of alternatives, generates all possible followup states from these states and chooses the best n configurations again. At the end, it takes the best of the n configurations as output. This way, it can select a path through the search space that is not the highest-rated one at every point in time.

shift $< S, w|I, O > \rightarrow < S|w, I, O >$; Moves the leftmost element of the input to the top of the stack.

left-arc $< S|d, h|I, O > \rightarrow < S, h|I, O \cup \{(d, h)\} >$; Attaches the topmost element of the stack to the leftmost of the input and removes the dependent from the stack.

right-arc $< S|h, d|I, O > \rightarrow < S|h|d, I, O \cup \{(d, h)\} >$; Attaches the topmost element of the stack to the leftmost of the input and *pushes the dependent from the input to the stack.*

reduce $< S|d, I, O > \rightarrow < S, I, O >$ if $(d, \_) \in O$; Removes the top element from the stack if that element already has a head.

The shift and left-arc transitions are the same as in the arc-standard system, but the right-arc transition can now be applied as soon as both head and dependent are available because the dependent is not removed from the stack. To remove an element from the stack once no more words should be attached, the arc-eager system introduces the reduce action. With this transition system, a shift-reduce parser is as incremental as it can be: it generates edges as soon as both head and dependent are available. However, all words with heads outside the current prefix are left unattached.

The number of words necessarily left unattached when parsing with the arc-eager transition system depends on the language and the annotation scheme (see Table 5.1): Parsing the Universal Dependencies treebanks incrementally with the arc-eager transition system results in a high amount of unattached words due to the annotation scheme. For example, the annotation of content verbs as heads instead of the auxiliary verbs induces additional unattached nodes for UD-HDT when looking at sentence prefixes as the content verb appears often at the end of the sentence. Even treebanks with comparably low numbers of unattached words would still be parsed with nearly one word unattached in each prefix on average, with outliers of three or more words without a head in the prefix.

We can see that the arc-eager approach produces edges earlier but still leaves a considerable amount of words unattached. This problem is more severe when parsing UD treebanks, as the annotation guidelines result in more edges going to the right. These frameworks generate no predictions at all. While the information obtainable from a transition-based parser is incomplete, that output can still be used as an additional input signal to a parser creating more complete incremental structure as shown in Köhn and Menzel (2013).

	HDT	HDT-UD	FTB	FTB-UD	PTB-CoNLL	PTB-UD
mean	1.07	1.77	0.77	1.31	0.78	1.26
median	1	1	0	1	0	1
95%-quant.	4	5	3	4	3	4

Table 5.1.: Distribution of words left unattached when performing arc-eager parsing. Numbers obtained on the training set of each treebank by counting tokens attached out of prefix.

5.2. Graph-based Incremental Parsing

The other main school of dependency parsers I experiment with are *graph-based* parsers. These do not optimize an action sequence but learn some function f that maps a dependency tree to a score. MST parser (McDonald, Pereira, et al. 2005) uses a function that decomposes over the edges of the tree, i. e., the score of a tree is the sum of scores of the edges. With y being the set of the edges and $s(d, h)$ a scoring function over a single edge, the function to be maximized is

$$f(y) = \sum_{(d,h)\in y} s(d, h) \tag{5.1}$$

A maximum spanning tree algorithm can be applied to obtain the dependency structure maximizing f using this decomposition. In contrast to transition-based parsers, an MST parser does not impose any restrictions, such as projectivity, on the dependency structure. Based on this idea, other variants of MST parser use functions that do not decompose over the edges but also score edge pairs and triplets. Exact non-projective decoding with such a function is NP-hard (McDonald and Satta 2007). Therefore, these parsers are either confined to producing structures with specific properties (the parser of Koo and Collins (2010) can only create projective structures) or to inexact decoding, i. e., the tree produced is not necessarily the one maximizing f (e. g.Martins, Smith, and Xing (2009) or Koo, Rush, et al. (2010)). Despite being restricted to approximating the optimum, these parsers perform better than the ones performing exact decoding over decomposable objective functions.

There is a graph-based parsing framework performing inexact decoding for which different approaches for incremental parsing have been implemented in the past: the

Weighted Constraints Dependency Grammar (WCDG) parser formalism (Foth, Daum, and Menzel (2004), Schröder (2002), and Foth, Menzel, and Schröder (2000)). In contrast to the parsers referenced before, it uses a manually created grammar of constraints assigning weights to parts of a tree; the optimal dependency tree, according to the grammar, is searched by repeatedly modifying an initial tree. There have been several extensions to WCDG for producing incremental structures. Foth, Menzel, Pop, et al. (2000) produced incomplete, unconnected structures for sentence prefixes. Daum (2004) introduced a special node as a stand-in for all upcoming material. Beuck, Köhn, and Menzel (2011b) and Beuck, Köhn, and Menzel (2013) developed an extension to WCDG for producing incremental dependency structures with prediction nodes as described in Section 3.2; this extension is implemented in the jwcdg parser. jwcdg will serve as a comparison to the parsers developed as part of this thesis.[4] The WCDG approach has two major disadvantages: First, it can only create dependency structures for German using a specific annotation scheme (the HDT annotation scheme (Foth 2006)). Adapting it to a different language or annotation scheme would be expensive as a new grammar needs to be written. Second, the parser is very slow. Even with optimizations and guidance from a second parser, the parsing speed is measured in seconds per word (Köhn and Menzel 2013) – too slow for interactive use.

5.3. Incremental Graph-based Parsing with Dual Decomposition

I will now introduce the first incremental parser developed for this thesis, incTP. It uses TurboParser (Martins, Almeida, and Smith 2013) as a basis. I adapted TurboParser for incremental parsing because, in contrast to most other approaches, TurboParser does not impose structural constraints on the dependency trees it generates in its core algorithm. It enforces these constraints using (modifiable) hard constraints in the linear program it generates for each parsing problem.

TurboParser is a graph-based parser that formulates the problem of finding the correct dependency structure as an integer linear program (ILP) (see Chapter 29 of Cormen et al. (2001) for an introduction to linear programming). Each possible edge is represented by a binary variable, and the ILP is formulated in such a way that every valid solution to it is interpretable as a dependency tree. Because every variable represents one (directed) edge between two words, a variable assignment can be converted into a graph by selecting

[4] jwcdg can make use of several predictors to enhance its performance, but not when parsing incrementally. Therefore, the accuracies reported here cannot be compared to the ones published for non-incremental WCDG parsing performance.

precisely the edges that have a value of one.

An ILP consists of two parts – the objective function to be optimized (f) and a set of auxiliary conditions ensuring the well-formedness of the results. Martins, Smith, and Xing (2009) managed to formulate auxiliary conditions that make sure that every valid result of the ILP corresponds to a tree structure using only a polynomial number of helper variables and conditions with respect to the input length. Previous formulations required an exponential number, making parsing with ILPs much less practical (Riedel and Clarke 2006). While the objective function does not decompose over single edges, f can be decomposed into a sum of components, each scoring on one to three connected edges. Using factors containing more edges makes the ILP harder to optimize (Riedel and Clarke 2006).

As solving an ILP is NP-hard, TurboParser approximates the result by relaxing the problem into a (non-integer) linear program using a technique called *dual decomposition*. The dual decomposition method represents the ILP to be solved as *factor graphs*. These are bipartite graphs that have variable nodes on one side and factor nodes connecting to variable nodes on the other side.[5] The factors correspond to the parts of f and the hard constraints imposed by the ILP. The set of possible assignments to the factor graphs (for variables and factors) is Y. We can define a probability distribution over Y (and therefore over all possible trees for a given input) by defining

$$P_\theta(y) = \frac{1}{Z(\theta)} exp(\theta * y) \tag{5.2}$$

with θ being the weights of the model and $Z(\theta)$ being a normalization factor. The task is to find the most probable y (corresponding to a dependency tree) given the parser model θ using maximum-a-posteriori (MAP) decoding. As the denominator $Z(\theta)$ is constant, it can be dropped to find the maximizing argument. The optimization criterion f used by TurboParser is decomposable into smaller parts, each containing a subset of the ILP variables. Each part corresponds to a weight in θ.

TurboParser performs MAP decoding by *alternating directions dual decomposition* (Martins, Smith, Figueiredo, et al. 2011): For each part of f, a sub-problem is created using new variables so that each sub-problem can be easily optimized. Variables of different sub-problems corresponding to the same variable in f are linked by an agreement constraint. For example, if a variable μ_i of f is present in the parts α and β, this variable is copied as μ_i^α in α and as μ_i^β in β. Now the parts are iteratively optimized, with each variable, e. g. μ_i^α, being penalized for deviating from the average of all variables it is linked to (in this

[5] This description is following the notation by Martins (2012), which goes into much more detail.

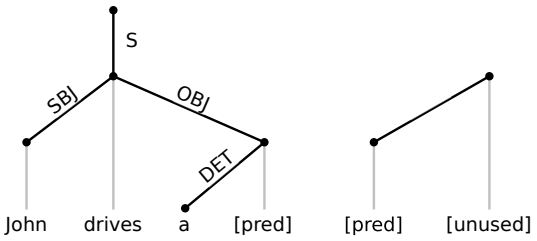


Figure 5.1.: A dependency structure with an *unused* node to denote that [pred] is not part of the analysis.

example, μ_i^α and μ_i^β should not deviate from $0.5(\mu_i^\alpha + \mu_i^\beta)$). As the process runs iteratively, the average from the previous iteration can be used for the quadratic penalizing term.

5.3.1. Performing predictive parsing with ILPs

Extending a dependency parser to incremental parsing with prediction nodes introduces a significant shift in the problem to be solved: While originally the problem was *where* to attach each word to (1), in the incremental case the additional problem arises *which* prediction nodes to include into the analysis (2). Problem (2), however, depends on the syntactic structure of the sentence prefix (i. e. on problem (1)). Therefore, it is not possible to determine the prediction nodes that should be included in the structure *before* parsing commences, but the decision has to be made *while* parsing; both problems have to be solved at the same time. As the set of nodes needs to stay fixed during parsing,[6] incTP varies the number of prediction nodes by an enhanced optimization problem that can mark nodes for deletion. These nodes are removed after the optimization step has finished. Prediction nodes are dynamically excluded by providing the parser with an additional node, named *unused*. It is always attached to the special node 0 (the root node of every analysis), and it can only dominate prediction nodes. *unused* and every prediction node it dominates is not considered part of the analysis and is removed from the parsers' output. The problem of whether a prediction node should be added to the analysis is reduced to the problem of where to attach that prediction node; Figure 5.1 shows an example.

[6] The nodes are encoded into the optimization problem, which stays fixed during parsing as parsing is exactly solving this problem.

5.3.2. Deciding on fixed sets of prediction nodes

To enable the parser to include prediction nodes into the syntactic structure it generates, a set of prediction nodes has to be provided. While this set could include any number of prediction nodes, we only include a set that covers most cases of prediction since rare prediction nodes have a very low a priori probability of being included. Additional prediction nodes make the parsing problem more complex. This set is sensitive to the annotation schema and has to be determined in advance; it can be obtained by generating incremental gold standards from a treebank as described in Section 4.3 and counting the occurrences of prediction nodes in them. Different possible sets can then be probed for the coverage they yield for the generated gold standards, i. e. the percentage of prefixes in the incremental gold standards using a subset of the prediction node set under consideration. The experiments in this chapter are carried out with language-dependent sets of prediction nodes that are a super-set for a large enough percentage ($> 90\%$) of the prediction nodes in the generated gold standards. Extending this coverage requires a significantly higher number of prediction nodes (cf. Beuck and Menzel (2013)).

The following experiments use the incremental gold standards for German and English as described in Section 4.3, both for training and evaluation. The prediction nodes are delexicalized in the following way: Their lexical identity is replaced with the string "[virtual]"[7]; the part-of-speech tag is generalized to a more coarse grained one – either to a tag representing noun-like words or a tag representing verb-like words (See Appendix C).

5.3.3. Incrementalizing TurboParser

Since in the TurboParser framework well-formedness is enforced by auxiliary constraints of the ILP, additional constraints on the shape of analyses can be imposed without changing the core algorithm of the parser. Each ILP is constructed over the in-prefix words, the maximum number of includable prediction nodes and the *unused* node. incTP uses three additional restrictions with respect to prediction nodes to enable the parser to add prediction nodes to an analysis selectively:

1. A prediction node that is attached to *unused* may not have any dependents. Otherwise, a prediction node with in-prefix dependents might be removed from the structure resulting in an ill-formed structure.

2. A prediction node may not be attached to 0 if it has no dependents. This especially prevents the prediction of isolated verbs that might otherwise happen because verbal prediction nodes are often attached to 0 in the training data.

[7] This is a reminiscence to the old name for prediction nodes — "virtual nodes" – used in previous literature.

3. Only prediction nodes may be attached to the *unused* node. This prevents in-prefix words from being removed from the dependency structure.

These constraints are enforced by adding factors to the factor graph of the ILP. Using the notation introduced in Section 3.2, for a dependency structure $\langle W, P, \pi, l\rangle$ with the *unused* node u being part of P and $z_{\langle h,d\rangle}$ representing the binary variable for the edge between a head h and a dependent d, the restrictions translate to these linear constraints with $\mathbb{I}$ being the indicator function[8]:

Using propositional logic, the three restrictions can be written as follows:

$$\neg(z_{\langle p,j\rangle} \wedge z_{\langle u,p\rangle}), \qquad p \in P \setminus \{u\}, j \in W \cup P \tag{5.3}$$

$$\bigwedge_{j \in W \cup P} \neg z_{\langle p,j\rangle} \Rightarrow \neg z_{\langle 0,p\rangle}, \qquad p \in P \setminus \{u\} \tag{5.4}$$

$$\neg z_{\langle u,i\rangle}, \qquad i \in W \tag{5.5}$$

Equations 5.3, 5.4, and 5.5 can be transformed into ILP constraints by formulating them as inequalities over integer variables:

$$\mathbb{I}(z_{\langle p,j\rangle}) + \mathbb{I}(z_{\langle u,p\rangle}) \leq 1, \qquad p \in P \setminus \{u\}, j \in W \cup P \tag{5.6}$$

$$\mathbb{I}(z_{\langle 0,p\rangle}) \leq \sum_{j \in W \cup P} \mathbb{I}(z_{\langle p,j\rangle}), \qquad p \in P \setminus \{u\} \tag{5.7}$$

$$\mathbb{I}(z_{\langle u,i\rangle}) = 0, \qquad i \in W \tag{5.8}$$

Besides adding these hard constraints to the ILP formulation, the algorithm of TurboParser does not need to be changed as incTP is run in a restart-incremental mode, i. e. each sentence prefix is translated into a new ILP to be solved without making use of previous results.

5.3.4. Training incTP

High-quality incremental parsing results can not be expected from models trained on whole-sentence annotations only. If incTP is trained on gold-standard incremental dependency structures (generated as described in Section 4.3), it includes every prediction node into every analysis because the training data does not include any non-attached prediction nodes. Therefore, the training data has to be augmented to contain non-included prediction nodes. The data can be augmented by adding prediction nodes to the generated

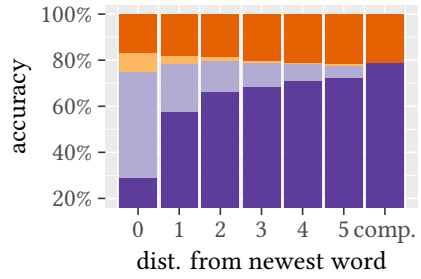
[8] That means $\mathbb{I}(z_{\langle h,d\rangle})$ is 1 if $\pi(d) = h$ and 0 otherwise.

	EN-EWT		DE (UD-HDT)		DE (HDT)		DE (jwcdg)	
	acc.	stab.	acc.	stab.	acc.	stab.	acc.	stab.
precision	79.7%	80.6%	86.0%	86.9%	65.7%	65.4%	29.7%	29.8%
avg. #pred	1.0		1.5		1.4		1.5	

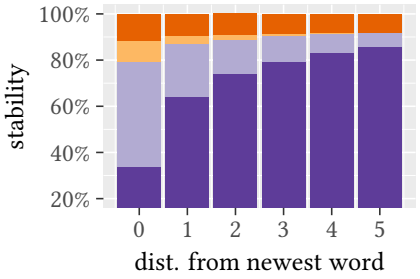
Table 5.2.: Percentage of prediction nodes that could be mapped to the gold standard (precision, unlabeled) both with respect to the gold standard (acc.) and with respect to the complete output of the parser (stab.). avg. #pred: average number of predicted prediction nodes per increment. EN-EWT: UD English EWT Treebank (Silveira et al. 2014) German: HDT with gold-standard part-of-speech tags. German (jwcdg): with predicted tags.

predictive dependency structures until they contain at least the set of prediction nodes later used during parsing; the *unused* node is always added to the training structures. For instance, for parsing the Hamburg Dependency Treebank incrementally, a set of two noun prediction nodes and one verb prediction node is used. Therefore, in each training increment that contains fewer than two noun prediction nodes, additional nodes attached to the unused node are introduced. If the increment contains no verb prediction, a verb prediction node is added and attached to the unused node. A predictive structure used for training looks like the structure shown in Figure 5.1. This approach ensures that the distribution of prediction nodes being attached to `unused` in the training data is similar to the probability they will have to be attached to `unused` in the test data. This way, incTP can learn the a priori probability that a prediction node of that type should be added to a dependency structure by the parser while parsing. incTP is trained on these extended predictive dependency structures, and no adaptation of the training algorithm is needed; the incrementality is transparent to the parser itself.

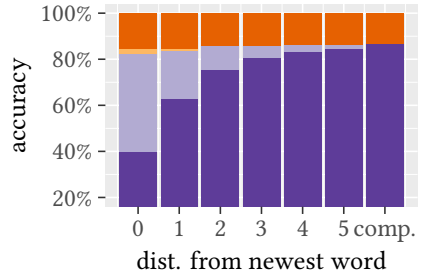
The training data is heavily skewed as words at the beginning of the sentences are more prevalent than the ones at the end. One might argue that this disproportion could deteriorate the parsing performance. However, a comparison with a version trained on non-incremental data shows that this has no noticeable effect on the parsing quality of complete sentences.



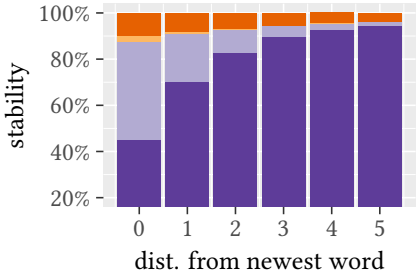
(a) Accuracy on UD-Szeged (Hungarian)



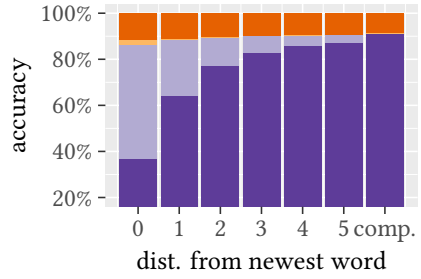
(b) Stability on UD-Szeged (Hungarian)



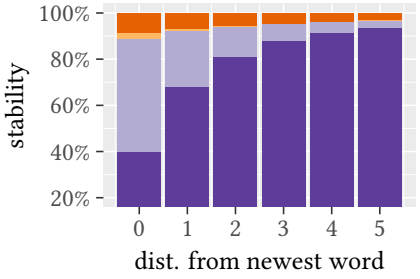
(c) Accuracy on UD-EWT (English)



(d) Stability on UD-EWT (English)

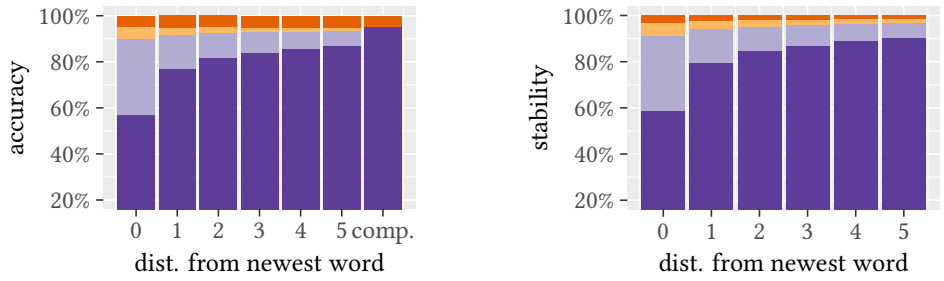


(e) Accuracy on UD-PTB (English)



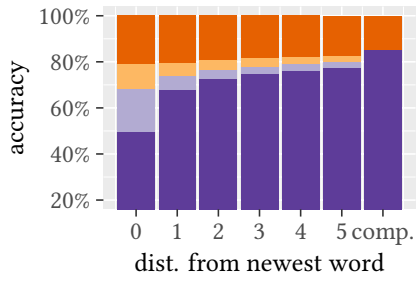
(f) Stability on UD-PTB (English)

Figure 5.2.: Results on non-German treebanks (unlabeled).
Correct: , correct prediction: , wrong prediction: , wrong:
X-axis: accuracy of the tokens N left of the newest one; comp: accuracy measured on complete sentences.

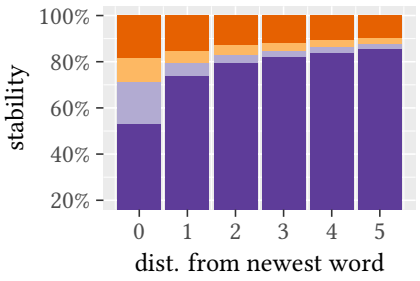


(a) Accuracy for incTP

(b) Stability for incTP



(c) Accuracy for jwcdg



(d) Stability for jwcdg

Figure 5.3.: Unlabeled results on the German HDT for incTP (with gold PoS) and jwcdg (with predicted tags).
Correct: , correct prediction: , wrong prediction: , wrong:

5.3.5. Evaluation

In this section, incTP will show its strengths using the metrics to evaluate predictive dependency parsers introduced in Chapter 4. We will see

- that it produces accurate structures for sentence prefixes including high-quality predictions.
- that the structures produced for complete sentences are as accurate as the ones by the original TurboParser (in contrast to other predictive parsing approaches, see Section 3.1.2).
- that it is fast enough for interactive use.

From all corpora, predictive dependency structures padded with unused prediction nodes have been created for training. The evaluation was performed using one prediction noun and one prediction verb for English and two prediction nouns and one prediction verb for German because these sets cover about 90% of the prefixes in both training sets. The UD treebanks were evaluated with a verb and a noun prediction node.

The only other parser implementing predictive dependency parsing as described in this thesis is jwcdg (Beuck, Köhn, and Menzel 2011b; Beuck, Köhn, and Menzel 2013), which will also be evaluated for comparison. jwcdg differs from most other parsers in that it does not act on pre-tagged data but runs an external tagger itself in a multi-tag mode (Foth and Menzel 2006).[9] A comparison between using gold-standard tags and predicted tags for incTP has shown that the output differs very little (Köhn and Menzel 2014). Therefore, even though a parser with gold-standard tags is compared with one using an automatic PoS tagger, the overall picture stays valid.

Incremental accuracy Figure 5.2 shows the evaluation results for parsing different treebanks using incTP. Figure 5.3 shows the results for parsing the German HDT with incTP and jwcdg. For all languages and treebanks except UD-Szeged, the attachment accuracy rises with the amount of context available. For UD-Szeged, which has the lowest overall accuracy, the correct attachments to a prediction node can not always be converted into a correct attachment in the prefix. Therefore the overall number of correct attachments drops slightly with increasing distance from the newest word.

The word five elements left of the newest word gets attached with an accuracy that is nearly as high as the accuracy for the whole sentence (*comp*) in all experiments. As can

[9] jwcdg can make use of additional predictors but they are not enabled in the experiments reported in this thesis as most of them do not work in the incremental parsing mode.

be seen for English, the accuracy is not only a function of the language but also depends on the corpus: these two differ in size, genre, and annotation scheme. When comparing jwcdg with incTP on the HDT, we can see that incTP performs significantly better than jwcdg. 89.9% of the newest words are attached correctly (unlabeled), whereas jwcdg only correctly attaches 68.5%. For complete sentences, the difference in accuracy is smaller but still about ten percentage points (incTP: 95.2%, jwcdg: 85.4%).

Prediction node precision and recall Regarding the prediction nodes, for both German and English, the unlabeled precision reaches more than 70% (see Table 5.2) for the UD treebanks. incTP has a lower prediction precision on the HDT schema, probably because the schema requires prediction chains not predictable by the parser.[10] Even the correct dependency label of upcoming words can be predicted with reasonably high precision. Interestingly, incTP not only has much higher precision than jwcdg, but it also achieves this while producing a similar amount of prediction nodes overall (see *avg. #pred* in Table 5.2).

Comparison to non-incremental parsing Training incTP on sentence prefixes could deteriorate the accuracy of the parser for complete-sentence analyses due to the overemphasis on the beginning of each sentence. However, this is not the case, and incTP achieves about the same accuracy for full sentences as the non-incremental TurboParser (Köhn and Menzel 2014). This also shows that the additional mechanism of prediction nodes has no adverse effects on the overall parsing accuracy.

Stability Figure 5.2 and Figure 5.3 show evaluation results with respect to the full-sentence output of each parser as a measure of stability. As the accuracy is approximately a lower bound for the stability of a parser,[11] it is not surprising that the experiments yielding the highest accuracy also yield the highest stability. incTP's stability turns out to be much higher than jwcdg's: it has a stability of 90% versus only 71% for the newest word. While the attachments of the newest words are already quite stable with about 90%, the stability for positions with more words as context to the right is even higher, with 96 to 97 percent. These high stability numbers are not that surprising because a parser with high accuracy inherently can not create unstable output – some part of that unstable output would have to be incorrect.

[10] see the discussion of *folding* in Section 4.3.

[11] If a parser has a stability of x, all the $1 - x$ changes can at most be from correct to incorrect or vice-versa and will counted as incorrect somewhere.

Speed incTP parses an increment in about 0.015 seconds, which is much faster than jwcdg where about eight seconds per word are needed to achieve a good accuracy[12] (Köhn and Menzel 2013).

5.4. Discussion

incTP shows that enhancing a processor producing structured output to be able to work incrementally is feasible with relatively little changes to the core of the processor if the task can be reformulated. This reformulation involves creating artificial gold standards for incremental input, which is unsatisfactory as it relies on linguistic intuition rather than on actual empirical data; phenomena on a lexical level can only be modeled to a limited extent. However, this fairly simple approach to incremental gold standard creation allows extending it to new languages in relatively little time. Adapting the rules to a new annotation standard (usually for a new language) takes about a day with the current framework already in place. The result is a restart-incremental processor – while performing a restart on every new word is not psycholinguistically plausible, restarts do happen in human sentence processing (Malsburg and Vasishth 2011).

Nonetheless, incTP analyzes sentences incrementally, produces connected dependency analyses at every point in time, and the intermediate structures produced are highly informative, including predictions for properties and structural embeddings of upcoming words. In contrast to previous approaches, incTP achieves state-of-the-art accuracy for whole sentences by abandoning strong monotonicity and aim at high stability instead, allowing the parser to improve intermediate results in light of new evidence.

[12] The speed of incTP and jwcdg was measured on a 48-core machine with four AMD Opteron 6168 processors. incTP used a single thread, jwcdg was run multi-threaded.

Chapter 6.

Transition-based Predictive Parsing

The previous chapter showed that it is possible to learn predictive parsers from automatically generated incremental gold standards. That approach, however, has two shortcomings: First, creating incremental gold standards requires domain knowledge (i. e., about the language and the annotation scheme). Second, even with this knowledge, the handwritten rules might be insufficient to capture all intricacies of predictability – the rules introduced in Section 4 largely ignore word identities and mainly rely on the dependency labels. Therefore, they are unable to model distributional properties of predictions such as determining whether an object should be predicted based on which verb it is governed by. Second, incTP works in a restart-incremental fashion and makes no use of previous computations. This is unsatisfactory from both a psycholinguistic point of view (as humans do not entirely reinterpret text they are reading all the time) and from an engineering point of view. Computation time is spent on re-computing structures that were thrown away instead of re-using the structure from previous increments. Additionally, a parser performing monotonic expansions has the potential to be used in experiments that force the parser to commit to a specific structure, a property useful for exploring garden-path phenomena.

This chapter introduces a parser (PreTra) without these two shortcomings: it does not need incremental gold standards as it can be trained on non-incremental ones, and it monotonically re-uses previous structures. Although PreTra uses a transition system, it produces predictive and connected structures – in contrast to standard shift-reduce based parsers (see Section 5.1).

6.1. A Transition System for Predictive Parsing

Transition-based dependency parsers in the line of Nivre (2003) combine a transition system with a fixed set of transitions with a classifier that rates this fixed set in a specific configuration. All such shift-reduce parsers generate edges in a more or less delayed

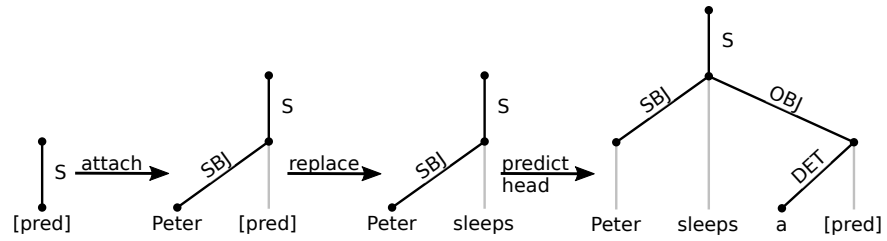


Figure 6.1.: A sequence of consecutive predictive annotations. The transitions – denoted as labels on the arrows between the annotations – are a subset of the ones described in Section 6.1.2.

fashion.[1] In contrast, a predictive parser should never delay an edge creation.

A shift-reduce parser usually employs a classifier to decide which action to take. To obtain a predictive parser working with a similar model, the generation of predictive structures would need to be reduced to a sequence of actions from a fixed action set. However, since the incremental structures determine both starting and end points for transition sequences (see Figure 6.1), such a fixed set of actions is hard to define. Whenever a new word is introduced, it can be attached to an existing node in the structure (where there is a varying number of candidates, see the first transition) it can replace one of the existing prediction nodes (of which there is a varying number, see the second transition), or it can require adding a new prediction node and include it somewhere (see third transition).

A shift-reduce parser does not score a tree directly. It instead generates the score via the transition sequence that created the tree. Several sequences might lead to the same tree (yielding different scores for the same structure), and training a parser should treat all possible paths leading to correct output as correct, see Goldberg and Nivre (2013)).

There is one piece of work that attempted to directly score (parts of) the dependency structure instead of only the actions: Bohnet and Kuhn (2012) augmented a transition-based parser with a graph-based completion model, which re-scores the structures in a beam[2] using features similar to the ones of TurboParser. This approach needs book-keeping about what parts of the sentence are already completely parsed and scored and in which parts new edges may be introduced later on. This approach improves upon only using a transition system and has an accuracy similar to graph-based parsers such as

[1] See Section 5.1.

[2] See definition in Section 6.1.1.

Martins, Smith, Xing, et al. (2010).

When designing a predictive parser, the intermediate states are tree structures. Therefore, it is possible to score the trees themselves instead of the actions without tracking partial structures. Once the actions are not scored at all anymore, the number of transitions can vary depending on the current state and allow a wide range of attachments, enabling to cover a wide range of structures. If several transition sequences yield the same tree structures, these structures by design have the same score and can be unified into a single output (as the transition sequence is of no interest).

In other words, when designing an incremental predictive parser, it is possible to use a transition system in which only the resulting dependency structures are scored, and not the transitions themselves. Without the need to classify the transitions, the set of transitions does not need to be fixed anymore; tree scoring can be completely decoupled from the transition system.

6.1.1. Overall Structure of the Transition Parser

PreTra employs beam-search between states. Starting from an initial state, it repeatedly creates new states using a transition function and performs a cut-off to obtain a beam:

Definition 8. A *beam* $\mathcal{B}_i$ is the list of the top N predictive structures according to a scoring function f for a given sentence prefix $w_1 \dots w_i$.

The parser always starts from a beam for the empty sentence prefix $\mathcal{B}_0$, which only contains a single structure consisting of a single prediction node attached to root (see Figure 6.1). For each following prefix $w_1 \dots w_i$, the parser produces a corresponding set of unlabeled predictive dependency structures[3] $\mathcal{S}_i \subset \langle w_1 \dots w_i, P, \pi \rangle$ based on the beam entries $\mathcal{B}_{i-1}$ for $w_1 \dots w_{i-1}$ by applying a transition function t to the new word w_i and every element of the beam:

Definition 9. A *transition function* t takes a predictive structure for the words $w_1 \dots w_{i-1}$ and a word w_i to include into that structure. It produces a set of followup structures containing that word:

$$t : \langle w_1 \dots w_{i-1}, P, \pi \rangle, w_i \rightarrow \mathbb{P}(\langle w_1 \dots w_i, P\text{'}, \pi\text{'} \rangle) \tag{6.1}$$

π' is an extension of π, i. e. the head of a node is only changed if the previous head was a prediction node that was replaced by w_i. In that case, the new head is w_i.

[3] See Definition 5 on page 40, but we will omit the labeling for now.

The followup states of a beam $\mathcal{B}_{i-1}$ given a new word w_i is simply $\mathcal{S}_i = \cup_{b \in \mathcal{B}_{i-1}} t(b, w_i)$.

The new beam is then formed by obtaining a score for every element of $\mathcal{S}_i$ using a scoring function f and the top N entries form the beam $\mathcal{B}_i$:

Definition 10. A *scoring function* f assigns a score to a predictive dependency structure:

$$f : \langle W, P, \pi \rangle \to \mathbb{R} \tag{6.2}$$

Scoring functions may be compositions of other scoring functions, e. g. by summing them ($f(s) = f_1(s) + f_2(s)$) or by chaining them: $f(s) = f_1(s, f_2(s))$

All that is needed to have a complete parser is a concrete transition function t and a scorer f.

6.1.2. The Transition System

A transition system in this framework has to provide a predictive structure for the empty input and a transition function t. As an initial state, we will use $\langle \emptyset, ([pred]), \{1 : 0\} \rangle$, i. e. a state containing no words and one prediction node that is attached to the root. This is a valid prediction as every dependency structure needs to have at least one element that is a root.

t should have a high coverage, i. e. the gold standard dependency tree for a sentence $w_1 \dots w_n$ needs to be derivable by n applications of t starting from the initial state for (nearly) every sentence. Without a high coverage, a parser using t will inherently not be able to produce accurate output. At the same time, the cardinality of the output should be reasonably small for efficiency reasons. The transition system I propose is built from parameterizable functions each mapping an input structure $s = \langle w_1 \dots w_{i-1}, P, \pi \rangle$, the new word w_i, and additional parameters to an output structure. These functions are the following, with parameters in parentheses:

attach$^s_{w_i}$(h) Attaches the new word w_{i+1} to an already existing word or prediction node $h \in W \cup P$:

$$\langle w_1 \dots w_i, P, \pi \rangle, w_{i+1}, h \mapsto \langle w_1 \dots w_i w, P, \pi \cup \{w_{i+1} : h\} \rangle$$

This is the simplest operation, e. g. for attaching an object to a verb in the prefix or an adjective to a predicted noun.

predictHead$^s_{w_i}$(h) Creates a new prediction node p , attaches the new word w_{i+1} to p and attaches p to an already existing word or prediction node $h \in W \cup P$:

$$\langle w_1 \dots w_i, P, \pi \rangle, w_{i+1}, h \mapsto \langle w_1 \dots w_{i+1}, P \cup \{p\}, \pi \cup \{w_{i+1} : p, (p, h)\} \rangle$$

This operation is needed if e. g. w_i is a determiner and the noun to this determiner needs to be predicted.

predictTwoHeads$^s_{w_i}$(h) Creates two new prediction nodes p_1 and p_2 , attaches the new word w_{i+1} to p_1, p_1 to p_2 and p_2 to $h \in W \cup P$:

$$\langle w_1 \dots w_i, P, \pi \rangle, w_{i+1}, h \mapsto \langle w_1 \dots w_{i+1}, P \cup \{p_1, p_2\}, \pi \cup \{w_{i+1} : p_1; p_1 : p_2; p_2 : h\} \rangle$$

This operation is needed for starting subordinate structures that are deeply nested.

replacePrediction$^s_{w_i}$(p) substitutes prediction node p with the new word w_i, inheriting all dependency relations:

$$\begin{aligned} \langle w_1 \dots w_{i-1}, P, \pi \rangle, w_i, p \mapsto \langle & w_1 \dots w_i, \\ & P \setminus \{p\}, \\ & \pi \setminus \{(p, h) : (p, h) \in \pi\} \\ & \quad \setminus \{(d, p) : (d, p) \in \pi\} \\ & \quad \cup \{(w_i, h) : (p, h) \in \pi\} \\ & \quad \cup \{(d, w_i) : (d, p) \in \pi\} \rangle \end{aligned}$$

When evaluating these functions with all possible values for their parameters, we obtain a set of successor structures (dependency structures for $w_1 \dots w_i$) from a single dependency structure for $w_1 \dots w_{i-1}$:

$$\begin{aligned} \text{attach}^s_{w_i} &= \{\text{attach}^s_{w_i}(h) : && h \in W \cup P\} \\ \text{predictHead}^s_{w_i} &= \{\text{predictHead}^s_{w_i}(h) : && h \in W \cup P\} \\ \text{predictTwoHeads}^s_{w_i} &= \{\text{predictTwoHeads}^s_{w_i}(h) : && h \in W \cup P\} \\ \text{replacePrediction}^s_{w_i} &= \{\text{replacePrediction}^s_{w_i}(p) : && p \in P\} \end{aligned}$$

We can now define the set of successor structures $\text{succ}(s, w)$ given a current structure s and a word w to be included as follows:

$$\text{succ}(s, w) = \text{attach}^s_w \cup \text{predictHead}^s_w \cup \text{predictTwoHeads}^s_w \cup \text{replacePrediction}^s_w$$

Using these functions, a predictive dependency structure can both be represented as a tuple $\langle W, P, \pi \rangle$ and as a sequence of applied transitions. For example, the last structure depicted in Figure 6.1 can be both represented as the tuple

$$\langle (\text{Peter}, \text{sleeps}, \text{a}), ([\text{pred}]), \{1:2, 2:0, 3:4, 4:2\} \rangle$$

and as a sequence representation:

$$\text{predictHead}(\text{replacePrediction}(\text{attach}(\text{emptyState}, 1, \text{Peter}), 0, \text{sleeps}), 2, \text{a})$$

This duality will be used for an optimization in Section 6.1.4

With the definition of *succ*, we have a transition function that can be used by PreTra to generate successor structures from on a predictive dependency structure and a word to be integrated into that structure. All generated structures are connected and contain prediction nodes if they are needed for connectedness. However, when using *succ* in a parser, this parser can not perform pure top-down prediction, i. e. it can only create prediction nodes that have a word from the prefix as (transitive) child.

6.1.3. Extending the Transition System to Perform Top-down Prediction

An enhancement to *succ* is needed to add the ability to perform top-down predictions, for which we will use an additional parametrizable function.

predict^s(h) creates a new prediction node p and attaches it to an already existing word or prediction node $h \in W \cup P$:

$$\langle w_1 \dots w_i, P, \pi \rangle, h \mapsto \langle w_1 \dots w_i, P \cup \{p\}, \pi \cup \{(p, h)\} \rangle$$

In contrast to the functions defined in the previous section, *predict* does not introduce a new word; the input structure and the resulting structure are structures for the same sentence prefix. Again, we define the set of all possible resulting structures:

$$\text{predict}^s = \{\text{predict}^s(h) : h \in W \cup P\}$$

This function can be extended to work on sets of structures S instead of single structures:

$$\text{predictmulti}(S) = \bigcup_{s \in S} \text{predict}^s$$

Depending on the current sentence prefix, different numbers of top-down predictions might be reasonable. Therefore, it is necessary to apply top-down prediction repeatedly. On the other hand, the set of predictive dependency structures needs to stay finite, requiring an upper bound for the number of prediction nodes. We will therefore define a function *succp* that repeatedly performs top-down prediction on predictive structures up to a fixed number n of prediction nodes and evaluates to all the structures generated by this procedure, including the intermediate ones.

With $\bigcirc^n f(x)$ being the n times repeated application of f on x (and $\bigcirc^0 f(x) = x$), *succ* can be augmented to include top-down prediction by adding zero to n prediction nodes to each structure generated by *succ*:

$$\text{succp}(s, w) = \{x : x \in \bigcup_{i=0}^{n} \bigcirc^i \text{predictmulti}(\text{succ}(s, w)) \,.\, ||P^x|| \leq n\}$$

The additional requirement $||P^x|| \leq n$ ensures that the number of prediction nodes in the resulting structures does not exceed n.

This transition function can produce structures similar to the ones produced by *incTP* (Section 5.3) and will be used throughout this chapter. A (predictive) structure can be represented in two ways in this framework: either by its dependency tree or by the sequence of actions that lead to the current structure.

6.1.4. Optimizations

While the definition of *succp* is sufficient from a theoretical point of view, the speed of the parser can be increased by slightly changing aspects of the transition system. These extensions do not alter the overall system but allow the parser to work with fewer trees while still obtaining the same overall accuracy.

PoS-based filtering *succp* potentially generates a very high number of successor states as every new word can be attached anywhere in the prefix either directly (using *attach*), with a new head (using *predictHead*) or even two predicted heads (using *predictTwoHeads*). Most of these attachments could be easily classified as incorrect, based only on the part of speech of the new word. For example, a determiner in German or English will never be attached to an adjective or another determiner. Following Bohnet and Kuhn (2012),[4] PreTra performs PoS-based filtering of edges. A word can only be attached to another word if the combination of head-PoS and modifier-PoS with the same direction has been

[4]Who attribute this optimization to Johansson and Nugues (2008), who, however, do not describe this approach in their paper.

observed at least once in the training data. A word may only be attached to a prediction node if some word with the same PoS was attached at least once to a word to the right of it. This filtering reduces the number of trees that have to be scored drastically while being very conservative in the pruning strategy: It is highly unlikely that a correct attachment is filtered this way, given enough training data.

Ephemeral top-down predictions The top-down predictions limit the probability of deriving a correct tree as an incorrect prediction of a prediction node can never be undone in subsequent analyses. The only way to recover from an incorrect prediction is to rely on beam search and hope that the version without this prediction is in the beam as well. If this alternative is not in the beam, it is advisable for the parser to not perform top-down prediction as the potential negative impact is much higher than the potential positive one. To not discourage top-down prediction, they are interpreted as being ephemeral, i. e., they are removed again before continuing parsing with the next word. This is achieved by using the sequence of actions that led to the tree structure and removing all top-down actions at the end of the sequence. As top-down predictions are always performed last, this results in a state where all top down predictions are removed to which no in-prefix words were attached. As *succp* generates new top-down predictions for the newly generated structures, the top-down predictions just removed are reintroduced in some of the generated structures to be scored.

6.2. Scoring Predictive Dependency Structures

With a transition function defined, all that is missing for a working parser is a scoring function to sort the resulting states. Because the scoring function is only used after the dependency trees have already been created, the function does not need to be decomposable. I will discuss two approaches to scoring: One essentially re-uses the scoring of incTP, making it possible to compare the restart-incremental approach to the transition-based one while keeping the scoring component as similar as possible. The other employs a neural-network-based scoring and shows that scoring composition is trivial using this approach of separating the transition system from the scoring component.

6.2.1. incTP-based Scoring

incTP introduced in Section 5.3 already has a well-defined scoring system composed of a sum of subgraph scores. This scoring mechanism was also used in other parsers, such as RBGParser (Zhang, Lei, et al. 2014), whose implementation is re-used for the experiments

in this section to extract feature vectors. The scorer uses a hash kernel (Bohnet 2010) to map each feature (i. e., a specific combination of edges) to an index up to a fixed upper bound. This defines a feature mapping:

Definition 11. A *feature mapping* $\phi(s)$ maps a predictive dependency structure s to a fixed-length feature vector $\{0,1\}^n$, where each dimension stands for a specific feature and the value of that dimension is 1 iff that feature is present in s.

To obtain a score for a dependency structure, its feature vector is multiplied with a learned weight vector $\vec{w} \in \mathbb{R}^n$. The resulting scoring function f_{TP} is simply the following:

$$f_{\mathrm{TP}}(s) = \phi(s) * \vec{w} \tag{6.3}$$

While the feature extraction and score lookup are reasonably fast, computing the feature vector and multiplying it with $\vec{w}$ still takes the majority of computing time of parsing. This score computation is sped up by caching the feature indices and corresponding weights for sub-graphs and reusing them when scoring other trees in the beam containing the same sub-graph. Such a cache can be re-used across different beam entries for the same beam as well as across structures for subsequent prefixes.[5]

6.2.2. NN-based Scoring

Other scoring functions can be used as well, and these do not need to be decomposable. However, non-decomposable scoring functions still need to be fast enough; initial experiments with child-sum tree-LSTMs (Tai, Socher, and Manning 2015) for scoring beam entries were promising but too slow for performing large-scale experiments with a speed of about six seconds per word. Instead, the second scoring evaluated here follows the BiLSTM-approach by Kiperwasser and Goldberg (2016).

Feature extraction Feature extraction is performed by LSTMs (Hochreiter and Schmidhuber 1997) with input gates and forget gates coupled (Cho, Merriënboer, et al. 2014) and peephole connections, which has shown to work well despite reduced complexity (Greff et al. 2017). The LSTMs maps a sequence of input vectors $i_1 \ldots i_n$ to a sequence of output vectors of the same length $o_1 \ldots o_n$. Due to the recurrent nature of the LSTM, an output o_j contains information about the inputs $i_1 \ldots i_j$. A BiLSTM employs two such encoders: one encoding the sentence prefix from left to right and one from right to left. The final

[5] The second optimization is not possible during training as the scores change between prefixes.

representation of the BiLSTM for the token at position j is the concatenation ($\circ$) of both BiLSTM outputs at position j, encoding both the left and the right context.

The inputs to the BiLSTM have to be vectors as well; these are pre-trained word embeddings e_w for the words W_i concatenated with learned PoS embeddings e_p:

$$i_j = e_w(W_j) \circ e_p(\text{pos}(W_j)) \tag{6.4}$$

Prediction nodes are not part of the prefix sequence and are not sequentially ordered. Therefore, they are not fed into the BiLSTM. Instead, a fixed representation with the same dimensionality as the output of the BiLSTM is learned and used for prediction nodes.

The forward LSTM can be reused from prefix to prefix, but the backward LSTM has to be recomputed for every prefix. Therefore, computing scores with this scoring function becomes proportionally more expensive with longer prefixes even when the computations for the previous prefix can be reused.

Tree scoring An edge with the head index being h and the dependent one d is scored by a simple multi-layer perceptron, with W^0, W^1, W^2 and b being the learned weights[6]:

$$\text{edgescore}(d, h) = W^0 \cdot \tanh(\text{BiLSTM}(h) \cdot W^1 + \text{BiLSTM}(d) \cdot W^2 + b) \tag{6.5}$$

The scoring function is the sum of all edge scores:

$$f_{\text{NN}}(s) = \sum_{d \in W \cup P} \text{edgescore}(d, \pi(d)) \tag{6.6}$$

The parameters to be optimized when training PreTra with f_{NN} are W^0, W^1, W^2, b, and the BiLSTM parameters.

The BiLSTM-based scoring consists of the sum of first-order edge scores, as in McDonald, Pereira, et al. (2005). The scorer is implemented using DyNet (Neubig et al. 2017). Similar to the incTP-based scoring, a caching mechanism makes sure that the results of overlapping computations between elements of the beam are reused. However, due to the infinite context introduced by the BiLSTMs, no sharing is possible between increments.

To evaluate whether this mode of scoring is also beneficial to predictive parsing, the PTB-based treebanks were parsed using the best-performing word embeddings from the

[6] The edge score might look unfamiliar, but this is only an optimization from the standard $(\text{BiLSTM}(h) \circ \text{BiLSTM}(d)) \cdot W$: rewriting the formula to $\text{BiLSTM}(h) \cdot W^1 + \text{BiLSTM}(d) \cdot W^2$ allows to reuse both parts of the sum between different edge score computations. This optimzation was taken from Kiperwasser and Goldberg (2016).

parser evaluation in Köhn (2016). These are dependency based[7] word embeddings (Levy and Goldberg 2014) trained on a Wikipedia corpus (Al-Rfou', Perozzi, and Skiena 2013). It turned out that the $f_{\mathrm{N}N}$ scorer performs worse than the $f_{\mathrm{T}P}$ one. The results will be discussed in Section 6.4.1 after first describing how to train PreTra.

6.3. Training PreTra

To train a parser, model updates have to be carried out. In non-incremental parsing, these training updates are performed against the gold-standard. incTP is trained on incremental gold standards; we, however, want to train an incremental parser without the need for an incremental gold standard. This section will introduce a method to select suitable trees to perform training updates against.

Because PreTra runs on a transition system, not all possible structures for a given sentence prefix can be constructed. Even if an incremental gold standard would be available, it still could happen that the gold-standard structure is not part of the generated trees. In this case, the structure with the least amount of errors with respect to the gold standard should be ranked highest by f. Therefore, all structures $s \in \mathcal{S}$ generated by the transition system need to be matched against the incremental gold standard to obtain the number of attachment errors (see Sections 4.1 and 4.4). However, as even with incremental gold standards, a matching is needed, we can make use of the matching mechanism to train PreTra on non-incremental gold standards and get rid of the need for incremental gold standards altogether with only minor adjustments.

6.3.1. Performing Updates Against Complete Sentence Annotations

Incremental gold standards have two drawbacks: The rules for them have to be manually created, and the structure they encode is not necessarily optimal (see the beginning of this chapter). By updating against complete sentence annotations, both problems go away. With the decoupling between candidate creation and scoring, this becomes possible with only minor adaptations of the parser.

Instead of mapping against an incremental gold standard to obtain the set of structures with the least amount of errors, this mapping is performed against the complete sentence structure, just as for evaluating predictive parsers (see Section 4.1). During training, each generated structure $s \in \mathcal{S}$ is not only scored, but also the number of mismatches to the

[7] "dependency based" only means that depdency structures are used to create the embeddings; the embeddings behave just as others when using them.

gold standard $e(s)$ is recorded.[8] Using a passive-aggressive update schedule (Crammer et al. 2006), the model is only updated if the highest-scoring tree is incorrect. In this case, the model is updated using loss-augmented inference (Taskar et al. 2005): instead of performing a training update from a least-error structure against the best-scoring structure, the update is performed using a large-margin objective: the score of a structure should decrease with the number of errors. Therefore, the number of errors are factored into the decision about the structure the update should be based upon. The update is performed against $s' = \arg\max_{s \in \mathcal{S}} f(s) + e(s)$. As the large margin objective gives preference to structures with many errors with respect to the target structure, weight updates are performed more often compared to using the structure with the highest score as a target. This approach proved to be beneficial to the training procedure of PreTra, leading to higher overall accuracy.

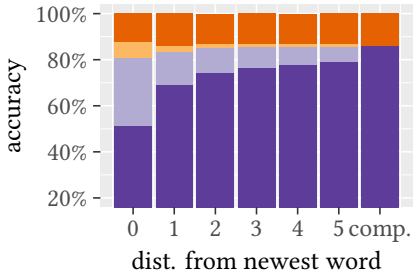
As there is no incremental gold standard to compare against, one of the structures with the least number of errors $l = \arg\min_{s \in \mathcal{S}} e(s)$ out of the current beam is used as training objective instead.

The model is then updated by increasing the weights of features in l but not in s' and decreasing the weights of features in s' but not in l. That means, for f_{TP}, an update for $\vec{w}$ is performed to increase $(\phi(l) - \phi(s')) * \vec{w}$ and for f_{NN}, the weights are updated to increase $f_{NN}(l) - f_{NN}(s')$. After such a training update, l is forced to stay in the beam to prevent updates based on spurious followup errors due to the inability to continue structure building from a good previous state.

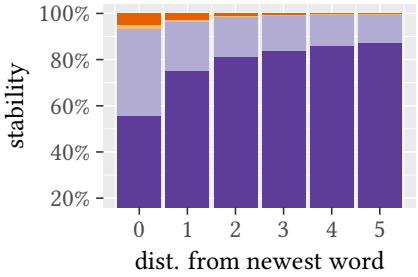
6.4. Experimental Results

PreTra is evaluated on the same set of treebanks as incTP; all results can also be found in a more detailed form in Appendix B. This selection allows us to inspect several properties: the performance of a restart-incremental parser (incTP) can be compared to a parser bound by a transition system (PreTra). Treebanks that are annotated with several annotation schemas enable us to assess which of these schemas is better suited for parsing with the proposed transition system – this mainly concerns content-head versus function-head annotation.

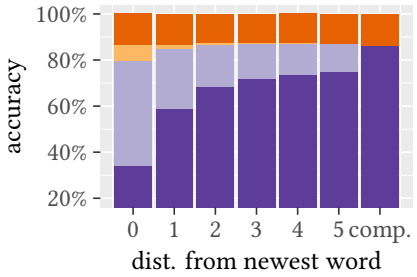
[8] Mismatches of prediction nodes are counted with a factor of 0.3 (chosen ad-hoc) to make them less severe than incorrect attachments in the prefix because prediction node attachment also depends on the mapping chosen.



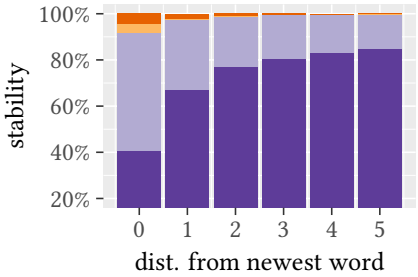
(a) Accuracy on the HDT (German)



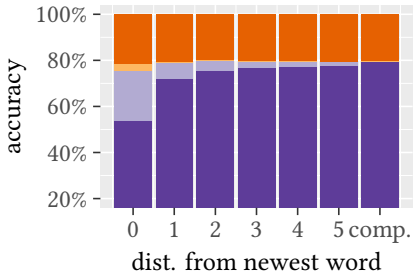
(b) Stability on the HDT (German)



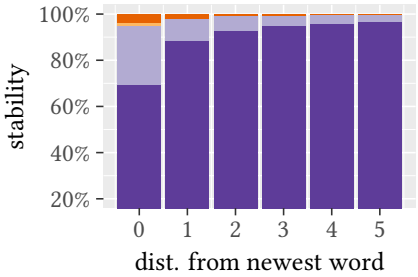
(c) Accuracy on the UD-HDT (German)



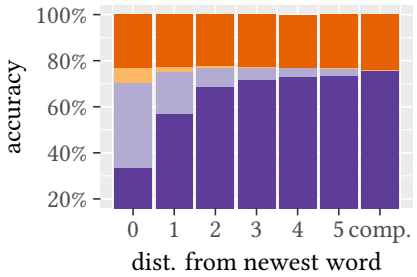
(d) Stability on the UD-HDT (German)



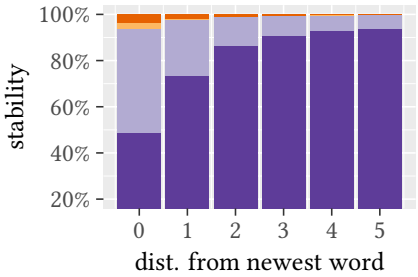
(e) Accuracy on the FTB (French)



(f) Stability on the FTB (French)

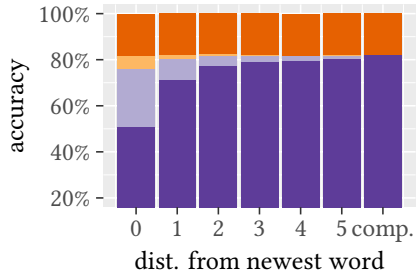


(g) Accuracy on the UD-FTB (French)

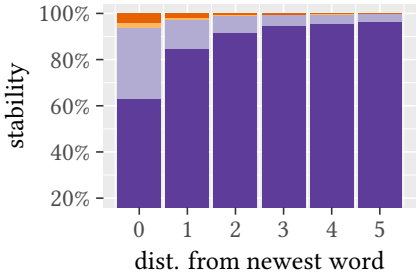


(h) Stability on the UD-FTB (French)

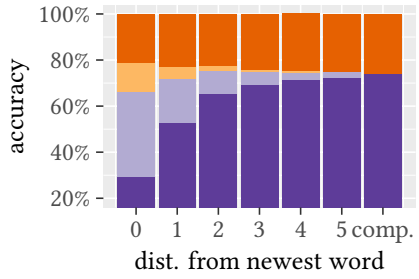
Figure 6.2.: Results of PreTra German and French treebanks.
Correct: , correct prediction: , wrong prediction: , wrong:



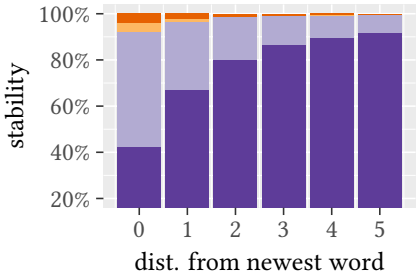
(a) Accuracy on the PTB (English)



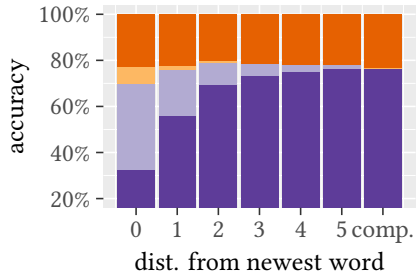
(b) Stability on the PTB (English)



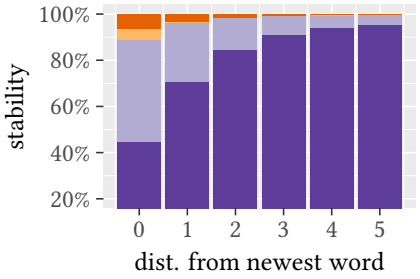
(c) Accuracy on the UD-PTB (English)



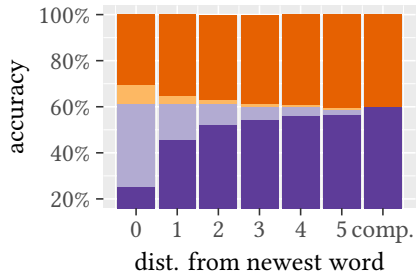
(d) Stability on the UD-PTB (English)



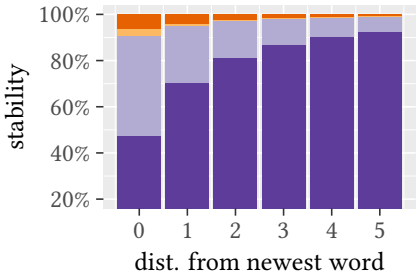
(e) Accuracy on the UD-English-EWT



(f) Stability on the UD-English-EWT



(g) Accuracy on the UD-Szeged (Hungarian)



(h) Stability on the UD-Szeged (Hungarian)

Figure 6.3.: Results of PreTra on English and Hungarian treebanks.
Correct: , correct prediction: , wrong prediction: , wrong:

6.4.1. Impact of Hyper-parameter Selection

There are several hyper-parameters that can be tuned: the scorer can use different sets of features, the beam size can be adjusted, the neural net scorer can be used to augment the f_{TP} and the parser can either be forced to keep top-down restrictions or not (see Section 6.1.4).

Exploring all possible combinations of hyper-parameters would lead to a combinatorial explosion, I therefore restricted the comparisons to the effects of single hyper-parameters. Using a reduced feature set[9] instead of the full one leads to a reduction of 0.5 to 1 percentage points in accuracy. Loss-augmented selection of the structure to perform an update against (as described above) increases the accuracy by at least 0.5 percentage points – the detailed results are shown in Appendix B. Interestingly, the beam size does not influence the parser accuracy much, a beam size of 10 yields similar results as a beam size of 50 does.

The NN-based parser achieves lower accuracies than f_{TP}, probably because it cannot distinguish between different edge combinations consisting only of prediction nodes. It achieves a full-sentence accuracy of 65.59% on UD-PTB and 71.98% on the PTB, whereas the TurboParser-inspired scoring achieves accuracies of 74.31% and 82.06%, respectively. Therefore, all other evaluations are performed using f_{TP} as a scorer.

6.4.2. Evaluating PreTra

Because PreTra does not rely on incremental gold standards, it can be evaluated on more treebanks than incTP; the results of PreTra can be seen in Figure 6.2 and Figure 6.3. While PreTra does not reach the accuracy of incTP, it still manages to parse with moderately high accuracy and high stability. Interestingly, the accuracy on the HDT is (in comparison to incTP) much lower than on the other treebanks: with a full-sentence accuracy of only 86.0% (incTP: 95.3%). In contrast, the difference on the UD annotated version of the HDT is smaller, with the accuracy of PreTra being higher (86.2%) and the accuracy of incTP being lower (95.0%) than on the original HDT annotations. Possible reasons will be discussed in the next section.

In contrast to incTP, the incremental accuracy does not rise but stays similar across the distances to the newest word for most treebanks. This indicates a reduced ability to recover from errors due to the beam search.

PreTra is quite fast with about 3ms processing time per word,[10] which is several times faster than incTP. The speed can be attributed to the parallelizable architecture and the

[9] The reduced feature set is not using great-grandparent, parent-sibling-child, and global feature templates.
[10] measured on an Intel Xeon E5-2630 v3 @ 2.40GHz

ability to re-use computation results both between elements in the beam and from previous increments. This speed enables the use in interactive systems without noticeable delay.

6.4.3. Search Errors Due To Beam Search

It is not surprising that PreTra is unable to match the performance of incTP as it is bound by the beam search and can not arbitrarily correct itself. The errors made by the parser can be due to three different reasons:

1. A correct structure is in the beam, but the scoring function rated a different one higher.
2. The transition system is unable to build the correct structure.
3. The transition system could have built the correct structure in principle, but a preceding structure to the one currently needed fell off the beam.

To see the impact of the third point, we can force the structure with the least amount of errors with regards to the gold standard to stay in the beam. This way, the parser is only limited by the quality of the scoring function and the coverage of the transition system. Evaluation results for this setting are shown in Figure 6.4. Unsurprisingly, the accuracy increases in this setting. More interesting is that this increase strongly depends on the treebank: While PreTra obtains a very high full-sentence accuracy on the FTB and PTB treebanks (FTB: 98.6%, UD-FTB 98.0, PTB: 97.6%, UD-PTB: 96.9%), the full-sentence accuracy on the HDT – 94.6% – is much lower and even lower than the results obtained by incTP (which does not have any guidance by the gold structure). As the scoring component of incTP and PreTra are very similar, this indicates that errors were introduced because the transition system is unable to generate the correct structures and that the annotation scheme by the HDT is not a good match for monotonic expansions of dependency structures. This mismatch can be observed in practice: for example, the HDT schema annotates verb chains in subordinate clauses such as “promoviert haben können hätte” (*to-do-a-Phd have could have*) as a chain with each verb being the child of the verb directly to the right of it. The subject of the verb chain is attached to the topmost element of the chain, whereas an object preceding the verb chain is attached to the lowest element of that chain. Therefore, a predictive parser only performing monotonic extensions (such as PreTra) would have to predict the exact number of verbs in the verb chain to create a correct and complete parse. This non-predictability is the reason why folding is employed when generating a gold standard for the HDT, as discussed in Section 4.3). The UD annotation does not have the same problem because the content bearing verb is the head and subject, objects, and all auxiliary verbs are attached to that verb.

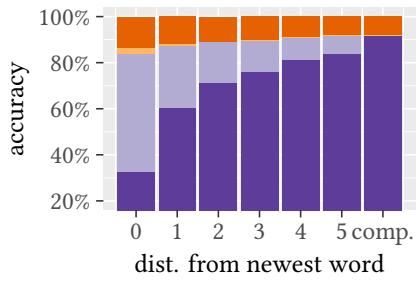
The high accuracies on the French and English treebanks show that the transition system can cover nearly all of the syntactic phenomena in those languages: high accuracies can only be achieved if the transition system creates fitting structures most of the time.

While forcing correct structures to stay in the beam is helpful to evaluate both the fit of the transition system and whether the scorer can pick the correct structures, the results should not be interpreted as the accuracy of PreTra with an infinite beam size. The very high accuracy (compared to incTP) stems from the fact that PreTra only has to select among a relatively small number of candidate structures for each increment.

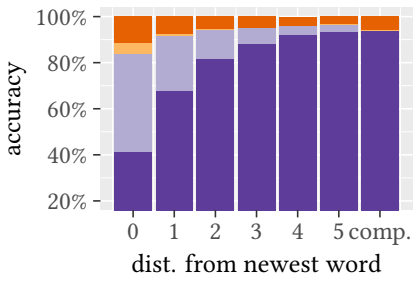
6.5. Summary and Discussion

In this chapter, I introduced a transition-based predictive parser, which, to my knowledge, is the first that is fully language-independent, can parse all input, provide accurate output, and uses little computation time at the same time.

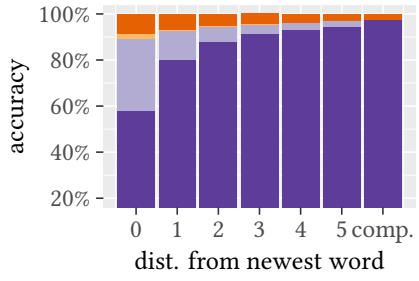
As the scoring function learned is independent of the transition system, it could, in principle, also be used to find structures by performing sampling techniques similar to the ones described by Zhang, Lei, et al. (2014). In that scenario, the transition system would be used during training to provide a restricted set of candidates, enabling matching to a non-incremental gold standard. As this matching is not needed when parsing, an unrestricted restart-incremental sampling could be used. From a psycholinguistic standpoint, this can be seen as reanalysis due to a comprehension failure (see Section 1.1). The reanalysis could also be triggered by an external critic to switch between transition-based and restart-incremental parsing. Preliminary experiments show that the accuracy of sampling-based parsing leads to low accuracy as it creates sub-structures not allowed by the transition system (e. g., multiple roots or having adjectives as roots) because they were never penalized when training with the transition system. Thus, the sampling needs to be restricted to structures similar to those the transition system can generate, or the scorer needs to be trained on sampled structures as well.



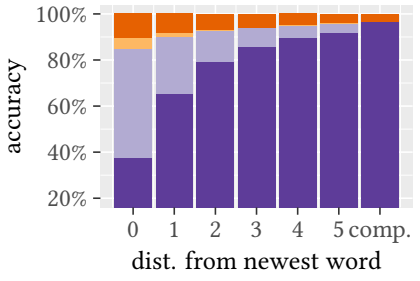
(a) Accuracy on the UD-Szeged (Hungarian)



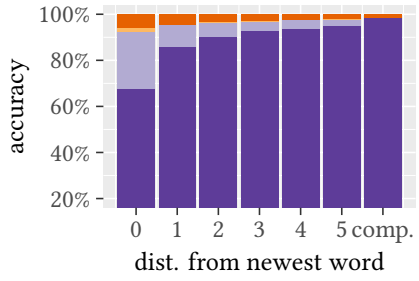
(b) Accuracy on the UD-EWT (English)



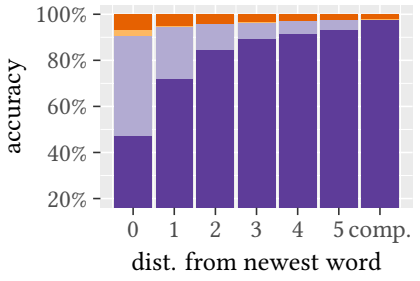
(c) Accuracy on the PTB (English)



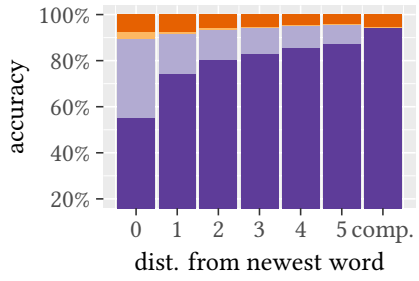
(d) Accuracy on the UD-PTB (English)



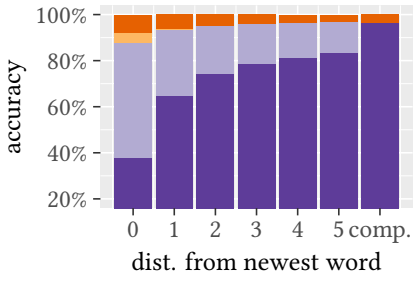
(e) Accuracy on the FTB (French)



(f) Accuracy on the UD-FTB (French)



(g) Accuracy on the HDT (German)



(h) Accuracy on the UD-HDT (German)

Figure 6.4.: Results of PreTra when forcing a structure with least errors to stay in the beam.

Chapter 7.

Incremental Parsing for Language Modeling

The previous two chapters evaluated predictive parsers with respect to other predictive parsers. This chapter shows that predictive parses – and especially the prediction nodes – can be useful for other processors. Language modeling will be used as a downstream process, a task in which (in most scenarios) the probability distribution of the next word given a prefix should be computed. Integrating the parser's predictions into N-gram language models also shows that the prediction nodes contain long-range information exceeding the restricted N-gram history.

Consider an incremental natural language understanding system with speech as input and a dependency parser somewhere in the understanding pipeline, for example, as part of a larger dialogue system. It will consist of a speech recognizer that generates hypotheses of what is said, followed by a text-based understanding system. Speech recognizers use language models to decode audio into text. Even though neural language models provide superior perplexity to n-gram language models, they are only used for post-hoc rescoring because they are computationally too expensive. While decoding takes place, n-gram models are still used because their speed is superior.[1]

7.1. Language Modeling

Language models define a probability distribution over sequences $P(w_1 \dots w_n)$. As this combined distribution can not easily be approximated from text, the decomposition based on the chain rule is used:

$$P(w_1 \dots w_n) = \Pi_{i=1}^{n} P(w_i | w_1 \dots w_{i-1})$$

[1] It is hard to find a source to show that something is *not* done – the current mainstream speech recognition toolkit kaldi (Povey et al. 2011), which is the basis for ample ASR research, uses n-grams for on-line decoding exclusively, with neural network language models only as post-hoc step.

The conditional probability $P(w_i|w_1 \dots w_{i-1})$ can be obtained from text (the training data) by counting the number of occurrences of $w_1 \dots w_i$ and $w_1 \dots w_{i-1}$ and dividing the first by the second. Here we can already see similarities to predictive parsing, which also has $w_1 \dots w_{i-1}$ as input but yields a syntactic structure as output.

Given a language model as described above, most words will be assigned a probability of zero in a given context as they were never seen in that context in the training data (i. e. $w_1 \dots w_i$ was never seen). In addition, most contexts themselves (i. e. $w_1 \dots w_{i-1}$) do not occur at all in the training data, yielding no usable probability distribution. Therefore, the N-gram approximation based on the Markov assumption is often used: ($P(w_i|w_1 \dots w_{i-1}) \approx P(w_i|w_{i-N} \dots w_{i-1})$). Interestingly, this approach, coupled with smoothing techniques such as modified Kneser-Ney-Smoothing (Chen and Goodman 1996), has been state of the art for a long time for large data sets. Only recently, LSTM-based approaches were published that outperform N-gram language models on large datasets as the Billion Word Corpus (BWC; Chelba, Mikolov, et al. (2013)) such as Józefowicz et al. (2016). Before that, neural approaches to language modeling were only evaluated on much smaller datasets such as the Penn Treebank (see Merity, Keskar, and Socher (2017) for a list of such publications).

Interestingly, N-gram models perform well despite relying on a fairly limited context. Increasing the context size does not increase the performance as the contexts are only very rarely seen even when creating the language model from huge datasets – the usual data sparsity problem. Nonetheless, sometimes, the context not modeled by N-grams influences on the possible continuations of a sequence. For example, the following two sentence prefixes share the same 4-gram context but have different possible continuations:

(7.1) The world that we are changing

(7.2) There is evidence that we are changing

Whereas the word "is" is a plausible continuation of Sentence 7.1 (e.g. "is turning for the better"), it is implausible for Sentence 7.2. In fact, Sentence 7.2 can not be continued the same as Sentence 7.1:

(7.3) * There is evidence that we are changing is turning for the better

In this example, an N-gram model fails to capture the long-distance dependencies, namely the information that Sentence 7.1 is still missing the verbal phrase but contains a complete noun phrase. In contrast, Sentence 7.2 already contains a verbal phrase, but its nested noun phrase is possibly incomplete – it could e. g. be continued by "There is evidence that we are changing the world". While I will focus on syntax in this chapter, it

is important to note that other types of long-distance dependencies also play an important role in predicting the continuation of a text such as pragmatics or semantics.

As already discussed in Section 1.1, humans can predict upcoming words quite well and distinguish between words that match a predicted structure and words that do not. In this chapter, I will lay out an approach to augment N-gram language models with syntactic information in a cost-efficient way.

7.2. Syntax-based Language Models

Syntactic structure of prefixes provides information about possible continuations. Combined with a probability model, they can be used to compute a probability distribution of the next word. In contrast to N-gram models, these models can make use of long-spanning context information due to the possibly long-ranging information in syntactic structures. Syntactic structures have, however, not been widely used for language modeling. The approaches proposed in the literature can be roughly divided into three clusters:

post-hoc re-ranking modifies the probabilities of sequences once they are complete

probability distributions over tree structures replaces n-gram probabilities with probability distributions over syntax trees

incorporating syntactic predictions extracts information from syntax structures and incorporates them into N-gram models as an additional information source

Recent work on incorporating syntactic information into language models has focused on re-ranking, based on scores for the syntax trees for the different possible sentences (Filimonov and Harper 2009; Charniak 2001; Tan et al. 2012), or by training a discriminative model (Collins, Roark, and Saraclar 2005). For interactive systems, re-scoring only complete output is unsatisfactory as the intermediate computations can not benefit. Also, re-ranking for language modeling is largely performed using neural language models today. These neural language models are both less complex than the syntax-based re-ranking approaches and seem to yield superior perplexity – even though the results reported are hard to compare due to the differences in the data and evaluation schema used for evaluation (see Merity, Keskar, and Socher (2017) for an overview of language model perplexities using different neural network architectures).

A special case of re-ranking is the task of sentence completion, where one word of a sentence is missing, and the correct word out of five possibilities needs to be selected to fill the gap (Zweig and Burges 2011). For evaluation purposes, the possible words are explicitly selected to have a similar probability under a plain N-gram model. Zhang, Lu,

and Lapata (2016) propose a generative LSTM that produces sentences via dependency structures instead of linearly – as a standard LSTM language model would do. They obtain state-of-the-art results for the sentence completion challenge by computing the probabilities of sentences obtained by filling the blank with all alternatives and choosing the most probable one. However, these models are not designed to predict the next word of a sentence. Hence, they can only be applied non-incrementally in a rescoring fashion, e. g., to improve ASR or parsing results after a first pass over the input data. Thus, they are not applicable to interactive use-cases that require incremental processing.

Parsers that define a probability distribution over the (partial) parse trees they derive can be used for language modeling, e. g. using a limited-domain hand-written PCFG with learned rule-weights (Jurafsky et al. 1995), or by learning structure using a hierarchical HMM (Schwartz et al. 2011). Using a trained parser for language modeling, Roark (2001) (see Section 3.1.1) estimates the probability of a word w_i following the current sentence prefix $w_1 \dots w_{i-1}$ by measuring the probability of all parse trees (within a beam) that are derivable for the prefix including that word (i. e., all parse trees over $w_1 \dots w_i$) divided by the probability of all parse trees for $w_1 \dots w_{i-1}$. To obtain a probability distribution over the possible upcoming words w_i, all parses for all possible $w_1 \dots w_i$ must be computed, which is computationally expensive. When obtaining a probability distribution for the next word, the cost grows with the vocabulary size.

There is one syntax-based language model that only creates syntactic structures for the context – excluding the word to be queried – and is therefore less impacted by large vocabulary sizes: Chelba and Jelinek (1998) use a tree adjoining grammar to parse the sentence prefix up to (but not including) the queried word, and the parser state (specifically, the top two elements of the stack) is used to condition a probability distribution for both the syntactic structure and the upcoming word. Because the probability distribution the parser learns is a joint distribution for the sentence and a syntax tree, the syntactic structure has to be summed out if only the probability of the next word irrespective of the intended syntactic structure is of interest. Therefore, the joint probability is summed over all possible trees for the prefix and again many parses are required for a single query, resulting in a high complexity.

7.3. Data Preparation

The following experiments use incTP, trained on the Penn Treebank transformed to dependencies using the LTH converter (Johansson and Nugues 2007). By design of the incremental training data, the parser predicts up to one upcoming verb and one upcoming noun, which results in a good coverage/precision trade-off for English. The parser and

		verb predicted		
		no	yes	sum
noun predicted	no	41.6	22.7	**64.3**
	yes	18.7	17.0	**35.7**
	sum	**60.3**	**39.7**	

Table 7.1.: Distribution (in percent) of prediction nodes in the syntactic structures generated by incTP for sentence prefixes in the Billion Word Corpus.

tagger models are the same as described in Chapter 5.2 and, as such, are not tuned for language modeling nor for the corpus used for estimating the LM.

All of the one billion sentence prefixes of the billion word corpus were parsed with incTP, so for each sentence prefix to be queried the corresponding dependency structure is available. The distribution of predictions made by the parser can be seen in Table 7.1. Throughout this chapter, the prediction nodes (a $v$erb and a $n$oun) will be shortened by vn. If in a specific syntactic structure, a verb or noun was predicted, it will be denoted by $\overset{+}{v}$ and $\overset{+}{n}$, respectively. If a verb or noun was not predicted, this is indicated by $\bar{v}$ and $\bar{n}$.

As can be seen in Table 7.1, both v and n split the prefixes into four parts of similar size, yielding an information of 1.90 bit due to the imbalances of the classes. All experiments in this chapter use only these two binary parts of information from the syntactic structure; the remaining structure is disregarded. The prediction nodes seemed to contain the most complementary and compact pieces of information, and language modeling already has to deal with severe data sparsity. All additional information used from syntactic structure only makes this problem worse. The configuration of prediction nodes the parser produced for a given input $w_1 \dots w_i$ will be called p_i in this chapter.

7.4. A First Language Model with Prediction Nodes

It could very well be that the predictions vn made by the parser only contain information also found in the N-gram context. In this case, the information about prediction nodes would inherently not be able to improve upon N-gram language models. A first simple experiment can gauge whether the prediction nodes encode additional information at all: The N-grams extracted from the text and forming the basis of the language model are split based on the parser prediction of their histories (i. e., the prediction nodes generated by the

parser when parsing the sentence prefix up to but not including the newest word of the N-gram) and four different N-gram models are trained based on this split. When querying the probability distribution $P(w_i|w_1 \dots w_{i-1})$, the parser's output for $w_1 \dots w_{i-1}$ (i. e. p_{i-1}) is used to determine which of the four language models to query.[2] The entropy of this model can be compared to one where the N-grams are put into four bins randomly with the same overall distribution. If the parser-based model has a lower entropy than the randomly shuffled one, there is useful information in the parser predictions.

It might be surprising that the parser-based model is not compared to a standard N-gram model. The reasoning behind this design is that splitting the source N-grams into four models makes each of the models perform worse due to increased data sparsity. Comparing the parser-based split model to a standard N-gram model would therefore also measure the effect of the additional data sparsity induced. Performing the split also on the baseline model results in only measuring the effect of the parser predictions as the data sparsity is the same for both models.

This model splitting approach is computationally effective: for a given query, the parser only needs to parse the context once (as the information we use from the parser does not depend on the word being queried) and after determining which of the sub language models to query, that model can not only be queried for the probability of a single word, but also for the complete probability distribution of possible continuations, which is especially useful in speech recognition scenarios.

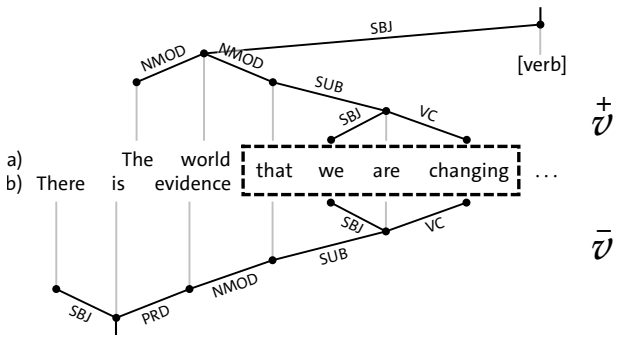
N-gram models are generated with SRILM (Stolcke et al. 2011) using standard settings (modified Kneser-Ney smoothing (Chen and Goodman 1996) and interpolation, a limited vocabulary of 100,000 words, and dropping singleton N-grams for N>2).[3]

7.4.1. Examples for the Effect of Prediction Integration

The non-locality of parsing predictions for the examples 7.1 and 7.2 is exemplified in Figure 7.1 (the example is restricted to just the verb prediction v). In the two sentences given in the figure, the local N-gram context is identical for either sentence prefix, a N-gram model even with N=5 would therefore produce the same probability distribution for both sentence continuations. The cause for the parser predictions lies further in the past than the N-gram model is able to integrate; the span from cause to effect could be

[2] It might be tempting to include the parser prediction directly into the N-gram context as an additional context word (using the N-grams $(w_{i-N} \dots w_{i-1} p_{i-1} w_i)$), but that would break the count-of-count assumptions for Kneser-Ney smoothing.

[3] SRILM allows to manipulate N-gram counts and is still able to compute correct backoff values unlike e. g. KenLM (Heafield 2011), which, however, would allow to include singleton N-grams due to its superior memory efficiency. Data sparsity (see below) would be less severe if singletons were included.



continuation	$\log_{10}$prob difference between $\overset{+}{v}/\bar{v}$ model
has/hasn	$+1.18$ / $+1.12$
will	$+1.10$
is/isn	$+1.08$ / $+0.97$
and	$+0.05$
in	-0.06
worlds	-0.34
it	-1.28
a/an	-1.41 / -1.35
our	-1.69
lives	-2.52

Figure 7.1.: Predictive parses for two sentence beginnings (left). While the parser predicts a verb to be neccessary to complete (a), it does not predict a verb for (b). The differences in the sentences (that cause the different parses) lie outside of the local N-gram context (dashed box). Possible continuations and the relative differences in assigned probabilities depending on $\overset{+}{v}/\bar{v}$ context are shown in tabular form (right).

even larger if e. g. "we" would be replaced by a long noun phrase. Querying the two language models split by whether the parser predicted an upcoming verb and computing the differences shows that the augmented language model is able to distinguish between the contexts: The table on the right of Figure 7.1 shows the $\log_{10}$ probability differences for some continuations. Words that can only reasonably follow prefix (a) have a higher probability under the $\overset{+}{v}$ model whereas words only fitting for prefix (b) have a higher probability under the $\bar{v}$ model. Words such as "and" and "in", which can appear in both contexts, have a similar probability.

The effect of parser prediction can also be seen on four-way splits (i. e. splitting according to the parser predictions vn). For this, we will look at 3-gram models with the 2-gram context being *the world*. The probabilities for the continuations of "the world" as well as the relative change to the standard model are shown in Table 7.2. Numbers of interest are in bold. Starting from the top, the suffix *'s* is assigned a high probability by the models where the parser is still expecting a noun, especially by the $\bar{v}\overset{+}{n}$ model. Structurally, a noun is unlikely to follow directly but it can fill a valency of a verb for which *the world* is a bad

	probabilities					relative change			
tri-gram	avg	$\overset{+}{v}\overset{+}{n}$	$\overset{+}{v}\overset{-}{n}$	$\overset{-}{v}\overset{-}{n}$	$\overset{-}{v}\overset{+}{n}$	$\overset{+}{v}\overset{+}{n}$	$\overset{+}{v}\overset{-}{n}$	$\overset{-}{v}\overset{-}{n}$	$\overset{-}{v}\overset{+}{n}$
the world 's	.33	.41	.37	.28	**.81**	×1.2	×1.1	÷1.2	×2.5
the world .	.14	.02	.03	**.19**	.01	**÷7.0**	**÷4.9**	×1.4	**÷13**
the world ,	.10	**.17**	.07	.11	.02	×1.6	÷1.5	×1.1	**÷4.9**
the world and	.02	.01	.01	**.03**	.004	÷2.2	÷2.2	×1.3	**÷5.3**
the world *everything else*	.41	.39	.52	.39	.16	÷1.1	×1.3	÷1.1	÷2.6

Table 7.2.: Example tri-grams and their probabilities in the full corpus (avg) vs. split by parser prediction. *everything else* is the sum of all other probabilities. Relative change: change of the split model probability with respect to the probability of the non-split (avg) model.

fit lexically. The connection *'s* makes *the world* a possessive modificator of the expected noun. The continuation . has the highest probabilty under the model not predicting any upcoming structure. This makes sense as a full stop usually denotes the end of a sentence, which means that the current prefix should form a complete sentence with a complete syntax structure. The comma is more likely when both a verb and a noun are predicted, presumably because this constellation dominates at the end of introductory prepositional phrases. Lastly, *and* is most likely under the $\overset{-}{v}\overset{-}{n}$ model, similarly to the full stop, as it often connects material to an already complete structure.

These observations show two points: first, at least some of the probability shifts are linguistically plausible (evaluating the whole language model for plausibility is a futile endeavor), and second, probability shifts based on the parser prediction do happen, i. e. there actually is some information in the parser predictions.

7.4.2. Evaluation of the Basic Split Model

As already briefly noted above, the approach of splitting the N-grams into four different models increases the data sparsity noticeably. This evaluation examines split N-gram models for N between 2 and 5; the entropy of 6-gram models is similar to the one of 5-gram models on "small" datasets such as the billion word corpus due to data sparsity. For each N, a four-way split model (vn), a model only split based on the predicted verbs (v), and a model only split based on the predicted nouns (n), the corresponding random split models and a standard model without splitting was created.

		syntax pred. A:	random splits B:	standard C:	syntax gain B-A:	splitting loss B-C:
2-grams	vn	7.819	7.968	7.881	0.149	0.087
	v	7.837	7.918		0.081	0.037
	n	7.861	7.917		0.057	0.036
3-grams	vn	6.988	7.209	6.938	0.221	0.271
	v	6.969	7.071		0.102	0.133
	n	6.959	7.067		0.108	0.129
4-grams	vn	6.735	7.025	6.633	0.290	0.392
	v	6.703	6.839		0.136	0.206
	n	6.671	6.830		0.159	0.197
5-grams	vn	6.685	6.994	6.566	0.309	0.428

Table 7.3.: Cross-entropies (in bit; lower is better) obtained by splitting the training data according to the parser's prediction (verb, noun, or both), vs. random splitting and comparison to standard models.

Table 7.3 shows the cross-entropies obtained by the different models as well as the gain of the model with splits based on parser prediction over the model with random splits and the increase of entropy induced by splitting the data (comparing random splits to the standard model). While the syntax gain is higher than the loss incurred from the data sparsity due to splitting the data for N=2, for higher-order language models the data sparsity is much more severe and cancels out the gain of using syntax information, even though that gain is also rising. The high syntax gain even for 5-grams shows that the parser predictions actually encode long-range information not encoded in the 4-word context. Both noun and verb predictions yield similar results and seem to provide similar (and additive) benefits.

Note that the entropies reported for this experiment are higher than the ones that will be reported in Section 7.5.1 because these models are not order-interpolated,[4] and the sentence ends were excluded from the perplexity computations. These changes are unlikely to change the overall picture of this experiment.

[4]Order-interpolated languge models interpolate N-gram probabilities with lower-order probabilities to obtain more robust probability distributions.

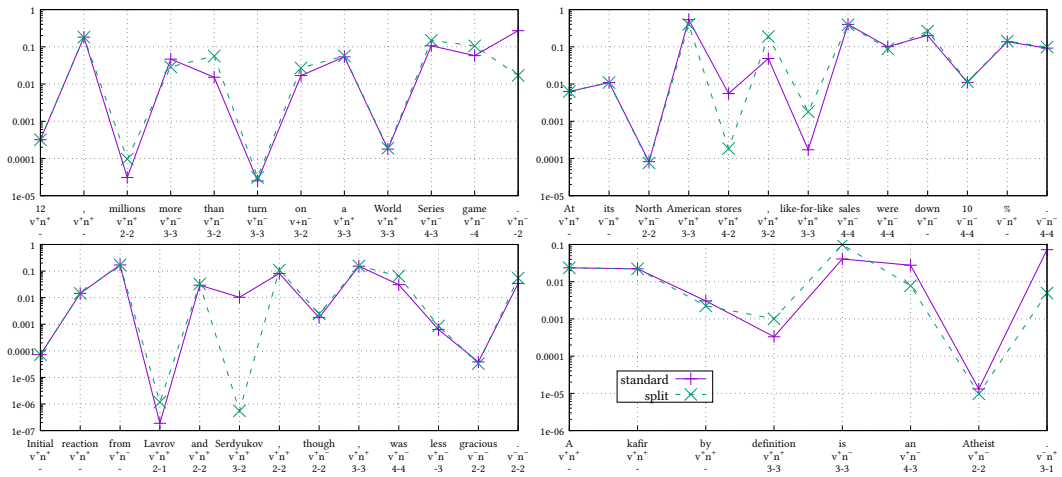


Figure 7.2.: Sentences with their per-word probabilities under standard and split models. Each word is annotated with the syntax prediction of its context and the the N-gram order the standard and split model had to back off to respectively, if backoff is needed.

7.4.3. Exemplary Comparison of Split and Standard Model Probabilities

We have now seen that the syntax-based split model contains useful information (compared to the random splits) but is unable to outperform standard N-gram models (probably due to data sparsity). We looked at probability assignments of the standard 5-gram model and compared those to the probabilities assigned by the syntax-split 5-gram model. We computed the probability differences on 500 short test sentences from the corpus and selected sentences exposing large differences. We found some patterns that are illustrated by the four example sentences in Figure 7.2. Each of the four plots shows the words on the x-axis, which of the split models was queried to obtain the probability for that word (i. e., the predictions of the parser for the prefix not including that word), and to which N-gram order the standard and the split model had to back off to. The logarithmic y-axis shows the probability that the standard and split models assign.

All sentence-initial words are assigned the same probability by both language models as long as the N-gram being queried spans back to the start of the sentence because the syntactic information can not contain additional information about the context outside the N-gram.

If both models can use the same order of N-grams (i. e. the split model does not have to back off more than the standard model), both models assign similar probabilities with the split model occasionally assigning a higher probability. The first three sentences also have data points where the split model assigns a higher probability even though it has to back off further than the standard model due to data sparsity. In the first sentence, the full stop is assigned a low probability by the split model because the parser is still expecting a verb, which is in fact missing from the sentence, and this sentence seems to be both ungrammatical and unfinished.

The split model fails under circumstances where the word to be predicted is only probable given a longer N-gram context than the split model is able to use. For example, the probability of "stores" is high given the context "North American", but much less likely given only the context "American" (see the second sentence). Similarly, "Serdyukov" is only probable if "Lavrov" is in the context: Anatoly Serdyukov was the Defense Minister of Russia from 2007 to 2012, Sergey Lavrov is the Foreign Minister since 2004. The syntax-split model has to back off to bigram probabilities, which removes "Lavrov" from the context and makes a continuation with another Russian politician highly improbable. Further analysis of this last example shows that there are two occurrences of the trigram in the training data, which fall into two different prediction states ($\overset{+}{v}\overset{+}{n}$ and $\overset{-}{v}\overset{+}{n}$). As singleton N-grams are ignored, both occurrences of the trigram are lost and not used in the split model.

7.4.4. Overall Differences Between Split and Standard Model

Up to now, we only looked at examples qualitatively. Figure 7.3 (left) shows a histogram of $\log_{10}$ probability differences between the two models. The majority of the differences are positive (55 %), but differences with high magnitude only occur in favor of the standard model, resulting in an overall disadvantage of the split model.

7.5. Beyond the Basic Split Model

As we have seen, the parser predictions provide helpful information, but the basic split approach introduced in the previous section is hindered by the data sparsity induced. In this section, approaches that yield a net gain when including parser predictions will be introduced. Experiments on two different kinds of language models are performed: N-gram language models using SRILM and maximum Entropy N-gram models (Rosenfeld 1994) using faster-rnnlm.[5]

[5] `github.com/yandex/faster-rnnlm`

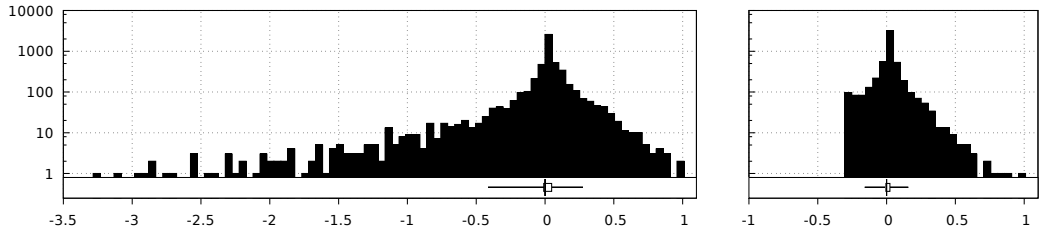


Figure 7.3.: Left: Histogram over $\log_{10}$prob differences between the split and the standard model. Right: Differences between the interpolated and standard model (see Section 7.5.1). Notice the logarithmic y-axis. The boxplots below show 25/75% (box) and 5/95% (whiskers) intervals and median.

7.5.1. Interpolating N-gram Models

To reduce the magnitude of the negative outliers due to data sparsity, the split model can be interpolated with a standard model. Building the average between two models to obtain a model superior to both parts seems counter-intuitive, but it actually works because entropy is computed by multiplying probabilities from the language model, and averaging two models with the same entropy can yield a model with lower entropy, as I will demonstrate with an example: Given a two-element binary task x_1, x_2 with observations $x_1 = 1$ and $x_2 = 1$ and two probability distributions a and b as well as the averaged distribution $\overline{ab}$ assigning

$$P_a(x_1 = 1) = 0.1 \qquad P_b(x_1 = 1) = 0.8$$
$$P_a(x_2 = 1) = 0.8 \qquad P_b(x_2 = 1) = 0.1$$
$$P_{\overline{ab}}(x_1 = 1) = 0.45 \qquad P_{\overline{ab}}(x_2 = 1) = 0.45$$

yields a cross-entropy of $(-\log_2(0.8) + \log_2(0.1))/2 = 1.82$ bits for models a and b, but a much lower entropy of $(-\log_2(0.45) + \log_2(0.45))/2 = 1.15$ bits for the averaged model $\overline{ab}$.

The effect of averaging can be seen on the right side of Figure 7.3: No probability of the interpolated model can be less than half of the standard model, removing the long tail of low probabilities from the left side of the figure. In other words, no $\log_{10}$prob difference is smaller than $log_{10}(0.5) \approx -0.3$. Although the magnitude of improvements

is also reduced, that effect is smaller, and the difference to the standard model turns from negative to positive.

There are, of course, several language models that can be combined in different ways; the results of different experiments are shown in Table 7.4; note that these models are – in contrast to the ones described in Section 7.4 – order-interpolated, resulting in a slightly lower entropy. Interpolating a model based on a split along the v predictions and a model based on n predictions improves performance over the four-way split (**1a**). This improvement is not surprising as each estimation is now roughly based on three quarters of the data (half from the v split and half from the n split, with an overlap) instead of only one quarter. The interpolation can also take place on the N-gram level instead of the resulting probability distributions (**1b**): A four-way split model can be constructed where each of the bins is assigned all the N-grams where the v prediction matches and the N-grams where the n prediction matches. For example, the $\overset{+}{v}\overset{+}{n}$ sub-model receives all N-grams for which either a verb or a noun were predicted; N-grams for which both a verb and a noun was predicted are counted twice as they are included once through both predictions.

The interpolation with the standard model can not only be performed with the four-way splitting model (**2a**) but also with the better count-based approach (**2b**). Using the count-based interpolation yields a gain of 0.08 bits cross-entropy. While sounding quite small, this is a perplexity improvement of 6 % (from 76.3 down to 72.0) while still having smaller models than unpruned N-gram models and retaining the speed of N-gram language models.

7.5.2. Adding Syntax Predictions to Maximum Entropy Models

To test whether the performance increase is only due to the peculiarities of one language model system, I extended a hash-based Maximum Entropy language model (the base implementation being from faster-rnnlm) to make use of parser predictions. A Maximum Entropy language model (MaxEnt-lm) computes the probability for a word given its context by learning scores $s(\text{N-gram})$ for N-grams and performing a softmax over the alternative words in the same context:

$$p(w_i|w_{i-N}\dots w_{i-1}) = \frac{exp(s(w_{i-N},\dots w_{i-1},w_i))}{\sum_{w'\in W} exp(s(w_{i-N},\dots w_{i-1},w'))} \tag{7.4}$$

N-grams never seen in the training data pose a similar problem as for N-gram language models. The MaxEnt-LMs are robust against this data sparsity because the scoring function

model type	setting	standard	syntax	gain
5-grams	**0.** plain splitting	6.253	6.374	−0.12
	1a. interp. from $v \times n$		6.254	−0.001
	1b. joint counts		6.187	0.07
	2a. vn interp. w/ std		6.234	0.02
	2b. joint counts interp. w/ std		6.170	**0.08**
MaxEnt	2-grams	8.41	8.31	0.10
	3-grams	7.86	7.78	0.08
	4-grams	7.88	7.87	0.01
	4-grams (large GPU)	7.55	7.49	**0.06**

Table 7.4.: Cross-entropies (in bit; lower is better) of the experiments presented in Section 7.5, for 5-grams and MaxEnt models, as discussed in the respective subsections, with the standard model, syntax-enhanced model, and gain (or loss) between the two. As can be seen, the baseline standard models are outperformed for all model types.

is a sum over lower-order scores:

$$s(w_i \dots w_{i-N}) = \sum_{j=0}^{N} f(w_i \dots w_{i-j}) \tag{7.5}$$

Note that this order-interpolation makes no additional assumptions about the structure of the context. It is therefore possible to augment the scoring function with the parser state as follows:

$$p(w_i | w_{i-1} \dots w_{i-N}, p_{i-1}) = \frac{exp(s(w_i, w_{i-1} \dots w_{i-N}, p_{i-1}))}{\sum_{w' \in W} exp(s(w', w_{i-1} \dots w_{i-N}, p_{i-1}))} \tag{7.6}$$

$$s(w_i \dots w_{i-N}, p_{i-1}) = \sum_{j=0}^{N} f(w_i \dots w_{i-j}, p_{i-1}) + f(w_i \dots w_{i-j}) \tag{7.7}$$

This approach is not possible with interpolated N-gram models because the interpolations are built on the assumption that a lower-order (N-1)-gram has the same count as all N-grams with that N-gram as context.

In the faster-rnnlm implementation, f hashes the N-gram (as well as lower-order N-grams) into a fixed-size table. It is therefore possible that two different N-grams share

weights, but this step is necessary to limit the memory consumption of the model. The experiment used a hash table with 400M elements, the maximum for the GPU card used.

Results are presented in Table 7.4, showing averages of several runs (to account for random initialization). The vn-enhanced models perform significantly better than the standard ones,[6] but the difference for 4-grams is small. This is likely due to an increased number of hash collisions; evaluating the MaxEnt-lm using a GPU with more memory and a hash table with 1,600M elements yields an improvement of the prediction-augmented model again.

7.6. Summary and Discussion

This chapter showed that predictive parsing can reduce the entropy of N-gram models without a need to create parses for the words queried. Moreover, the parser used is in no way optimized for language modeling and can be trained on a corpus different from the one used for training the language model. The information extracted from the parser is simple, encodable in two bits, and using it incurs no additional computational cost if a predictive parser is run in the same overall system as the language model anyway.

[6] Wilcoxon rank sum test based on multiple runs of each experiment condition, $p < .01$.

Chapter 8.

Conclusions

This thesis set out to investigate how incremental and predictive parsing can be performed. A significant problem for incremental parsing (and other types of incremental processing, see Chapter 2) is the lack of a gold standard for each input. While the complete input does have a gold standard, a partial one does not. This thesis dealt with this problem in two ways. First, by creating incremental gold standards out of the non-incremental ones (see Chapter 3), which enabled the training of a restart-incremental predictive parser (Chapter 5). This parser has high accuracy, but heuristics are needed to create the gold standard. These heuristics limit the data sets the parser can be trained on and are only approximations of what a good gold standard should look like. The second approach to incremental predictive parsing (Chapter 6) mitigates the need for an incremental gold standard by rephrasing the parsing problem as a ranking problem of (not too many) candidate structures that can all be compared to the non-incremental gold standard. The structure created by the parser that fits the non-incremental gold standard best is used as the target for training. To create these candidate structures, a transition system is employed that has a high coverage but still generates few enough structures for the resulting parser to be several times faster than the restart-incremental one, trading accuracy for speed.

In contrast to previous work on predictive parsing (targeting, e. g., psycholinguistic adequacy (Demberg, Keller, and Koller 2013) or language modeling (Jurafsky et al. 1995)), incTP and PreTra are not optimized for any task except parsing. Still, their output proves to be of merit for other tasks such as language modeling (see Chapter 7).

As for every work, there are still plenty of open questions, problems and opportunities for further research left. A description of some of these forms the end of this thesis.

Psycholinguistic plausibility It remains to be seen whether incTP and PreTra exhibit similar effects concerning psycholinguistic adequacy, e. g. if the non-monotonic changes in their output are good predictors of surprisal or reading times. A good correlation of those changes with reading times would explain away some of the stated linguistic plausibility of other approaches (such as PLTAG) because psycholinguistic implausible parsers (e. g.

incTP) exhibit similar effects.

Mixing transition systems with reanalysis The two parsers presented – incTP and PreTra – work complementary: incTP performs restart-incrementality whereas PreTra only performs monotonic extensions.[1] As psycholinguistic research has shown that neither approach alone models human processing well (Malsburg and Vasishth 2011), a combination of both would be desirable. I. e., a parser that can switch from extension to reanalysis whenever the probability of a misunderstanding of the sentence is high. This combined approach could lead to a faster parser than incTP is (which currently recomputes too much) and higher accuracy than the one of PreTra (which can currently not recover from errors made in all entries of the beam).

Training without incremental gold standards In contrast to PreTra, incTP needs pre-generated incremental gold standards. It may be possible to use the structures generated while training PreTra as a basis for training incTP or a similar restart-incremental parser. In that case, the transition system would primarily be used to constrain the automatic generation of gold-standard data, but the resulting parser would not need to use it.

Qualitative Analyses Non-incremental parsers can be evaluated qualitatively by looking at the kind of errors they make. This type of evaluation would undoubtedly be useful for incremental parsers as well but is more complicated due to the new dimension the incremental processing introduces. In addition to e. g. asking *whether* a prepositional phrase is correctly attached, one needs to reason about *when* they are attached where. As this additional complexity makes drawing valid conclusions from qualitative analyses even harder, they were omitted from this thesis. Performing this analysis might nonetheless be fruitful, especially when compared to the kind of constructions where humans exhibit reading difficulties.

Incremental parsing in multi-modal settings The architecture of PreTra especially lends itself into experiments on multi-modal processing. The scoring of structures is distinct from the structure generation and can therefore be easily augmented with priming from e. g. a visual modality. This would allow modeling parsing effects in a visual world similar to the effects reported by Tanenhaus et al. (1995).

Syntax annotations motivated by incremental parsing Regarding parsing effects, the experiments with PreTra have also shown that the choice of annotation schema has

[1] PreTra still produces non-monotonic output whenever the best hypothesis in the beam changes.

an impact on the accuracy that could theoretically be achieved by a transition-based predictive parser. Especially the low attachment of arguments to verb chains – as done in the HDT annotation – results in the need for reanalysis. This insight could be a motivation for annotations that are better-suited for incremental parsing, such as the UD annotations. However, even with a function-head annotation, attaching arguments to verb chains high rather than low would enable better incremental parsing accuracy.

Incremental non-syntax structured prediction tasks I hope that the findings can carry over to other structured prediction tasks. For example, the general approach of performing transition-based beam search and updating against an element of the beam instead of a gold standard seems to be more widely applicable, e. g. for incremental semantic parsing for which some preliminary work is already available (Konstas et al. 2014; Sayeed and Demberg 2012). However, the grounding of the output is even less clear than for syntax parsing that makes it even harder to attribute parts of the output to parts of the input.

Making use of predictive parsers in incremental systems While the task of predictive parsing is interesting in itself and can be a tool to understand the dynamic processes of language better, predictive parsers could also be used in applications, as briefly discussed in Chapter 7. Some applications that used to include syntax parsers for feature extraction have switched to use neural network based extractions on the input sequences instead. However, even for those applications, recent work has shown that making use of syntactic structure provides benefits. Incremental systems, for which incremental parsers were not readily available to be used, can now employ the parsers presented in this thesis, and non-incremental systems relying on syntactic structure have one obstacle less to be incrementalized.

Bibliography

Abeillé, Anne, Lionel Clément, and François Toussenel (2003). "Building a Treebank for French". In: *Treebanks*. Ed. by Anne Abeillé. Dordrecht: Kluwer (cit. on p. 43).

Aït-Mokhtar, S., J.-P. Chanod, and C. Roux (2002). "Robustness beyond shallowness: incremental deep parsing". In: *Natural Language Engineering* 8.2-3, pp. 121–144 (cit. on p. 11).

Aly, Rami, Shantanu Acharya, Alexander Ossa, Arne Köhn, Chris Biemann, and Alexander Panchenko (2019). "Every Child Should Have Parents: A Taxonomy Refinement Algorithm Based on Hyperbolic Term Embeddings". In: *Proceedings of the 57th Conference of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 4811–4817 (cit. on p. 133).

Andor, Daniel, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins (2016). "Globally Normalized Transition-Based Neural Networks". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 2442–2452 (cit. on p. 60).

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *CoRR* abs/1409.0473. arXiv: 1409.0473 (cit. on p. 21).

Bangalore, Srinivas and Aravind K. Joshi (1999). "Supertagging: An Approach to Almost Parsing". In: *Computational Linguistics* 25.2 (cit. on p. 37).

Baumann, Timo (2013). "Incremental Spoken Dialogue Processing: Architecture and Lower-level Components". PhD thesis. Universität Bielefeld, Germany (cit. on p. 27).

Baumann, Timo, Michaela Atterer, and David Schlangen (2009). "Assessing and Improving the Performance of Speech Recognition for Incremental Systems". In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Boulder, Colorado: Association for Computational Linguistics, pp. 380–388 (cit. on pp. 19, 20, 27).

Baumann, Timo, Okko Buß, and David Schlangen (2011). "Evaluation and Optimisation of Incremental Processors". In: *Dialogue & Discourse* 2.1. Special Issue on Incremental Processing in Dialogue, pp. 113–141. ISSN: 2152-9620 (cit. on pp. 25, 51).

Baumann, Timo, Arne Köhn, and Felix Hennig (2018). "The Spoken Wikipedia Corpus collection: Harvesting, alignment and an application to hyperlistening". In: *Language Resources and Evaluation*. ISSN: 1574-0218 (cit. on p. 132).

Beuck, Niels, Arne Köhn, and Wolfgang Menzel (2011a). "Decision Strategies for Incremental POS Tagging". In: *Proceedings of the 18th Nordic Conference of Computational Linguistics NODALIDA 2011*. Ed. by Bolette Sandford Pedersen, Gunta Nešpore, and Inguna Skadina. Vol. 11. NEALT Proceedings. Northern European Association for Language Technology (NEALT), pp. 26–33 (cit. on pp. 19, 134).

– (2011b). "Incremental parsing and the evaluation of partial dependency analyses". In: *Proceedings of the 1st International Conference on Dependency Linguistics*. Depling 2011 (cit. on pp. 26, 37, 44, 47, 49, 64, 72, 134).

– (2013). "Predictive Incremental Parsing and its Evaluation". In: *Computational Dependency Theory*. Ed. by Kim Gerdes, Eva Hajičová, and Leo Wanner. Vol. 258. Frontiers in Artificial Intelligence and Applications. IOS press, pp. 186–206 (cit. on pp. 37, 51, 61, 64, 72, 129).

Beuck, Niels and Wolfgang Menzel (2013). "Structural Prediction in Incremental Dependency Parsing". In: *Computational Linguistics and Intelligent Text Processing*. Ed. by Alexander Gelbukh. Vol. 7816. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 245–257. ISBN: 978-3-642-37246-9 (cit. on pp. 49, 52, 54, 55, 67, 159).

Bohnet, Bernd (2010). "Top Accuracy and Fast Dependency Parsing is not a Contradiction". In: *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*. Beijing, China, pp. 89–97 (cit. on p. 83).

Bohnet, Bernd and Jonas Kuhn (2012). "The Best of Both Worlds – A Graph-based Completion Model for Transition-based Parsers". In: *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Avignon, France: Association for Computational Linguistics, pp. 77–87 (cit. on pp. 76, 81).

Bohnet, Bernd, Ryan McDonald, Emily Pitler, and Ji Ma (2016). "Generalized Transition-based Dependency Parsing via Control Parameters". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 150–160 (cit. on p. 60).

Borges Völker, Emanual, Maximilan Wendt, Felix Hennig, and Arne Köhn (2019). "HDT-UD: A very large Universal Dependencies treebank for German". In: *Proceedings of the Universal Dependencies Workshop at SyntaxFest 2019* (cit. on pp. 43, 133).

Charniak, Eugene (2001). "Immediate-Head Parsing for Language Models". In: *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics*. Toulouse, France: Association for Computational Linguistics, pp. 124–131 (cit. on p. 95).

Chelba, Ciprian and Frederick Jelinek (1998). "Exploiting Syntactic Structure for Language Modeling". In: *Proceedings of the 36th Annual Meeting of the Association for Computational*

Linguistics and 17th Int. Conf. on Computational Linguistics. Vol. 1. Montréal, Canada: ACL, pp. 225–231 (cit. on p. 96).

Chelba, Ciprian, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn (2013). "One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling". In: *CoRR* abs/1312.3005 (cit. on p. 94).

Chen, Stanley F. and Joshua Goodman (1996). "An Empirical Study of Smoothing Techniques for Language Modeling". In: *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*. Santa Cruz, California, USA: Association for Computational Linguistics, pp. 310–318 (cit. on pp. 94, 98).

Chiang, David (2000). "Statistical Parsing with an Automatically-Extracted Tree Adjoining Grammar". In: *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*. Hong Kong: Association for Computational Linguistics, pp. 456–463 (cit. on p. 37).

Cho, Kyunghyun and Masha Esipova (2016). "Can neural machine translation do simultaneous translation?" In: *CoRR* abs/1606.02012. arXiv: 1606.02012 (cit. on p. 25).

Cho, Kyunghyun, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734 (cit. on p. 83).

Collins, Michael, Brian Roark, and Murat Saraclar (2005). "Discriminative Syntactic Language Modeling for Speech Recognition". In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*. Ann Arbor, USA: ACL, pp. 507–514 (cit. on p. 95).

Comerford, Liam, David Frank, Ponani S. Gopalakrishnan, Ramesh A. Gopinath, and Jan Sedivý (2001). "The IBM Personal Speech Assistant". In: *ICASSP* (cit. on p. 9).

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (2001). *introduction to algorithms (second edition)*. The MIT Press (cit. on p. 64).

Crammer, Koby, Ofer Dekel, Joseph Keshet, and Shai Shalev-Shwartz andYoram Singer (2006). "Online Passive-Aggressive Algorithms". In: *Journal of Machine Learning Research* 7, pp. 551–585 (cit. on p. 86).

Currey, Anna and Kenneth Heafield (2019). "Incorporating Source Syntax into Transformer-Based Neural Machine Translation". In: *Proceedings of the Fourth Conference on Machine Translation (Volume 1: Research Papers)*. Florence, Italy: Association for Computational Linguistics, pp. 24–33 (cit. on p. 29).

Daum, Michael (2004). "Dynamic Dependency Parsing". In: *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together* (cit. on p. 64).

Dean, Thomas and Mark Boddy (1988). "An Analysis of Time-Dependent Planning". In: *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 49–54 (cit. on p. 11).

Demberg, Vera and Frank Keller (2008). "A Psycholinguistically Motivated Version of TAG". In: *Proceedings of the Ninth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+9)*. Tübingen, Germany, pp. 25–32 (cit. on p. 36).

Demberg, Vera, Frank Keller, and Alexander Koller (2013). "Incremental, Predictive Parsing with Psycholinguistically Motivated Tree-Adjoining Grammar". In: *Computational Linguistics* 39.4, pp. 1025–1066 (cit. on pp. 36, 37, 109).

DeVault, David, Kenji Sagae, and David Traum (2009). "Can I Finish? Learning When to Respond to Incremental Interpretation Results in Interactive Dialogue". In: *Proceedings of the SIGDIAL 2009 Conference*. London, UK: Association for Computational Linguistics, pp. 11–20 (cit. on p. 12).

Dyer, Chris, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith (2015). "Transition-Based Dependency Parsing with Stack Long Short-Term Memory". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 334–343 (cit. on p. 59).

Filimonov, Denis and Mary Harper (2009). "A Joint Language Model With Fine-grain Syntactic Tags". In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, pp. 1114–1123 (cit. on p. 95).

Foth, Kilian A. (2006). *Eine umfassende Constraint-Dependenz-Grammatik des Deutschen*. ger (cit. on p. 64).

Foth, Kilian A., Michael Daum, and Wolfgang Menzel (2004). "Interactive grammar development with WCDG". In: *The Companion Volume to the Proceedings of 42st Annual Meeting of the Association for Computational Linguistics*. Barcelona, Spain, pp. 122–125 (cit. on p. 63).

Foth, Kilian A., Arne Köhn, Niels Beuck, and Wolfgang Menzel (2014). "Because Size Does Matter: The Hamburg Dependency Treebank". In: *Proceedings of the Language Resources and Evaluation Conference 2014*. Ed. by Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis. LREC. Reykjavik, Iceland: European Language Resources Association (ELRA), pp. 2326–2333 (cit. on pp. 31, 43, 130).

Foth, Kilian A. and Wolfgang Menzel (2006). "Hybrid Parsing: Using Probabilistic Models as Predictors for a Symbolic Parser". In: *Proceedings of the 21st International Conference*

on Computational Linguistics and 44th Annual Meeting of the ACL. Sydney, Australia: Association for Computational Linguistics, pp. 321–328 (cit. on p. 72).

Foth, Kilian A., Wolfgang Menzel, Horia F. Pop, and Ingo Schroder (2000). "An Experiment On Incremental Analysis Using Robust Parsing Techniques". In: *COLING 2000 Volume 2: The 18th International Conference on Computational Linguistics* (cit. on p. 64).

Foth, Kilian A., Wolfgang Menzel, and Ingo Schröder (2000). "A Transformation-based Parsing Technique with Anytime Properties". In: *4th Int. Workshop on Parsing Technologies, IWPT-2000*. Trento, Italy, pp. 89–100 (cit. on p. 64).

Friedrich, Max, Arne Köhn, Gregor Wiedemann, and Chris Biemann (2019). "Adversarial Learning of Privacy-Preserving Text Representations for De-Identification of Medical Records". In: *Proceedings of the 57th Conference of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 5829–5839 (cit. on p. 133).

Gibson, Edward and Tessa Warren (2004). "Reading-Time Evidence for Intermediate Linguistic Structure in Long-Distance Dependencies". In: *Syntax* 7.1, pp. 55–78. ISSN: 1467-9612 (cit. on p. 10).

Goldberg, Yoav and Joakim Nivre (2013). "Training Deterministic Parsers with Non-Deterministic Oracles". In: *Transactions of the Association for Computational Linguistics* (cit. on pp. 59, 76).

Goldman-Eisler, Frieda (1972). "Segmentation of input in simultaneous translation". In: *Journal of Psycholinguistic Research* 1.2, pp. 127–140. ISSN: 1573-6555 (cit. on p. 19).

Gómez-Rodríguez, Carlos and Joakim Nivre (2010). "A Transition-Based Parser for 2-Planar Dependency Structures". In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden, pp. 1492–1501 (cit. on pp. 59, 60).

Greff, K., R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber (2017). "LSTM: A Search Space Odyssey". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10, pp. 2222–2232. ISSN: 2162-237X (cit. on p. 83).

Grissom II, Alvin, He He, Jordan Boyd-Graber, John Morgan, and Hal Daumé III (2014). "Don't Until the Final Verb Wait: Reinforcement Learning for Simultaneous Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1342–1352 (cit. on pp. 22, 25).

Gu, Jiatao, Graham Neubig, Kyunghyun Cho, and Victor O.K. Li (2017). "Learning to Translate in Real-time with Neural Machine Translation". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain: Association for Computational Linguistics, pp. 1053–1062 (cit. on pp. 21–23).

Guhe, Markus (2007). *Incremental Conceptualization for Language Production*. Lawrence Erlbaum Associates, Inc. ISBN: 0-8058-5624-2 (cit. on p. 15).

Hajič, Jan, Barbora Vidová Hladká, and Petr Pajas (2001). "The Prague Dependency Treebank: Annotation Structure and Support". In: *Proceedings of the IRCS Workshop on Linguistic Databases*. University of Pennsylvania, Philadelphia, USA, pp. 105–114 (cit. on p. 31).

Han, Kyu J., Akshay Chandrashekaran, Jungsuk Kim, and Ian R. Lane (2018). "The CAPIO 2017 Conversational Speech Recognition System". In: *CoRR* abs/1801.00059. arXiv: 1801.00059 (cit. on p. 13).

Hansen, Brian, David G. Novick, and Stephen Sutton (1996). "Prevention and repair of breakdowns in a simple task domain". In: *Proceedings of the AAAI-96 Workshop on Detecting, Repairing, and Preventing Human-Machine Miscommunication*, pp. 5–12 (cit. on p. 12).

He, He, Jordan Boyd-Graber, and Hal Daumé III (2016). "Interpretese vs. Translationese: The Uniqueness of Human Strategies in Simultaneous Interpretation". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 971–976 (cit. on p. 23).

He, He, Alvin Grissom II, John Morgan, Jordan Boyd-Graber, and Hal Daumé III (2015). "Syntax-based Rewriting for Simultaneous Machine Translation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 55–64 (cit. on pp. 22, 23, 25).

Heafield, Kenneth (2011). "KenLM: Faster and Smaller Language Model Queries". In: *Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation*. Edinburgh, Scotland, United Kingdom, pp. 187–197 (cit. on p. 98).

Hennig, Felix and Arne Köhn (2017). "Dependency Tree Transformation with Tree Transducers". In: *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*. Gothenburg, Sweden: Association for Computational Linguistics, pp. 58–66 (cit. on pp. 43, 131).

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Computation* 9.8, pp. 1735–1780. eprint: https://doi.org/10.1162/neco.1997.9.8.1735 (cit. on p. 83).

Honnibal, Matthew and Mark Johnson (2014). "Joint Incremental Disfluency Detection and Dependency Parsing". In: *Transactions of the Association for Computational Linguistics* 2 (cit. on pp. 13, 59).

Hough, Julian and Matthew Purver (2014). "Strongly Incremental Repair Detection". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*

(EMNLP). Doha, Qatar: Association for Computational Linguistics, pp. 78–89 (cit. on p. 13).

Huang, Liang and Kenji Sagae (2010). "Dynamic Programming for Linear-Time Incremental Parsing". In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden, pp. 1077–1086 (cit. on pp. 59, 60).

Johansson, Richard and Pierre Nugues (2007). "Extended Constituent-to-Dependency Conversion for English". In: *Proceedings of the 16th Nordic Conference of Computational Linguistics NODALIDA-2007*. Ed. by Kadri Muischnek Joakim Nivre Heiki-Jaan Kaalep and Mare Koit. University of Tartu, Tartu, pp. 105–112 (cit. on pp. 43, 96).

– (2008). "Dependency-based Syntactic–Semantic Analysis with PropBank and NomBank". In: *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*. Manchester, England: Coling 2008 Organizing Committee, pp. 183–187 (cit. on pp. 81, 159).

Joshi, Aravind K., Leon S. Levy, and Masako Takahashi (1975). "Tree adjunct grammars". In: *Journal of Computer and System Sciences* 10.1, pp. 136–163. ISSN: 0022-0000 (cit. on p. 35).

Joshi, Aravind K. and Yves Schabes (1997). "Tree-Adjoining Grammars". In: *Handbook of Formal Languages: Volume 3 Beyond Words*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 69–123. ISBN: 978-3-642-59126-6 (cit. on pp. 35, 36).

Józefowicz, Rafal, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu (2016). "Exploring the Limits of Language Modeling". In: *CoRR* abs/1602.02410. arXiv: 1602.02410 (cit. on p. 94).

Jurafsky, Daniel, Chuck Wooters, Jonathan Segal, Andreas Stolcke, Eric Fosler, Gary Tajchaman, and Nelson Morgan (1995). "Using a stochastic context-free grammar as a language model for speech recognition". In: *Acoustics, Speech, and Signal Processing. ICASSP-95., International Conference on*. Vol. 1. Detroit, USA: IEEE, pp. 189–192 (cit. on pp. 96, 109).

Kay, Martin, Jean Mark Gawron, and Peter Norvig (1994). *Verbmobil: a translation system for face-to-face dialog*. CSLI lecture notes 33. CSLI (cit. on p. 11).

Kiperwasser, Eliyahu and Yoav Goldberg (2016). "Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations". In: *Transactions of the Association for Computational Linguistics* 4, pp. 313–327. ISSN: 2307-387X (cit. on pp. 59, 83, 84).

Köhn, Arne (2015). "What´s in an Embedding? Analyzing Word Embeddings through Multilingual Evaluation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 2067–2073 (cit. on p. 130).

Köhn, Arne (2016). "Evaluating Embeddings using Syntax-based Classification Tasks as a Proxy for Parser Performance". In: *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*. Berlin: Association for Computational Linguistics, pp. 67–71 (cit. on pp. 85, 130).

– (2018). "Incremental Natural Language Processing: Challenges, Strategies, and Evaluation". In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 2990–3003 (cit. on p. 132).

Köhn, Arne and Timo Baumann (2016). "Predictive Incremental Parsing Helps Language Modeling". In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, pp. 268–277 (cit. on p. 131).

Köhn, Arne, Timo Baumann, and Oskar Dörfler (2018). "An Empirical Analysis of the Correlation of Syntax and Prosody". In: *Proceedings of Interspeech 2018*, pp. 2157–2161 (cit. on p. 132).

Köhn, Arne, U Chun Lao, AmirAli B Zadeh, and Kenji Sagae (2014). "Parsing Morphologically Rich Languages with (Mostly) Off-The-Shelf Software and Word Vectors". In: *Proceedings of the 2014 Shared Task of the COLING Workshop on Statistical Parsing of Morphologically Rich Languages* (cit. on pp. 58, 130).

Köhn, Arne and Wolfgang Menzel (2013). "Incremental and Predictive Dependency Parsing under Real-Time Conditions". In: *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*. Hissar, Bulgaria: INCOMA Ltd. Shoumen, BULGARIA, pp. 373–381 (cit. on pp. 11, 62, 64, 73, 129).

– (2014). "Incremental Predictive Parsing with TurboParser". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 803–808 (cit. on pp. 37, 72, 73, 129).

Köhn, Arne, Florian Stegen, and Timo Baumann (2016). "Mining the Spoken Wikipedia for Speech Data and Beyond". In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. Portorož, Slovenia: European Language Resources Association (ELRA), pp. 4644–4647 (cit. on p. 131).

Köhn, Christine and Arne Köhn (2018). "An Annotated Corpus of Picture Stories Retold by Language Learners". In: *Proceedings of the Joint Workshop on Linguistic Annotation, Multiword Expressions and Constructions (LAW-MWE-CxG-2018)*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 121–132 (cit. on p. 132).

Köhn, Christine, Tobias Staron, and Arne Köhn (2016). "Parsing Free-Form Language Learner Data: Current State and Error Analysis". In: *Proceedings of KONVENS 2016*. Bochum (cit. on p. 131).

Koller, Alexander, Timo Baumann, and Arne Köhn (2018). "DialogOS: Simple and extensible dialog modeling". In: *Proceedings of Interspeech*, pp. 167–168 (cit. on p. 132).

Koller, Alexander and Marco Kuhlmann (2011). "A generalized view on parsing and translation". In: *Proceedings of the 12th International Conference on Parsing Technologies (IWPT)*. Dublin (cit. on p. 35).

Konstas, Ioannis, Frank Keller, Vera Demberg, and Mirella Lapata (2014). "Incremental Semantic Role Labeling with Tree Adjoining Grammar". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 301–312 (cit. on p. 111).

Koo, Terry and Michael Collins (2010). "Efficient third-order dependency parsers". In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. ACL '10. Uppsala, Sweden: Association for Computational Linguistics, pp. 1–11 (cit. on p. 63).

Koo, Terry, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag (2010). "Dual decomposition for parsing with non-projective head automata". In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. EMNLP '10. Cambridge, Massachusetts: Association for Computational Linguistics, pp. 1288–1298 (cit. on p. 63).

Lerner, Gene H. (2002). "Turn-Sharing: The Choral Co-Production Of Talk In Interaction". In: *The Language of Turn and Sequence*. Ed. by Cecilia E. Ford, Barbara A. Fox, and Sandra A. Thompson. Oxford University Press, pp. 225–256 (cit. on pp. 9, 12).

Levelt, Wilem J. M. (1989). *Speaking: From Intention to Articulation*. The MIT Press (cit. on pp. 10, 23).

Levy, Omer and Yoav Goldberg (2014). "Dependency-Based Word Embeddings". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 302–308 (cit. on p. 85).

Ma, Ji, Yue Zhang, and Jingbo Zhu (2014). "Punctuation Processing for Projective Dependency Parsing". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 791–796 (cit. on p. 44).

Malsburg, Titus von der and Shravan Vasishth (2011). "What is the scanpath signature of syntactic reanalysis?" In: *Journal of Memory and Language* 65.2, pp. 109–127. ISSN: 0749-596X (cit. on pp. 10, 11, 74, 110).

Marcus, Mitchell, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger (1994). "The Penn Treebank: Annotating Predicate Argument Structure". In: *Proceedings of the Workshop on Human Language*

Technology. HLT '94. Plainsboro, NJ: Association for Computational Linguistics, pp. 114–119. ISBN: 1-55860-357-3 (cit. on pp. 31, 37).

Martins, André Filipe Torres (2012). "The Geometry of Constrained Structured Prediction: Applications to Inference and Learning of Natural Language Syntax". PhD thesis. UNIVERSIDADE TÉCNICA DE LISBOA (cit. on p. 65).

Martins, André Filipe Torres, Miguel Almeida, and Noah A. Smith (2013). "Turning on the Turbo: Fast Third-Order Non-Projective Turbo Parsers". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Sofia, Bulgaria, pp. 617–622 (cit. on p. 64).

Martins, André Filipe Torres, Noah A. Smith, Mario Figueiredo, and Pedro Aguiar (2011). "Dual Decomposition with Many Overlapping Components". In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK.: Association for Computational Linguistics, pp. 238–249 (cit. on p. 65).

Martins, André Filipe Torres, Noah A. Smith, and Eric P. Xing (2009). "Concise Integer Linear Programming Formulations for Dependency Parsing". In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Suntec, Singapore, pp. 342–350 (cit. on pp. 63, 65).

Martins, André Filipe Torres, Noah A. Smith, Eric P. Xing, Pedro M. Q. Aguiar, and Mário A. T. Figueiredo (2010). "Turbo parsers: dependency parsing by approximate variational inference". In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. EMNLP '10. Cambridge, Massachusetts: Association for Computational Linguistics, pp. 34–44 (cit. on p. 77).

Mazzei, Alessandro and Vincenzo Lombardo (2007). "BUILDING A WIDE COVERAGE DYNAMIC GRAMMAR". In: *Applied Artificial Intelligence* 21.4-5, pp. 281–296. eprint: `https://doi.org/10.1080/08839510701252346` (cit. on p. 36).

Mazzei, Alessandro, Vincenzo Lombardo, and Patrick Sturt (2007). "Dynamic TAG and Lexical Dependencies". In: *Research on Language and Computation* 5.3, pp. 309–332. ISSN: 1572-8706 (cit. on p. 36).

McDonald, Ryan, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee (2013). "Universal Dependency Annotation for Multilingual Parsing". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 92–97 (cit. on p. 31).

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajič (2005). "Non-projective Dependency Parsing Using Spanning Tree Algorithms". In: *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Process-*

ing. HLT '05. Vancouver, British Columbia, Canada: Association for Computational Linguistics, pp. 523–530 (cit. on pp. 63, 84).

McDonald, Ryan and Giorgio Satta (2007). "On the Complexity of Non-Projective Data-Driven Dependency Parsing". In: *Proceedings of the Tenth International Conference on Parsing Technologies*. Prague, Czech Republic: Association for Computational Linguistics, pp. 121–132 (cit. on p. 63).

McGraw, Ian and Alexander Gruenstein (2012). "Estimating Word-Stability During Incremental Speech Recognition". In: *Interspeech* (cit. on p. 20).

Merity, Stephen, Nitish Shirish Keskar, and Richard Socher (2017). "Regularizing and Optimizing LSTM Language Models". In: *CoRR* abs/1708.02182. arXiv: 1708.02182v1 (cit. on pp. 94, 95).

Mieno, Takashi, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura (2015). "Speed or Accuracy? A Study in Evaluation of Simultaneous Speech Translation". In: *16th Annual Conference of the International Speech Communication Association (InterSpeech 2015)*. Dresden, Germany (cit. on p. 21).

Milde, Benjamin and Arne Köhn (2018). "Open Source Automatic Speech Recognition for German". In: *Proceedings of the 13th ITG conference on Speech Communication* (cit. on pp. 13, 131).

Milde, Benjamin, Jonas Wacker, Stefan Radomski, Max Mühlhäuser, and Chris Biemann (2016). "Ambient Search: A Document Retrieval System for Speech Streams". In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, pp. 2082–2091 (cit. on p. 9).

Neubig, Graham, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin (2017). "DyNet: The Dynamic Neural Network Toolkit". In: *arXiv preprint arXiv:1701.03980* (cit. on p. 84).

Nivre, Joakim (2003). "An Efficient Algorithm for Projective Dependency Parsing". In: *Proceedings of IWPT 03* (cit. on pp. 59, 75).

– (2004). "Incrementality in Deterministic Dependency Parsing". In: *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*. Ed. by Frank Keller, Stephen Clark, Matthew Crocker, and Mark Steedman. Barcelona, Spain: Association for Computational Linguistics, pp. 50–57 (cit. on p. 60).

– (2007). "Incremental Non-Projective Dependency Parsing". In: *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Compu-*

tational Linguistics; Proceedings of the Main Conference. Rochester, New York: Association for Computational Linguistics, pp. 396–403 (cit. on p. 59).

Nivre, Joakim (2008). "Algorithms for Deterministic Incremental Dependency Parsing". In: *Computational Linguistics* 34.4, pp. 513–553. eprint: https://doi.org/10.1162/coli.07-056-R1-07-027 (cit. on pp. 59, 60).

Nivre, Joakim, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret (2007). "The CoNLL 2007 Shared Task on Dependency Parsing". In: *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pp. 915–932 (cit. on p. 44).

Och, Franz Josef and Hermann Ney (2003). "A Systematic Comparison of Various Statistical Alignment Models". In: *Computational Linguistics* 29.1, pp. 19–51 (cit. on p. 18).

Paetzel, Maike, Ramesh Manuvinakurike, and David DeVault (2015). ""So, which one is it?" The effect of alternative incremental architectures in a high-performance game-playing agent". In: *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Prague, Czech Republic: Association for Computational Linguistics, pp. 77–86 (cit. on p. 13).

Povey, Daniel, Arnab Ghoshal, Gilles Boulianne, Lukáš Burget, Ondřej Glembek, K. Nagendra Goel, Mirko Hannemann, Petr Motlíček, Yanmin Qian, Petr Schwarz, Jan Silovský, Georg Stemmer, and Karel Veselý (2011). "The Kaldi Speech Recognition Toolkit". In: *Proceedings of ASRU 2011*. Hilton Waikoloa Village Resort, Hawaii, US: IEEE Signal Processing Society, pp. 1–4. ISBN: 978-1-4673-0366-8 (cit. on p. 93).

Al-Rfou', Rami, Bryan Perozzi, and Steven Skiena (2013). "Polyglot: Distributed Word Representations for Multilingual NLP". In: *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 183–192 (cit. on p. 85).

Riedel, Sebastian and James Clarke (2006). "Incremental Integer Linear Programming for Non-projective Dependency Parsing". In: *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*. Sydney, Australia: Association for Computational Linguistics, pp. 129–137 (cit. on p. 65).

Roark, Brian (2001). "Probabilistic Top-Down Parsing and Language Modeling". In: *Computational Linguistics* 27.2, pp. 249–276. eprint: https://doi.org/10.1162/089120101750300526 (cit. on pp. 34, 42, 96).

Roark, Brian and Mark Johnson (1999). "Efficient probabilistic top-down and left-corner parsing". In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*. College Park, Maryland, USA: Association for Computational Linguistics, pp. 421–428 (cit. on p. 34).

Rodriguez, Carlos Gomez, Francesco Sartorio, and Giorgio Satta (2014). "A Polynomial-Time Dynamic Oracle for Non-Projective Dependency Parsing". In: *Proc. of the 2014*

Conference on Empirical Methods in Natural Language Processing (EMNLP 2014) (cit. on p. 59).

Rosenfeld, Ronald (1994). "Adaptive Statistical Language Modeling: A Maximum Entropy Approach". PhD thesis. Pittsburgh, USA: Carnegie Mellon University (cit. on p. 103).

Sagae, Kenji, Gwen Christian, David DeVault, and David Traum (2009). "Towards Natural Language Understanding of Partial Speech Recognition Results in Dialogue Systems". In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*. Boulder, Colorado: Association for Computational Linguistics, pp. 53–56 (cit. on p. 12).

Saunders, Danielle, Felix Stahlberg, Adrià de Gispert, and Bill Byrne (2018). "Multi-representation ensembles and delayed SGD updates improve syntax-based NMT". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 319–325 (cit. on p. 29).

Sayeed, Asad and Vera Demberg (2012). "Incremental Neo-Davidsonian semantic construction for TAG". In: *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*. Paris, France, pp. 64–72 (cit. on p. 111).

Schlangen, David, Timo Baumann, and Michaela Atterer (2009). "Incremental Reference Resolution: The Task, Metrics for Evaluation, and a Bayesian Filtering Model that is Sensitive to Disfluencies". In: *Proceedings of SigDial 2009*. London, UK (cit. on p. 13).

Schlangen, David and Gabriel Skantze (2009). "A General, Abstract Model of Incremental Dialogue Processing". In: *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*. Athens, Greece: Association for Computational Linguistics, pp. 710–718 (cit. on pp. 18, 28).

Schröder, Ingo (2002). "Natural Language Parsing with Graded Constraints". PhD thesis. Universität Hamburg (cit. on p. 64).

Schuster, Sebastian and Christopher D. Manning (2016). "Enhanced English Universal Dependencies: An Improved Representation for Natural Language Understanding Tasks". In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. Ed. by Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Helene Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis. Portorož, Slovenia: European Language Resources Association (ELRA). ISBN: 978-2-9517408-9-1 (cit. on p. 43).

Schwartz, Lane, Chris Callison-Burch, William Schuler, and Stephen Wu (2011). "Incremental Syntactic Language Models for Phrase-based Translation". In: *Proceedings of the*

49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies. Portland, USA: ACL, pp. 620–631 (cit. on p. 96).

Seddah, Djamé, Eric De La Clergerie, Benoît Sagot, Héctor Martínez Alonso, and Marie Candito (2018). "Cheating a Parser to Death: Data-driven Cross-Treebank Annotation Transfer". In: *Proceedings of the 11th Language Resources and Evaluation Conference*. Miyazaki, Japan: European Language Resource Association (cit. on p. 43).

Selfridge, Ethan, Iker Arizmendi, Peter Heeman, and Jason Williams (2011). "Stability and Accuracy in Incremental Speech Recognition". In: *Proceedings of the SIGDIAL 2011 Conference*. Portland, Oregon: Association for Computational Linguistics, pp. 110–119 (cit. on pp. 20, 29).

Shen, Libin and Aravind K. Joshi (2005). "Incremental LTAG Parsing". In: *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing* (cit. on p. 35).

Shen, Tianxiao, Tao Lei, and Regina Barzilay (2016). "Making Dependency Labeling Simple, Fast and Accurate". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 1089–1094 (cit. on p. 58).

Silveira, Natalia, Timothy Dozat, Marie-Catherine de Marneffe, Samuel Bowman, Miriam Connor, John Bauer, and Christopher D. Manning (2014). "A Gold Standard Dependency Corpus for English". In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)* (cit. on pp. 43, 69).

Skantze, Gabriel and Anna Hjalmarsson (2013). "Towards incremental speech generation in conversational systems". In: *Computer Speech & Language* 27.1. Special issue on Paralinguistics in Naturalistic Speech and Language, pp. 243–262. ISSN: 0885-2308 (cit. on p. 23).

Stolcke, Andreas, Jing Zheng, Wen Wang, and Victor Abrash (2011). "SRILM at sixteen: Update and outlook". In: *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. Waikoloa, USA: IEEE (cit. on p. 98).

Stoness, Scott C., James Allen, Greg Aist, and Mary Swift (2005). "Using real-world reference to improve spoken language understanding". In: *AAAI Workshop on Spoken Language Understanding*, pp. 38–45 (cit. on p. 13).

Sturt, Patrick and Vincenzo Lombardo (2005). "Processing coordinated structures: Incrementality and connectedness". In: *Cognitive Science* 29, pp. 291–305 (cit. on p. 10).

Tai, Kai Sheng, Richard Socher, and Christopher D. Manning (2015). "Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1:*

Long Papers). Beijing, China: Association for Computational Linguistics, pp. 1556–1566 (cit. on p. 83).

Tan, Ming, Wenli Zhou, Lei Zheng, and Shaojun Wang (2012). "A scalable distributed syntactic, semantic, and lexical language model". In: *Computational Linguistics* 38.3, pp. 631–671. eprint: `https://doi.org/10.1162/COLI_a_00107` (cit. on p. 95).

Tanenhaus, MK, MJ Spivey-Knowlton, KM Eberhard, and JC Sedivy (1995). "Integration of visual and linguistic information in spoken language comprehension". In: *Science* 268.5217, pp. 1632–1634. ISSN: 0036-8075. eprint: `http://science.sciencemag.org/content/268/5217/1632.full.pdf` (cit. on pp. 10, 110).

Taskar, Ben, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin (2005). "Learning Structured Prediction Models: A Large Margin Approach". In: *Proceedings of the 22Nd International Conference on Machine Learning*. ICML '05. Bonn, Germany: ACM, pp. 896–903. ISBN: 1-59593-180-5 (cit. on p. 86).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., pp. 6000–6010 (cit. on p. 21).

Vincze, Veronika, Dóra Szauter, Attila Almási, György Móra, Zoltán Alexin, and János Csirik (2010). "Hungarian Dependency Treebank". In: *Proceedings of LREC*. Valletta, Malta (cit. on p. 43).

Wahlster, Wolfgang, ed. (2000). *Verbmobil: Foundations of Speech-to-Speech Translation*. Springer-Verlag Berlin Heidelberg. ISBN: 978-3-540-67783-3 (cit. on pp. 9, 12).

Walker, Willie, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woelfel (2004). *Sphinx-4: A flexible open source framework for speech recognition*. Tech. rep. SMLI TR-2004-139. Menlo Park, CA, USA: Sun Microsystems, Inc. (cit. on p. 20).

Wirén, Mats (1992). "Studies in Incremental Natural-Language Analysis". Ph.D. thesis. Linköping University (cit. on p. 28).

Young, Stephen John, NH Russell, and JHS Thornton (1989). *Token passing: a simple conceptual model for connected speech recognition systems*. Tech. rep. Cambridge, UK: University of Cambridge: Department of Engineering (cit. on p. 20).

Zhang, Meishan, Zhenghua Li, Guohong Fu, and Min Zhang (2019). "Syntax-Enhanced Neural Machine Translation with Syntax-Aware Word Representations". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 1151–1161 (cit. on p. 29).

Zhang, Xingxing, Liang Lu, and Mirella Lapata (2016). "Top-down Tree Long Short-Term Memory Networks". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, USA: ACL, pp. 310–320 (cit. on p. 95).

Zhang, Yuan, Tao Lei, Regina Barzilay, Tommi Jaakkola, and Amir Globerson (2014). "Steps to Excellence: Simple Inference with Refined Scoring of Dependency Trees". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 197–207 (cit. on pp. 82, 91).

Zweig, Geoffrey and Christopher J.C. Burges (2011). *The Microsoft Research Sentence Completion Challenge*. Tech. rep. MSR-TR-2011-129. Redmond, USA: Microsoft Research (cit. on p. 95).

Appendix A.

Publications

These are the publications I (co-)authored while working on this dissertation. If I only contributed to a part of the findings of a paper, my part is explicitly described.

Arne Köhn and Wolfgang Menzel (2013). "Incremental and Predictive Dependency Parsing under Real-Time Conditions". In: *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*. Hissar, Bulgaria: INCOMA Ltd. Shoumen, BULGARIA, pp. 373–381

Content An investigation into the real-time ability of predictive parsing with jwcdg.

Niels Beuck, Arne Köhn, and Wolfgang Menzel (2013). "Predictive Incremental Parsing and its Evaluation". In: *Computational Dependency Theory*. Ed. by Kim Gerdes, Eva Hajičová, and Leo Wanner. Vol. 258. Frontiers in Artificial Intelligence and Applications. IOS press, pp. 186–206

Content Introduces jwcdg as a predictive parser and a method to evaluate predictive parsers.

Contribution Co-developed jwcdg, a translation of WCDG to Java, co-developed the evaluation schema. The extension of the WCDG formalism for predictive parsing was exclusively done by Niels Beuck.

Arne Köhn and Wolfgang Menzel (2014). "Incremental Predictive Parsing with TurboParser". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 803–808

Content Introduced incTP, the restart-incremental predictive parser discussed in Chapter 5.

Arne Köhn, U Chun Lao, AmirAli B Zadeh, and Kenji Sagae (2014). "Parsing Morphologically Rich Languages with (Mostly) Off-The-Shelf Software and Word Vectors". In: *Proceedings of the 2014 Shared Task of the COLING Workshop on Statistical Parsing of Morphologically Rich Languages*

Content An investigation into how well a single parser system can parse morphological rich languages and an approach to perform word segmentation using parsing.

Contribution I performed the experiments for the first part, the idea for segmentation was by me, but implemented and evaluated by U Chun Lao.

Kilian A. Foth, Arne Köhn, Niels Beuck, and Wolfgang Menzel (2014). "Because Size Does Matter: The Hamburg Dependency Treebank". In: *Proceedings of the Language Resources and Evaluation Conference 2014*. Ed. by Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis. LREC. Reykjavik, Iceland: European Language Resources Association (ELRA), pp. 2326–2333

Content The Hamburg Dependency Treebank, which is used for evaluation in this thesis.

Contribution I performed extensive cleanup and corrections to the HDT to bring it in a publishable state; the parser evaluations were also done by me.

Arne Köhn (2015). "What´s in an Embedding? Analyzing Word Embeddings through Multilingual Evaluation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 2067–2073

Content Evaluates what information is encoded in static word embeddings; used to select embeddings for the experiment in Chapter 6.

Arne Köhn (2016). "Evaluating Embeddings using Syntax-based Classification Tasks as a Proxy for Parser Performance". In: *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*. Berlin: Association for Computational Linguistics, pp. 67–71

Content Extends the previous evaluation to gauge whether the evaluation results carry over to dependency parsing.

Christine Köhn, Tobias Staron, and Arne Köhn (2016). "Parsing Free-Form Language Learner Data: Current State and Error Analysis". In: *Proceedings of KONVENS 2016*. Bochum

Content An evaluation of how well dependency parsers work on language learner text and what specific problems they face

Contribution I contributed to parts of the gold standard and extended jwcdg to work in the joint model that was evaluated.

Arne Köhn and Timo Baumann (2016). "Predictive Incremental Parsing Helps Language Modeling". In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, pp. 268–277

Content Shows that the predictions in predictive parsing contain relevant information not found in simple n-gram models and that these predictions can be used to enhance n-gram language models. All experiments of Chapter 7 are published here.

Contribution Joint work with Timo Baumann.

Arne Köhn, Florian Stegen, and Timo Baumann (2016). "Mining the Spoken Wikipedia for Speech Data and Beyond". In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. Portorož, Slovenia: European Language Resources Association (ELRA), pp. 4644–4647

Content The first version of the Spoken Wikipedia Corpora.

Contribution I generated the corpora using code mainly written by the other authors, extracted the statistics for the paper and wrote large parts of the paper.

Felix Hennig and Arne Köhn (2017). "Dependency Tree Transformation with Tree Transducers". In: *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*. Gothenburg, Sweden: Association for Computational Linguistics, pp. 58–66

Content First steps of converting the HDT to Universal Dependencies.

Contribution Based on the Bachelor's thesis by Felix Hennig, which I supervised.

Benjamin Milde and Arne Köhn (2018). "Open Source Automatic Speech Recognition for German". In: *Proceedings of the 13th ITG conference on Speech Communication*

Content A libre state of the art speech recognition system for German.

Contribution The pipeline to use the Spoken Wikipedia Corpora as training data for kaldi.

Timo Baumann, Arne Köhn, and Felix Hennig (2018). "The Spoken Wikipedia Corpus collection: Harvesting, alignment and an application to hyperlistening". In: *Language Resources and Evaluation*. ISSN: 1574-0218

Content The Spoken Wikipedia Corpora, extended version.

Contribution For this extension, we completely revamped the annotation schema collaboratively, Felix Hennig re-worked relevant parts of the software, which Timo Baumnan and I supervised.

Alexander Koller, Timo Baumann, and Arne Köhn (2018). "DialogOS: Simple and extensible dialog modeling". In: *Proceedings of Interspeech*, pp. 167–168

Content A libre system to build dialogue systems.

Contribution I supervised a group of students to make DialogOS free software, replacing all proprietary components and modernizing the code base.

Arne Köhn, Timo Baumann, and Oskar Dörfler (2018). "An Empirical Analysis of the Correlation of Syntax and Prosody". In: *Proceedings of Interspeech 2018*, pp. 2157–2161

Content An investigation into how syntax correlates with prosodic features.

Contribution The underlying data was generated by Oskar Dörfler as part of his Bachelor's thesis, which I supervised. The evaluation is new and was performed by me.

Arne Köhn (2018). "Incremental Natural Language Processing: Challenges, Strategies, and Evaluation". In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 2990–3003

Content A review over existing approaches to incremental NLP. It forms the basis of Chapter 2.

Christine Köhn and Arne Köhn (2018). "An Annotated Corpus of Picture Stories Retold by Language Learners". In: *Proceedings of the Joint Workshop on Linguistic Annotation, Multiword Expressions and Constructions (LAW-MWE-CxG-2018)*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 121–132

Content A corpus of annotated learner data.

Contribution Nearly all work was done by Christine Köhn. I contributed statistics about the corpus as well as a computation and discussion of the inter-annotator agreement.

Rami Aly, Shantanu Acharya, Alexander Ossa, Arne Köhn, Chris Biemann, and Alexander Panchenko (2019). "Every Child Should Have Parents: A Taxonomy Refinement Algorithm Based on Hyperbolic Term Embeddings". In: *Proceedings of the 57th Conference of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 4811–4817

Content Improving unsupervised taxonomy induction. In large parts a result of a bachelor's project I taught.

Contribution Provided ideas and wrote parts of the paper.

Max Friedrich, Arne Köhn, Gregor Wiedemann, and Chris Biemann (2019). "Adversarial Learning of Privacy-Preserving Text Representations for De-Identification of Medical Records". In: *Proceedings of the 57th Conference of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 5829–5839

Content How to make training data shareable if the underlying text contains sensitive information?

Contribution The result of a master's thesis I advised.

Emanual Borges Völker, Maximilan Wendt, Felix Hennig, and Arne Köhn (2019). "HDT-UD: A very large Universal Dependencies treebank for German". In: *Proceedings of the Universal Dependencies Workshop at SyntaxFest 2019*

Content The conversion of the HDT to Universal Dependencies.

Contribution I supervised this work.

These publications were published before the official start of my dissertation, but contain research relevant the the dissertation:

Niels Beuck, Arne Köhn, and Wolfgang Menzel (2011a). "Decision Strategies for Incremental POS Tagging". In: *Proceedings of the 18th Nordic Conference of Computational Linguistics NODALIDA 2011*. Ed. by Bolette Sandford Pedersen, Gunta Nešpore, and Inguna Skadina. Vol. 11. NEALT Proceedings. Northern European Association for Language Technology (NEALT), pp. 26–33

Content A discussion about incremental processing using PoS tagging as an example.

Contribution I programmed the software and performed the evaluation.

Niels Beuck, Arne Köhn, and Wolfgang Menzel (2011b). "Incremental parsing and the evaluation of partial dependency analyses". In: *Proceedings of the 1st International Conference on Dependency Linguistics*. Depling 2011

Content Predictive parsing using the WCDG approach

Contribution I performed the evaluations.

Appendix B.

Parsing evaluation results

Legend:

dist distance to newest word

compl measurements for **compl**ete sentences

cip **c**orrect **i**n **p**refix

cpred **c**orrect **pred**iction

wip **w**rong **i**n **p**refix

wpred **w**rong **pred**iction

Hyperparameters:

beam beam size during training and decoding for PreTra

standard feature set (PreTra) disables the following features from the RBGParser scoring: great-great-grandparent, parent-sibling-child and global features

no morph. features (incTP) input does not contain gold-standard morphological features (same as PreTra)

gold morph. features (incTP) input contains gold-standard morphological features

standard feature set (incTP) uses arc-factored, consecutive sibling, grandparent

full feature set (incTP) uses artitrary sibling, head bigram, grand-sibling, tri-sibling features in addition to standard.

Measurements are reported as distribution over the different (non-)error categories for the given distance (or complete sentence). Accuracy measured against the non-incremental gold standard as described in Section 4.2. Stability measured against the parser's complete output as described in Section 4.2.3.

FTB

PreTra, gold forced in beam

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	67.37	25.13	1.48	6.01	67.04	25.16	1.52	6.28
1	85.97	9.34	0.3	4.39	85.73	9.35	0.3	4.62
2	90.29	6.24	0.05	3.42	90.07	6.24	0.05	3.64
3	92.62	4.31	0.03	3.04	92.5	4.3	0.03	3.17
4	94.01	3.49	0.01	2.49	93.85	3.49	0.02	2.65
5	94.92	2.96	0.01	2.12	94.76	2.96	0.01	2.28
compl	98.58	0.0	0.0	1.42				

PreTra, beam=10, no cost-augmented training, standard feature set

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	53.57	22.17	2.5	21.77	71.11	26.51	0.86	1.53
1	70.72	7.04	0.66	21.58	89.28	9.87	0.16	0.7
2	74.03	4.05	0.33	21.6	93.19	6.33	0.04	0.44
3	75.21	2.66	0.13	22.0	95.23	4.5	0.02	0.25
4	75.67	2.11	0.08	22.15	96.03	3.82	0.0	0.14
5	75.98	1.79	0.05	22.18	96.58	3.32	0.0	0.1
compl	77.67	0.0	0.0	22.33				

PreTra, beam=10

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	54.96	22.49	2.54	20.02	70.73	26.77	0.95	1.55
1	72.1	7.1	0.79	20.01	89.25	9.85	0.29	0.61
2	75.43	4.11	0.34	20.12	93.24	6.35	0.05	0.36
3	76.69	2.67	0.15	20.49	95.19	4.55	0.04	0.23
4	77.19	2.11	0.1	20.6	95.97	3.88	0.02	0.14
5	77.47	1.79	0.07	20.67	96.51	3.37	0.02	0.1
compl	79.34	0.0	0.0	20.66				

PreTra, beam=50

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	53.8	21.8	2.74	21.66	69.27	25.78	1.34	3.6
1	71.84	6.93	0.81	20.43	88.42	9.5	0.32	1.75
2	75.58	4.22	0.29	19.91	92.93	6.33	0.05	0.69
3	76.88	2.75	0.11	20.26	95.03	4.5	0.03	0.44
4	77.38	2.18	0.07	20.37	95.89	3.84	0.01	0.27
5	77.66	1.87	0.06	20.41	96.43	3.37	0.01	0.19
compl	79.71	0.0	0.0	20.29				

PreTra, beam=10, no cost-augmented training

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	51.67	19.7	5.87	22.76	66.82	29.38	1.72	2.07
1	68.67	6.68	1.87	22.77	85.36	13.02	0.63	0.98
2	72.31	3.9	1.09	22.71	89.91	9.26	0.26	0.56
3	73.66	2.67	0.5	23.17	92.35	7.17	0.1	0.38
4	74.26	2.08	0.37	23.28	93.48	6.22	0.04	0.26
5	74.62	1.74	0.3	23.34	94.37	5.43	0.03	0.18
compl	76.35	0.0	0.0	23.65				

UD-FTB

PreTra, gold forced in beam

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	47.34	43.32	2.63	6.71	47.12	43.21	2.69	6.97
1	72.27	22.26	0.67	4.8	71.97	22.19	0.7	5.14
2	84.74	11.02	0.13	4.11	84.24	10.94	0.14	4.69
3	89.36	7.19	0.05	3.41	88.9	7.17	0.05	3.89
4	91.73	5.38	0.02	2.87	91.3	5.38	0.02	3.3
5	93.36	4.32	0.02	2.31	92.86	4.3	0.02	2.82
compl	97.92	0.0	0.0	2.08				

PreTra, beam=50

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	33.57	36.93	6.19	23.31	48.63	45.23	2.35	3.79
1	56.79	18.57	1.99	22.65	73.48	24.22	0.41	1.89
2	68.74	8.37	0.56	22.32	86.49	12.42	0.07	1.01
3	71.72	5.32	0.22	22.74	90.6	8.79	0.02	0.59
4	73.09	3.79	0.12	23.0	92.72	6.92	0.01	0.35
5	73.67	3.1	0.07	23.16	93.87	5.92	0.01	0.2
compl	75.84	0.0	0.0	24.16				

PreTra, beam=10

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	34.59	39.2	4.23	21.97	50.91	46.71	1.0	1.38
1	57.01	18.93	1.49	22.57	75.05	24.06	0.16	0.74
2	68.71	8.31	0.52	22.46	87.57	11.97	0.03	0.42
3	71.43	5.31	0.23	23.03	91.29	8.45	0.01	0.25
4	72.69	3.8	0.12	23.39	93.24	6.62	0.0	0.14
5	73.28	3.12	0.07	23.53	94.3	5.63	0.0	0.07
compl	75.48	0.0	0.0	24.52				

PreTra, beam=10, no cost-augmented training

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	31.27	35.77	8.65	24.31	47.43	49.33	1.6	1.64
1	54.14	17.36	3.64	24.87	71.9	26.64	0.43	1.02
2	66.03	7.79	1.18	25.0	85.18	14.08	0.11	0.63
3	68.99	5.04	0.47	25.5	89.56	10.0	0.04	0.39
4	70.21	3.62	0.26	25.92	91.75	8.01	0.01	0.22
5	70.82	2.98	0.16	26.05	92.9	6.94	0.0	0.16
compl	72.44	0.0	0.0	27.56				

PreTra, beam=10, no cost-augmented training, standard feature set

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	33.4	36.11	4.14	26.35	54.25	42.93	0.99	1.83
1	54.01	17.4	1.39	27.21	76.8	22.02	0.17	1.01
2	64.49	7.73	0.46	27.31	88.2	11.16	0.03	0.61
3	67.16	4.92	0.17	27.75	91.87	7.77	0.01	0.35
4	68.42	3.53	0.08	27.97	93.82	5.97	0.0	0.21
5	68.91	2.88	0.05	28.16	94.85	5.04	0.0	0.11
compl	71.11	0.0	0.0	28.89				

incTP full feature set, no morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	39.51	42.98	1.76	15.74	45.48	43.66	2.35	8.51
1	62.84	20.75	0.69	15.72	71.31	21.37	0.95	6.37
2	74.98	9.31	0.28	15.44	84.31	9.71	0.34	5.64
3	78.64	5.83	0.14	15.4	88.79	6.22	0.16	4.82
4	80.33	4.13	0.07	15.47	91.1	4.45	0.08	4.38
5	81.14	3.27	0.05	15.55	92.42	3.49	0.05	4.03
compl	84.04	0.0	0.0	15.96				

incTP standard feature set, no morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	39.49	42.87	1.88	15.76	45.12	43.23	2.45	9.2
1	62.83	20.73	0.7	15.74	70.92	21.21	0.98	6.89
2	74.87	9.43	0.28	15.42	83.75	9.74	0.35	6.16
3	78.46	5.94	0.13	15.48	88.3	6.27	0.15	5.28
4	80.28	4.23	0.08	15.41	90.7	4.48	0.09	4.73
5	81.08	3.37	0.05	15.5	92.1	3.52	0.06	4.32
compl	84.15	0.0	0.0	15.85				

incTP standard feature set, gold morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	40.79	43.15	1.85	14.21	46.0	43.29	2.39	8.32
1	64.3	20.97	0.71	14.02	71.58	21.36	0.93	6.13
2	76.41	9.54	0.31	13.74	84.23	9.92	0.37	5.48
3	80.14	6.08	0.17	13.61	88.7	6.44	0.18	4.68
4	81.89	4.37	0.1	13.63	91.19	4.65	0.12	4.05
5	82.8	3.49	0.06	13.66	92.48	3.67	0.07	3.78
compl	86.22	0.0	0.0	13.78				

HDT

PreTra, gold forced in beam

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	55.02	34.7	2.66	7.62	54.99	33.49	3.83	7.7
1	74.45	17.08	0.85	7.63	74.92	16.98	1.61	6.48
2	80.44	13.19	0.54	5.83	80.87	13.18	1.14	4.8
3	83.08	11.15	0.4	5.37	83.67	11.33	0.82	4.19
4	85.54	9.58	0.33	4.56	86.04	9.74	0.68	3.55
5	87.34	8.31	0.25	4.1	88.01	8.42	0.54	3.03
compl	94.55	0.0	0.0	5.45				

PreTra, beam=10

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	52.67	29.7	7.19	10.44	57.45	39.37	1.05	2.13
1	69.75	13.91	3.26	13.08	76.13	22.54	0.42	0.91
2	73.9	10.29	2.48	13.33	80.88	18.41	0.2	0.52
3	75.66	8.69	1.92	13.72	83.34	16.16	0.11	0.39
4	76.78	7.51	1.59	14.12	85.14	14.55	0.07	0.25
5	78.01	6.51	1.28	14.2	86.9	12.89	0.05	0.16
compl	85.41	0.0	0.0	14.59				

PreTra, beam=50

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	51.24	29.52	6.84	12.4	55.76	37.36	2.08	4.8
1	69.29	14.0	2.81	13.9	75.21	21.44	0.47	2.88
2	74.47	10.6	1.99	12.94	81.17	17.42	0.17	1.25
3	76.43	8.98	1.56	13.03	83.87	15.35	0.1	0.68
4	77.91	7.78	1.29	13.02	85.82	13.68	0.08	0.43
5	79.08	6.71	1.04	13.16	87.5	12.15	0.03	0.33
compl	86.0	0.0	0.0	14.0				

PreTra, beam=10, no cost-augmented training

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	53.03	28.85	7.29	10.83	59.07	37.28	1.51	2.13
1	70.1	13.35	3.09	13.45	77.95	20.91	0.28	0.87
2	74.32	9.88	2.29	13.5	82.53	16.91	0.11	0.45
3	75.89	8.42	1.79	13.9	84.8	14.82	0.09	0.29
4	77.09	7.3	1.47	14.14	86.43	13.33	0.04	0.2
5	78.06	6.32	1.21	14.41	87.93	11.91	0.02	0.14
compl	84.96	0.0	0.0	15.04				

PreTra, beam=10, no cost-augmented training, standard feature set

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	52.37	29.26	7.01	11.35	58.15	38.26	1.29	2.31
1	69.34	13.47	3.28	13.9	76.99	21.69	0.29	1.03
2	73.54	9.88	2.46	14.12	81.7	17.57	0.14	0.59
3	75.02	8.33	1.92	14.73	83.9	15.63	0.06	0.41
4	76.17	7.14	1.66	15.03	85.65	14.07	0.02	0.25
5	77.04	6.14	1.4	15.43	87.12	12.69	0.02	0.17
compl	83.95	0.0	0.0	16.05				

incTP full feature set, no morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	57.15	32.7	5.4	4.74	58.59	32.71	5.64	3.06
1	77.07	14.51	3.16	5.26	79.69	14.73	3.32	2.27
2	81.93	10.61	2.56	4.91	84.58	10.73	2.66	2.03
3	83.88	8.96	2.04	5.12	87.01	9.0	2.2	1.79
4	85.51	7.58	1.78	5.13	88.85	7.65	1.89	1.61
5	86.86	6.52	1.56	5.07	90.35	6.58	1.63	1.44
compl	95.26	0.0	0.0	4.74				

incTP full feature set, gold morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	57.85	33.01	5.69	3.45	59.19	32.84	6.07	1.9
1	77.6	14.69	3.34	4.36	80.01	14.83	3.54	1.63
2	82.32	10.68	2.76	4.24	84.82	10.75	2.94	1.49
3	84.31	9.09	2.2	4.4	87.19	9.08	2.38	1.35
4	85.98	7.68	1.98	4.36	88.98	7.68	2.14	1.19
5	87.3	6.59	1.72	4.38	90.49	6.6	1.84	1.06
compl	95.72	0.0	0.0	4.28				

incTP standard feature set, no morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	57.07	32.63	5.36	4.93	58.56	32.6	5.65	3.19
1	76.98	14.44	3.2	5.38	79.59	14.67	3.3	2.44
2	81.83	10.51	2.62	5.04	84.47	10.68	2.69	2.16
3	83.75	8.88	2.07	5.3	86.87	8.93	2.21	1.99
4	85.37	7.54	1.81	5.28	88.78	7.67	1.89	1.67
5	86.64	6.47	1.57	5.32	90.21	6.59	1.61	1.59
compl	94.99	0.0	0.0	5.01				

incTP standard feature set, gold morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	57.76	32.94	5.71	3.59	59.19	32.81	6.08	1.92
1	77.49	14.62	3.36	4.53	79.97	14.78	3.55	1.71
2	82.23	10.65	2.77	4.35	84.73	10.74	2.9	1.62
3	84.25	9.02	2.22	4.52	87.2	9.03	2.37	1.41
4	85.83	7.65	1.97	4.55	88.97	7.68	2.11	1.24
5	87.18	6.55	1.74	4.53	90.45	6.61	1.81	1.12
compl	95.58	0.0	0.0	4.42				

UD-German-HDT

PreTra, gold forced in beam

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	37.76	49.95	4.48	7.8	38.67	49.19	4.95	7.19
1	64.78	28.41	0.67	6.14	65.7	28.22	0.87	5.21
2	74.49	20.48	0.25	4.79	75.3	20.24	0.34	4.12
3	78.69	17.17	0.11	4.02	79.44	16.96	0.15	3.45
4	81.34	15.17	0.06	3.43	82.0	14.97	0.09	2.94
5	83.53	13.41	0.05	3.01	84.2	13.24	0.07	2.49
compl	96.42	0.0	0.0	3.58				

PreTra, beam=50

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	33.86	45.85	6.82	13.47	40.85	50.91	3.89	4.35
1	59.12	25.69	1.96	13.23	67.29	29.98	0.62	2.11
2	68.3	18.31	0.88	12.5	76.98	21.95	0.12	0.95
3	71.71	15.33	0.5	12.46	80.79	18.68	0.04	0.5
4	73.54	13.51	0.35	12.61	83.07	16.62	0.02	0.29
5	75.06	11.92	0.29	12.74	84.94	14.87	0.01	0.18
compl	86.2	0.0	0.0	13.8				

incTP full feature set, no morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	39.71	53.32	1.4	5.57	41.53	53.42	1.68	3.38
1	65.14	28.36	0.68	5.82	67.86	28.82	0.64	2.68
2	74.01	19.88	0.38	5.73	76.88	20.28	0.36	2.47
3	77.52	16.65	0.22	5.61	80.46	16.98	0.24	2.32
4	79.74	14.62	0.15	5.49	82.77	14.87	0.15	2.21
5	81.61	12.85	0.12	5.42	84.71	13.07	0.12	2.1
compl	94.98	0.0	0.0	5.02				

incTP standard feature set, no morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	39.54	53.09	1.59	5.78	41.45	53.25	1.85	3.46
1	64.88	28.22	0.75	6.16	67.77	28.73	0.72	2.77
2	73.77	19.8	0.41	6.02	76.82	20.27	0.39	2.53
3	77.25	16.6	0.24	5.91	80.4	16.97	0.26	2.38
4	79.44	14.56	0.17	5.83	82.71	14.86	0.17	2.26
5	81.31	12.84	0.12	5.73	84.65	13.08	0.13	2.14
compl	94.6	0.0	0.0	5.4				

PTB

PreTra, gold forced in beam

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	57.89	31.09	2.53	8.49	58.0	30.53	2.86	8.6
1	80.19	12.35	0.59	6.87	80.16	12.25	0.66	6.94
2	88.02	6.57	0.2	5.21	88.02	6.5	0.22	5.25
3	91.23	4.32	0.1	4.35	91.37	4.29	0.1	4.24
4	93.21	3.13	0.06	3.6	93.33	3.13	0.06	3.48
5	94.46	2.51	0.03	3.0	94.54	2.52	0.04	2.9
compl	97.57	0.0	0.0	2.43				

PreTra-NN-scorer

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	46.67	22.17	5.85	25.32	63.9	27.92	3.41	4.76
1	61.97	8.19	1.8	28.04	82.2	14.26	1.06	2.48
2	66.73	3.8	0.67	28.8	89.24	8.76	0.37	1.63
3	68.25	2.49	0.26	29.0	91.98	6.75	0.18	1.09
4	68.92	1.85	0.14	29.1	93.46	5.59	0.13	0.83
5	69.66	1.53	0.09	28.72	94.37	4.96	0.06	0.6
compl	71.98	0.0	0.0	28.02				

PreTra, beam=50

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	51.0	24.77	6.06	18.17	63.14	30.62	2.25	3.98
1	71.31	9.09	1.7	17.9	84.47	12.98	0.51	2.04
2	77.31	4.51	0.56	17.62	91.75	7.33	0.08	0.84
3	79.04	2.76	0.26	17.95	94.47	5.01	0.06	0.46
4	79.66	2.0	0.17	18.18	95.73	3.95	0.03	0.28
5	80.26	1.65	0.1	17.98	96.42	3.37	0.02	0.19
compl	82.06	0.0	0.0	17.94				

PreTra, beam=10

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	52.01	26.7	5.11	16.18	64.45	32.45	1.45	1.64
1	71.85	9.42	1.52	17.21	85.59	13.41	0.34	0.66
2	77.14	4.49	0.58	17.78	92.17	7.34	0.09	0.4
3	78.61	2.66	0.29	18.44	94.74	5.01	0.02	0.22
4	79.22	1.9	0.17	18.7	95.92	3.93	0.01	0.14
5	79.87	1.54	0.11	18.49	96.63	3.28	0.0	0.09
compl	81.7	0.0	0.0	18.3				

PreTra, beam=10, no cost-augmented training

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	49.09	21.54	9.83	19.54	63.01	33.04	1.87	2.09
1	68.28	7.7	3.06	20.96	83.38	14.88	0.47	1.27
2	73.76	3.86	1.08	21.3	90.2	8.99	0.11	0.7
3	75.28	2.38	0.5	21.84	92.88	6.63	0.03	0.47
4	75.87	1.78	0.28	22.07	94.27	5.4	0.02	0.32
5	76.39	1.47	0.19	21.95	95.03	4.75	0.02	0.21
compl	77.76	0.0	0.0	22.24				

incTP full feature set, no morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	52.37	0.0	34.08	13.55	57.16	0.0	33.55	9.29
1	74.66	0.0	11.47	13.87	81.61	0.0	11.32	7.07
2	80.8	0.0	5.36	13.84	89.28	0.0	5.2	5.52
3	82.87	0.0	3.15	13.98	92.5	0.0	2.97	4.53
4	83.94	0.0	2.16	13.9	94.06	0.0	2.04	3.9
5	84.57	0.0	1.69	13.74	94.84	0.0	1.6	3.56
compl	86.79	0.0	0.0	13.21				

incTP standard feature set, no morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	52.15	0.0	33.99	13.86	56.85	0.0	33.5	9.65
1	74.42	0.0	11.49	14.09	81.17	0.0	11.43	7.4
2	80.62	0.0	5.35	14.03	88.75	0.0	5.23	6.03
3	82.69	0.0	3.11	14.2	92.17	0.0	2.97	4.86
4	83.75	0.0	2.16	14.09	93.85	0.0	2.04	4.11
5	84.35	0.0	1.71	13.95	94.64	0.0	1.62	3.74
compl	86.41	0.0	0.0	13.59				

UD-PTB

PreTra, gold forced in beam

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	37.48	47.33	4.75	10.44	38.11	46.83	5.2	9.87
1	65.18	25.0	1.56	8.27	65.49	25.03	1.74	7.73
2	78.99	13.74	0.66	6.6	78.81	13.81	0.72	6.66
3	85.66	8.18	0.29	5.87	85.68	8.34	0.33	5.66
4	89.55	5.5	0.15	4.8	89.52	5.55	0.18	4.75
5	91.88	4.12	0.1	3.9	91.82	4.23	0.11	3.85
compl	96.89	0.0	0.0	3.11				

PreTra-NN-scorer

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	27.0	36.7	9.58	26.72	44.25	43.57	7.11	5.07
1	46.01	18.32	3.66	32.02	66.22	27.85	2.49	3.44
2	56.92	8.87	1.53	32.67	79.17	17.1	0.94	2.78
3	60.94	4.72	0.6	33.74	86.01	11.49	0.41	2.09
4	62.39	2.92	0.3	34.4	89.72	8.6	0.18	1.5
5	63.21	2.1	0.16	34.53	91.77	6.84	0.14	1.25
compl	65.59	0.0	0.0	34.41				

PreTra, beam=50

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	29.28	36.92	12.83	20.97	42.16	50.15	3.69	4.01
1	52.91	19.01	5.1	22.99	67.15	29.45	1.15	2.24
2	65.33	9.9	2.33	22.45	80.11	18.4	0.33	1.16
3	69.34	5.41	1.1	24.15	86.41	12.71	0.15	0.74
4	71.19	3.44	0.61	24.76	89.6	9.8	0.08	0.52
5	72.07	2.57	0.38	24.97	91.59	8.06	0.04	0.32
compl	74.31	0.0	0.0	25.69				

PreTra, beam=10

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	30.14	36.87	13.76	19.23	43.77	52.56	2.19	1.48
1	54.34	18.14	5.62	21.89	69.28	29.46	0.53	0.72
2	65.83	9.27	2.57	22.34	81.44	17.9	0.18	0.48
3	69.4	5.03	1.19	24.38	87.18	12.49	0.06	0.27
4	71.11	3.29	0.61	24.99	90.06	9.78	0.02	0.14
5	71.91	2.45	0.38	25.26	91.66	8.25	0.02	0.08
compl	74.21	0.0	0.0	25.79				

PreTra, beam=10, no cost-augmented training

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	30.85	39.16	9.74	20.25	46.8	48.64	2.65	1.92
1	53.96	18.95	3.89	23.2	71.28	26.95	0.67	1.11
2	65.31	9.38	1.76	23.56	83.57	15.56	0.17	0.69
3	68.92	4.84	0.81	25.42	89.61	9.93	0.06	0.41
4	70.5	3.04	0.38	26.08	92.5	7.24	0.02	0.24
5	71.25	2.2	0.19	26.37	94.05	5.82	0.0	0.13
compl	73.96	0.0	0.0	26.04				

incTP full feature set, no morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	36.98	49.34	2.22	11.46	39.92	49.03	2.63	8.42
1	64.13	24.11	0.75	11.0	68.4	24.3	0.83	6.46
2	77.14	12.44	0.42	10.01	81.64	12.49	0.46	5.41
3	83.29	6.97	0.23	9.51	88.53	6.96	0.25	4.26
4	86.06	4.55	0.11	9.28	91.9	4.49	0.14	3.46
5	87.54	3.31	0.08	9.07	93.61	3.29	0.09	3.01
compl	91.43	0.0	0.0	8.57				

incTP standard feature set, no morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	35.94	48.73	2.68	12.65	39.38	48.42	3.12	9.08
1	63.09	23.98	0.91	12.01	67.75	24.22	1.06	6.98
2	76.32	12.39	0.51	10.78	81.09	12.48	0.58	5.85
3	82.35	6.97	0.23	10.45	88.0	7.0	0.27	4.74
4	85.16	4.57	0.14	10.13	91.37	4.54	0.16	3.93
5	86.58	3.36	0.11	9.96	93.19	3.35	0.12	3.33
compl	90.55	0.0	0.0	9.45				

incTP standard feature set, gold morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	36.89	49.34	2.37	11.39	39.84	49.06	2.81	8.3
1	63.99	24.21	0.84	10.97	68.03	24.49	0.94	6.54
2	77.02	12.55	0.44	9.99	81.25	12.71	0.48	5.57
3	83.01	7.11	0.22	9.65	88.18	7.15	0.26	4.4
4	85.74	4.66	0.12	9.48	91.58	4.64	0.13	3.65
5	87.23	3.43	0.1	9.24	93.47	3.38	0.12	3.03
compl	91.34	0.0	0.0	8.66				

UD-English-EWT

PreTra, gold forced in beam

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	41.48	42.61	4.39	11.52	42.35	43.01	4.69	9.96
1	67.88	23.68	0.79	7.65	68.81	23.83	0.86	6.5
2	81.82	12.55	0.21	5.41	81.88	12.4	0.23	5.48
3	88.24	6.84	0.07	4.85	88.56	6.86	0.08	4.51
4	91.84	4.26	0.03	3.86	91.95	4.26	0.06	3.73
5	93.68	3.07	0.01	3.24	93.6	3.16	0.01	3.22
compl	94.11	0.0	0.0	5.89				

PreTra, beam=50

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	32.56	37.42	7.38	22.64	44.83	44.18	4.69	6.29
1	55.92	19.83	2.02	22.23	70.65	25.35	0.71	3.28
2	69.23	9.86	0.75	20.15	84.67	13.74	0.11	1.49
3	73.44	4.82	0.35	21.39	91.08	8.1	0.04	0.79
4	75.33	2.69	0.17	21.81	94.05	5.53	0.01	0.42
5	76.32	1.76	0.11	21.82	95.44	4.29	0.0	0.27
compl	76.63	0.0	0.0	23.37				

PreTra, beam=10

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	34.91	41.35	4.05	19.69	48.75	46.37	1.97	2.91
1	57.27	20.74	1.0	20.99	73.81	25.2	0.14	0.85
2	69.17	9.76	0.44	20.63	86.79	12.73	0.03	0.45
3	73.49	4.59	0.21	21.72	92.9	6.84	0.01	0.24
4	75.52	2.41	0.09	21.98	95.69	4.17	0.0	0.14
5	76.31	1.49	0.06	22.14	97.03	2.89	0.0	0.08
compl	77.37	0.0	0.0	22.63				

PreTra, beam=10, no cost-augmented training

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	32.17	34.88	10.85	22.09	45.56	46.42	4.2	3.82
1	53.27	17.79	4.11	24.83	69.75	27.43	0.89	1.93
2	65.28	8.85	1.63	24.24	82.22	16.45	0.19	1.14
3	69.23	4.5	0.69	25.58	88.69	10.64	0.04	0.64
4	71.13	2.57	0.35	25.95	92.11	7.59	0.02	0.28
5	71.84	1.7	0.28	26.18	93.71	6.09	0.01	0.19
compl	72.91	0.0	0.0	27.09				

PreTra, beam=10, no cost-augmented training, standard feature set

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	32.64	37.17	9.14	21.04	44.84	47.51	4.23	3.42
1	53.8	18.79	3.29	24.12	69.32	27.54	1.22	1.93
2	65.99	9.27	1.31	23.43	82.33	16.23	0.3	1.14
3	69.91	4.71	0.53	24.85	88.9	10.33	0.08	0.68
4	72.16	2.68	0.28	24.88	91.96	7.59	0.02	0.42
5	73.16	1.82	0.16	24.86	93.83	5.93	0.02	0.22
compl	73.89	0.0	0.0	26.11				

incTP full feature set, no morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	39.91	42.6	2.22	15.28	45.1	42.39	2.6	9.91
1	62.95	20.87	0.94	15.23	70.0	21.0	1.05	7.95
2	75.65	10.02	0.41	13.92	82.77	10.03	0.48	6.72
3	80.72	5.07	0.14	14.07	89.53	4.97	0.18	5.31
4	83.28	2.88	0.11	13.73	92.69	2.87	0.1	4.34
5	84.41	1.95	0.05	13.58	94.48	1.75	0.09	3.68
compl	86.74	0.0	0.0	13.26				

incTP full feature set, gold morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	39.33	42.23	2.43	16.01	45.1	41.95	2.81	10.14
1	62.24	20.83	0.96	15.98	69.96	20.98	1.11	7.94
2	74.71	9.85	0.41	15.03	82.76	9.83	0.54	6.86
3	80.16	4.9	0.11	14.82	89.62	4.67	0.3	5.42
4	82.86	2.73	0.09	14.33	92.79	2.57	0.15	4.48
5	83.86	1.9	0.02	14.21	94.41	1.6	0.13	3.86
compl	85.94	0.0	0.0	14.06				

incTP standard feature set, no morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	40.0	42.52	2.17	15.31	45.48	42.49	2.41	9.62
1	62.76	20.82	0.91	15.51	70.17	21.13	0.99	7.71
2	75.03	9.99	0.41	14.58	82.69	10.11	0.44	6.75
3	80.2	5.0	0.15	14.65	89.46	4.93	0.24	5.37
4	82.95	2.87	0.09	14.09	92.7	2.86	0.11	4.33
5	83.99	1.92	0.07	14.01	94.33	1.74	0.09	3.84
compl	86.46	0.0	0.0	13.54				

incTP standard feature set, gold morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	39.31	42.13	2.43	16.14	45.0	42.04	2.55	10.41
1	62.28	20.62	0.98	16.12	69.95	20.79	1.11	8.16
2	74.72	9.95	0.37	14.97	82.68	9.92	0.54	6.86
3	79.94	4.97	0.15	14.94	89.51	4.85	0.26	5.38
4	82.54	2.84	0.08	14.54	92.68	2.75	0.13	4.44
5	83.62	1.96	0.06	14.36	94.24	1.67	0.12	3.96
compl	86.16	0.0	0.0	13.84				

UD-Hungarian-Szeged

PreTra, gold forced in beam

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	32.86	50.94	2.78	13.43	35.46	48.46	4.05	12.03
1	60.3	27.14	0.74	11.82	61.45	26.35	1.38	10.82
2	71.21	17.61	0.41	10.77	72.34	17.2	0.71	9.74
3	76.32	13.12	0.3	10.26	77.57	12.86	0.45	9.12
4	81.01	9.95	0.24	8.8	81.91	9.76	0.3	8.03
5	83.94	7.97	0.15	7.94	84.77	7.77	0.23	7.22
compl	91.89	0.0	0.0	8.11				

PreTra, beam=50

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	25.15	36.31	7.83	30.7	47.29	43.48	3.03	6.19
1	45.52	15.83	3.38	35.26	70.15	24.99	0.81	4.05
2	52.42	8.76	2.02	36.8	81.06	16.22	0.32	2.4
3	54.56	5.44	1.38	38.62	86.91	11.44	0.13	1.52
4	56.18	3.72	0.98	39.12	90.2	8.6	0.07	1.13
5	56.32	2.52	0.65	40.51	92.65	6.59	0.02	0.73
compl	60.0	0.0	0.0	40.0				

PreTra, beam=10

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	26.42	34.48	7.95	31.15	52.57	41.96	2.34	3.12
1	46.01	14.56	2.81	36.61	74.96	22.57	0.21	2.26
2	51.75	7.72	1.54	38.99	84.89	13.53	0.12	1.46
3	53.41	5.18	0.74	40.67	89.76	9.17	0.02	1.06
4	54.85	3.21	0.62	41.31	92.63	6.7	0.0	0.67
5	54.93	2.16	0.37	42.54	94.46	5.08	0.0	0.46
compl	58.52	0.0	0.0	41.48				

incTP full feature set, no morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	29.16	45.75	8.18	16.9	33.66	45.71	9.12	11.5
1	57.54	20.74	3.72	18.0	64.27	22.87	3.59	9.27
2	66.48	13.27	1.91	18.34	74.06	14.96	1.81	9.17
3	68.73	10.0	1.06	20.21	79.12	11.2	1.18	8.51
4	71.14	7.11	0.68	21.07	83.19	7.99	0.68	8.14
5	72.65	5.21	0.44	21.7	85.75	5.88	0.34	8.03
compl	79.12	0.0	0.0	20.88				

incTP full feature set, gold morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	29.54	46.84	7.87	15.75	33.36	47.08	8.47	11.09
1	58.39	21.51	3.64	16.46	63.85	23.38	3.68	9.09
2	67.43	13.86	1.88	16.83	73.8	14.76	2.35	9.09
3	70.56	10.35	1.01	18.08	78.88	11.03	1.52	8.57
4	73.26	7.27	0.79	18.68	82.93	7.85	0.89	8.33
5	74.83	5.25	0.52	19.4	86.07	5.82	0.49	7.61
compl	81.15	0.0	0.0	18.85				

incTP standard feature set, no morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	29.01	45.72	8.08	17.19	33.49	45.66	8.91	11.94
1	56.75	20.33	4.03	18.89	63.53	22.85	3.72	9.9
2	65.77	13.15	1.95	19.14	73.44	14.85	1.99	9.71
3	68.04	9.73	1.09	21.15	78.54	10.91	1.37	9.18
4	70.28	7.06	0.68	21.97	82.45	8.05	0.86	8.65
5	71.98	5.34	0.37	22.32	85.45	6.01	0.44	8.1
compl	78.14	0.0	0.0	21.86				

incTP standard feature set, gold morph. features

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	29.32	46.92	7.63	16.14	32.92	47.03	8.39	11.65
1	57.69	21.16	3.96	17.19	62.84	23.33	4.05	9.78
2	66.92	13.73	1.95	17.4	72.92	15.14	2.38	9.56
3	69.78	10.33	1.09	18.8	77.94	11.3	1.58	9.18
4	72.51	7.17	0.8	19.52	82.27	7.8	1.06	8.87
5	74.21	5.36	0.5	19.93	85.47	5.98	0.54	8.02
compl	80.0	0.0	0.0	20.0				

jwcdg

	accuracy				stability			
dist	cip	cpred	wpred	wip	cip	cpred	wpred	wip
0	49.81	18.71	10.65	20.84	52.96	18.43	10.07	18.54
1	67.99	6.25	5.24	20.52	73.71	5.9	4.89	15.5
2	72.59	3.81	4.45	19.15	79.61	3.54	4.11	12.74
3	74.83	3.2	3.59	18.38	81.95	2.94	3.36	11.76
4	76.37	2.73	3.05	17.85	83.9	2.42	2.9	10.78
5	77.63	2.39	2.62	17.35	85.66	2.13	2.42	9.79
compl	85.35	0.0	0.0	14.65				

Appendix C.

Predictability sets

These are the dependency relations marked as *predictable* and *lex_predictable* for the incremental gold standard generation algorithm in Section 4.3.

HDT

These sets were taken from Beuck and Menzel (2013) and not modified further to ensure compatibility of results.

predictable : SUBJ, SUBJC, PN, CJ

lex_predictable : OBJA, OBJD, OBJC, PRED, OBJP, AUX, PART, S

PoS : noun-like: NN and NE, verb-like: VVFIN, VAFIN. All other PoS are not modified.

EN-LTH

Conversion of the Penn Dependency Treebank with the LTH converter (Johansson and Nugues 2008).

predictable : SBJ

lex_predictable : PMOD, ROOT, VC, CONJ, IM, PRD, SUB, OPRD, PRT, LGS, LOC-PRD, EXTR, DTV, PUT, PRD-PRP, PRD-TMP, LOC-OPRD

Noun-like PoS : NN, NNS, NNP, NNPS, PRP, $, CD, JJ, IN, RP, TO

Verb-like PoS : VBD, VBP, VBZ, VB, MD, VBN

UD

predictable : nsubj, nsubj:pass, csubj, csubj:pass, root

lex_predictable : obj, aux, aux:pass,ccomp, xcomp, compound:prt,cop, expl,expl:pv, expl:pass, expl:impers, iobj, obl, obl:npmod, obl:tmod, orphan, fixed, case, goeswith

PoS : No distinction was made between noun-like and verb-like PoS.

Eidesstattliche Versicherung Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, den Unterschrift