Online Learning of Latent Linguistic Structure with Approximate Search

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung.

Vorgelegt von

Anders Björkelund

aus Lund, Schweden

Hauptberichter:	Prof. Dr. Jonas Kuhn
Mitberichter:	Prof. Dr. Anders Søgaard

Tag der mündlichen Prüfung: 30. November 2018

Institut für Maschinelle Sprachverarbeitung der Universität Stuttgart

2019

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Stuttgart, January 22, 2019

Place, Date

Anders Björkelund

Contents

List of Figures

List of Tables

Abstract

Automatic analysis of natural language data is a frequently occurring application of machine learning systems. These analyses often revolve around some linguistic structure, for instance a syntactic analysis of a sentence by means of a tree. Machine learning models that carry out **structured prediction**, as opposed to simpler machine learning tasks such as classification or regression, have therefore received considerable attention in the language processing literature.

As an additional twist, the sought **linguistic structures** are sometimes not directly modeled themselves. Rather, prediction takes place in a different space where the same linguistic structure can be represented in more than one way. However, in a standard supervised learning setting, these **prediction structures** are not available in the training data, but only the linguistic structure. Since multiple prediction structures may correspond to the same linguistic structure, it is thus unclear which prediction structure to use for learning. One option is to treat the prediction structure as **latent** and let the machine learning algorithm guide this selection.

In this dissertation we present an abstract **framework** for structured prediction. This framework supports latent structures and is agnostic of the particular language processing task. It defines a set of hyperparameters and task-specific functions which a user must implement in order to apply it to a new task. The advantage of this modularization is that it permits comparisons and reuse across tasks in a common framework.

The framework we devise is based on the **structured perceptron** for learning. The perceptron is an online learning algorithm which considers one training instance at a time, makes a prediction, and carries out an update if the prediction was wrong. We couple the structured perceptron with **beam search**, which is a general purpose search algorithm. Beam search is, however, only approximate, meaning that there is no guarantee that it will find the optimal structure in a large search space. Therefore special attention is required to handle search errors during training. This has led to the development of special update methods such as *early* and *max-violation* updates.

The contributions of this dissertation sit at the intersection of machine learning and natural language processing. With regard to language processing, we consider three tasks: Coreference resolution, dependency parsing, and joint sentence segmentation and dependency parsing. For **coreference resolution**, we start from an existing latent tree model and extend it to accommodate non-local features drawn from a greater structural context. This requires us to sacrifice exact for approximate search, but we show that, assuming sufficiently advanced update methods are used for the structured perceptron, then the richer scope of features yields a stronger coreference model. We take a transition-based approach to **dependency parsing**, where dependency trees are constructed incrementally by transition system. Latent structures for transition-based parsing have previously not received enough attention, partly because the characterization of the prediction space is non-trivial. We provide a thorough analysis of this space with regard to the Arc-Standard with Swap transition system. This characterization enables us to evaluate the role of latent structures in transition-based dependency parsing. Empirically we find that the utility of latent structures depend on the choice of approximate search – for greedy search they improve performance, whereas with beam search they are on par, or sometimes slightly ahead of, previous approaches. We then go on to extend this transition system to do **joint sentence segmentation and dependency parsing**. We develop a transition system capable of handling this task and evaluate it on noisy, non-edited texts. With a set of carefully selected baselines and data sets we employ this system to measure the effectiveness of syntactic information for sentence segmentation. We show that, in the absence of obvious orthographic clues such as punctuation and capitalization, syntactic information can be used to improve sentence segmentation.

With regard to machine learning, our contributions of course include the framework itself. The task-specific evaluations, however, allow us to probe the learning machinery along certain boundary points and draw more general conclusions. A recurring observation is that some of the standard update methods for the structured perceptron with approximate search – e.g., early and max-violation updates – are inadequate when the predicted structure reaches a certain size. We show that the primary problem with these updates is that they may **discard training data** and that this effect increases as the structure size increases. This problem can be handled by using more advanced update methods that commit to using all the available training data. Here, we propose a new update method, **DLaSO**, which consistently outperforms all other update methods we compare to. Moreover, while this problem potentially could be handled by an increased beam size, we also show that this cannot fully compensate for the structure size and that the more advanced methods indeed are required.

Überblick

Bei der automatisierten Analyse natürlicher Sprache werden in der Regel maschinelle Lernverfahren eingesetzt, um verschiedenste linguistische Information wie beispielsweise syntaktische Strukturen vorherzusagen. **Structured Prediction** (dt. etwa Strukturvorhersage), also der Zweig des maschinellen Lernens, der sich mit der Vorhersage komplexer Strukturen wie formalen Bäumen oder Graphen beschäftigt, hat deshalb erhebliche Beachtung in der Forschung zur automatischen Sprachverarbeitung gefunden.

In manchen Fällen ist es vorteilhaft, die gesuchte **linguistische Struktur** nicht direkt zu modellieren und stattdessen **interne Repräsentationen** zu lernen, aus denen dann die gewünschte linguistische Information abgeleitet werden kann. Da die internen Repräsentationen allerdings selten direkt in Trainingsdaten verfügbar sind, sondern erst aus der linguistischen Annotation inferiert werden müssen, kann es vorkommen, dass dabei mehrere äquivalente Strukturen in Frage kommen. Anstatt nun vor dem Lernen eine Struktur beliebig auszuwählen, kann man diese Entscheidung dem Lernverfahren selbst überlassen, welches dann selbständig die für das Modell am besten passende auszuwählen lernt. Unter diesen Umständen bezeichnet man die interne, nicht a priori bekannte Repräsentation für eine gesuchte Zielstruktur als **latent**.

Diese Dissertation stellt ein **Structured Prediction Framework** vor, mit dem man den Vorteil latenter Repräsentationen nutzen kann und welches gleichzeitig von konkreten Anwendungsfällen abstrahiert. Diese Modularisierung ermöglicht die Wiederverwendbarkeit und den Vergleich über mehrere Aufgaben und Aufgabenklassen hinweg. Um das Framework auf ein reales Problem anzuwenden, müssen nur einige Hyperparameter definiert und einige problemspezifische Funktionen implementiert werden.

Das vorgestellte Framework basiert auf dem **Structured Perceptron**. Der Perceptron-Algorithmus ist ein inkrementelles Lernverfahren (eng. online learning), bei dem während des Trainings einzelne Trainingsinstanzen nacheinander betrachtet werden. In jedem Schritt wird mit dem aktuellen Modell eine Vorhersage gemacht. Stimmt die Vorhersage nicht mit dem vorgegebenen Ergebnis überein, wird das Modell durch ein

entsprechendes Update angepasst und mit der nächsten Trainingsinstanz fortgefahren. Der Structured Perceptron wird im vorgestellten Framework mit **Beam Search** kombiniert. Beam Search ist ein approximatives Suchverfahren, welches auch in sehr großen Suchräumen effizientes Suchen erlaubt. Es kann aus diesem Grund aber keine Garantie dafür bieten, dass das gefundene Ergebnis auch das optimale ist. Das Training eines Perceptrons mit Beam Search erfordert deshalb besondere Update-Methoden, z.B. *Early-* oder *Max-Violation-Updates*, um mögliche Vorhersagefehler, die auf den Suchalgorithmus zurückgehen, auszugleichen.

Diese Dissertation ist an der Schnittstelle zwischen maschinellem Lernen und maschineller Sprachverarbeitung angesiedelt. Im Bereich Sprachverarbeitung beschäftigt sie sich mit drei Aufgaben: Koreferenzresolution, Dependenzparsing und Dependenzparsing mit gleichzeitiger Satzsegmentierung. Das vorgestellte Modell zur **Koreferenzresolution** ist eine Erweiterung eines existierenden Modells, welches Koreferenz mit Hilfe latenter Baumstrukturen repräsentiert. Dieses Modell wird um Features erweitert, mit denen nicht-lokale Abhängigkeiten innerhalb eines größeren strukturellen Kontexts modelliert werden. Die Modellierung nicht-lokaler Abhängigkeiten macht durch die kombinatorische Explosion der Features die Verwendung eines approximativen Suchverfahrens notwendig. Es zeigt sich aber, dass das so entstandene Koreferenzmodell trotz der approximativen Suche dem Modell ohne nicht-lokale Features überlegen ist, sofern hinreichend gute Update-Verfahren beim Lernen verwendet werden. Für das **Dependenzparsing** verwenden wir ein transitionsbasiertes Verfahren, bei dem Dependenzbäume inkrementell durch Transitionen zwischen definierten Zuständen konstruiert werden. Im ersten Schritt erarbeiten wir eine umfassende Analyse des latenten Strukturraums eines bekannten Transitionssystems, nämlich Arc-Standard mit Swap. Diese Analyse erlaubt es uns, die Rolle der latenten Strukturen in einem transitionsbasierten Dependenzparser zu evaluieren. Wir zeigen dann empirisch, dass die Nützlichkeit latenter Strukturen von der Wahl des Suchverfahrens abhängt – in Kombination mit Greedy-Search verbessern sich die Ergebnisse, in Kombination mit Beam-Search bleiben sie gleich oder verbessern sich leicht gegenüber vergleichbaren Modellen. Für die dritte Aufgabe wird der Parser noch einmal erweitert: wir entwickeln das Transitionssystem so weiter, dass es neben **syntaktischer Struktur auch Satzgrenzen vorhersagt** und testen das System auf verrauschten und unredigierten Textdaten. Mit Hilfe sorgfältig ausgewählter Baselinemodelle und Testdaten messen wir den Einfluss syntaktischer Information auf die Vorhersagequalität von Satzgrenzen und zeigen, dass sich in Abwesenheit orthographischer Information wie Interpunktion und Groß- und Kleinschreibung das Ergebnis durch syntaktische Information verbessert.

Zu den wissenschaftlichen Beiträgen der Arbeit gehört einerseits das Framework selbst. Unsere problemspezifischen Experimente ermöglichen es uns darüber hinaus, die Lernverfahren zu untersuchen und allgemeinere Schlußfolgerungen zu ziehen. So finden wir z.B. in mehreren Experimenten, dass die etablierten Update-Methoden, also Early- oder Max-Violation-Update, nicht mehr gut funktionieren, sobald die vorhergesagte Struktur eine gewisse Größe überschreitet. Es zeigt sich, dass das Hauptproblem dieser Methoden das **Auslassen von Trainingsdaten** ist, und dass sie desto mehr Daten auslassen, je größer die vorhergesagte Struktur wird. Dieses Problem kann durch bessere Update-Methoden vermieden werden, bei denen stets alle Trainingsdaten verwendet werden. Wir stellen eine neue Methode vor, **DLaSO**, und zeigen, dass diese Methode konsequent bessere Ergebnisse liefert als alle Vergleichsmethoden. Überdies zeigen wir, dass eine erhöhte Beamgröße beim Suchen das Problem der ausgelassenen Trainingsdaten nicht kompensieren kann und daher keine Alternative zu besseren Update-Methoden darstellt.

Acknowledgements

First and foremost I am indebted to my advisor Jonas Kuhn for supporting me throughout this thesis work. You have a remarkable skill of distant supervision where, on the one hand you have let me pursue my own ideas without interfering, yet you could always jump into whichever topic I was currently working on, spawning new ideas and asking the right questions. I am also thankful to Anders Søgaard who accepted to be my external reviewer. Anders' work has been very inspiring and I have followed it throughout my years as a doctoral student. I take great pride in having you on my committee. Moreover, I should express my gratitude to Andres Bruhn and Ngoc Thang Vu for volunteering to be on the committee and participating in a smooth defense.

My journey towards this dissertation started as an undergraduate student in Lund. Pierre Nugues and Richard Johansson introduced me to the subject of computational linguistics. While I was sometimes ambivalent about Pierre's advice back in Lund, in hindsight I have realized that his ideas and comments typically were more motivated and profound than I was able to comprehend at the time. I am also grateful to Love Hafdell, my best buddy and colleague back in Lund. If it wasn't for you, I would probably have left academia after the master's. Finally, Jacek Malec played a role as an external senior academic. I'm very appreciative of our conversations about doctoral studies and academic life which have been valuable at critical decision points in my carreer.

The transition to Stuttgart would not have been possible without Bernd Bohnet. You were instrumental in my moving to Stuttgart and your work has had a large bearing on this dissertation. My first office mate in Stuttgart was Richárd Farkas. Although it was great to leave the CS department in Lund for a proper CL department, sharing the office with a computer scientist provided a sense of familiarity when switching universities and countries. Not least did I enjoy the occasional beers we had – *Egészségedre, rúgjunk be bazdmeg!*

In any case, however, the most inspiring person I have worked with in Stuttgart was Wolfgang Seeker. I never ceased to be impressed by your skills of both linguistics and

computer science – to me you are a true *computational linguist*. Although our conversations often ranged way outside the scope of work, be it politics or random geekery, your knowledge of the subject matter and your ability to keep both topics in mind at all times must not be underestimated. As much as I wished you all the best when you left the IMS after graduating, it was with mixed emotions that I saw you depart given that you defined a very strong gold standard – as a colleague, as a scientist, and, perhaps most importantly, as a friend.

Another critical development during my dissertation work was when Agnieszka Faleńska joined the IMS. You have been a bright beacon of sanity in my day-to-day life. This applies both to our work at the IMS, but maybe more vitally as a friend, not least because of our common endeavors in parenting. Although I am excited to leave Stuttgart after all these years I will miss our dates at playgrounds and restaurants, venting topics related both to work and parenthood.

More generally, it is essential to recognize the importance of an institution like the IMS. Before I came to Stuttgart I couldn't have imagined the good spirit and breadth of work going on at this department. I've had the pleasure to work with and be inspired by numerous great minds including André Blessing, Fabienne Braune, Fabienne Cap, Özlem Çetinoğlu, Grzegorz Dogil, Diego Frassinelli, Markus Gärtner, Boris Haselbach, Charles Jochim, Kerstin Jung, Hans Kamp, Wiltrud Kessler, Maximilian Köper, Gabriella Lapesa, Florian Laws, Lukas Michelbacher, Thomas Müller, Sebastian Padó, Tillmann Pross, Nils Reiter, Arndt Riester, Christian Rohrer, Antje Roßdeutscher, Christian Scheible, Dominik Schlechtweg, Sabine Schulte im Walde, Hinrich Schütze, Antje and Katrin Schweitzer, Jason Utt, Michael Walsh, Xiang Yu, Alessandra Zarcone, and Sina Zarieß.

I would also like to convey my appreciation to the people who typically do not get sufficient recognition in our day-to-day work. This includes the secretaries Sabine Dieterle, Sybille Laderer, Sabine Mohr, and Barbara Schäfer, who always handled all administrative issues perfectly. From the technical perspective, however, Edgar Hoch must be mentioned. You are a brilliant sysadmin and I think you get considerably less credit than you deserve. Thank you for maintaining our infrastructure, day and night, weekdays and weekends, workdays and holidays. Without you the IMS would not hold up.

Finally, I am also indebted to my local and non-local family (no pun intended, though see Chapter 3): Gisela, Yarle, and Lars, as well as Ute, Jürgen, Laura, Jens, and Sebastian. Most importantly, however, this work could not have been completed without the limitless support of Emma and Kathrin. Thank you for cheering me up when I was in doubt. And thank you for bearing with my mental and physical absence during the completion of this dissertation. I dedicate this dissertation to you.

Chapter 1

Introduction

Predicting a **linguistic structure** over a discrete text input is a frequent problem in language processing (Smith, 2011). Part-of-speech tagging, syntactic parsing, and coreference resolution are some classical problems. Although these problems may be quite unrelated linguistically, they share a property that is intimidating from a computational perspective: The space of potential output structures tends to be exponential in the size of the input. For instance, the number of possible part-of-speech tag sequences for a sentence is exponential in the number of words of the sentence. It is therefore not computationally tractable to enumerate all possible sequences in order to find the highest scoring sequence according to a some scoring function that conditions the score on a large structural context in the tag sequence.

Nevertheless, efficient methods for these problems exist. They often rely on breaking the problem into smaller pieces that can be handled efficiently. In some cases such a simplification reduces the problem into a sequence of classification problems. But classifiers are optimized with respect to the *local* decisions (e.g., the part-of-speech tag of a single word in a sentence) in such a sequence rather than the *global* output structure (e.g., an assignment of a sequence of part-of-speech tags for a full sentence). This is part of the motivation for **structured predictors** which are optimized towards a global scoring function that takes the full output structure into consideration. Moreover, structured predictors do not suffer from the typical drawbacks of classifiers such as local normalization and error propagation and have a more principled approach to how *structural context* in the output can be integrated as features in the model (e.g., exploiting part-of-speech tags of surrounding words).

The large space of outputs nonetheless prohibits enumeration of all candidates. Thus, structured predictors often also rely on making simplifying assumptions such as breaking

the problem into subproblems. One option is to impose restrictions on the structural context visible to each subproblem such that they can be easily and efficiently solved independently. The limited scope with respect to structural context then enables *exact search*, meaning that when the solutions to these subproblems are put together to form the overall output structure, the output remains optimal with respect to a global scoring function.

Another option, which is the one we will be pursuing in this dissertation, is to relax the requirement that the predicted output structure is optimal with respect to a global scoring function. The advantage is that the limitations on the scope of features vanishes and a larger structural context can be exploited. The price we pay for this is that we have to resort to **approximate search**. This may involve straightforward **greedy search**, where the locally optimal choice is made for each subproblem. Greedy search is, however, unlikely to yield the globally optimal solution, as this might depend on making non-optimal choices for some subproblems. We may thus want to apply a search algorithm that explores multiple choices for some subproblems. One of the most common approximate search algorithms found in the language processing literature is **beam search** where not only the highest locally scoring solution is explored, as in greedy search, but a number of parallel paths are explored simultaneously. Beam search is attractive because it is only a constant factor slower than greedy search, yet it explores several solutions to the individual subproblems in parallel. Moreover, it does not require specific knowledge or assumptions about the problem at hand, as is the case of heuristic search methods, such as, e.g., A* search.

The search problem arises from the exponential space of output structures. But for many language processing tasks prediction is carried out in a yet larger space. For instance, in transition-based dependency parsing a dependency tree is constructed through an incremental derivation using a transition system (Nivre, 2008). In this case the search problem is equivalent to finding the optimal sequence of transitions to derive a given dependency tree. Typically, however, there is more than one transition sequence that can derive the same dependency tree, i.e., there is a many-to-one mapping from derivations to trees. Given annotated training data it is then an open question which sequences to use for training. The simplest approach is to use a heuristic that deterministically selects one *canonical sequence* for each dependency tree. Whether these canonical sequences also constitute the easiest to learn or predict is an open question. In the domain of coreference resolution it has been shown that simple deterministic sequence derivations (Soon et al., 2001) can be largely improved over either by carefully adapting the heuristic (Ng and Cardie, 2002b) or by outsourcing these decisions to the machine learning component and

treating them as **latent** (Fernandes et al., 2012). The idea is that the derivation is not specified beforehand, but that the machine learning algorithm chooses which derivation to learn. The underlying assumption is that the latent derivations should be easier to learn and generalize better to unseen data at test time.

One of the most commonly used machine learning algorithms for structured prediction is the **structured perceptron** (Collins, 2002), an extension of the perceptron algorithm (Rosenblatt, 1958) to structured outputs. It is an error-driven online algorithm that processes training instances one at a time and applies updates if the current model fails at the current instance. It is attractive because it is very efficient, yet it has proven to be very strong empirically. The combination of approximate search with the structured perceptron has been successfully applied to a variety of language processing tasks. It has been also shown that dedicated methods such as *early* (Collins and Roark, 2004) or *max-violation* (Huang et al., 2012) *updates* must be used in order to handle search errors during learning. Even though the use of these techniques is widespread, most previous work have utilized them for a single language processing task and a systematic understanding of how they relate to each other within and across specific tasks is lacking.

In this dissertation we formulate the structured perceptron with beam search into a common abstract framework which also facilitates the inclusion of latent structure. The framework is agnostic of the specific machine learning task and can be applied to a given task by defining appropriate functions and data structures. Of course, the framework also comes with its own set of task-agnostic hyper-parameters and discrete choices, such as the choice of update or search method. We instantiate this framework and carry out systematic comparisons of this learning machinery for several tasks. This enables us to view established approaches to specific tasks from a higher vantage point and consider similarities and dissimilarities between task-specific approaches in a broader perspective. In this manner, we probe known tasks along different axes by moving, e.g., between exact search and approximate search, between greedy and beam search, or from static to latent structure. This enables us to investigate questions such as which *role the **size** of the input and output* plays, how ***latent structure** behaves when comparing **greedy** and **beam** search*, and the *importance of the **update methods** used for the **structured perceptron** with **inapproximate search***.

One of the chief insights from our empirical evaluation relates the *size* of the input and output to the *choice of update* methods. With our selection of tasks and experiments, we find that for large instances the choice of update method must be made with special care. The established techniques do not satisfactorily carry over to these problems as they tend to **discard training data**. Instead, these problems require techniques that ensure to use

all available training data, such as *Learning as Search Optimization* (LaSO; Daumé III and Marcu, 2005) or, as we propose, **delayed** LaSO (DLaSO; Björkelund and Kuhn, 2014). We show how DLaSO outperforms other update methods and can be successfully applied to problems with very large inputs.

In addition to the high-level perspective on structured prediction with approximate search, we also contribute to the understanding of the state of the art with respect to specific language processing tasks. For **coreference resolution**, we extend a state-of-the-art latent structure coreference model (Fernandes et al., 2012) to incorporate **non-local features** that exploit a greater structural context, turning what is known as a *mention-pair* model into an *entity-mention* model. In terms of **dependency parsing**, we characterize the level of *spurious ambiguity* in the SwapStandard (Nivre, 2009) transition system and devise a **non-deterministic oracle** for this system. For **sentence segmentation** and **dependency parsing** we construct a transition system that carries out these two tasks **jointly**, operating at the level of documents. The system processes a document from left to right and introduces sentence boundaries in tandem with building a syntactic representation. In a set of controlled experiments we use this system to show how **syntax** can be **helpful** for **sentence segmentation**.

In the next section we will situate the discussion above with respect to one particular task, namely coreference resolution. In Section 1.2 we introduce the additional tasks of dependency parsing and sentence segmentation more briefly. An outline of the dissertation as a whole is provided in Section 1.3. Section 1.4 contains a list of publications.

1.1 An Example from Coreference Resolution

To make the above discussion a bit more concrete we now consider an example. *Coreference resolution* is the task of grouping referring expressions (or *mentions*) of the same entity in a text into equivalence classes (or *clusters*). Specifically, each cluster contains all and only all mentions of a specific entity. An example text taken from the OntoNotes corpus is shown in Figure 1.1. The coreferring mentions in the text are enclosed by square brackets and the subscripts signify which equivalence class a given mention belongs to. For instance, the first entity, a, corresponding to the company *Drug Emporium Inc.*, is referred to four times: once as a proper name (a_1), and three times as common noun phrases (a_2, a_3 and a_4), where the head nouns may or may not be identical. The second entity in the example refers to the person *Gary Wilber*, who is referred to twice as explicit proper noun phrases (b_1 and b_3), and once by a pronoun (b_2). Note that the text additionally

includes references to several other entities, e.g., the persons *Philip T. Wilber* and *Robert E. Lyons III*, as well as positions within the company, e.g., *chairman* and *chief operating officer*. These entities are however only referred to once. In the coreference jargon these types of mentions are called *singletons*.[1]

> [Drug Emporium Inc.]$_{a_1}$ said [Gary Wilber]$_{b_1}$ was named CEO of [this drug-store chain]$_{a_2}$. [He]$_{b_2}$ succeeds his father, Philip T. Wilber, who founded [the company]$_{a_3}$ and remains chairman. Robert E. Lyons III, who headed the [company]$_{a_4}$'s Philadelphia region, was appointed president and chief operating officer, succeeding [Gary Wilber]$_{b_3}$.

Figure 1.1: An excerpt of a document with the mentions from two clusters marked. The subscripts indicate which clusters the mentions belong to.

The traditional data-driven approach to coreference resolution (Soon et al., 2001) has been to decompose the problem into a set of classification problems, where pairs of two mentions are considered in isolation. This way, a binary classifier which discriminates between coreferent and disreferent pairs can be applied. An assignment of clusters can subsequently be constructed by placing mentions that are deemed coreferent into one cluster, i.e., taking the transitive closure over all (or a select subset of) coreferent pairs of mentions. While this approach renders the task very accessible from a machine-learning perspective, it suffers from three primary drawbacks:

- **Reduction to Classification.** This approach does not model a coreference assignment across a document as a whole. This means that documents of varying length yield a varying number of training examples, possibly creating an imbalance in the number of training instances with respect to training documents.

- **Limitation of Feature Scope.** As a consequence, the notion of a document as a coherent text with a set of coreferent clusters is lost. This means that features that consider a greater structural context are not accessible.

- **Choice of Pairs.** The number of possible mention pairs grows quadratically with the number of mentions. It is non-trivial to decide which pairs to use as training instances. Considering all possible pairs will lead to a very skewed training set, where most pairs are disreferent.

[1] To keep the example simple we do not bracket these singleton clusters. As for the technical treatment of singletons, we will return to this in Chapter 3.

Structured Approach. To address these points, we take a structured modeling approach. Rather than training a classifier for individual pairs of mentions, we aim to predict an assignment of clusters for a full document. Motivated by previous work (Bengtson and Roth, 2008; Fernandes et al., 2012, *inter alia*), which has shown that the pairwise models can be very strong in certain cases, we nevertheless want to maintain a model that establishes pairwise links between mentions. We thus take a graph-based approach, where mentions correspond to nodes and coreferent nodes are linked. Specifically, we symbolize a pair of coreferent mentions by a directed arc, pointing from left to right with respect to the linear order of the text. Additionally, the first mention of each cluster has an incoming arc from a virtual dummy mention ROOT which is placed before all mentions in the document. By imposing that every mention has exactly one incoming arc the graph in fact must be a *directed tree* rooted at the virtual mention. Modeling the clusters in a document as a tree implicitly creates an internal structure, where one out of potentially many incoming arcs must be singled out for each mention when representing a given coreference clustering.

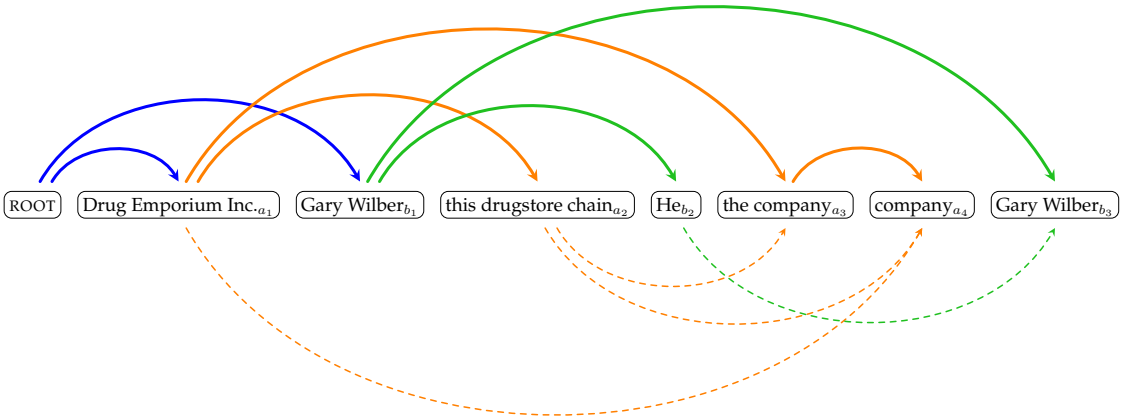


Figure 1.2: Example of correct pairwise coreference links using the example text from Figure 1.1. Arcs drawn in orange and green connect mentions within the two clusters from the example. While all arcs are correct, we argue that the solid ones are preferable over the dashed ones.

As an illustration, consider Figure 1.2, which shows the mentions from the example as a graph with pair-wise links. The links from the virtual mention are marked separately in blue, while the orange and green arcs connect mentions within one of the two clusters. However, for several mentions there is more than one correct incoming arc. Each mention

has one incoming solid arc drawn on top, and potentially more drawn as a dashed lines underneath. This implies that there is more than one possible tree that encodes the same clustering. Here, we want to emphasize an important distinction between *output space* and *prediction space*. Output space corresponds to the ultimate linguistic structure we aim to find (in this case clusters without internal structure), whereas prediction space denotes the space of structures we are learning and predicting (in this case directed trees over the mentions). It is easy to see that any clustering can be represented by at least one tree and, as we just pointed out, more than one tree may represent the same clustering. That is, the prediction space is larger than the output space and we can construct a many-to-one surjective mapping from prediction space into output space.

Latent Structure. As a consequence of the many-to-one mapping between prediction and output spaces, two important and related issues arise: (1) Are there better or worse structures with respect to learning and generalization? and (2) How do we choose which structures we want to use for training?

We believe that the solid arcs drawn above the mentions in Figure 1.2 should be easier to learn than using some of the dashed arcs drawn below the mentions. For instance, the second mention of *Gary Wilber* (b_3) can be linked either to the pronoun *he* (b_2), or the preceding noun phrase *Gary Wilber* (b_1). Among the two competing choices, we argue that the arc (b_1, b_3) should be easier to learn and generalize better since the two involved mentions have an exact string match over proper nouns. Similarly, one could argue that the mention *the company* (a_3) should be linked with *Drug Emporium Inc.* (a_1), since a company is likely to be coreferent with a noun phrase ending in *Inc.*.

As for the second issue, which structure to try to learn, one option is to apply a heuristic that selects one out of several incoming arcs. Such a heuristic can have a varying level of complexity, ranging from associating every mention with its closest preceding coreferent mention (Soon et al., 2001) to trying to enforce certain preferential hierarchies between, e.g., pronouns, common noun phrases, and proper noun phrases (Ng and Cardie, 2002b). But designing such a heuristic is non-trivial, and it needs to be done anew for each new task that is considered. We will therefore pursue a different strategy: Instead of trying to design a clever heuristic ourselves, we will outsource these decisions to the machine learning algorithm. As part of the learning process, we will query the learner for a *latent structure* and use this as the goal for learning (Fernandes et al., 2012). The expectation is that latent structure should be easier to learn and generalize better to unseen data, e.g., as in the example in Figure 1.2. In the next chapter we will review how we accomplish this by means of the perceptron learning algorithm.

A Note on Complexity. Using terminology from combinatorial mathematics, a coreference clustering is called a *partition* of a set: A finite set of objects is subdivided into non-empty subsets, such that every element is in exactly one subset. The number of possible partitions for a set of n objects is given by the Bell number B_n. The Bell numbers grow extremely quickly: for 7 mentions, as in the example from Figure 1.1, there are 877 possible partitions. For 10 or 20 mentions, there are $115,975$ or $51,724,158,235,372$, respectively.

But recall that a partition belongs in the *output space*, that is, there is no internal structure (a tree) in the subsets. The *prediction space*, i.e., the space of trees over the mentions, grows even faster. Specifically, for n mentions, the possible number of trees is given by $n!$.[2,3] For example, for 7 mentions the number of possible trees is $5,040$.

The main point of this exercise is to illustrate that we can generally not enumerate all possible assignments, neither in prediction space nor in output space.

Features, Scoring, and Search. Assume that there is a scoring function that assigns a score to an arc between two mentions. This scoring function can take into consideration features of the two involved mentions, but it cannot access mentions farther away in the tree structure. That is, the score of an arc is independent of other arcs. Moreover, let the score of a full coreference assignment, i.e., a tree over the mentions, correspond to the sum of the scores of the individual arcs. We call such a model *arc-factored*. The task of finding the highest scoring tree can be regarded as a *search problem*. In an arc-factored model, the search problem can be solved quite easily: Make a left-to-right pass over the mentions and for each mention select the highest scoring incoming arc. In terms of complexity, this procedure needs to consider on the order of $O(n^2)$ arcs and this is also the overall time complexity of this procedure. Given that the number of possible trees grows tremendously faster, this is somewhat surprising. The reason for this discrepancy is that the optimal incoming arc for each mention can be selected locally, without considering other arcs, which is precisely the underlying assumption of an arc-factored model. However, the approach we just outlined suffers from exactly the problem discussed previously – pairs of mentions are regarded in isolation and the greater structural context (i.e., the tree) cannot be taken into consideration while making individual choices.

The search procedure outlined above – making a left-to-right pass over the mentions

[2]This is *not* the number of trees in the general case over n nodes. It applies to the specific case where nodes are arranged in a linear order and each node is connected to exactly one preceding node.

[3]The formula can be derived easily: The k'th mention with respect to the linear order has $(k-1)+1=k$ possible choices for an arc originating on its left – the $(k-1)$ preceding mentions, plus the virtual dummy node. Thus, the number of possible trees for n mentions is given by $1 \cdot 2 \cdot ... \cdot (n-1) \cdot (n) = n!$

– implicitly constructs the tree as it goes. It is thus possible to exploit the partially built structure to the left of a given mention to extract features. These features may be predicated both on the presence or absence of arcs on the left. For instance, when considering how to connect a pronoun such as *he*, it may be preceded by proper name mentions of persons of male gender as well as other mentions using the same pronoun (*he*). The internal structure amongst these mentions, i.e., the presence or absence of arcs, may then be helpful to retrieve an appropriate coreferent mention for this pronoun. While this means that we can access a greater structural context, it also means that the model is no longer arc-factored. The search procedure is then no longer guaranteed to return the highest scoring tree. There might be cases where the optimal local choice, i.e., the highest scoring incoming arc, does not lead to the optimal global choice, i.e., the highest scoring tree. Rather, the optimal global choice might depend on some locally sub-optimal decisions.

In the most general case we would like a scoring function that can consider an arbitrary amount of contextual structure, i.e., it can follow any other arcs while computing the score of a single arc. In this case there is no choice but to enumerate all possible trees and score them one by one, which, as noted above, is not computationally tractable. In conclusion, we are left with a trade-off between scope of features and guarantees on optimality of the predicted structure: With a narrow enough scope of features, the optimal structure can be guaranteed, but as the scope of features is extended, this guarantee is lost. In terms of the search problem, we say in the former case that the problem can be solved using an *exact search* procedure, whereas in the latter we have to rely on *approximate search*.

Learning with Approximate Search. Learning with approximate search for structured prediction is a well-studied problem in the NLP and machine learning communities. A particularly established approach is the combination of beam search and the structured perceptron. It was first introduced by Collins and Roark (2004) in the context of phrase structure parsing. They noted that the learning process tended to be unsuccessful when search errors caused the correct solution to be lost by the approximate search. Instead, they proposed to make parameter updates only with respect to the prefix of decisions leading up to the point where the correct solution was lost and showed that this led to better results. In the context of the coreference example above, this update strategy, known as *early updates*, makes a left-to-right pass over the document pursuing multiple possible trees. When it reaches a point where the correct solution is no longer reachable, it stops, updates, skips the remainder of the document, and moves on to the next document. This effectively means that large portions of the training data may be discarded.

The theoretical justification for early updates was later given by Huang et al. (2012), who showed that search errors should not be used to penalize the parameters of a model. They also proposed an alternative update method, *max-violation updates*, that makes better use of training data. Specifically, it proceeds longer than early updates and therefore converges faster. Nevertheless, there is no guarantee that max-violation updates use the full input/output structure for learning.

Combining beam search with the structured perceptron using early or max-violation updates have subsequently been applied successfully to a wide range of language processing tasks (see Section 2.5 for a longer discussion). However, some of the problems we will be considering, e.g., coreference resolution, have considerably larger search spaces where neither early nor max-violation updates are able to exploit the training data sufficiently. To this end, we will introduce a novel update method, DLaSO, an adaptation of the Learning as Search Optimization (LaSO) framework (Daumé III and Marcu, 2005). This update method makes maximal use of training data and is a key ingredient in order to apply the beam search and perceptron idea to some of the tasks we consider.

1.2 Additional Tasks

In addition to coreference resolution, this dissertation considers two other language processing tasks: Transition-based dependency parsing and sentence segmentation. We will define the tasks more precisely in the relevant content chapters, but below we provide a quick introduction to each of them.

Transition-based Dependency Parsing. Dependency Syntax is a framework to describe the syntactic relations between words in a sentence. It organizes the words of a sentence as a graph by means of asymmetric binary relations between words. These relations encode the syntactic structure of the sentence, e.g., what is the subject of a verb, or which noun phrase a prepositional phrase modifies. *Dependency parsing* is the task of finding the correct dependency graph for a given input sentence. There is a wide range of linguistic theories that model the syntactic structure of a sentence through dependencies, such as Functional Generative Description Grammar (Sgall et al., 1986), Meaning Text Theory (Mel'čuk, 1988), or Word Grammar (Hudson, 1984). Nivre (2005) provides a brief but broad overview of these linguistic theories as well as an (somewhat aged) overview of computational approaches to dependency parsing. A more recent and on-going initiative, aimed at developing a unified and language-independent dependency-based framework for representing the syntactic structure of a sentence is the Universal Dependencies

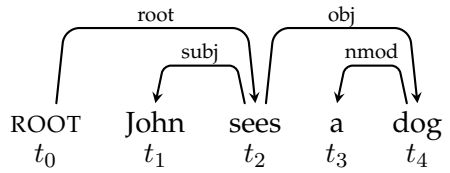


Figure 1.3: An example dependency tree. The grammatical role of each word is indicated on the incoming arc from its governor.

project (Nivre et al., 2016).[4] While these formalisms agree on using a graph to model the syntax of a sentence, they vary in design decisions regarding the representation of certain linguistic phenomena. Moreover, in some formalisms, the dependency graph is further constrained to be a tree. This is also the setting that we will consider in this dissertation. Besides that, our approach makes no assumptions on how certain linguistic phenomena are represented, nor the choice of labels used on the arcs in the dependency trees.

An example dependency tree is displayed in Figure 1.3. The figure shows directed arcs that connect the words. Moreover, the arcs are assigned labels that denote the grammatical function that holds between connected words. A pair of tokens connected through an arc is typically referred to as *governor-modifier* or *head-dependent* pair.[5] Arcs are drawn pointing from the governor to the modifier. For instance, in the example, *sees* is the governor of *John*. Additionally, the arc is labeled to indicate *John* as the subject of *sees*.

There are two main algorithmic approaches to dependency parsing. *Graph-based* approaches tackle the problem in much the same way as we discussed coreference resolution above: Words are considered nodes and possible arcs between them are assigned a score. However, most of these approaches typically employ exact search algorithms, have a rather limited feature scope, and high asymptotic time complexities (see Section 4.6 for a longer discussion). We will instead be pursuing a *transition-based* approach. Here, dependency trees are incrementally constructed through a sequence of transitions, rather similar to shift-reduce parsing for context free grammars. Transition-based approaches are efficient in terms of time complexity and enable the use of a rich feature scope, however they do sacrifice exactness of search. Moreover, they suffer from the problem of *spurious ambiguity*, i.e., that a tree can be derived by more than one transition sequence. For example, when deriving the dependency tree in Figure 1.3, the SwapStandard transition system has the option to attach the left and right modifiers of *sees* in arbitrary order.

[4] `http://universaldependencies.org/`

[5] In this dissertation we will, however, use the governor-modifier terminology and reserve the term head for the head word of phrases in phrase-structure trees, as discussed in Chapter 3.

That is, attaching *John* to *sees* may happen before or after attaching *dog*. In other words, the transition system permits (at least) two possible derivations of the example dependency tree. We thus face a similar situation as for coreference: prediction space (the space of transition sequences) will be larger than output space (the space of dependency trees).

Our contributions with respect to transition-based parsing are two-fold. First, we exhaustively describe the spurious ambiguities of SwapStandard and characterize the full prediction space of this system. This allows us to create a novel heuristic (or *static oracle* as it is known in the dependency parsing literature) that minimizes the length of a transition sequence that can be used for training. We empirically compare this oracle with previous static ones in two regards: First, by considering their coverage on standard training data sets. Second, by evaluating dependency parsers trained with the different oracles.

Second, the characterization of spurious ambiguity allows us to create *non-deterministic oracles* that allow for multiple transition sequences for a given sentence. This enables us to move from the traditional training regime of transition-based parsers, where a single static transition sequence is used for training, to a situation where the sequence is left latent. That is, similarly to the case of coreference resolution, moving from a pre-defined tree structure to one that is dynamically induced by the machine learning component. We conduct experiments comparing static and non-deterministic oracles using both a greedy, classifier-based parser, as well as one using a structured model with beam search. Interestingly, we find that, while the use of a non-deterministic oracle can improve performance for the greedy parser, it makes little difference to the performance of a beam search parser.

Sentence Segmentation. *Sentence Segmentation* is the task of dividing a sequence of words in a text (or *document*) into units corresponding to sentences. Systems for many other NLP tasks that operate at the level of sentences, e.g., syntactic parsing, tend to take the sentence segmentation for granted. But when dealing with real-world texts sentence segmentation is an essential component in a processing pipeline. The task has, however, historically not received much attention in the scientific community. There are two primary reasons for this: First, most research on other NLP tasks is carried out on data sets that have been prepared for experimentation on the task at hand. This isolates the problem under consideration and simplifies the experimental setup. Considering the example of syntactic parsing again, the standard experimental setup starts out with data sets that

Raw-steel production by the nation's mills decreased 0.7% last week to 1,816,000 tons from 1,828,000 tons the previous week, the American Iron and Steel Institute said. Last week's output fell 1.7% from the 1,848,000 tons produced a year earlier. The industry used 81.6% of its capability last week, compared with 82.2% the previous week and 86.2% a year ago. The American Iron and Steel Institute reported: The capability utilization rate is a calculation designed to indicate at what percent of its production capability the industry is operating in a given week.

Figure 1.4: An example document annotated with sentence boundaries shown. Sentence-initial words are underlined.

have already been split into sentences.[6] This follows naturally since syntactic analyses typically only encompass single sentences. Moreover, standard evaluation metrics for syntactic parsing may not be applicable if a sentence (as characterized by a gold standard corpus) has been divided into two or more sentences in a prediction scenario.

The second reason why sentence segmentation has received little attention is that it has partly been regarded as a solved task. As many data sets are drawn from copy-edited sources where capitalization of sentence-initial words and sentence-final punctuation provide very strong cues to the beginning and end of a sentence, the problem is not obvious in the scientific community. Figure 1.4 shows an example document taken from the Wall Street Journal (WSJ) corpus, one of the most established English data sets used for experimentation on syntactic parsing. The words that begin a new sentence are underlined. The example illustrates how orthographic clues are very strong indicators of where sentences end and begin – every new sentence in the example starts with a capitalized letter and is preceded by punctuation marking the end of the previous sentence. For texts such as the one in the example, sentence segmentation is indeed not a tough challenge. However, for texts that do not follow standard orthographic conventions, such as web content or historical manuscripts, sentence segmentation can be much more difficult. An example from the spoken domain, that can be regarded as potential output of a speech recognizer can be found in Figure 5.1 in Chapter 5.

As for historical manuscripts, a Latin example based on the New Testament is given in Figure 1.5. The example shows the first four verses of Matthew 2 from the 4th century Vulgate Bible. This Bible translation has historically been typeset with varying levels of consistency with respect to capitalization and punctuation. The example is drawn from the PROIEL treebank (Haug and Jøhndal, 2008) and we have followed their version of

[6] Another step that is often taken for granted in the context of syntactic parsing is tokenization.

[1]cum ergo natus esset Iesus in Bethleem Iudaeae in diebus Herodis regis ecce magi ab oriente venerunt Hierosolymam [2]dicentes ubi est qui natus est rex Iudaeorum vidimus enim stellam eius in oriente et venimus adorare eum [3]audiens autem Herodes rex turbatus est et omnis Hierosolyma cum illo [4]et congregans omnes principes sacerdotum et scribas populi sciscitabatur ab eis ubi Christus nasceretur

"[1]Now when Jesus was born in Bethlehem of Judaea in the days of Herod the king, behold, there came wise men from the east to Jerusalem, [2]Saying, Where is he that is born King of the Jews? for we have seen his star in the east, and are come to worship him. [3]When Herod the king had heard these things, he was troubled, and all Jerusalem with him. [4]And when he had gathered all the chief priests and scribes of the people together, he demanded of them where Christ should be born."

Figure 1.5: An example of a historical text, in this case the first four verses of Matthew 2 from the Latin Vulgate Bible (above) and its corresponding translation from King James Version (below). Verse numbers are superimposed as superscripts and sentence-initial tokens are underlined.

sentence boundaries and orthographic conventions faithfully. In addition we have superimposed verse numbers (which were not present in the 4th century).[7] For comparison, the example also includes a translation into (relatively) modern English taken from the King James Bible.[8]

The Bible example in Figure 1.5 demonstrates two things. First of all, the obvious fact that the Latin version lacks the standard orthographic clues which renders the task much more difficult in the first place. Second, it shows that the definition of sentences may be rather blurry. While we follow the PROIEL treebank for the sentence boundaries in the Latin version, we have used our own judgment for the English translation. We do not claim that our judgment forms a gold standard, but choose to point it out here in order to illustrate that even for a language processing task that may at first seem somewhat mundane, e.g., sentence segmentation, there are plenty of pitfalls and potential decisions that could be argued one way or another. In fact, even the previous example in Figure 1.4 displays some choice that could be discussed: In particular, the (potential) sentence following the phrase *The American Iron and Steel Institute reported:* could be regarded as its own sentence, but was not regarded as such by the creators of the WSJ corpus. Similar

[7]Verse numbers are taken from `https://www.biblestudytools.com/vul/matthew/2.html`

[8]While the first version of King James Bible was published in the early 17th century, this translation is from the *Standard Version*, dated 1769. The translation is taken from `https://www.kingjamesbibleonline.org/Matthew-Chapter-2/`

issues also arise in edited texts that may contain lists or quoted speech, where list items or quoted speech sometimes can be regarded as separate sentences, sometimes not.

In this dissertation we will approach the sentence segmentation problem *jointly* with dependency parsing. To this end we develop a transition system that processes full documents and splits it into sentences while simultaneously creating a syntactic analysis. Using this system we show that solving these tasks jointly can improve sentence segmentation results, indicating that sentence segmentation can profit from syntax. From the machine learning perspective the move from single-sentence dependency parsing to document parsing facilitate a comparison of the structured learning framework when moving from relatively short transition sequences (as in the case of sentences), to very long ones (i.e., parsing full documents). The experimental results strongly indicate how the update methods for approximate search must be thoughtfully chosen in order to leverage all available training data.

1.3 Outline and Contributions

The remainder of this dissertation is organized as follows:

Chapter 2 discusses the algorithmic framework that we will apply for the language processing tasks. It reviews the structured perceptron and beam search, as well as the dedicated update methods from the literature that are required for learning in this setting. Additionally, we introduce our own update method, *Delayed Learning as Search Optimization*, DLASO, that builds upon the Learning as Search Optimization framework (Daumé III and Marcu, 2005). We string these pieces together into a common framework which abstracts away from a specific language processing task. This framework will subsequently be instantiated in a task-specific manner in each of the following content chapters.

In **Chapter 3** we show how the beam search and structured perceptron framework can be applied to coreference resolution. This work builds upon recent successes that induce latent structure for coreference resolution (Fernandes et al., 2012; Durrett and Klein, 2013). The chief contribution is that we are able to fuse pair-wise models (Soon et al., 2001; Ng and Cardie, 2002b), that have a limited scope of features, with entity-mention models (Rahman and Ng, 2009), where features can be drawn from a larger structural context. A strictly pair-wise model can be regarded as a special case of our abstraction. In terms of learning, we demonstrate that, de-

spite the richer scope of features, early and max-violation updates are unable to surpass an arc-factored model with a more limited scope of features where the search problem can be solved exactly. We demonstrate that this is caused by the large size of the search problems and the fact that early and max-violation updates tend to discard training data. However, when we use DLaSO for learning, the extended scope of features yields results that outperform the arc-factored baseline.

In **Chapter 4** we turn our attention to transition-based dependency parsing. We do this in the context of the ArcStandard with Swap (Nivre, 2009) transition system. We make a careful analysis of spurious ambiguities in this system and devise a set of non-deterministic oracles. We show that these oracles can be used to induce latent transition sequences, as opposed to previous approaches, where transition sequences have been fixed a priori by a static oracle. Empirically, we find that the difference between non-deterministic and static oracles is subject to whether a classifier or a structured model is used: It is generally insignificant when beam search is used, but in the greedy setting, the non-deterministic oracles outperforms their static counterparts.

Chapter 5 extends the transition system from the previous chapter to parse not just sentences but full documents. To this end, we introduce a transition system that jointly segments sentences and parses them in one go. As in the case of coreference resolution, we find that early and max-violation updates tend to discard too much training data, yielding unsatisfactory results compared to a greedy baseline. By applying DLaSO updates we show that this problem can be circumvented. For the empirical evaluation we carefully select our baselines to isolate the utility of syntax. Our empirical results suggest that, given appropriate conditions with respect to orthographic clues and syntactic complexity, the extra access to syntactic context is beneficial when segmenting sentences.

Chapter 6 concludes this dissertation. We revisit the contributions and view them in the larger context of language processing. These contributions are broken down along the two axes of machine learning and language processing. Then we go on to discuss our contributions in the with regard to machine learning and language processing and potential implications to future work. This includes a main take-away message from the content chapters and its relationship to other language processing tasks, but also a discussion on how our machine learning contributions relate to deep learning and recent NLP research based on neural models.

1.4 Publications

This dissertation is primarily based on work from three publications:

- Björkelund, A. and Kuhn, J. (2014). Learning Structured Perceptrons for Coreference Resolution with Latent Antecedents and Non-local Features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 47–57, Baltimore, Maryland, USA. Association for Computational Linguistics

  The research presented in this dissertation drawn from this paper is based solely on my own contributions. Jonas Kuhn had an advisory role as I carried out this work. He also provided feedback on early drafts of the paper before its publication.

- Björkelund, A. and Nivre, J. (2015). Non-Deterministic Oracles for Unrestricted Non-Projective Transition-Based Dependency Parsing. In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 76–86, Bilbao, Spain. Association for Computational Linguistics

  The research presented in this dissertation drawn from this paper is based solely on my own contributions. Joakim Nivre had an advisory role as I carried out this work. He also provided feedback on early drafts of the paper before its publication.

- Björkelund, A., Faleńska, A., Seeker, W., and Kuhn, J. (2016). How to Train Dependency Parsers with Inexact Search for Joint Sentence Boundary Detection and Parsing of Entire Documents. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1924–1934, Berlin, Germany. Association for Computational Linguistics

  The research presented in this dissertation drawn from this paper is based solely on my contributions. Agnieszka Faleńska and Wolfgang Seeker assisted during preliminary experiments as I developed the ideas. Agnieszka Faleńska, Jonas Kuhn, and Wolfgang Seeker also provided feedback on early drafts of the paper before its publication.

To facilitate reproducibility and further research we have released the software implementations used in these papers (and in this dissertation) under open source licenses. They are available for download on the author's website.

In addition, the work that has led up to this dissertation has been shaped by experiences from work that has gone into other publications. Although they do not constitute a core part of this dissertation, it has been part of my project work while working towards this dissertation and the ideas therein have to a greater or lesser extent influenced this final product:

- Björkelund, A. and Farkas, R. (2012). Data-driven Multilingual Coreference Resolution using Resolver Stacking. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 49–55. Association for Computational Linguistics
- Bohnet, B., Björkelund, A., Kuhn, J., Seeker, W., and Zarriess, S. (2012). Generating Non-Projective Word Order in Statistical Linearization. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 928–939. Association for Computational Linguistics
- Björkelund, A. and Kuhn, J. (2012a). Comparing Non-projective Strategies for Labeled Graph-Based Dependency Parsing. In *Proceedings of COLING 2012: Posters*, pages 135–144. The COLING 2012 Organizing Committee
- Björkelund, A. and Kuhn, J. (2012b). Phrase Structures and Dependencies for End-to-End Coreference Resolution. In *Proceedings of COLING 2012: Posters*, pages 145–154. The COLING 2012 Organizing Committee
- Pradhan, S., Moschitti, A., Xue, N., Ng, H. T., Björkelund, A., Uryupina, O., Zhang, Y., and Zhong, Z. (2013). Towards Robust Linguistic Analysis using OntoNotes. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 143–152. Association for Computational Linguistics
- Björkelund, A., Cetinoglu, O., Farkas, R., Mueller, T., and Seeker, W. (2013). (Re)ranking Meets Morphosyntax: State-of-the-art Results from the SPMRL 2013 Shared Task. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 135–145. Association for Computational Linguistics
- Björkelund, A., Eckart, K., Riester, A., Schauffler, N., and Schweitzer, K. (2014). The Extended DIRNDL Corpus as a Resource for Coreference and Bridging Resolution. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*. European Language Resources Association (ELRA)

- Björkelund, A., Çetinoğlu, Ö., Faleńska, A., Farkas, R., Müller, T., Seeker, W., and Szántó, Z. (2014). The IMS-Wrocław-Szeged-CIS Entry at the SPMRL 2014 Shared Task: Reranking and Morphosyntax meet Unlabeled Data. In *Notes of the SPMRL 2014 Shared Task on Parsing Morphologically-Rich Languages*, Dublin, Ireland

- Faleńska, A., Björkelund, A., Çetinoğlu, Ö., and Seeker, W. (2015). Stacking or Supertagging for Dependency Parsing – What's the Difference? In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 118–129. Association for Computational Linguistics

- Versley, Y. and Björkelund, A. (2016). Off-the-Shelf Tools. In Poesio, M., Stuckardt, R., and Versley, Y., editors, *Anaphora Resolution: Algorithms, Resources, and Applications*, pages 237–266. Springer Berlin Heidelberg, Berlin, Heidelberg

- Björkelund, A., Falenska, A., Yu, X., and Kuhn, J. (2017). IMS at the CoNLL 2017 UD Shared Task: CRFs and Perceptrons Meet Neural Networks. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 40–51. Association for Computational Linguistics

Chapter 2

Machine Learning Framework

The previous chapter used the example of coreference resolution to outline the main challenges when faced with learning structured predictors. In this chapter, we will formalize these ideas and describe the algorithmic machinery that we apply to our selection of NLP tasks in subsequent chapters. Although we will use the coreference resolution example from the previous chapter to exemplify the concepts we discuss, we will keep the presentation at a level abstract enough not to be tied to a specific task.

This chapter will start with some basic definitions followed by a discussion on how to represent the linguistic data in vector space, i.e., feature extraction. After considering search algorithms we continue with discussing the (structured) perceptron and how to use it in the context of approximate search. A discussion of related work precedes the conclusion, where we summarize the learning framework. To this end, we will present a "recipe" of functions and (abstract) data structures that are required to instantiate the general framework. In subsequent content chapters, the framework will be instantiated with respect to particular tasks to validate the generality.

2.1 Notation and Objective

We will use uppercase calligraphic letters $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ to denote the *input*, *output*, and *prediction* spaces (cf. Chapter 1, page . Note that output and prediction spaces are in fact a function of the input, i.e. in coreference resolution the space of possible coreference clusters and coreference trees depends on the size of the input, although we will not make this notationally explicit as it is always clear from context what input is concerned. Members of these sets will be notated with bold lowercase letters. For instance, in the case of coreference resolution, an $\mathbf{x} \in \mathcal{X}$ denotes an input document, a $\mathbf{y} \in \mathcal{Y}$ a coreference

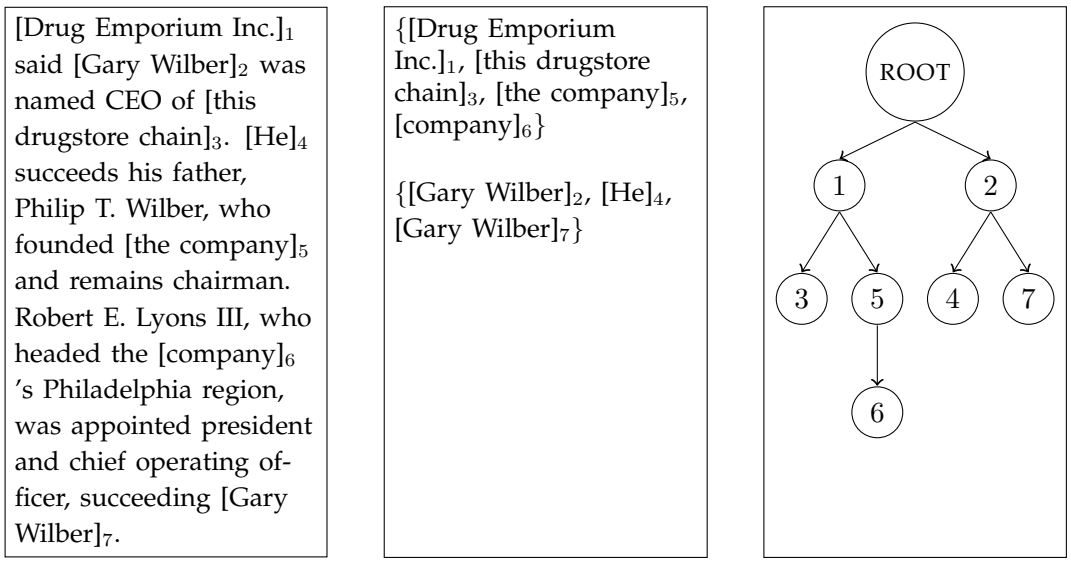


Figure 2.1: Examples of input, output, and prediction spaces.

assignment (i.e., a partition, or clustering), and a $\mathbf{z} \in \mathcal{Z}$ an assignment of pairwise coreference links (a coreference tree over mentions). This is illustrated in Figure 2.1, where the coreference example from the previous chapter is displayed in all three forms. The figure shows the mentions in the input document labeled 1 through 7 (left), the output (center), and a potential prediction (right) that encodes the clustering following the tree from Figure 1.2 in the introduction.

We will also assume that there is a mapping $\pi(z) : \mathcal{Z} \mapsto \mathcal{Y}$ that can map a prediction into output. Finally, we will also define a special subset $\mathcal{Z}_y$ of the prediction space $\mathcal{Z}$, which denotes all predictions that correspond to the same $\mathbf{y}$. In other words, they are equivalent under the mapping π, or $\mathcal{Z}_y = \{\mathbf{z} \in \mathcal{Z} | \pi(\mathbf{z}) = \mathbf{y}\}$. For example, in the case of coreference, all members of $\mathcal{Z}_y$ correspond to the specific coreference partition $\mathbf{y}$.

A prediction z in turn is made up of a sequence of state and decision pairs that we may expand as $\mathbf{z} = [z_0, z_1, ..., z_{n-1}] = [(s_0, d_0), (s_1, d_1), ..., (s_{n-1}, d_{n-1})]$, where s_i and d_i denote states and decisions, respectively. The states and decisions are task-specific and we define them in the corresponding content chapters but we can use coreference to give an example. The search algorithm for coreference that we sketched in Chapter 1 makes a left-to-right pass over the mentions, selecting for each mention the highest scoring incoming arc. In this case, each step correspond to one such (s_i, d_i) pair, where the decision d_i denotes the chosen arc, and the state s_i denotes the mention which is currently being

processed. Note that the last state-decision pair, (s_{n-1}, d_{n-1}), where decision d_{n-1} is applied to state s_{n-1} corresponds to a subsequent state s_n. As for the mapping π, which maps predictions into outputs, we will assume that it is also applicable to a prefix of a sequence, yielding a partial output (e.g., a partial coreference partition).

Given an input $\mathbf{x} \in \mathcal{X}$, the prediction problems we are concerned with can be summarized by the equation

$$\hat{\mathbf{z}} = \arg\max_{\mathbf{z} \in \mathcal{Z}} F(\mathbf{x}, \mathbf{z}). \tag{2.1}$$

Here, $\hat{\mathbf{z}}$ denotes the best prediction and F is a function that assigns a numerical score to an output structure $\mathbf{z}$ given the input $\mathbf{x}$. The $\arg\max$ expression looks for the $\mathbf{z} \in \mathcal{Z}$ that maximizes the value of F. Solving this problem is often not easy and it is sometimes referred to as the *arg max problem*, the *inference problem*, or the *decoding problem*. The difficulty lies in how large the space $\mathcal{Z}$ is and whether there are intrinsic properties of the problem that can be exploited advantageously to search this space efficiently. In the simplest case the decision sequence includes only one decision from a finite set of options, in which case we are dealing with a *classification problem*. If the sequence is longer it might still exhibit certain computational properties that allow for clever algorithmic treatment such as dynamic programming. In this dissertation we will make no such assumptions, but we will make two assumptions about the scoring function F: First, we assume that it is linear with respect to a global feature extraction function $\Phi : \mathcal{X} \times \mathcal{Z} \mapsto \mathbb{R}^m$ that maps the representation of an input and a prediction structure into high-dimensional vector space. This means that the score of a prediction structure $\mathbf{z}$ can be computed as the scalar product of a weight vector $\mathbf{w} \in \mathbb{R}^m$ and the feature representation of a prediction $\mathbf{z}$ as extracted by Φ.

The second assumption is that the feature extraction function factors over the local state-decision pairs of a sequence. That is, the *global* feature function Φ is composed of a sum of features extracted from each state-decision pair in a sequence by a *local* feature extraction function ϕ. In practice, this means that Equation 2.1 can be rewritten as

$$\hat{\mathbf{z}} = \arg\max_{\mathbf{z} \in \mathcal{Z}} F(\mathbf{x}, \mathbf{z}) = \arg\max_{\mathbf{z} \in \mathcal{Z}} \Phi(\mathbf{x}, \mathbf{z}) \cdot \mathbf{w} = \arg\max_{\mathbf{z} \in \mathcal{Z}} \sum_{(s,d) \in \mathbf{z}} \phi(\mathbf{x}, s, d) \cdot \mathbf{w}, \tag{2.2}$$

where $\mathbf{w} \in \mathbb{R}^m$ denotes the weight vector and $\phi : \mathcal{X} \times (S \times D) \mapsto \mathbb{R}^m$ denotes a feature extraction function that operates on the level of single state-decision pairs (with S and D denoting the space of possible states and decisions, respectively). The definition of ϕ

is task-specific and one of the components that a user is required to define in order to apply the learning machinery. Note that the addends in the sum on the right hand side correspond to individual scores of each state-decision pair in the sequence.

2.2 Feature Extraction Function

The feature extraction function $\phi(\mathbf{x}, s, d)$ is one of the most important user-defined components in the learning machinery. It needs to extract sufficiently descriptive features such that good and bad decisions can be discriminated. The co-domain of ϕ is a high-dimensional real-valued vector space $\mathbb{R}^m$ for some large m. While there are no strict requirements on the feature vectors, we consistently implement the feature extraction function by means of binary indicator functions and it can be regarded as $\phi = (\phi_1, ..., \phi_m)$, where each ϕ_i corresponds to some binary predicate. Two examples based on coreference resolution could be the following:

$$\phi_{752}(\mathbf{x}, s, d) = \begin{cases} 1 & \text{if the antecedent in decision } d \text{ is a pronoun} \\ 0 & \text{otherwise;} \end{cases}$$

$$\phi_{1,432,451}(\mathbf{x}, s, d) = \begin{cases} 1 & \text{if the last word of the antecedent in decision } d \text{ is } \textit{company} \\ 0 & \text{otherwise.} \end{cases}$$

These two examples and their indices are arbitrarily chosen but give a sense of how the feature vector is constructed. In practice feature functions are not designed at this level of granularity, but rather through the use of *feature templates*. Feature templates act as rules that instantiate multiple indicator functions. For instance, a feature template that indicates the last word of the antecedent mention in decision d. For a large vocabulary of thousands of words, say k, occurring in this position during training, this feature template spawns k unique indicator functions, where $\phi_{1,432,451}$ would be one of them.

As we are using a linear model, we also introduce cross-products between feature templates to increase separability. For instance, the template discussed above, that extracts the last word of the antecedent under decision d could be combined with a template that extracts the last word of the other mention involved in decision d (i.e., the mention to which the arc points). The combination of these two templates would then denote a new template comprised of yet more indicator functions than either of the constituent features due to multiplicativity.

For an efficient implementation of feature extraction we follow Bohnet (2010). First, lexicalized and other non-closed sets of values (e.g., part-of-speech tags, lemmas, etc) in the training data are gathered into category-specific symbol tables, assigning a unique integer to each value. Then, to encode a feature template the corresponding values are packed into a binary representation in the form of a `long` (i.e., a 64-bit integer) using bit shifting operations. The long value is then mapped into the dimensions of $\mathbb{R}^m$ using a hash function. Although this can potentially cause collisions between individual indicator functions it considerably speeds up feature extraction compared to using a hash function that handles collisions properly. Moreover, it means that the indicator functions need not be defined before training, but that new indicators can be introduced on the fly while learning, which occurs frequently in the presence of features predicated on structural context. Feature hashing has empirically been shown to be very robust, even when collisions are common (Ganchev and Dredze, 2008; Weinberger et al., 2009).

2.3 Navigating Prediction Space

With the formalities in place we now consider the problem of finding optimal derivations. First we will consider how the spaces $\mathcal{Z}$ and $\mathcal{Z}_y$ can be regarded as *search trees*. We will then continue to discuss how to navigate, i.e., search, these trees.

2.3.1 States, Decisions, and Search Trees

Recall the procedure to find the optimal coreference assignment discussed in Chapter 1: move left to right over the mentions in a document and, for each mention, find the highest scoring coreferent mention on the left. The space of possible decisions sequences $\mathcal{Z}$ can intuitively be visualized as a *search tree*.[1] A sketch of a search tree for the running coreference example (Figure 2.1) is shown in Figure 2.2. The nodes in the tree correspond to states, and the arcs between them to decisions. The labels on the arcs denote which pair-wise link would be introduced by applying the given decision. For instance, following an arc labeled $1 \mapsto 3$ in the search tree corresponds to the decision of selecting the arc between mentions labeled 1 and 3 for the coreference tree. The path corresponding to the example derivation from Figure 2.1 is highlighted in green. It should be noted that depending on the search space, it can sometimes be represented more compactly as a

[1] Not to be confused with coreference trees or dependency trees, nor with search trees designed for fast retrieval (i.e., search), such as binary search trees or B-trees. The search trees we discuss function as a visualization of possible paths from a start state to a goal state in the classical AI sense (Russell and Norvig, 2003, p. 69).

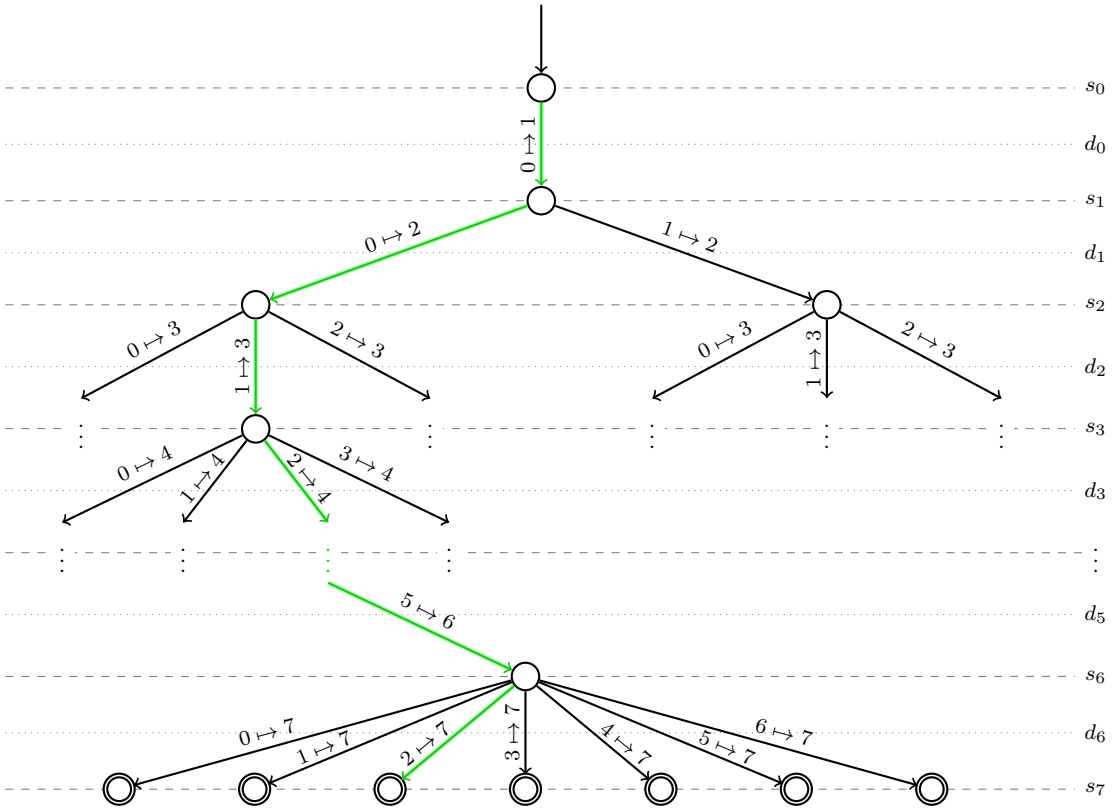


Figure 2.2: Search tree corresponding to the example from Figure 2.1. The green path corresponds to the specific coreference tree shown in Figure 2.1. The dashed and dotted horizontal lines indicate the progression of states (nodes) through decisions (arcs) numbered $s_0, s_1, ...$ and $d_0, d_1, ...$, respectively. The initial state is s_0 (at the top) and the final and terminal state is s_7 (bottom).

graph, where different (partial) paths can lead to equivalent states. While this is not the case for our coreference model, it happens in transition-based parsing, and we will see examples of such search graphs in Chapter 4.

The tree includes two special types of states: the *initial state* state s_0, and *terminal states*. The initial state, at the top, is indicated with an incoming arc. Terminal states, at the bottom, are indicated by doubly circled nodes. Any path from the initial state to any terminal state corresponds to a unique coreference tree, i.e., a unique $\mathbf{z} \in \mathcal{Z}$. The outgoing arcs from a state in the search tree show all possible decisions in this state. Specifically, we require a special function PERMISSIBLE(s) that can be used to probe a state for the set of decisions that are permissible in state s. A state is defined to be terminal when PERMISSIBLE returns the empty set. To simplify notation, we define a boolean-valued function TERMINAL(s) that returns true when a state is terminal and false otherwise, i.e., whether PERMISSIBLE returns the empty set or not.

The score of states and decisions is defined in accordance with Equation 2.2. That is, each decision is assigned a score given by the scalar product of the weight vector and the feature vector associated with this state-decision pair as extracted by the local feature extraction function ϕ. The score of a state is the sum of the scores of all decisions that lead to that state. In terms of the search tree in Figure 2.2 this means that we can think of individual arcs being associated with the score for this particular state-decision pair, and the score of a state is the sum along the path from the initial state.

The state representation needs to be defined for the specific task at hand such that it (a) accommodates the functions PERMISSIBLE and (b) that it enables access to features for the feature extraction function ϕ. Using the coreference example, this includes (a) a pointer to the next mention to process, in order to properly return the set of permissible decisions, as well as (b) a sufficiently rich representation of prediction and/or output structure built so far. That is, it may include the partial coreference tree built so far, such that feature extraction can leverage preceding decisions to link particular mentions. In addition to task-specific properties of states, we will assume that every state holds (1) a backpointer to its previous state, (2) a record of what decision was taken in the previous state, and (3) the current score of the state. The backpointers and previous decisions are necessary to track the path in the search tree, whereas the current score of the state is needed to be able to rank states.

In Figure 2.2, the path corresponding to the coreference tree from Figure 2.1 was highlighted in green. There may, however, be more than one path that results in the correct output structure $\mathbf{y}$, i.e., the set $\mathcal{Z}_y$. To facilitate learning, we require a means to assess whether a state in the search tree is on any of the correct paths through the tree. Specifically, the user must define a task-specific function ORACLE$(s, \mathbf{y})$ that returns the subset of permissible decisions that stay on any correct path. In terms of search trees ORACLE defines a sub-tree of the full search tree that encodes $\mathcal{Z}_y$. That is, there is a parallelism, where PERMISSIBLE can be used to draw up the full prediction space $\mathcal{Z}$, and ORACLE can be used to draw the constrained space $\mathcal{Z}_y$. In the former case, any well-formed output structure is possible, whereas the latter case only allows for paths that result in the given output $\mathbf{y}$.

Defining the function ORACLE may be of varying difficulty. As we have seen, the case of coreference resolution is next to trivial – return the subset of mentions on the left with which the next mention is coreferent. But for other tasks, e.g., those we pursue in Chapters 4 and 5, this function may also be more involved.

2.3.2 Greedy and Beam Search

The simplest way to navigate the search tree is to select the best option in every state, called *greedy search*. Pseudocode for this procedure is shown in Algorithm 2.1. Starting in the initial state, it considers all permissible decisions and selects the highest scoring one (line 4). Then, this decision is applied (indicated by the $\circ$ operator) to the current state and the resulting state replaces the current (line 5). Application of the decision creates a new state with a backpointer to the previous state and a score equal to the sum of the scores of the previous state and the chosen decision. This procedure is repeated until the current state is terminal, after which the terminal state is returned.

Algorithm 2.1 Greedy Search

Input: Input structure $\mathbf{x}$, feature extraction function ϕ, weight vector $\mathbf{w}$
Output: The terminal state s where each decision was the locally optimal one

```
1: function GreedySearch(x, φ, w)
2:     s = InitialState(x)
3:     while ¬Terminal(s) do                          ▷ While not done,
4:         d̂ = arg max_{d∈Permissible(s)} w · φ(x, s, d)   ▷ find best decision,
5:         s = d̂ ∘ s                                  ▷ and apply it
6:     return s                                       ▷ Return terminal state
```

While greedy search is very efficient, it might not find the terminal state with maximal score (also known as the *global optimum*), i.e., the path in the search tree where the *sum* of the scores of every decision is maximal. This situation is illustrated in Figure 2.3. The figure shows an example search tree where the arcs have been labeled with the scores of each decision and the states have been labeled with the score of that state. The path corresponding to greedy search is highlighted in blue. However, the final state along that path is not the global optimum. Instead, the optimal path, highlighted in orange, relies on taking a sub-optimal decision in the initial state and then continues to a terminal state that has a greater total score.

A naive solution to find the optimal path in the search tree would be to do exhaustive search, i.e., evaluating all possible paths through the search tree to the terminal states and then returning the optimal one. Although the search trees in the examples we have seen seem rather small, the size of the prediction spaces $\mathcal{Z}$ that we are dealing with is vastly larger thus precluding exhaustive search. Instead we have to find a compromise between exploring multiple paths simultaneously but doing so efficiently. In this dissertation, we rely on **beam search** as an approximate search strategy.

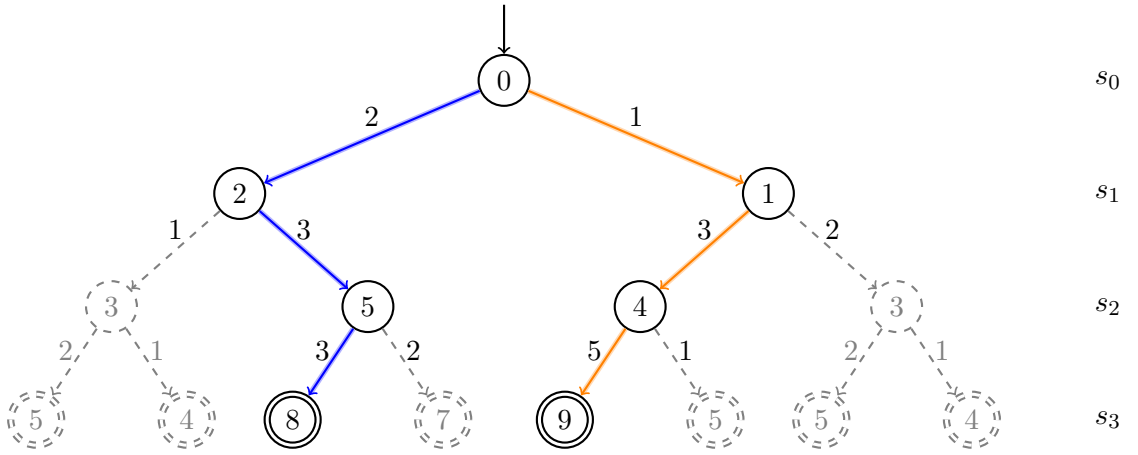


Figure 2.3: An example search tree where greedy search is unable to find the global optimum. The path chosen by greedy search is highlighted in blue. The optimal path is highlighted in orange. The states explored correspond to using beam search with a beam width of 2.

The intuition behind beam search is to keep a small set of candidate paths in memory (the *agenda*) and explore them simultaneously. The agenda iteratively gets expanded and the resulting states are pruned and put back into the agenda for a next round. Specifically, at each time step all the current items are expanded and their scores are computed. Then, the expansions are pruned and only the top k scoring expansions are retained and put back into the agenda. The parameter k, known as the *beam width*, controls how many paths are being explored simultaneously. In addition to greedy search, Figure 2.3 also shows how beam search would operate assuming a beam width of 2. The decisions that are grayed out would be pruned since their score is not among the top 2. In this example, the optimal path (orange) would be found by beam search.

The chief motivation for beam search is that it provides a reasonable trade-off between speed and exploration of sub-optimal intermediate states. In the extreme case when $k = 1$, beam search collapses into greedy search. Conversely, with $k = \infty$, it corresponds to breadth-first search, which would exhaustively search the full search space.

Pseudocode for beam search is shown in Algorithm 2.2. As input, the algorithm takes the input structure x (e.g., a sentence or a document), the beam parameter k, the local feature extraction function ϕ, and the weight vector used for scoring. The algorithm starts by seeding the agenda with the initial state. The main loop of the algorithm (lines 4-13) iterates over the states in the current agenda. For each state in the agenda, all permissible decisions are examined (lines 7-12) and new states are either put into a list of new states new for the next round (if the new state is not terminal), or in a separate list of

terminal states terminal (if the new state is terminal). Finally, the non-terminal new states are pruned to keep only the k best, and the loop is repeated. Once the agenda is empty, the best item from the list of terminal states is returned (line 14). We note that the pseudocode does not actually make use of the ϕ or $\mathbf{w}$, but point out that these are implicitly used when spawning a new state (line 8), as this operation is assumed to update the score of the new state.

Algorithm 2.2 Beam search

Input: Input structure $\mathbf{x}$, feature extraction function ϕ, weight vector $\mathbf{w}$, and beam size k

Output: The terminal state s from the beam with the highest total score of all explored states

```
1:  function BeamSearch(x, k, φ, w)
2:      agenda = {InitialState(x)}                  ▷ Seed agenda with inital state
3:      terminal = { }                              ▷ Output set for terminal states
4:      while NonEmpty(agenda) do
5:          new = { }
6:          for s ∈ agenda do                       ▷ For every item in agenda,
7:              for d ∈ Permissible(s) do           ▷ consider the perimissible decisions
8:                  s' = d ∘ s                      ▷ Create new state
9:                  if Terminal(s') then
10:                     terminal = terminal ∪ s'    ▷ Place new state in set of terminals,
11:                 else
12:                     new = new ∪ s'              ▷ or set of new
13:         agenda = ExtractKBest(new, k)           ▷ Keep k best in agenda
14:     return ExtractKBest(terminal, 1)            ▷ Return best terminal state.
```

In the coreference example we have seen all paths through the search tree have the same length (or equivalently, the depth of the search tree is defined by the input). This is not the situation in the parsing tasks we will consider in Chapters 4 and 5, where the length of the paths may vary. To accommodate this we extend Algorithm 2.2 to handle this. Specifically, we let the list of terminal states terminal grow until it is the size of the beam k. Then, when resetting the agenda (line 13 in Algorithm 2.2) we only add a new state if its score is *greater than the score of the worst scoring state* in the set of terminal states. To keep this computation efficient, we implement terminal as a binary heap with the smallest value in the root (i.e., a min-heap; see, e.g., Cormen et al. (2001, pp. 127-129)). This data structure allows for constant time lookup of the item with minimal score, enabling quick filtering of potential expansions that are not deemed good enough.

In terms of time complexity, beam search is often regarded as adding a multiplicative constant of the beam width k. That is, if the input can be processed by greedy search in $O(n)$ where n measures the size of the input, beam search requires $O(kn)$ time. While this is true, a naive implementation might fail to accommodate this for two reasons. First, when exploring the possible options (lines 6 to 13 in Algorithm 2.2), there is an implicit copy operation of the state taking place (line 8) since multiple states will be spawned from the state that is currently being explored. To maintain $O(kn)$ time complexity, it is critical to ensure that the copy operation can run in constant time (as opposed to being a function of the size of the input, e.g., $O(n)$). Although other constants might play a bigger role and time efficiency may not be the most critical aspect when running experiments in a scientific context, it could have serious repercussions when such a system is employed in a production environment. Goldberg et al. (2013) provide a thorough discussion describing how the state representation can be kept small enough to allow for constant time copy in the context of transition-based dependency parsers (cf. Chapters 4 and 5, where we also employ this technique; however the same underlying idea is also applicable to our approach to coreference resolution in Chapter 3).

The second issue with regard to complexity is the extraction of the k best candidates (line 13 in Algorithm 2.2). The simplest approach here would be to implement new as a list and then sort the list according to score and extract the top k items. Since sorting requires $O(m \log m)$ time for a list of m items, this would increase the complexity of the overall algorithm. This issue can be overcome by applying order statistic selection algorithms (Blum et al., 1973; Cormen et al., 2001, pp. 184-192) to find the k'th best element in linear time, and then making a second (linear time) pass over the expansions to extract the candidates that should be retained in the beam. Another option to keep the expand-prune step efficient is to also implement the list new as a binary min-heap. Although this has asymptotically worse complexity, it is very fast in practice since the k best expansions in the beam tend to cluster together and most other expansions can be quickly discarded by knowing the worst scoring item in the list new.

2.3.3 Deriving Latent Structure

Before concluding the discussion on search, we would like to draw the attention to how latent structure can be found. We have seen above that the search space $\mathcal{Z}$ can be explored using algorithms such as greedy and beam search. Both of these algorithms rely on the function PERMISSIBLE to explore the space of possibilities. Given an input-output pair $(\mathbf{x}, \mathbf{y})$, a latent structure $\mathbf{z} \in \mathcal{Z}_y$ that encodes the output $\mathbf{y}$ in the prediction space $\mathcal{Z}$ can

be found by means of the ORACLE function. Specifically, by replacing PERMISSIBLE with ORACLE in either Algorithms 2.1 or 2.2, the final state will be ensured to belong to the set of correct prediction structures. As is the case of greedy and beam search in general, there is no guarantee that this would result in the globally optimal $\mathbf{z} \in \mathcal{Z}_y$, although we note that the search space is considerably smaller. In the content chapters where we rely on latent structure (Chapters 3 and 4), we apply beam search with the same beam width as for prediction to find the latent structure to be learned.

2.4 Learning the Weight Vector

So far we have defined the task, feature functions, and search algorithms. The missing component at this point is how to obtain the weight vector $\mathbf{w}$. In this work we rely on the *perceptron* (Rosenblatt, 1958), or more precisely its closely related cousin the **Passive-Aggressive** (PA) algorithm (Crammer et al., 2006). While the perceptron and PA algorithms can be used to learn classifiers, they can be extended for *structured prediction*, commonly referred to as the *structured perceptron* (Collins, 2002; Crammer et al., 2006). The key ingredient here is that the global feature function Φ decomposes into the sum of a local feature extraction function ϕ applicable to an individual state-decision pair (cf. Section 2.1). In this section we will first consider the PA algorithm in the simpler setting where exact search is available. Then we will continue with the case where approximate search is used and discuss potential issues and solutions.

2.4.1 The Perceptron and the Passive-Aggressive Algorithm

The (structured) perceptron is an online algorithm that processes training data points $(\mathbf{x}, \mathbf{y})$ one by one. For each training instance, it makes a prediction. If the prediction is correct, the algorithm moves on to the next training instance. If the prediction is incorrect, the weight vector is updated to favor the correct solution. In our setting the situation is a bit more complex since prediction is carried out in $\mathcal{Z}$-space and not in $\mathcal{Y}$-space. As discussed above, a latent structure can be found by running the search algorithm over the constrained space $\mathcal{Z}_y$. For the time being we assume that the search problem in $\mathcal{Z}$-space can be solved exactly by a search procedure called EXACT. This is for instance the situation when an arc-factored model is used for coreference resolution, as was discussed in the introductory example in Chapter 1.

Pseudocode for the PA algorithm with latent structure is shown in Algorithm 2.3. It starts by initializing the weights to the $\mathbf{0}$-vector and then makes a number of passes (or

epochs) over the training data, considering one training instance at a time. It starts by deriving an instance from the latent space which we aim to learn by decoding in $\mathcal{Z}_y$-space (line 5).[2] It then makes a prediction in $\mathcal{Z}$-space (line 6). If the predicted structure $\hat{\mathbf{z}}_\mathbf{i}$ is identical to the latent one (i.e., $\tilde{\mathbf{z}}_\mathbf{i}$) then no action is taken, and the algorithm continues with the next training instance. If the prediction $\hat{\mathbf{z}}_\mathbf{i}$ is not the same as the latent structure $\tilde{\mathbf{z}}_\mathbf{i}$, an update is carried out (lines 7 to 10). The update is the only thing that differentiates the PA algorithm from the vanilla perceptron. The PA algorithm introduces a custom loss function $\textsc{Loss}(\hat{\mathbf{z}}, \tilde{\mathbf{z}})$ which measures the compatibility between a prediction and the correct structure with the intuition that a better (but still wrong) prediction should receive a lower loss than a worse prediction. The loss function must be defined by the user for the specific task at hand. For instance, when comparing a predicted coreference tree to a latent gold coreference tree, the loss function might count the number of erroneous arcs in the prediction compared to the gold.

Algorithm 2.3 Passive-Aggressive algorithm with latent structure

Input: Training data $D = (\mathbf{x}_i, \mathbf{y}_i)$, number of training epochs T, loss function $\textsc{Loss}$

Output: Weight vector $\mathbf{w}$

1: **function** $\textsc{PAPerceptron}(D, T, \textsc{Loss})$
2: $\quad\mathbf{w} = \overrightarrow{0}$ ▷ Initialize weights to the zero vector
3: $\quad$**for** $t \in 1..T$ **do** ▷ Loop T times (epochs)
4: $\quad\quad$**for** $(\mathbf{x}_i, \mathbf{y}_i) \in D$ **do** ▷ For each instance
5: $\quad\quad\quad\tilde{\mathbf{z}}_i = \textsc{Exact}(\mathbf{x}_i, \mathbf{w}, \phi, \mathcal{Z}_y)$ ▷ Decode latent
6: $\quad\quad\quad\hat{\mathbf{z}}_i = \textsc{Exact}(\mathbf{x}_i, \mathbf{w}, \phi, \mathcal{Z})$ ▷ Predicted structure
7: $\quad\quad\quad$**if** $\tilde{\mathbf{z}}_i \neq \hat{\mathbf{z}}_i$ **then** ▷ If wrong, update
8: $\quad\quad\quad\quad\Delta = \Phi(\hat{\mathbf{z}}_i) - \Phi(\tilde{\mathbf{z}}_i)$ ▷ Distance vector
9: $\quad\quad\quad\quad\tau = \frac{\Delta \cdot \mathbf{w} + \textsc{Loss}(\hat{\mathbf{z}}_i, \tilde{\mathbf{z}}_i)}{\|\Delta\|^2}$ ▷ Scaling factor
10: $\quad\quad\quad\quad\mathbf{w} = \mathbf{w} + \tau\Delta$ ▷ Passive-aggressive update
11: $\quad$**return** $\mathbf{w}$ ▷ Return learned weights

To explain the scaling factor τ we first need to consider the weight vector in a geometric sense. A common view on the weight vector is to regard it as a hyperplane, where training instances (data points in the vector space) are either on one side or the other of the hyperplane. The *margin* of training instance is the distance between that point in vector space and the hyperplane corresponding to the weight vector. *Margin-based* methods

[2] This should be contrasted with using a heuristic to derive the $\mathcal{Z}$-space structure. If a heuristic is used, no latent structure is required. Moreover, the target $\mathcal{Z}$-space structure can be computed once and for all before training begins. This is the situation in Chapter 4 when we compare with static oracles, and consistently throughout Chapter 5, where no latent structure is employed.

of learning weight vectors aim at not only finding a weight vector (hyperplane) that separates the training data, but one that does so with a certain minimum margin for training instances on either side of the hyperplane. The PA algorithm, which can be regarded as a margin-based extension of the perceptron algorithm, aims at maintaining a margin the size of the loss function between the current training instance and the hyperplane. The *aggressive* part in the name of the algorithm comes from the fact that every update is aggressive enough to ensure this margin, while at the same time moving the hyperplane as little as possible.[3] The computation of τ on line 9 ensures exactly this (Crammer et al., 2006). If τ is instead set to 1, the algorithm collapses into the regular structured perceptron and no margin constraints are enforced.

In practice, we deviate from the pseudo-code in Algorithm 2.3 in two ways: First, the training instances are shuffled between every epoch so as to simulate a "random" stream of training examples. Second, after training we return the average of all weight vectors seen during training, known as parameter averaging or *the averaged perceptron* (Collins, 2002). Parameter averaging has been shown to approximate the more expensive *voted perceptron* which involves multiple weight vectors used for prediction (Freund and Schapire, 1999; Collins, 2002). The intuition between the voted perceptron (and, consequently, also the averaged perceptron) is that, because of the online nature of the perceptron, it is very sensitive to the order of the instances in the training data. In particular, the very last (or most recent, during training) instance seen has an disproportionate bias on the final weight vector compared to instances seen much earlier. Therefore, letting the weight vectors vote, or just considering their average, softens this effect and makes the final weight vector less sensitive to the order of the training data. The motivation for using the averaged perceptron rather than the voted one is strictly based on efficiency – the voted perceptron relies on using *all weight vectors* seen during training and thus means the computations involved at test time would use a involve a considerable number of scalar products as opposed to a single one.[4]

Computing the average of all weight vectors may be an expensive operation since it involves a linear pass over the weight vector after every instance in order to aggregate a sum. However, Daumé III (2006, *pp. 9-10*) provides an elegant solution to this problem by showing that the average weight factor can be unfolded as a telescopic sum over the distance vectors involved in the updates. This way, only the weights that change

[3] The *passive* part of the name comes from the simple fact that when a prediction is correct no action is taken, i.e., the algorithm is passive.

[4] To make it perfectly clear, we emphasize that the average of the weight vectors is only computed after training has completed. That is, while training, the most recent weight vector is used consistently and the averaged weight vector is only computed after Algorithm 2.3 has finished.

during an update need to be processed and the average weight vector can be computed efficiently at the end of learning.

2.4.2 Learning with Approximate Search

Until now we have assumed that the search problem can be solved exactly. In the pseudocode for the perceptron (Algorithm 2.3) we used the place-holder function name EXACT to stand in place for $\arg\max_{\mathbf{z} \in \mathcal{Z}} F(\mathbf{x}, \mathbf{z})$ (cf. Equation 2.1 in Section 2.1). In practice this means that EXACT returns the highest scoring candidate $\mathbf{z} \in \mathcal{Z}$ and, by extension, that the perceptron will only apply an update if the correct solution $\tilde{\mathbf{z}}$ obtains a lower score than the best prediction $\hat{\mathbf{z}}$. As we will see (Figure 2.4, page 38), naively replacing EXACT with an approximate search algorithm breaks this guarantee. That is, it could be the case that an update is made when the correct solution $\tilde{\mathbf{z}}$ scores higher than the best prediction $\hat{\mathbf{z}}$ under the current weights. But because $\tilde{\mathbf{z}}$'s path on the search tree was pruned, it was never given a fair chance to compete with the final prediction $\hat{\mathbf{z}}$. Applying an update in this situation makes no sense, since the weight vector actually correctly scores the right solution highest; Huang et al. (2012) call this an *invalid update*. They go on to show that the convergence proof for the structured perceptron does not require exact search, but it does require valid updates. Any update where the prediction $\hat{\mathbf{z}}$ scores higher than the correct solution $\tilde{\mathbf{z}}$ is by definition valid. Critically, these updates do not necessarily have to involve complete sequences (i.e., a full path through the search tree), but they can also be based on prefixes of correct and predicted sequences.

Early Updates

To ensure valid updates when using beam search, Collins and Roark (2004) propose to halt search when the correct sequence is pruned from the beam. Then a perceptron update between the prefixes of the correct sequence and the highest scoring prediction is applied. After the update, the algorithm proceeds with the next training instance. This procedure has subsequently become known as **early update**. Early updates ensure valid updates since at that point it is clear that the correct solution scores worse than the best prediction (otherwise it would not have been pruned).

Applying early updates requires a tighter coupling between the search algorithm and the perceptron. Algorithm 2.4 shows pseudocode of beam search coupled with the perceptron. The algorithm combines the ideas from the PA algorithm and beam search and some of the steps in each of those have been condensed into function calls in the pseudocode. It makes a number of epochs over the training data and applies beam search for

Algorithm 2.4 Passive-Aggressive algorithm with beam search and latent structure.

Input: Training data $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$, number of epochs T, loss function LOSS

Output: Weight vector $\mathbf{w}$

```
1:  function PABEAMEARLY(D,T,LOSS)
2:      w = 0⃗
3:      for t ∈ 1..T do
4:          for (x_i, y_i) ∈ D do
5:              z̃ = FINDLATENT(x_i, w, φ, k)              ▷ Derive latent structure
6:              agenda = {INITIALSTATE(x)}                 ▷ Inital state
7:              terminal = { }                             ▷ Terminal (finished) states
8:              t = 0
9:              while NONEMPTY(agenda) do
10:                 t = t + 1                              ▷ Increment step counter
11:                 agenda = EXPANDANDPRUNEBEAM(agenda)    ▷ Expand beam
12:                 if t < |z̃| ∧ z̃_[1:t] ∉ agenda then
13:                     ẑ_[1:t] = EXTRACTKBEST(agenda, 1)
14:                     PAUPDATE(z̃_[1:t], ẑ_[1:t])         ▷ Early update
15:                     next                               ▷ Continue with next instance
16:                 else if z̃ ∉ terminal then
17:                     ẑ_[1:t] = EXTRACTKBEST(agenda, 1)
18:                     PAUPDATE(z̃, ẑ_[1:t])               ▷ Early update
19:                     next                               ▷ Continue with next instance
20:             ẑ = EXTRACTKBEST(terminal, 1)
21:             if z̃ ≠ ẑ then
22:                 PAUPDATE(z̃, ẑ)                         ▷ Full update
23:     return w
```

every instance. The very first step for each instance is to decode the latent gold structure (line 5), denoted by the function FINDLATENT. This is accomplished by applying beam search itself in the constrained space $\mathcal{Z}_{\mathbf{y}}$, using the same beam size k. This sequence will then act as the correct sequence for this instance during this epoch. Beam search then commences to explore the full space $\mathcal{Z}$ by seeding the agenda with the initial state and then continuously expanding and pruning the agenda. The call EXPANDANDPRUNEBEAM on line 11 corresponds to the steps in regular beam search (cf. lines 6 to 13 in Algorithm 2.2) where the permissible options of the current agenda are all explored and the top k are retained in the beam. While doing so, a step counter t is incremented for every expansion. After each expansion, a check is carried out whether the correct item is still in the beam or among the set of terminal expansions (lines 12 or 16). The two if blocks handle the situation when the length of the sequences is not guaranteed to be fixed (cf.

the discussion on beam search from the previous section). That is, when the step counter has advanced longer than the length of the correct sequence, the check boils down to whether the correct solution is in the set of terminals or not. If either of these situations occur, an *early* passive-aggressive update (PAUPDATE; cf. lines 8-10 in Algorithm 2.3) is carried out between the (prefix of) the correct sequence and the best item in the agenda. We here use the notation $\mathbf{z}_{[1:t]}$ to denote the prefix of the first t state-decision pairs from the sequence $\mathbf{z}$. After an early update the remainder of this instance is skipped and the algorithm proceeds with the next instance (lines 15 and 19). If the correct solution survives in the beam and ends up in the set of terminals after beam search has completed, a final check is made whether the correct solution obtained the maximum score (line 21). If it did not, a regular passive-aggressive update is carried out (line 22). We note in passing that the pseudocode does not make explicit use of the loss function LOSS, but that this function is implicitly used by the call to PAUPDATE.

Max-violation and Latest Updates

A major drawback of early updates is that it discards the remainder of the current instance when an update is made. This tends to require many training epochs (Huang et al., 2012) and can, as we will see in Chapters 3 and 5, in the worst case be detrimental to learning to the point where early updates underperform compared to simpler baselines. The central theme, and the requirement for convergence, in Huang et al.'s (2012) analysis of the structured perceptron is *violation*, i.e., the difference in scores between the correct solution and the best current prediction. With this in mind they propose two alternative updates: *max-violation* and *latest* updates. Compared to early update the idea is to continue beam search even after the correct item is pruned and then track the difference in scores between the correct solution and the best current prediction. A max-violation update corresponds to selecting the pair of prefixes where the difference in score (i.e., the violation) is maximal. A latest update keeps going further and would choose the longest prefixes which is still a valid update.

An alternative graphical depiction of these updates is shown later in this section when we summarize this section (Figure 2.5, page 41), but Figure 2.4 contains an example search tree displaying the points of early, max-violation, and latest updates. The numbers along the arcs denote the hypothetical score of making that decision, whereas the numbers in the nodes denote the score of the corresponding states (i.e., the sum of the scores along the arcs to that node). A beam width of 2 is assumed and states in the beam are drawn with solid borders. The correct sequence is highlighted in green. Dashed states

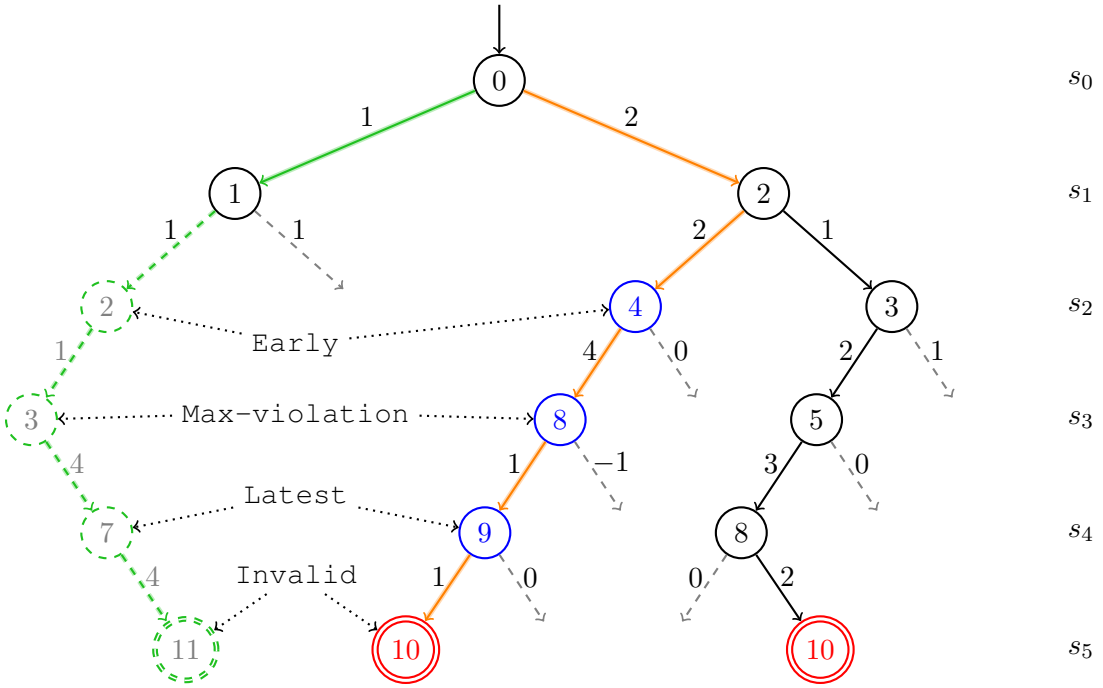


Figure 2.4: An example search tree illustrating early, max-violation, and latest update. The correct path is indicated in green. A beam size of 2 is assumed, and states in the beam are indicated by solid borders. Blue and red states correspond to valid and invalid updates with respect to the green (correct) states, respectively. The potential states involved in each update pair are indicated by dotted arrows.

and expansions are not included in the beam as they score too low and get pruned. The figure shows how the correct path is pruned during the second expansion. This means that at state s_2 an early update can be applied, with a violation of $4 - 2 = 2$. In the next step, s_3, the violation between correct and best prediction increases to $8 - 3 = 5$ and this is also the place for the maximum violation. The last chance to make a valid update is at s_4, where the violation is only $9 - 7 = 2$. At the level of terminal states s_5, there is no valid update since the violation $10 - 11 = -1$ is now negative.

LaSO

One motivation for max-violation and latest updates is that they make better use of the training data than early updates. Nevertheless, they are all subject to partial updates, where parts of the current training instance is discarded. Daumé III and Marcu (2005) present *Learning as Search Optimization* (LaSO), which situates perceptron learning specifically in the context of search trees that are explored with approximate search. In essence,

they propose a different strategy where multiple updates can be applied for a single instance, ensuring that no training data is discarded. Specifically, they propose to carry out beam search (or any other approximate search algorithm) to the point of early update, carry out the update, and then *continue* decoding the current instance from the gold state that was used for the update. This is accomplished by re-seeding the agenda with a partially decoded state, i.e., inserting the gold state into an empty agenda and resuming beam search. This procedure is repeated until the full instance is decoded and the correct sequence has been found. In terms of pseudocode, this essentially (but not entirely; see next paragraph) boils down to removing the **next** statements in Algorithm 2.4.

The LaSO updates are attractive since they make full use of the training data. However, they do not properly comply with our view on latent structure. Recall that the latent gold structure is computed every time processing of a new instance begins (line 5 in Algorithm 2.4). Since partial updates are made to the weights while decoding an instance, the score of potential latent structures may change. Specifically, the score of the best correct path in the search tree (e.g., the green path in Figure 2.4) may change. To make matters worse, such an update could even imply that the optimal $\tilde{\mathbf{z}} \in \mathcal{Z}_y$ changes. Simply continuing decoding with the initial goal may thus lead to invalid updates. When we apply LaSO updates we therefore recompute the latent gold structure and use the prefix of length t_u from this gold sequence to re-seed the beam, given that the update occurred at time point t_u.

DLaSO

Due to the online nature of the perceptron, it is quite sensitive to the most recent update. Since LaSO updates happen within instances, this gives an unusual feedback during learning, which may make decoding the remainder of the instance artificially easier. As we will see in Chapter 3, LaSO yields fewer mistakes during training and thus also fewer updates. While this feedback may make training easier (i.e., more correct predictions), such feedback is not available at test time, and the LaSO learning setting therefore does not properly mimic the setting at test time. We therefore proposed *Delayed LaSO* (DLaSO) updates (Björkelund and Kuhn, 2014). DLaSO adopts the idea that multiple updates can be carried out for a single instance, however the updates are delayed until the full instance has been processed. While LaSO applies an update as soon as the correct sequence is pruned from the beam (i.e., an early update), DLaSO stores the update, reseeds the beam with the correct item at this point and continues until the next early update. Once the full instance has been processed, all updates are applied in one go. Storing the

update hinges on the fact that a PA update essentially relies on two components: (1) the distance vector Δ, i.e., the feature vectors of the correct and predicted sequences subtracted from each other, and (2) the loss between the correct and predicted sequences (cf. Algorithm 2.3). To carry out an update at the end of the instance we sum the distance vectors as well as the loss of all updates and then apply a single PA update.

The description of LaSO implies that multiple pairs of loss and distance vectors will overlap in their prefixes with respect to the sequence. Each pair covers the sequence from the start to the current point, hence the summed loss and feature vectors will overlap over prefixes of the sequences within a set of DLaSO updates. However, since the beam is reseeded with the correct prefix after each early stop, the prefixes (and thus also their loss and distance vectors) of the restarts will cancel out between the start and the point of restart (where it was reseeded). As a consequence, for every update pair of distance vector and loss within a full DLaSO update a specific point in the transition sequence will only be accounted for once in one of these individual update pairs.

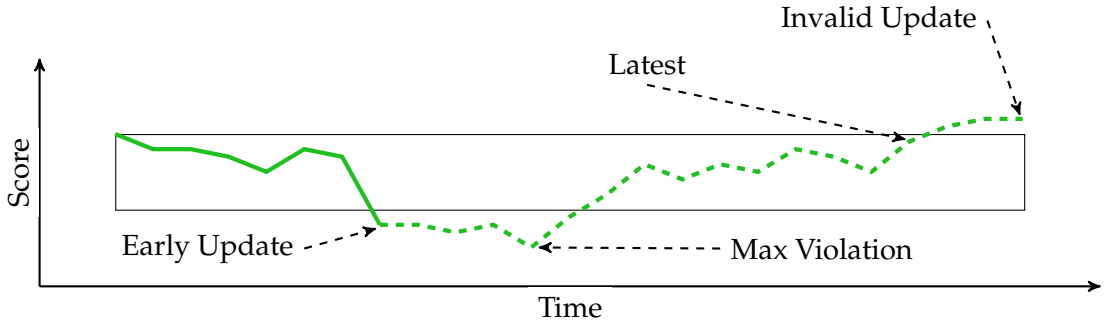
To summarize and relate to LaSO, DLaSO follows LaSO in that the complete training instance is always used during every epoch. But they differ in how exactly updates are carried out – LaSO applies updates as soon as possible, whereas DLaSO collects them and applies them at the end of an instance. We argue that the DLaSO update method better mimics the test-time scenario since it does not return feedback within instances during learning. In Chapter 3 we will consider this point in greater detail.

Summary of Updates

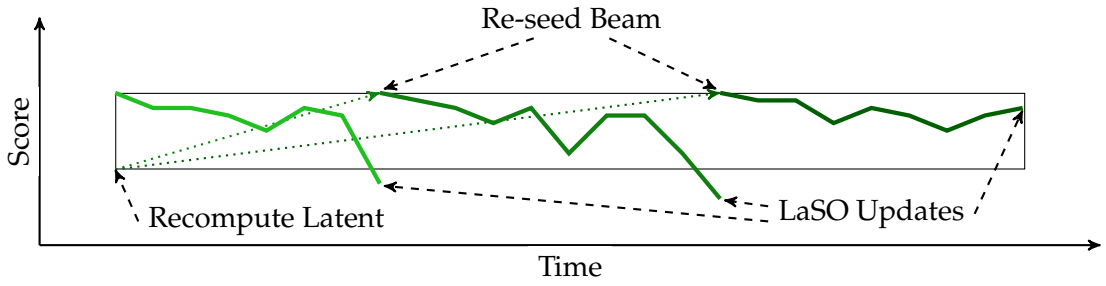
We have now seen five different update strategies for the structured perceptron with approximate search: early updates, max-violation and latest updates, LaSO and DLaSO updates. A schematic view of how they work with respect to beam search is shown in Figure 2.5. All three figures illustrate the beam as a rectangle where higher score means higher up along the vertical axis. The horizontal axis denotes the incremental steps taken for each expansion of the beam. The correct item in the beam is highlighted in green.

Figure 2.5a shows early, max-violation, and latest updates. About one third into decoding the instance the correct sequence is pruned and is now outside the beam. At this point an early update is applicable, and the remainder of the instance would then be discarded. But the correct sequence, now dashed to indicate that it is no longer in the beam, remains to have a lower score than the best items in the beam. At a later point, the difference in scores between the best item in the beam and the correct sequence is the greatest, at which point a max-violation update is applicable. From that point on the

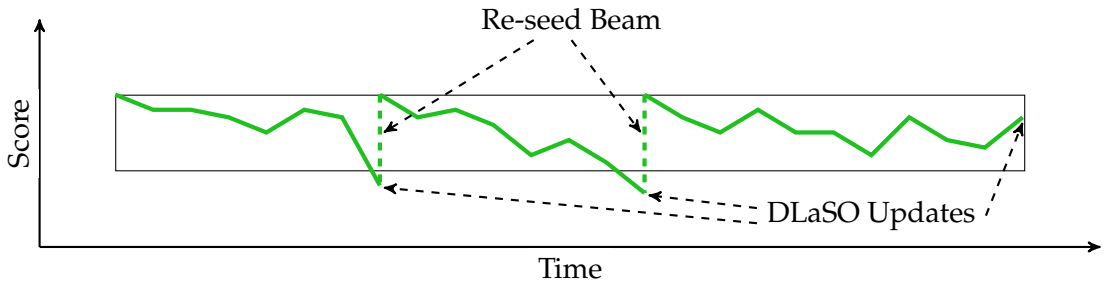
relative difference in scores between the correct sequence and the best item in the beam is shrinking. Shortly before the end, the chance to make the latest update occurs. After that point the correct item scores higher than the best item in the beam and an update at here would be invalid.[5]



(a) Early, Max-violation, Latest.



(b) LaSO.



(c) DLaSO.

Figure 2.5: Graphical summary of the behavior of the different updates.

In Figure 2.5b the LaSO update is shown. It starts off the same way as in the previous figure, but it applies an early update when the correct sequence is pruned from the beam.

[5]While the illustration in Figure 2.5a only shows invalid updates towards the end, this does not have to be the case. Invalid updates can be present anywhere along the time axis. The example here was chosen deliberately to keep things simple.

It then recomputes the latent gold, reseeds the beam at this point and continues. The same thing happens once again, and at the end of decoding the correct sequence is not the top scoring one, resulting in a third update. Note that in contrast to the previous figure, we may be dealing with three different latent gold structures here (as indicated by the different shades of green).

Finally, Figure 2.5c shows a similar figure but for DLaSO updates. Again, the initial steps of beam search are identical until the point of an early update. But here, instead of applying the update immediately, the corresponding vector and loss are saved for a later update. The beam, however, is reset to hold only the correct gold item at this point and beam search is resumed, implying that the correct sequence is now the best item in the beam. This means that, in contrast to the setting for LaSO, it is the *same* gold structure that survives for the entire instance.

2.5 Related Work

In the previous sections we have covered the machinery we will be using in this dissertation. We now take a step back and discuss it in the broader context of online learning, structured prediction, and search with respect to NLP. A comprehensive survey of all these aspects is clearly outside our scope, but we intend to cover the most relevant related work below.

2.5.1 Online Learning and Structured Prediction

Structured problems are omnipresent in NLP, including tasks such as part-of-speech tagging (i.e., sequence labeling), syntactic parsing, or coreference resolution. For problems that can be approached as a sequence of decisions, such as the ones we consider in this dissertation, a straightforward simplification is to apply a (discriminatively trained) classifier iteratively after each step in the sequence. These models are sometimes called *locally normalized* since each decision is made in isolation. While locally normalized models may perform reasonable, they are sensitive to mistakes in the sequence. When a mistake is made, this creates a situation that may be rather dissimilar to how the classifier was trained. In the worst case this can throw off the classifier such that it makes more mistakes in subsequent steps, an effect known as *error propagation*. A classifier also has poor options to exploit structural features in the output. For instance, when deciding on the antecedent for one mention during a coreference assignment, the question is how much of the additional structure is available for feature extraction. In the simplest case, a clas-

sifier can take no structural context into consideration. The other option is to provide the classifier with gold standard structural context (e.g., the antecedents of some or all preceding mentions). While the former case clearly lacks contextual information, the latter suffers from the problem of unrealistically good contextual information at training time since the contextual features have to be drawn from the correct output.

A more principled approach is to model the entire structured output, leading to structured predictors. Most structured prediction techniques take ideas from classification algorithms and generalize them to larger output spaces. To handle the increase in possible outputs and interactions, certain simplifying assumptions have to be made. One option is to decompose the problem in some problem-specific manner as in the case of conditional random fields (Lafferty et al., 2001) or Maximum Margin Markov networks (M^3; Taskar et al., 2004) where marginal probabilities can be efficiently computed despite the large output space. Another option is to let learning be entirely driven by predictions and consider only one or a few erroneous structures for updates as in SVM_{struct} (Tsochantaridis et al., 2005), or as we have seen in this chapter, the structured perceptron (Collins, 2002). As these models are *globally normalized*, they account for the dependence within the output and thus alleviate the error propagation problem.

The perceptron was first proposed by Rosenblatt (1958) as a binary classifier. Block (1962) and Novikoff (1963) present the first convergence proofs that assert that, assuming the training data is linearly separable the weight vector will converge to one that obtains no errors on the training data after a finite amount of steps. Freund and Schapire (1999) provide a more modern treatment, extend the algorithm to multi-class classification and provide convergence bounds. The bounds and proof technique has subsequently been carried over to the structured perceptron (Collins, 2002) and the structured perceptron with approximate search (Huang et al., 2012). Crammer et al. (2006) introduced the closely related margin-based PA algorithm and also provide convergence proofs for binary, multi-class, and structured settings. The structured perceptron and its relatives have become a very common technique for a variety of NLP problems both when exact search is possible, as in part-of-speech tagging (Collins, 2002; Huang et al., 2012) and named entity recognition (Ratinov and Roth, 2009), or in combination with beam search and early updates on tasks such as Chinese word segmentation (Zhang and Clark, 2007), natural language generation (Bohnet et al., 2011, 2012), event extraction (Li et al., 2013), syntactic parsing (Huang, 2008; Zhang and Clark, 2008), and rule selection for machine translation (Yu et al., 2013), to name a few.

The (structured) perceptron should also be regarded with respect to batch-learning algorithms. Batch-learning algorithms typically involve a convex objective function that

is optimized with gradient descent or similar methods. These methods tend to be slow relative to the perceptron since they depend on the expensive operation of computing the gradient over the full training data after each update. With sufficiently large and redundant data sets, as is the case for many NLP applications, including the ones we consider, it can be argued that the extra overhead of computing these gradients does not provide sufficient benefit compared to the additional computational effort. Indeed, this has in part also motivated the proliferation of stochastic gradient descent (Bottou, 2004; Tsuruoka et al., 2009) and related methods such as AdaGrad (Duchi et al., 2011) in the NLP literature. In the most extreme case, the perceptron can be viewed as stochastic gradient descent with a batch-size of 1 (Zhang et al., 2014a), allowing for a properly regularized objective function.

2.5.2 Search in NLP

Due to the structured nature of natural language and natural language processing tasks, search problems are frequent in the NLP literature. General introductions to search can be found in most AI text books (Russell and Norvig, 2003). Edelkamp and Schroedl (2010) provide a thorough survey of heuristic search algorithms.

Beam search was first introduced in the HARPY speech recognition system (Lowerre, 1976), although the term was first used by Reddy (1977). Unfortunately beam search is not optimal, i.e., it does not necessarily find the highest scoring path in the search tree. It has nevertheless become an attractive method since it is efficient and easy to implement. The combination of beam search with the structured perceptron has already been mentioned above, but we note in passing that beam search has also been used with classifiers or other non-structured models for tasks such as coreference resolution (Luo et al., 2004), dependency parsing (Johansson and Nugues, 2006, 2007b), and statistical machine translation (Koehn et al., 2003; Tillmann and Ney, 2003).

A discussion on search cannot be concluded without mentioning heuristic search methods, such as (greedy) best-first search and A* search (Hart et al., 1968). These search methods rely on a heuristic function that, given a node in the search tree, provide an estimate on the *score of the remaining path to a terminal state*.[6] Such a heuristic function is said to be admissible if it never underestimates the score of the remaining path. Both best-first and A* keep an agenda of next items to explore but, unlike beam search, they

[6]While we aim to maximize the score of a path through the search tree, A* and other search algorithms are often presented in the AI literature using *costs* of paths and the objective is to minimize the cost. Flipping the signs and reversing minimization/maximization makes these situations equivalent, but we choose to ground the discussion in how we have presented the problem earlier in this chapter.

are not synchronized in terms of the number of steps taken. The agenda is typically implemented as a priority queue and the most promising item in this queue is explored next. Best-first search prioritizes the next item that has the maximal score according to the heuristic function, whereas A* prioritizes the next item that has a maximal sum of the score to the current node and the score of the remaining path according to the heuristic function. It has been shown that if the heuristic function is admissible A* is optimal, whereas best-first search is not.

As A* is optimal it is a very attractive search algorithm. Unfortunately, designing an admissible heuristic function is challenging and depends on exploiting intricacies of the problem at hand. The most prominent example of A* in the NLP literature is probably its applications to probabilistic context free grammar (PCFG) parsing (Klein and Manning, 2002, 2003). They propose multiple heuristic functions, but these functions all depend on the fact that the probabilities can be estimated based on the grammar rules. For instance, Klein and Manning (2003) pre-compute the potential costs of composite production rules offline using the grammar table in a PCFG in order to obtain estimates on productions at parse time. This fact can then be used to create strong heuristic functions for PCFG parsing. This idea has subsequently been taken further to create k-best parse lists using A* (Pauls and Klein, 2009; Pauls et al., 2010). In the same vein, A* parsing has also been applied to CCG parsing (Auli and Lopez, 2011; Neubig et al., 2012) as well as machine translation using synchronous grammars (Zhang and Gildea, 2006, 2008). However, both with respect to the specific tasks we consider as well as for structured perceptrons in general (where scores can, in principle, attain arbitrary values between positive and negative infinity), we are not aware of heuristic functions or applications of A* and similar algorithms from the literature.

2.5.3 Latent Structure

The use of latent structure is becoming increasingly common. Although the term is similar, it differs conceptually from latent variables in generative frameworks, as they typically do not involve any structure. Though it should also be mentioned that the perceptron has been adapted to incorporate latent variables in the generative sense (Sun et al., 2009). In the coreference community latent structures have become a standard approach in the manner discussed in Chapter 1 already (Yu and Joachims, 2009; Fernandes et al., 2012; Chang et al., 2013; Durrett and Klein, 2013, *inter alia*). Latent structure also pops up in other problems such as semantic parsing, both coupled with the perceptron (Zhao and Huang, 2015) as well as in generative frameworks (Richardson and Kuhn, 2016).

The idea of latent structure is also closely related to other ideas building on the perceptron such as *guided learning* (Shen et al., 2007; Shen and Joshi, 2008; Goldberg and Elhadad, 2010) and *training with exploration* (Goldberg and Nivre, 2012, 2013). In guided learning, a classifier is used to select from a large set of decisions (e.g., introducing an arc in a dependency tree). Among the many options, several ones may be correct. As long as the classifier selects correct decisions during training, no updates are made, but when the classifier makes a mistake, the current weights are used to decide on a latent gold decision to use for update. Training with exploration is more or less the same idea, with the addition that sometimes mistakes are accepted in order to explore erroneous parts of the search space. While these techniques primarily (at least with this terminology, see next paragraph) have been used in the parsing community, guided learning has also been applied to the task of coreference resolution (Stoyanov and Eisner, 2012).

Training with exploration can in turn be regarded as an instance of imitation learning. Here, the idea is that a search space is explored as learning progresses but an *expert policy* can be queried for the right action at any point. Typically the search space is explored in a stochastic manner, where incorrect actions sometimes are taken. The expert policy then provides the best given action at any point. A classifier is initially trained to mimic the expert policy, but gradually the classifier is left to traverse the search space on its own (as in training with exploration) and both updates and instance selection are driven by gradually moving from the expert policy to the learned policy guided by the reward function. Two well-known approaches to imitation learning include frameworks such as SEARN (Daumé III et al., 2009) and DAGGER (Ross et al., 2011). A similar approach is that of reinforcement learning (Sutton and Barto, 1998; Maes et al., 2009). The main difference between imitation and reinforcement learning is the requirements on the expert policy. In imitation learning, the assumption is that the expert knows the search space well enough to guide the search in the best possible direction. In reinforcement learning this assumption is weakened, and the expert is replaced with a reward function that gives a measure on the goodness of a given state. The reward function is again used to guide the exploration of the search space, however with the caveat that it might not properly approximate an expert in a given state.

2.6 Conclusion

The previous sections have outlined the building blocks that are necessary to train the structured perceptron with approximate search. This machinery constitutes the main ve-

hicle for learning that we will rely on in subsequent chapters. In particular, we have described five different types of update strategies for the structured perceptron with approximate search: early, max-violation, and latest updates, as well as LaSO and DLaSO. In addition to the update strategy, instantiation of the framework relies on the definition of a number of task-specific functions and data structures. First, the user needs to formulate the search problem and the relationship between output space $\mathcal{Y}$ and prediction space $\mathcal{Z}$-space. This involves defining a sequential search problem, where a well-formed output structure $\mathbf{y} \in \mathcal{Y}$ can be created by a sequence of steps $\mathbf{z} \in \mathcal{Z}$. Additionally, the user needs to provide the following functions and data structures:

- The mapping $\pi : \mathcal{Z} \mapsto \mathcal{Y}$ that maps a prediction structure $\mathbf{z}$ to output space $\mathcal{Y}$.
- A feature extraction function $\phi : \mathcal{X} \times (S \times D) \mapsto \mathbb{R}^m$ which extracts appropriate features in order to learn and predict the correct decisions. It can exploit all available partial structure in order to create a high-dimensional representation of the current state.
- The function PERMISSIBLE which returns the set of all permissible decisions in a given state.
- The function ORACLE which returns the set of correct decisions in a given state.
- The function LOSS which computes a structured loss between a correct sequence and a prediction. It may be arbitrarily complex, but must be applicable also on a partial sequence $\mathbf{z}_{1:i}$, with or without the aid of the mapping π.
- A state representation that includes the basic requirements stated earlier, i.e., backpointers to previous state, decision taken in previous state, and score of the state. The state representation then needs to be extended in a task-specific manner to accommodate the other user-defined functions including the feature extraction function ϕ as well as PERMISSIBLE and ORACLE.

In addition, two hyperparameters are also left open to the user. They include the number of training epochs for the perceptron and the beam width used for beam search.

In the remainder of this dissertation we will carry out a careful evaluation on learning in this framework using real NLP problems. We will define the necessary functions and vary the set of hyperparameters and update methods: Chapter 3 will carry out a comprehensive analysis of the different update methods for coreference resolution with latent structure. In Chapter 4 we will introduce latent structure into transition-based

dependency parsing, while keeping the update strategy fixed. Finally, in Chapter 5 we will drop the latent structure and instead focus on the update methods when sequences become very large.

Chapter 3

Coreference Resolution

3.1 Introduction

In recent years much work on coreference resolution has been devoted to increasing the expressivity of the classical *mention-pair model*, in which each coreference classification decision is limited to information about two mentions that make up a pair. This shortcoming has been addressed by *entity-mention models*, which relate a candidate mention to a (potentially partial) cluster of mentions predicted to be coreferent so far (Ng, 2010). As discussed in the introduction already, these two approaches sit on opposing sides with regard to inference: In the mention-pair model exact search can be carried out to yield the optimal output structure, however at the expense of richer features that could consider a greater structural context. The entity-mention model, on the other hand, extends the scope of features to consider multiple mentions at the same time, although this requires abandoning exact search.

Nevertheless, models that are based on pairs of mentions have empirically proven very competitive. This has, at least in part, been driven by the introduction of latent structure to coreference. Specifically, for a given mention in a document, which mention, or *antecedent*, on its left is the best match in terms of learning and generalizability.[1] The use of latent antecedents, i.e., that the mention on the left is derived dynamically by the machine-learning model was part of training, has received considerable attention since it was used by the winning system (Fernandes et al., 2012) on the CoNLL 2012 Shared Task (Pradhan et al., 2012) on multilingual coreference resolution.

The system of Fernandes et al. (2012) basically implemented the model that was dis-

[1] We note in passing that, following standard practice in the coreference resolution literature, we overload the term antecedent to mean any mention to the left with which a mention is coreferent.

cussed in the Chapter 1: Model the mentions of a document as a tree, let the antecedents of each mention be latent, and try to predict a tree as a structure (as opposed to learning a classifier that classifies individual pairs of mentions). There was, however, one major caveat to their implementation – they relied on graph algorithms to do inference over the possible trees. Specifically, they used the Chu-Liu-Edmonds (CLE) algorithm (Chu and Liu, 1965; Edmonds, 1967) to find the maximum scoring directed tree assuming scores over all potential arcs are provided. The very nature of this inference algorithm requires an arc-factored model that is difficult to extend to richer features. However, under the constrained setting discussed in Chapter 1, where the tree is only right-branching, the inference problem can equally well be solved using a left-to-right greedy pass (also discussed in Chapter 1). While these two approaches have the same asymptotic time complexity ($O(n^2)$, in the number of mentions, assuming Tarjan's (1977) improved version of CLE), an inference procedure that makes a left-to-right pass is much more suitable to be extended with beam search.

In this chapter, we follow Fernandes et al. (2012) and develop a coreference model building on latent antecedents, or rather, latent trees, but replace the inference algorithm with such a left-to-right pass. Our starting point will be an arc-factored model in which the search problem can be solved exactly. We will then extend this model to include features that span more than two mentions. Since exact inference now becomes intractable, we will apply the framework from the previous chapter using beam search during the left-to-right pass and couple this with the update methods we have discussed. From the perspective of Chapter 2, this will enable us to carry out careful analysis of the update methods. We will compare the situation where exact search but weak features are available (i.e., an arc-factored model) to the approximate search with rich features. The evaluation shows that update methods that do not commit to using all available training data – early update, max-violation, and latest – underperform compared to LaSO and DLaSO, sometimes so severely that the simpler arc-factored model cannot be beaten. With regard to the area of coreference resolution research, the tree formulation can be considered an entity-mention model, where single mentions are compared to (partially) completed entities. However, unlike most entity-mention models, the proposed one includes mention-pair models as a special case and moving between the two settings can be done seamlessly, without requiring a change in choice of machine-learning or features. In that regard, we demonstrate that our entity-mention model, when trained using appropriate update methods, outperforms our strong mention-pair baseline.[2]

[2]The numbers from the experimental section of this chapter differ slightly from those presented in the paper on this topic (Björkelund and Kuhn, 2014), as we modified the system to properly implement the

3.2 Tree-based Coreference Resolution

Coreference resolution should be regarded in the greater context of several discourse-related NLP tasks that have sometimes been studied in isolation, and sometimes conflated under different names (e.g., pronoun resolution, anaphora resolution, anaphoricity detection and, in a broader scope, prediction of information structure). As usual in NLP, this has been, at least to a certain extent, driven by what annotated data sets are available. One critical detail in the task definition of coreference resolution that varies in the literature is the treatment of *singletons*, i.e., clusters that contain only a single mention. In this section we will define the task in a way suitable to the data sets and experimental evaluation with which we are concerned and, consequently, singleton clusters will not be allowed. In the related work (Section 3.5), we will return with a discussion on the treatment of singletons in other data sets as well as the family of tasks related to coreference resolution.

One not entirely insignificant detail that we have not yet addressed is where the mentions come from. The system we have implemented for the experimental evaluation can in fact be regarded as a pipeline (see Figure 3.2 in Section 3.2.2), where a high-recall mention extractor is used to extract mentions from input documents according to a set of rules operating over certain layers of linguistic annotation we presuppose as input (including, e.g., constituency trees for the input sentences). The mention extractor in turn is preceded by a set of rules that identifies the head words of each constituent. While these components could be discussed at length in their own right, they are kept constant throughout all our experiments and thus have little bearing on the main topics we are concerned with in this dissertation, i.e., learning structured perceptrons with approximate search. Nevertheless, in order to suitably document the full experimental pipeline we will provide a brief summary on how these components work and will describe them in turn after the task definition below.

3.2.1 Task Definition

The particular flavor of coreference resolution we are tackling is sometimes called noun phrase coreference resolution, which, despite what the name suggests, typically also encompasses pronouns, including possessive pronouns. At a high-level, we assume an

framework described in the previous chapter. The version used for the original conference paper differed to a small extent in the choice of latent trees used for updates and restarts with LaSO and DLaSO. This does not affect the overall conclusions, however, but the modifications puts all three instantiations of the framework used in this dissertation on the same foundation, as defined in Chapter 2.

[Walter Sisulu]$_{a_1}$ and [the African National Congress]$_{b_1}$ came [home]$_{c_1}$ yesterday. After 26 years in prison, [Mr. Sisulu, the 77-year-old former secretary-general of the liberation movement,]$_{a_2}$ was dropped off at [[his]$_{a_3}$ house]$_{c_2}$ by a prison services' van just as the sun was coming up. At the same time, [six [ANC]$_{b_2}$ colleagues, five of whom were arrested with [him]$_{a_4}$ in 1963 and sentenced to life imprisonment,]$_{d_1}$ were reunited with [their]$_{d_2}$ families at various places around [the country]$_{e_1}$. And as [the graying men]$_{f_1}$ returned to [their]$_{f_2}$ homes, [the ANC, outlawed in [South Africa]$_{e_2}$ since 1960 and still considered to be the chief public enemy by [the white government]$_{e_3}$,]$_{b_3}$ defiantly returned to the streets of the [country]$_{e_4}$'s black townships.

Figure 3.1: An example of a partial document with mentions labeled for coreference taken from the OntoNotes corpus.

input $\mathbf{x} \in \mathcal{X}$ to be document and the output structure $\mathbf{y} \in \mathcal{Y}$ to be a partition over mentions, that is, a collection of sets of mentions. We will shortly turn to a more formal definition, but pause to display a high-level example in Figure 3.1. The figure follows the same pattern as shown in the introduction, where a text (the input $\mathbf{x}$) has been labeled for mentions that are coreferent. The coreferent mentions (i.e., the output $\mathbf{y}$) are indicated by the subscripts on the bracketed mentions. The example illustrates one aspect that was not evident in the example from Chapter 1, namely that mentions may be nested, e.g. as in *[[his] house]* on the third line in Figure 3.1.

Coreference resolution is typically carried out after a number of preceding steps in an NLP pipeline and we will consequently assume quite rich annotations accessible in the input. Specifically, an input $\mathbf{x}$ is assumed to be an input document that has been split into sentences, tokenized, part-of-speech tagged, parsed with a constituency parser, and tagged for named entities. In early work on coreference resolution gold standard mentions were often included as part of the input, however we do not make this assumption.

To formalize the input structure we first define a few basic concepts. A **token** t is a tuple $\langle i, f_1, ..., f_k \rangle$, where i denotes a positional index, and the $f_1, ..., f_k$ denote token-level atomic features. A **sentence** S is a set of tokens $\{t_1, t_2, ..., t_n\}$, where the subscripts correspond to the positional indices associated with the tokens. The atomic features of tokens include the surface forms of tokens as well as token-level annotations such as part-of-speech tags or morphological information. Finally, a **span** p is a continuous subset of tokens of a sentence $\{t_k, t_{k+1}, ..., t_{k+m}\}$ of cardinality at least one. That is, a single token may constitute a span. A **named entity** n is a pair $\langle p, l \rangle$ of a span p and a label l. A **constituency tree** is a pair $\langle S, C \rangle$ of a sentence S and representation of a parse tree of S

using a context free grammar.[3]

We are now ready to define an input $\mathbf{x}$ more precisely. An input $\mathbf{x} = \langle S, C, N \rangle_{i=1}^{n}$ is a sequence of tuples consisting of a sentence, a constituency tree, and a set $N = \{n_0, ..., n_k\}$ of named entity annotations of the sentence. The named entity annotations in N are assumed to be non-overlapping.

Before turning to the output structure, we also introduce the term **mention**. A mention is simply a span in a sentence, with the exception of a special mention m_0 that has no surface realization but acts as an abstract mention for every document (i.e., a *root*). When convenient, we will sometimes regard the pair $\langle \mathbf{x}, M \rangle$, where $M = \{m_0, m_1, ..., m_n\}$ is a set of mentions over the document $\mathbf{x}$.

The output structure $\mathbf{y}$ can be formalized as a set of sets $\{\Gamma_i\}_{i=1}^{n}$, where each Γ_i is a set of **mentions** $\{m^{(1)}, ..., m^{(k)}\}$, where a mention is a span. Additionally, we impose the following constraints on the output structure:

1. **Unique mention constraint**: No two mentions in M cover the same span;
2. **Unique cluster constraint**: Every mention in M is in at most one cluster Γ;
3. **No singleton constraint**: Every cluster Γ has cardinality at least 2.

With these definitions at hand, we conclude by defining a total order over the mentions in M. We will use the notation $m_i \prec_{\text{ment}} m_j$ to denote that mention m_i precedes m_j. First of all, the root mention m_0 precedes any other mention in a document. For the other mentions, the order by and large follows the obvious ordering of the sentences and tokens in $\mathbf{x}$. That is, for non-overlapping mentions m_i and m_j, $m_i \prec_{\text{ment}} m_j$ precisely when the span corresponding to m_i precedes the span corresponding to m_j. If the corresponding spans are overlapping, then $m_i \prec_{\text{ment}} m_j$ if the first token of m_i precedes the first token of m_j. In the case where the mentions begin on the very same token, the longer span is taken to precede the shorter. By the unique mention constraints, no two mentions will refer to the exact same span, and we thus have a total order.

3.2.2 Head Word and Mention Extraction

Figure 3.2 shows an overall view of the pipeline in the coreference resolution setup. The raw document $\mathbf{x}$ is first fed to a component that labels the head words of the non-terminals in the constituency tree (this additional level of annotation is indicated by pass-

[3] We abstain from a formal definition of constituency trees and refer the reader to standard text books, e.g., Jurafsky and Martin (2009), for further details.

ing the input $\mathbf{x}'$ on to the next step). Then, the mention extraction component extracts a set of mentions (M), after which the coreference resolver sets in.

Figure 3.2: Overall pipeline for coreference resolution.

Head words of phrases determine the syntactic category of that phrase, e.g., the head word of a noun phrase is typically its last noun. Extracting head words of mentions is standard practice in coreference resolution since heads words provide a strong basis for features in a machine-learning system. With regard to parsing, the use of head words can be traced back to work on constituency parsing (Magerman, 1994), where head words have been observed to be of important utility to improve parsing performance (Collins, 1999). The basic idea is to define rules specific to the different non-terminals that scan either from left to right or right to left and select the first token with a certain part-of-speech tag as the head. This idea also underpins common conversion schemes for converting constituency treebanks into dependency treebanks (de Marneffe et al., 2006; Johansson and Nugues, 2007a). Head-finding rules are treebank- and language-specific, and for the data sets we experiment with (see Section 3.3.1) we use the head finding rules of Choi and Palmer (2010) for Arabic and English, and Zhang and Clark's (2011) for Chinese.

The overall architecture of the mention extraction component is shown in Figure 3.3. It is implemented as a set of rules primarily operating on parts of speech and the constituency tree of input sentences. Additionally, named entities may also be extracted. In a first step, a number of rules extract spans. Then, spans are merged and turned into mentions (ensuring uniqueness). After that, potentially some filters are applied to remove some mentions.

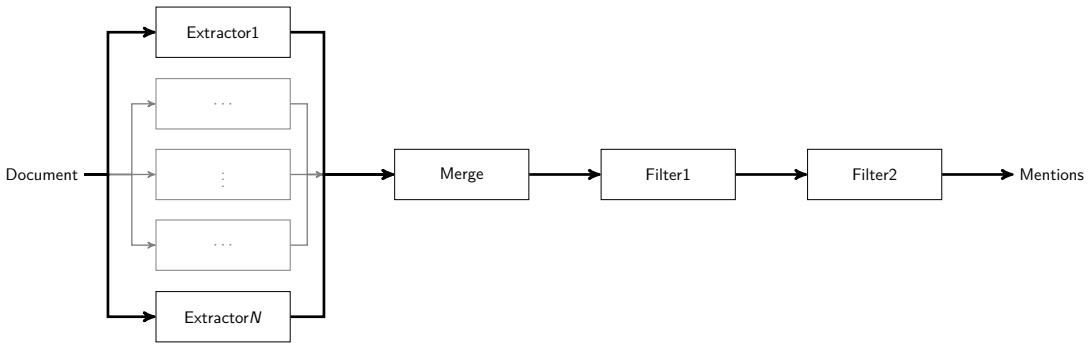


Figure 3.3: Overall architecture of mention extraction.

Similar to head word extraction, also the mention extraction and filtering rules are language- and treebank-specific. Specifically, we extract the spans matching the following rules:

- For Arabic, Chinese, and English: All noun phrases (as indicated by the non-terminal label NP);
- For Arabic, Chinese, and English: All personal and possessive pronouns (as indicated by the part of speech tags PRP and PRP$ in Arabic and English, and PN in Chinese);
- For English: All named entities.

Additionally, for English two filters are applied. The first looks for spans that have the same head noun and removes the shorter span (this rule is applied exhaustively until no two spans have the same head noun). The second filter aims at three types of expletive or generic pronouns that frequently occur without referring to a real-world entity. This includes expletive usages of *it*, generic uses of *you*, as well as generic use of *we*. While the two former are well-known linguistic phenomena, the latter might come as a surprise. This is largely an artifact of the annotations in the data sets. It includes transcripts of broadcast news and conversations, where the hosts sometimes use the pronoun *we* to refer to the viewers and hosts collectively, such as in *We now turn to ....* These uses of *we* are not labeled for coreference.

The pronoun filter is implemented as a binary maximum entropy classifier (implemented using the MALLET toolkit (McCallum, 2002)). One classifier is trained for each pronoun individually, and the training instances are drawn from the training set, and includes all the uses of the corresponding pronoun. The features used for the classifier are primarily drawn from previous work on classifying expletive uses of *it* (Boyd et al., 2005). In order to maintain a high recall, rather than using a default probability threshold of 0.5, we require that the classifier assigns a probability of at least 0.95 for the pronouns to be non-referential in order to filter them out.

The mention extraction component at work can be exemplified by means of the English input sentence in Figure 3.4. Extracted mentions are indicated by a box around the tokens and named entity, or around the non-terminal nodes. This includes all noun phrases as well as the named entities *Walter Sisulu*, *African National Congress*, and *yesterday*. While the first and the latter of these coincide with noun phrases, the *African National Congress* does not. However, it has the same head word as the longer constituent

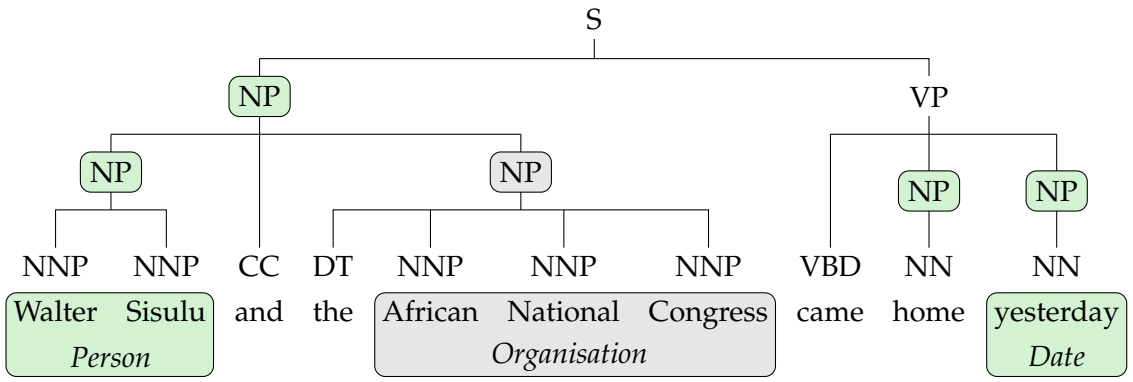


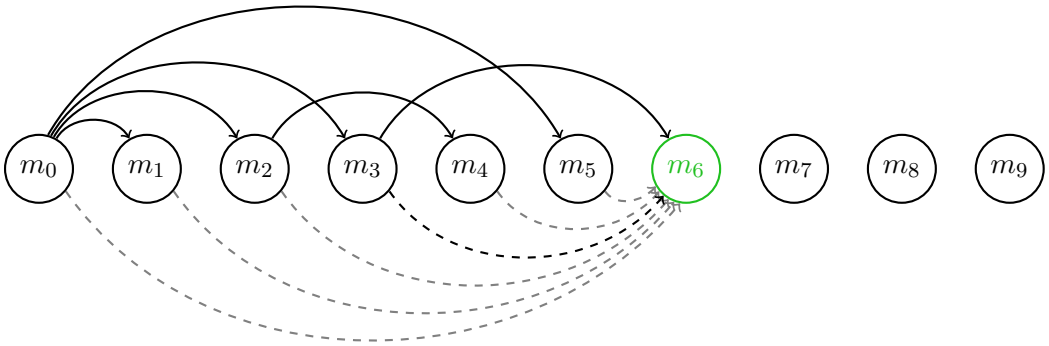
Figure 3.4: Example input structure for mention extraction corresponding to the first sentence of the example in Figure 3.1. Extracted mentions are indicated by a shaded box around the tokens or on the non-terminal node. Extracted mentions with a gray background are pruned, only the ones with a green background are retained (for the phrase *Walter Sisulu* and the word *yesterday*, the named entity annotations and noun phrases node coincide).

the African National Congress and is therefore removed by the first filter. To make matters worse, however, this constituent in turn shares the head with the longer coordinated subject *Walter Sisulu and the African National Congress* and is therefore also filtered out. This de facto means that the rules are inadequate for this example, as the noun phrase *the African National Congress* does in fact participate in a coreference cluster (cf. Figure 3.1).

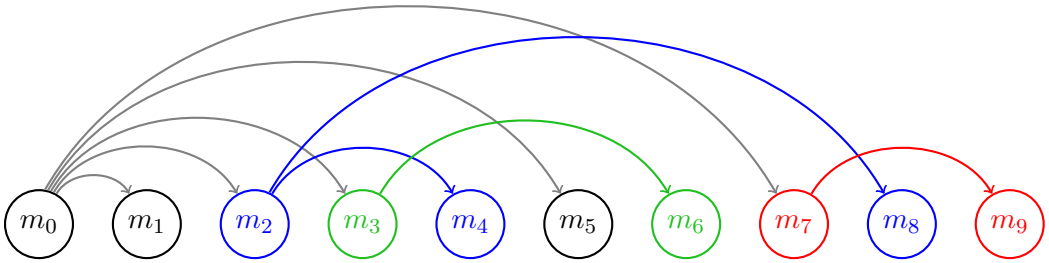
3.2.3 Tree-based Coreference Resolution

The basic idea for the coreference resolution procedure was already outlined in the introduction (cf. Section 1.1). To create the output structure $\mathbf{y}$, i.e., a coreference clustering over the mentions, a detour is made through the larger search space $\mathcal{Z}$, where mentions are represented as nodes in a tree and an arc between two nodes indicates they are coreferent. The projection function π that maps from latent space $\mathcal{Z}$ into output space $\mathcal{Y}$ is fairly straightforward. For a tree $\mathbf{z}$, all subtrees under the root are selected and placed in their own sets. Then, all singleton sets are discarded. The last step here is critical, as it ensures the no singleton constraint (cf. Section 3.2.1). This is also the main motivation why the mention extraction step aims at high recall – false negative mentions that are never extracted will have no chance to participate in a cluster, but false positive mentions can still be discarded by being connected to the root and not having any outgoing arcs.

To decode the tree structure the mentions are first sorted according to the total order defined in Section 3.2.1. Then, a left-to-right pass is made and for each *active mention*



(a) Partially finished decoding process. Current active mention m_6.



(b) Complete coreference tree.

(c) Corresponding output partition.

Figure 3.5: Overall depiction of the decoding process. In (a) the left to right pass is in progress and m_6 is the active mention. In (b) the tree has been completed. In (c) the tree has been reduced to clusters.

all arcs coming from the left are considered. That is, the PERMISSIBLE decisions at each mention include all pairs of the active mention and the mentions that precede it according to the total order (including the root m_0). Figure 3.5 displays the general flow of the procedure. Figure 3.5a displays a potential snapshot of decoding in progress where the active mention is m_6. The preceding mentions have already been assigned incoming arcs (shown above the mentions), and the incoming dashed arcs indicate the current options

for m_6 (shown below the mentions). In this example, we assume that the incoming arc from m_3 is the highest scoring one (as indicated by the black color) and it is thus selected. The process then continues in the same manner with mentions m_7 through m_9. After processing all mentions, a complete tree has been constructed, as depicted in Figure 3.5b. In the final step the tree is mapped to the output structure, by removing all singletons and placing mentions that are linked in their own clusters (Figure 3.5c).

As discussed in Chapter 2, the latent structure is derived by decoding in the constrained space $\mathcal{Z}_{\mathbf{y}}$. But instead of considering decisions that PERMISSIBLE yields, we constrain the search space to allow only those decisions given by ORACLE. While PERMISSIBLE allows all incoming arcs from the left of the active mention, ORACLE only allows for those that correspond to the correct clustering according to $\mathbf{y}$. Specifically, if the active mention is the first of its cluster according to the ordering, ORACLE returns only the arc from the root node m_0. Otherwise, it returns all incoming arcs originating from other mentions that belong to the same cluster as the active mention.

3.3 Experimental Setup

With the task and model defined in the previous section we turn to the experimental setup. In this section we will discuss data sets and evaluation metrics. We will then provide the missing details in order to instantiate the model using the framework from Chapter 2.

3.3.1 Data Set

We use the data set from the CoNLL 2012 Shared Task (Pradhan et al., 2012) for all experiments. The data set comprises three languages – Arabic, Chinese, and English – and is drawn from the OntoNotes corpus (Weischedel et al., 2011), a very large multi-lingual corpus annotated for a wide set of linguistic phenomena. The annotations include all the layers we assume in our input, such as named entities, parts of speech, and constituency trees. The CoNLL 2012 data set has become the de facto standard benchmark data set for coreference resolution for these three languages. We will follow the closed track setting from the 2012 shared task, which assumes that the pre-processing layers are automatically predicted, thus emulating a realistic evaluation scenario. Specifically, we use the predictions that were provided as part of the shared task. We also adopt the same split into training, development, and test sets from the shared task. The only external resource (which was also allowed in the shared task setting) that we use is a data set of gender

statistics for English (Bergsma and Lin, 2006) that we use for English to assign gender and number to noun phrases.

3.3.2 Evaluation Metrics

Evaluation of automatic coreference resolution is challenging since there is a wide variety of things that can go wrong: a predicted cluster can miss a few correct mentions or contain mentions of several clusters. Conversely, the mentions of a single correct cluster can be distributed over several clusters. To make matters worse, assuming gold standard mentions are not given, it is highly probable that correct mentions are missed and also that spurious mentions are introduced. These so-called *twinless* mentions have caused further confusion as to how evaluation should be carried out. In general, this has led not only to several evaluation metrics, but also several variants of the same metric that handle twinless mentions differently (Stoyanov et al., 2009; Rahman and Ng, 2009; Cai and Strube, 2010, *inter alia*).

We will abstain from a lengthy discussion on all available metrics and settle with those established in the CoNLL 2011 and 2012 Shared Tasks (Pradhan et al., 2011, 2012), which are also the ones we use in this chapter: MUC (Vilain et al., 1995), B^3 (Bagga and Baldwin, 1998a), and $CEAF_e$ (Luo, 2005). Additionally, the arithmetic mean of these three metrics' F_1 have become the de facto standard CoNLL metric (sometimes also called MELA; Denis and Baldridge (2009)). Most of our analysis is carried out using this number alone, since it captures all metrics and distills them into a single figure. Below we will outline the basic ideas and potential drawbacks of the three metrics. Pradhan et al. (2014) and Luo and Pradhan (2016) provide more detailed definitions and discussions of corner cases for the involved metrics. For evaluation in the experiments we use Pradhan et al.'s (2014) implementation of the metrics.

MUC The MUC metric (Vilain et al., 1995) is the oldest metric and was established with the Message Understanding Conference (MUC) evaluation campaign, which led to the first standard data set for coreference. MUC is often said to be a *link-based* metric as it considers possible pairwise links between mentions. Here the number of links within clusters is kept minimal to keep the cluster connected, i.e., for a cluster of n mentions, $n - 1$ links are needed. Recall is computed by counting the number of common links between gold and a system response divided by the number of links in the gold partition. Conversely, precision is computed by dividing the common links by the number of links in the system response.

MUC has primarily been criticized for two reasons. First, it tends to favor larger clusters since the computation of recall is fairly generous. In fact, a prediction placing all (true) mentions in a single cluster will receive perfect recall, while still maintaining somewhat reasonable precision. Second, since the metric is link-based, it cannot handle singletons.

B^3 Bagga and Baldwin (1998a) take aim at the aforementioned drawbacks of MUC and argue that merging two smaller or two larger entities should be penalized differently (with a higher penalty for merging two larger entities). They proposed the B^3 metric, which is often called a *mention-based* metric as it places the individual mentions at the center during evaluation. B^3 zooms in on each mention individually and, for each mention, computes the size of the intersection between the gold cluster for that mention and the system cluster containing the same mention. Then, for each mention, recall is computed by dividing this number by the size of the gold cluster, whereas precision divides it by the size of the predicted cluster. Recall and precision for an entire document is then computed by summing up the corresponding ratios for each mention and dividing by the number of total mentions in gold and system response, respectively.

$CEAF_e$ Luo (2005) notes that B^3 (and also MUC) allows the same entity to be counted more than once. Since the numerator for precision and recall in B^3 considers the intersection between gold and system clusters, this intersection may be used several times while scoring different mentions belonging to the same cluster. This leads to counter-intuitive effects similar to the problems with MUC, where placing all mentions in a giant cluster leads to perfect recall. Conversely, leaving all mentions in a single cluster leads to perfect precision.

Instead, Luo (2005) argues that a cluster (and thus also each mention) should only be used once and proposes the Constrained Entity-Alignment F-Measure (CEAF). The idea is to create an alignment between gold and system clusters and then score the pairs using a similarity metric. Since clusters are only used once, CEAF is often regarded as an *entity-based* metric. Finding the alignment can be treated as a maximum bipartite matching problem, where each cluster is regarded as a vertex, and edges between them are assigned a weight according to the similarity metric. This problem can be solved efficiently with the Hungarian algorithm (Kuhn, 1955; Munkres, 1957).

The open issue is how to define the similarity metric between clusters. One option is to define the similarity as 1 if two clusters are identical, and 0 otherwise. However, this gives no credit to partial matches and may thus be of little use to measure small,

incremental improvements. Instead, Luo (2005) proposes two different similarity metrics (denoted ϕ_3 and ϕ_4 in his paper). Recall is computed by summing the similarities of every pair of aligned clusters and dividing it by the self-similarity of the gold cluster. Precision is computed analogously. The similarity metric ϕ_3 simply computes the number of shared mentions between the two clusters, whereas ϕ_4 computes the mention F-measure between the clusters. These two versions of CEAF have subsequently become called mention-based CEAF, CEAF$_m$, when using ϕ_3 and entity-based CEAF, CEAF$_e$, when using ϕ_4. CEAF$_m$ should be regarded as a misnomer in relation to the link-based and mention-based metrics MUC and B^3 – CEAF is inherently an entity-based metric, as it aims to align entities in gold and system prediction.

3.3.3 Instantiation

The implementation of the coreference system employs the framework from Chapter 2. Note that in addition to beam search and its array of update methods for approximate search, we can also carry out exact decoding assuming the feature set is arc-factored, i.e., that features only involve pairs of mentions connected by an arc. In terms of the required hyperparameters and functions required, some of these have been discussed above already, e.g., PERMISSIBLE and ORACLE. The beam size, update methods, and number of epochs will be varied during the experiments but 20 and 25 are used as default values for beam size and number of epochs, respectively.[4]

Loss Function. The loss function we use is shown in Equation 3.1. It is composed of two parts, mention mistakes and root mistakes. When evaluating a prediction $\hat{\mathbf{z}}$, every mention that has received an erroneous incoming arc is considered a mistake. If the correct antecedent is another mention it is classified as a MENTIONMISTAKE. If the correct antecedent is the root mention m_0, then it is classified as a ROOTMISTAKE. The loss function in Equation 3.1 computes a linear combination of these mistakes, where root mistakes are weighted by a factor 1.5 following Fernandes et al. (2012). We did not attempt to tune this factor and it is kept constant throughout.

$$\text{LOSS}(\tilde{\mathbf{z}}, \hat{\mathbf{z}}) = \text{MENTIONMISTAKES}(\tilde{\mathbf{z}}, \hat{\tilde{\mathbf{z}}}) + 1.5 \cdot \text{ROOTMISTAKES}(\tilde{\mathbf{z}}, \hat{\mathbf{z}}) \tag{3.1}$$

[4] As stated in Chapter 1 the implementation we used for the experiments in this chapter is available on the author's website.

Feature Extraction. The feature sets are language-specific and manually tuned. We abstain from a long discussion of each and every possible template but summarize briefly what categories of features we use and their origin.

Traditional machine-learning approaches to coreference resolution have relied on small sets of **categorical** features that are typically computed by some more or less complex rules over the input (Soon et al., 2001; Ng and Cardie, 2002b). This includes, e.g., binary indicators as to whether a mention is a pronoun or not; string matching under certain relaxations such as removing determiners and quotation marks; or rules that attempt to extract aliases (for instance, *International Business Machines* can be referred to as *IBM*). Similarly, assigning gender and number can be done by tables of pronouns, or which we do in English, extended also to cover common nouns by looking them up in automatically mined frequency statistics (Bergsma and Lin, 2006). Moreover, we also include binary indicators for whether a mention appears within quotes, or whether two mentions along an arc are nested. Finally, every mention is assigned a particular *type*: pronoun, common noun phrase, or name. Pronouns and names are detected by part of speech tags, and all other mentions are regarded as common noun phrases.

A second set of features are **lexicalized** features that extract surface forms of the words in, and possibly surrounding, a mention. This can be either the full surface string of the mention or target a specific words such as the surface form of the head word, the first or last words of a mention, or even a word preceding or following the mention. While lexical features had been used (Rahman and Ng, 2009) before the CoNLL 2011 and 2012 Shared Tasks, these shared tasks and the much larger data set OntoNotes increased the usage of lexicalized features for coreference resolution (Björkelund and Nugues, 2011; Durrett and Klein, 2013).

We also use **grammatical** features drawn from the parse trees in the data (Yang et al., 2006; Rahman and Ng, 2011a). This includes encoding the labels of the non-terminals corresponding to a mention or the parent non-terminal, but also the subcategorization of the parent constituent which provides information on how this mention is situated in the parse tree. Additionally, the path through the parse tree between a pair of mentions is also used (Gildea and Jurafsky, 2002). These types of features may capture typical grammatical constraints on coreference, e.g., such as Chomsky's (1981) binding principles. Feature templates that extract paths in the parse tree remove the work of manually constructing heuristic rules corresponding to these principles and may also capture further generalizations. In that vein, we do not only extract paths when both mentions occur in the same sentence. When two mentions are in separate sentences, the path traverses the parse trees through an auxiliary non-terminal that sits above each sentence's parse

tree. These features give a rough indication to how deeply embedded each mention is in their corresponding sentences.

Numerical features of **distance** are also included. The numerical values are always discretized into buckets. We use two flavors of distance between two mentions: The distance in terms of mentions, and the distance in terms of sentences (Soon et al., 2001).

Additionally, the particular composition of the data sets motivate two more features. As the OntoNotes is drawn from multiple genres the peculiarities of the genres may cause certain typical coreference relations to be more or less frequent. We therefore include the genre as a feature. Moreover, some of the genres include transcribed speech from television where speaker information is provided. For this reason we also use a binary indicator feature that flags if the mentions were said by the same speaker or not. This is critical to get the usage of first and second person pronouns right in spoken discourse.

Finally, as we extended the arc-factored model to include features that can span more than two mentions we also experimented with *non-local* features that are drawn from a greater context in the coreference tree. Of course, all the features described above are also applicable in this context. For instance, a lexicalized template that extracts the head word of the antecedent's antecedent could be used. During development we also experimented with non-local features drawn from previous work on entity-mention models (Luo et al., 2004; Rahman and Ng, 2009), however they did not improve performance in preliminary experiments. The one exception is the **size** of a cluster under consideration (Culotta et al., 2007). As with the distance features, these numerical values are discretized into buckets. Similarly, we also use a feature that indicates the **cluster start** distance, which is the distance in mentions from the beginning of the document to where a potential entity is first mentioned. Again, these numerical values are bucketed.

More generally, theories on discourse and information structure have suggested that certain patterns occur as to how an entity is introduced and subsequently referred to throughout discourse (Prince, 1981; Grosz et al., 1983, 1995). For instance, a new entity might be introduced by a long common noun phrase or a name in object position. Subsequent references to the same entity may be shorter (common) noun phrases or pronouns occurring syntactically in subject position. We therefore compute the **shape** of a cluster by encoding the linear "shape" in terms of mention types as a sequence. For instance, the clusters representing *Gary Wilber* and *Drug Emporium Inc.* from the example in Chapter 1, would be represented as NPN and NCCC, respectively. Where N, P, and C denote the names, pronouns, and common noun phrases, respectively. Moreover, inspired by the Entity Grid (Barzilay and Lapata, 2008), which models the evolution of entities with respect to whether they are in subject or object position, we aim to capture the **cluster**

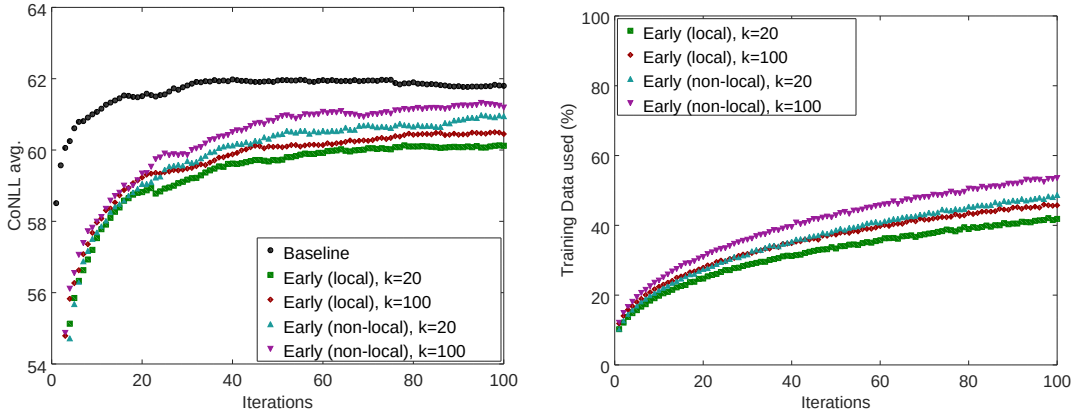
syntactic context. We approximate the grammatical functions of a mention by the path in the parse tree from the mention to the root of its parse tree, similar to the syntactic features mentioned above. The partial paths of mentions in a cluster inform the model about the local syntactic context of a potential cluster.

Armed with the above selection of feature templates we tuned the feature sets for each language. First, as a baseline we started from the feature set from a previous coreference system we had developed (Björkelund and Farkas, 2012) which roughly encompasses features of all categories discussed above except the non-local features. We then optimized the feature sets for an arc-factored model by doing greedy forward/backward feature selection over a pool of templates discussed above, as well as conjunctions among them. In order not to taint the development set, this was performed over the training set of each language, split into two parts where 75% was used for training, and 25% for testing. Feature templates were incrementally added or removed in order to optimize the CoNLL average. The idea is that this **local** feature set is the strongest arc-factored model that allows for exact search and can then be used as a baseline when comparing with models that use **non-local** features and beam search. After freezing the local feature set, the feature selection procedure was repeated to find the optimal non-local feature set.

3.4 Experiments

The experimental evaluation will focus on the relationship between the update methods on one hand, and the feature sets and decoding algorithm on the other. In summary, for each language we have defined two feature sets: The *local* feature set, which only considers features of two mentions. That is, it is an arc-factored model and exact search can be carried out by greedy decoding. The second feature set extends the local one with *non-local* features. While this provides more contextual information to the machine learning model, it also comes at the cost of rendering exact search intractable. Moreover, this implies that regular perceptron updates are not applicable and we have to apply the alternative update methods discussed in Chapter 2.

In this section we will regard the arc-factored model trained with regular PA updates (cf. Algorithm 2.3) and the local feature set as the baseline. This will be compared to the non-local feature set, where we will evaluate each of the update types for approximate search in a series of experiments. While these experiments could be squeezed into fewer plots comparing more things in one go, we have chosen to incrementally present the results for the various update strategies in order to keep the points clear and isolated.



(a) CoNLL Accuracy **(b)** Training Data Usage

Figure 3.6: Learning curve over training iterations comparing the baseline to early updates. In (a) the CoNLL accuracy on the English development set is shown as a function of the number of training iterations. In (b) the amount of training data used is displayed.

3.4.1 Early Updates

We first consider early updates, the oldest of the update strategies for perceptrons with approximate search. Figure 3.6 shows a comparison of the baseline, which only uses the local feature set, and the system trained with early updates. Recall that early updates interrupt the training as soon as the correct item falls off the agenda and immediately makes an update. This implies that the remainder of the training document is discarded and not used in this iteration.

The left plot in Figure 3.6 shows the CoNLL accuracy on the English development set as a function of training iterations. The choice of using English as an example is arbitrary as the curves look roughly the same for Arabic and Chinese. In addition to the baseline, four versions of early updates are used, varying either the feature set or the beam size. The most obvious result is that the baseline is the strongest system. The chief reason for this is that the baseline is trained with exact search and all training data is used, while this is not certain in the case of early updates. This is depicted in the right plot in Figure 3.6, where the amount of training data used is plotted against iterations for each of the early update systems. As the plot shows, early updates rarely even use more than half the training data.[5]

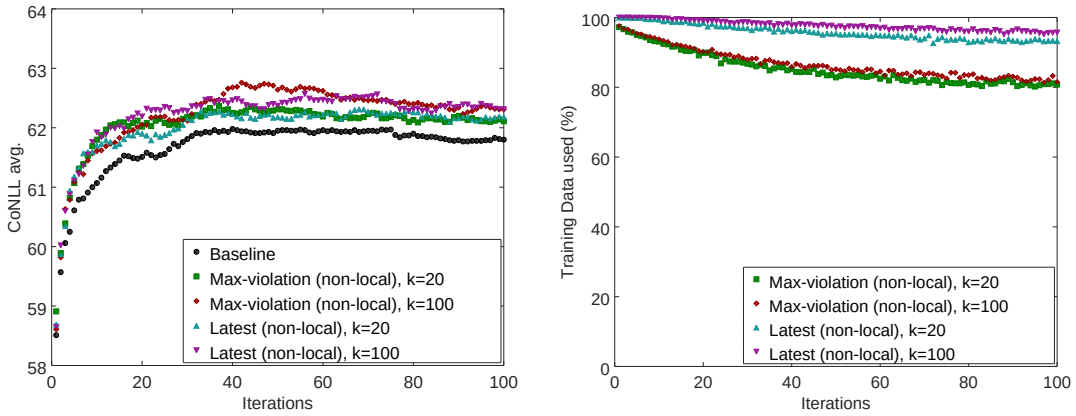
[5]The baseline, which is based on exact search and the regular perceptron, would always use 100% of the training data and is for simplicity omitted from this and subsequent similar plots.

It should of course be noted that training with early updates using only the local feature set makes little sense since here the search problem can be solved exactly (i.e., what the baseline does). However, the four versions using early updates illustrate very well the two possible solutions to improve learning and accuracy – either increase the beam size or improve the feature set.

In conclusion, Figure 3.6 tells us that, despite access to richer features, early updates are not sufficient to outperform the baseline. The main reason is that early updates happen too early and therefore too much training data is ignored.

3.4.2 Max-violation and Latest

By comparing early updates to the baseline we have established that early updates are not sufficiently strong to compete with the baseline even with access to richer features. We saw that the main reason is that early updates discard too much training data. In Figure 3.7 we compare the baseline to Max-violation and Latest updates using the non-local feature set. Again the comparison is made for English on the development sets, but the situation for Arabic and Chinese is essentially the same. Just like early updates, max-violation and latest updates do not commit to using all the available training data, however, they are aimed at making better use of it by going further. For max-violation the aim is to maximize the violation, whereas latest take the last possible item where violation is ensured.



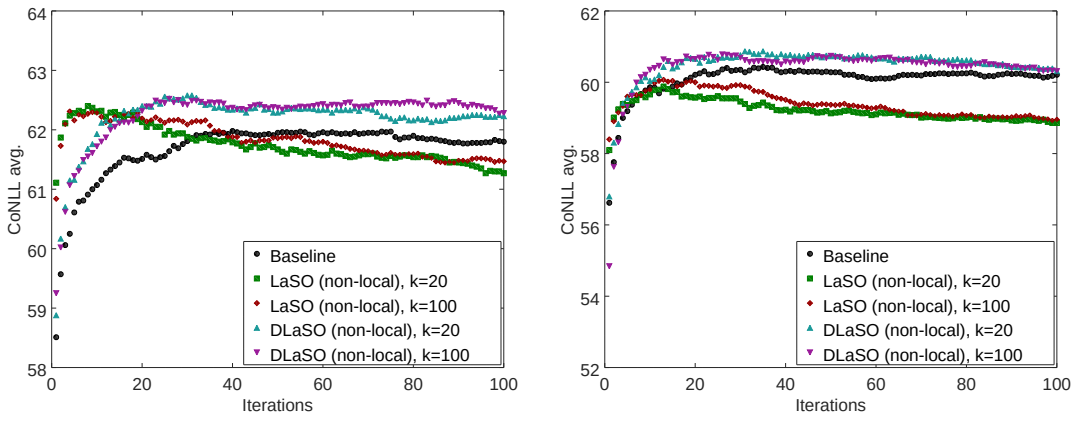
(a) CoNLL Accuracy **(b)** Training Data Usage

Figure 3.7: Learning curve over training iterations comparing the baseline to max-violation and latest updates. In (a), the CoNLL accuracy on the English development set is shown as a function of the number of training iterations. In (b) the amount of training data used is displayed.

In this case we see that the stronger non-local feature set combined with sufficiently advanced updates outperform the baseline. Varying the beam gives the expected result, with slight improvements when a greater beam is used. In terms of accuracy, max-violation and latest are roughly trailing each other, albeit with some small peaks for max-violation. Interestingly the training data usage decreases over the iterations although it stays higher than 80% throughout. Intuitively it is no surprise that it starts close to 100%, since in early iterations the weights are far from fit to the training data. This means that every decision in the sequence easily can be wrong and thus contributes violation. Longer sequences then have a chance to aggregate a greater violation.

3.4.3 LaSO and DLaSO

We finally turn to comparing LaSO and DLaSO with the baseline. Figure 3.8 shows the same type of learning curves that we have seen for the other update methods. In this case we display plots for English and Chinese. For Arabic the trends are similar, but more pronounced, which can probably be attributed to the fact that Arabic has the smallest training and development sets and also the lowest overall scores, making the CoNLL average more sensitive to small changes. In contrast to the other update methods we have seen in earlier plots, LaSO and DLaSO, as well as the baseline, make full use of all available training data and plots of training data usage are omitted since they would constantly be at 100%.

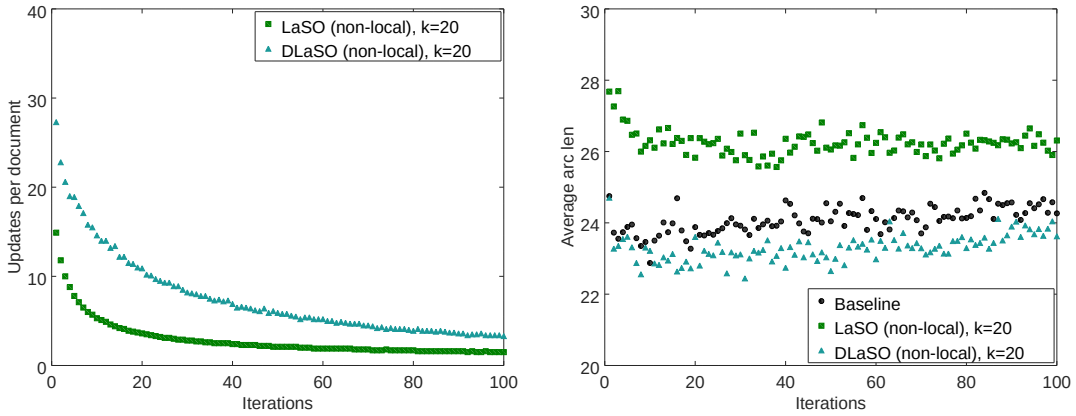


(a) English **(b)** Chinese

Figure 3.8: Learning curve over training iterations comparing the baseline to LaSO and DLaSO updates on English (a) and Chinese (b).

The plots in Figure 3.8 again show that the baseline can be outperformed by means of appropriate update methods that can leverage the extended feature sets. An increased beam size, however, rarely leads to any noticeable gains. Comparing the two update methods it appears that LaSO peaks quickly, and then starts declining. DLaSO on the other hand takes a bit longer to peak, but then remains relatively steady throughout. Overall, the global peak of DLaSO is greater than that of LaSO, which is particularly pronounced for Chinese (and also for Arabic, cf. Section 3.4.4).

Even though LaSO and DLaSO are very similar architecturally, they differ significantly in the details of training. We look at two different ways of quantifying some aspects of the learning process: First, how many updates happen per document on average during an iteration. Although the updates take place at different points in time – multiple times within a document for LaSO vs. all updates collectively at the end of a document for DLaSO – this gives an indication of how frequently the correct item falls off the beam when training with either of these methods. Second, we wish to get an impression of what the latent trees that are used for updates look like. One way of quantifying this is the average distance between the mentions that are connected by an arc. In this case we calculate this number only over arcs that connect true mentions, i.e., the distance between the root m_0 and the first mention of each cluster are not counted (also implying that singletons are excluded). Figure 3.9 shows plots of how these numbers vary as a function of the training iterations. While we have selected the English data set and a beam size of 20 for these plots, the general picture stays the same across all languages and beam sizes.



(a) Updates per document **(b)** Average arc length

Figure 3.9: Comparison of LaSO and DLaSO during training on English using a beam width of 20. (a) shows the number of updates per document. (b) displays the average length of arcs in the latent tree.

It is important to keep in mind that the numbers in Figure 3.9 show averages across all documents across an entire iteration, and we must interpret them cautiously and not draw too far-fetching conclusions. Nevertheless, the plots point to some general trends. As for the number of updates, LaSO consistently makes fewer updates per instance than DLaSO. Another way of putting this is that the correct solution stays in the beam longer. When it comes to the average arc length, LaSO prefers longer arcs. This is a bad sign, as distance tends to be a strong discriminator for coreference in pair-wise models, where longer distances reduce the likelihood of two mentions being coreferent (Soon et al., 2001; Ng and Cardie, 2002b). More generally, we argue that these figures highlight the inherent difference between LaSO and DLaSO – while the latter only updates after processing an entire document, the former updates the weights within documents. The online nature of the perceptron makes it quite sensitive to the most recent update(s) and LaSO's increased arc length and reduced number of updates per document can be regarded as a manifestation of this. First, since the learner receives feedback about erroneous incoming arcs within documents, it updates the weights in favor of likely, as well as (and maybe even more importantly) in disfavor of unlikely, antecedents. Moreover, as many features are lexicalized and therefore relatively rare, this yields strong (dis-)preferences while decoding the remainder of the document, resulting in fewer mistakes. Second, since a LaSO update requires recomputing the latent gold structure, this effect will carry over and select latent antecedents that were seen in the earlier updates within an instance.

The discussion above should make us wary. If we take a step back and consider the difference between training time and test time, it is clear that LaSO provides too much feedback within instances. Receiving feedback half-way through a document does not properly emulate the situation at test time, where, by definition, no feedback is available. In conclusion, we reason that DLaSO is a more intuitive update method than LaSO itself. When combined with latent structure this effect is exacerbated and the appeal of LaSO is yet lower. Empirically, DLaSO updates yield models that are at least as good as LaSO updates. When using latent structure DLaSO additionally has the advantage of being faster, as the latent structure does not have to be recomputed.

3.4.4 Summary of Update Methods

To conclude the discussion about update methods we summarize the main points. We have compared the baseline – i.e., an arc-factored model that uses only *local* features that only scope individual pairs of mentions – with extended models that have an increased scope over *non-local* features. The introduction of non-local features entails that the search

	Arabic		Chinese		English	
	CoNLL Score	Iteration	CoNLL Score	Iteration	CoNLL Score	Iteration
Baseline	45.72	41	57.90	57	60.79	47
Early	42.53	95	57.83	100	60.98	95
Max	46.32	37	59.96	55	62.37	37
Latest	46.23	68	60.19	69	62.31	68
LaSO	45.27	8	59.88	13	62.40	8
DLaSO	47.52	30	60.87	31	62.58	30

Table 3.1: Maximum accuracy from the learning curves seen extended to all data sets. In all cases a beam size of 20 is used and results are evaluated on the development sets.

problem cannot be solved exactly and this in turn requires alternative update methods for the structured perceptron that ensure violation.

Table 3.1 shows the maximum accuracy on the development sets for each language for each update method using a beam size of 20 where we have trained the system for 100 epochs and evaluated it after each epoch. While the baseline uses only the local feature set, all the other methods use the extended non-local feature set. In addition, the table indicate after which iteration the maximum was attained. The table generally corroborates the findings that we have gone through above: Early updates struggle even to outperform the baseline – for Arabic the discrepancy is considerable, for Chinese and English it is minuscule (see also Section 3.4.1). The chief reason is that early updates discard too much training data, although these models do improve slowly, as indicated by the fact that the maximum is typically attained during very late iterations. As discussed in Section 3.4.2, max-violation and Latest outperform the baseline. Generally they tend to peak somewhere roughly in the middle of the 100 epochs, in favor of max-violation which peaks earlier. Finally, as we saw in Section 3.4.3, LaSO and DLaSO also outperform the baseline, with LaSO peaking quite early. All in all however, DLaSO obtains the best results across the board.

3.4.5 Test Set Results

So far this chapter has entirely been concerned with comparing feature sets and update methods. In a greater context we also believe that the analysis and breakdown provided above are the more valuable experimental results in this chapter. Nevertheless, for the sake of completeness and potential value as comparison for future work, we provide

	MUC			B^3			$CEAF_e$			CoNLL
	Rec	Prec	F_1	Rec	Prec	F_1	Rec	Prec	F_1	avg.
Arabic										
Baseline	46.16	51.92	48.87	42.38	50.40	46.04	51.76†	47.32	49.44	48.11
Non-local	48.38	54.68†	51.34†	44.47	50.66	47.36	49.80	50.78†	50.28	49.66†
Chinese										
Baseline	60.90	69.85	65.07	51.69	63.45	56.97	54.60†	58.15	56.32	59.45
Non-local	62.02†	69.92	65.73	53.25†	62.44	57.48	54.24	59.57	56.79	60.00
English										
Baseline	66.45	72.99	69.57	53.43	61.58	57.22	53.37	55.94	54.63	60.47
Non-local	67.28	73.93†	70.45†	54.69	62.30	58.25†	52.56	58.76†	55.49	61.39†

Table 3.2: Overall comparison of the system with and without non-local features on the three data sets. Significant changes at the $p < 0.01$ level are indicated by †.

the results on the test sets as well. While our system defined a new state-of-the-art at the time of publishing (Björkelund and Kuhn, 2014), these results have (naturally) been superseded since (Wiseman et al., 2016; Clark and Manning, 2016; Lee et al., 2017).

As the general conclusion of the preceding experiments is that DLaSO yields the strongest non-local models, we here compare it with the baseline on the test sets. For these experiments the system was run in a default setting using exact search for the arc-factored local system, and beam search with a beam width of 20 for the non-local system. Both systems were trained for 25 iterations.

The results, including the constituent results of the CoNLL average as well as their values of precision and recall, are shown in Table 3.2.[6] Significant changes at the $p < 0.01$ level are indicated by †. For significance testing we use the Wilcoxon signed-rank test. First of all, the table displays a general improvement across the board, as indicated by the consistent improvements in F-measures and the CoNLL average. Interestingly, recall for $CEAF_e$ tends to go down when moving from the local to the non-local model, sometimes significantly so. This probably indicates that the system with the richer feature set constructs slightly smaller, but more precise, clusters.

[6]These results appear lower than those we have previously published (Björkelund and Kuhn, 2014) since we only trained on the training sets. The CoNLL 2012 Shared Task set an obscure precedent, where it was allowed to train the final system on the concatenation of training and development sets, generally resulting in greater figures. For the sake of this comparison, where we compare the baseline local system against the non-local system trained with DLaSO, we have chosen to only train of the training sets and evaluate on the development sets.

3.5 Related Work

Before wrapping up this chapter we now consider previous work and position our work in a greater context. As for the most immediate related work on coreference resolution, Ng (2010, 2017) provides comprehensive overviews of progress during the last decades. For a yet broader perspective, we recommend the books by Mitkov (2002) or Poesio et al. (2016), the latter in which we contributed to a chapter on off-the-shelf tools (Versley and Björkelund, 2016).

3.5.1 Noun Phrase Coreference Resolution

Machine-learning approaches to noun phrase coreference resolution started gaining traction in the mid-90s (McCarthy and Lehnert, 1995; Aone and Bennett, 1995, *inter alia)*. Research was furthered by the Message Understanding Conference (MUC) campaigns that defined a standard benchmark data set. Since Soon et al. (2001) set a new state of the art with a pair-wise model using decision trees, considerable effort has gone into improving this model. This includes swapping out the machine learning algorithms for more feature-intensive classifiers such as support vector machines (Rahman and Ng, 2011c), log-linear models (Versley et al., 2008; Björkelund and Nugues, 2011), or perceptrons (Bengtson and Roth, 2008; Stoyanov et al., 2009). Additionally, the choices of how to create training instances (Ng and Cardie, 2002b) and/or how to derive clusters from pairwise classifications (Björkelund and Farkas, 2012) have been explored. Of course, new feature templates have also been designed and empirically evaluated (Ng and Cardie, 2002b; Bengtson and Roth, 2008).

In parallel, the architecture has also been reconsidered and the alternative entity-mention model has been proposed (Luo et al., 2004; Yang et al., 2008). This model is motivated by how the pair-wise model lacks in expressivity. The most commonly used features in this type of model aim to model agreement within clusters, such as gender and number agreement (Rahman and Ng, 2009). More generally, these approaches typically alter the basic feature definitions from pairwise models when introducing entity-level features, making a proper comparison difficult. This contrasts with our work, as our mention-pair model simply constitutes a special case of the non-local system.

The use of latent antecedents received considerable attention after Fernandes et al. (2012) presented their tree model in the CoNLL 2012 shared task, though also Chang et al. (2012) used latent antecedents in the same shared task. The idea of latent antecedents can at least be traced back to Yu and Joachims (2009). It has since been applied extensively

in the literature (Chang et al., 2013; Durrett and Klein, 2013; Martschat and Strube, 2015). It should however be noted that the idea of selecting antecedents intelligently can be argued to originate from earlier work on training instance creation for pair-wise classifiers (Ng and Cardie, 2002b), the novel thing with latent antecedents is that no manual heuristic is required.

In terms of algorithmic modeling of coreference, the role of the root node should also be properly recognized. While early mention-pair models exclusively focused on pairs of mentions, the root node plays a role in that a machine-learned model can learn to favor arcs from the root node. The consequence is that these models can learn to favor that a mention is *not* coreferent with preceding mentions. In the tree model this effect is facilitated by the root node, but even when a tree is not explicit it the modeling approach, this effect can be accommodated by introducing a dummy mention that plays the same role both in pair-wise models (Morton, 2000; Durrett and Klein, 2013) or as a dummy entity in entity-mention models (Rahman and Ng, 2011c).

3.5.2 Coreference and Related Tasks

Coreference resolution could be regarded as a specific type of the more general problem called *anaphora resolution*. In linguistic terms, an anaphor refers to an expression that does not stand on its own (e.g., a pronoun), but relies on some other contextual expression (e.g., a noun phrase that the pronoun refers to), an antecedent, for proper interpretation. Historically, pronoun resolution, i.e., finding the antecedents of pronouns in a text has been extensively analyzed as a task on its own, both using rule-based and heuristic methods (Hobbs, 1978; Lappin and Leass, 1994) as well as statistical approaches (Cherry and Bergsma, 2005; Charniak and Elsner, 2009).

For coreference resolution anaphoricity has received attention in the context of mention detection. The large discrepancy between mentions annotated for coreference and mentions extracted by a typical mention extractor (e.g., the one we described in Section 3.2.2) has triggered work on *anaphoricity detection* (Ng and Cardie, 2002a; Uryupina, 2003). Here an anaphoric noun phrase is not regarded in the linguistic sense of anaphoricity, but rather along the lines of whether the noun phrase participates in a coreference cluster or not. This approach has been applied in a pipeline fashion, where a classifier that classifies candidate mentions as anaphoric or non-anaphoric is involved at the mention detection stage (Ng and Cardie, 2002a; Poesio et al., 2004). While these approaches retain a pipeline relationship between mention extraction and coreference resolution, extensions that aim to globally consolidate the results of independent anaphoricity and

coreference classifiers using techniques such as integer linear programming (Denis and Baldridge, 2007) or graph-cut algorithms (Ng, 2009) have also been proposed.

Although classifiers for anaphoricity have been shown to improve overall performance of coreference resolution systems, we believe that models that maintain a strict separation between anaphoricity classification and coreference resolution during learning or inference are inherently inadequate. The tree-based approach pursued in this chapter aims at high-recall mention detection, allowing the coreference resolver to discard mentions as part of the resolution process. Taking this idea further, Lassalle and Denis (2015) extend the latent tree-based model to also include an anaphoricity model. Critically, in their model learning and inference of anaphoricity and coreference is carried out jointly.

Instead of approaching anaphoricity specifically for coreference resolution, it is also a topic of research from the perspective of more linguistics and pragmatics. In this case, classifications of noun phrases are often regarded in terms of *givenness* (Prince, 1981). That is, a noun phrase could be known to a reader (or hearer) from the given document, from background knowledge, or from other contextual factors. These types of annotations are of concern when annotating *information status*. Annotation schemes that aim at describing referential expressions in terms of information status have been developed (Nissim et al., 2004; Baumann and Riester, 2012, 2013) and the task of classifying the information status of noun phrases has also been considered in isolation (Nissim, 2006; Rahman and Ng, 2011b; Cahill and Riester, 2012).

In addition to labeling noun phrases as given or unknown, a particular type of relation that has received considerable attention as part of information status annotating is *bridging* (Asher and Lascarides, 1998; Poesio and Vieira, 1998). A typical example to illustrate this phenomenon would be the following utterance: *John bought a car yesterday. The steering wheel was dirty.* Here there is a clear relationship between the car and the steering wheel, although the two noun phrases clearly do not refer to the exact same entity. While annotation of coreference is arguably much easier (Deemter and Kibble, 2000), annotation and automatic prediction of bridging anaphors have also been examined thoroughly (Lassalle and Denis, 2011; Markert et al., 2012; Hou et al., 2013; Rösiger and Teufel, 2014; Rösiger et al., 2018).

Before concluding this discussion, we loop back to the core task we have discussed – coreference resolution – and note that it has also been explored in cross-document settings, where coreferent mentions across multiple documents should be clustered together (Bagga and Baldwin, 1998b; Gooi and Allan, 2004; Rao et al., 2010). This task is also closely related to that of entity-linking or entity-grounding, where entities in one

or multiple documents (potentially expressed by several references in each document) should be grounded in a knowledge base, such as, e.g., Wikipedia (Cucerzan, 2007; Hajishirzi et al., 2013; Durrett and Klein, 2014).

3.6 Conclusion

In this chapter we have applied the framework from Chapter 2 to the task of coreference resolution. We have relied on latent structure for learning and started with a baseline arc-factored model (Fernandes et al., 2012). In this case exact (greedy) search can be used to find the optimal tree using a feature set which is *local* to individual pairs of mentions, not considering a greater context. On the other hand, an arc-factored model is known to be insufficient in terms of access to features, and it is desirable to extend the scope of features to encompass a larger non-local context. By introducing a larger feature context – i.e., *non-local* features – we also sacrificed the ability to carry out exact search and thus resorted to beam search as the approximate search method.

Couched in this setting we have carried out an extensive analysis of the update methods for the perceptron with approximate search. We demonstrated that, despite access to richer features, plain early updates (Collins and Roark, 2004) are not sufficiently strong to outperform an arc-factored baseline. The reason is that early updates discard too much training data during learning. While max-violation and latest updates (Huang et al., 2012) manage to outperform the local baseline, they too do not make proper use of all available training data. We therefore additionally compared to LaSO (Daumé III and Marcu, 2005) and DLaSO (Björkelund and Kuhn, 2014) updates. Here we saw that, although both update methods outperform the baseline, DLaSO yields better results. We argued that the problem with LaSO updates is that feedback is provided within training instances, which differentiates training time from test time in a very artificial way. In an analysis, we showed that LaSO tends to make fewer updates and create longer distance arcs in the latent trees, and effect that we ascribe to receiving feedback within instances.

Chapter 4

Transition-based Dependency Parsing

4.1 Introduction

In the previous chapter, we saw how latent trees provide a strong basis for learning a coreference resolver. We now turn to dependency syntax, which has become probably the most widely used syntactic framework in the computational linguistics community. Not only is dependency parsing, i.e., deriving a dependency-syntactic representation of a sentence, a very prominent NLP task in the scientific community, but dependency parsers are also utilized in a wide range of scientific and industrial applications as a processing tool, either as a step in an NLP pipeline, or as a means to derive statistics from large automatically parsed corpora.

Our work is couched in the context of *transition-based* dependency parsing, where a dependency tree is constructed incrementally by a sequence of state transitions, processing the words of a sentence from left to right. In the literature, this paradigm is often contrasted with *graph-based* approaches that model the inherent graph structure of a dependency tree. Throughout this chapter we will focus on transition-based parsing, but a brief summary of graph-based methods is provided in the related work (Section 4.6).

Since Yamada and Matsumoto (2003) presented the first transition system for dependency parsing, a wide range of other systems have been presented. Irrespective of system, however, one common issue with these systems is that they exhibit *spurious ambiguity*, i.e., that the same syntactic structure can be derived in multiple ways. Using the terminology we established in Chapter 2, a transition sequence corresponds to a prediction structure (from $\mathcal{Z}$-space), whereas the actual dependency tree is the output struc-

ture (from $\mathcal{Y}$-space). Most previous work has, however, handled spurious ambiguity by defining a single canonical $\mathbf{z}$ to be learned at training time.[1] The oracle functions used to derive such canonical sequences are known in the transition-based parsing literature as **static oracles** (Goldberg and Nivre, 2012, 2013).

In this chapter we explore the possibility of training a transition-based parser with latent transition sequences for one of the most frequently used transition systems. The main motivation for this is that the canonical $\mathbf{z}$'s from previous work are not motivated on any particular grounds and there is no obvious reason to assume that these $\mathbf{z}$'s are the best to use for training. It is thus an open question how latent representations fare both for greedy parsers and beam search parsers. In order to map out the space $\mathcal{Z}_{\mathbf{y}}$ for learning we will need to define the ORACLE function to allow for spurious ambiguity. Coming back to the established terminology in the dependency parsing community, the oracle functions we are seeking have been dubbed **non-deterministic oracles** (Goldberg and Nivre, 2013).

The experimental evaluation will be carried out using the transition system ArcStandard (Nivre, 2004) with Swap (Nivre, 2009), henceforth referred to as SwapStandard. This transition system has been shown to be very competitive (Bohnet and Nivre, 2012; Bohnet et al., 2013). Moreover, it does not impose any constraints on the structural expressivity with respect to the output trees. Specifically, unlike most other transition systems, SwapStandard is able to derive any non-projective arcs in an output dependency tree which makes it attractive from an application point of view. The increase in relative expressivity, however, also leads to an increase in spurious ambiguity, which is a second reason for choosing this system – with more spurious ambiguity, we expect to be in a better position to measure the impact of latent structures in transition-based dependency parsers.

As we will see, the ORACLE function is not as straightforward as it is for coreference resolution. We will provide a careful analysis of spurious ambiguities in the SwapStandard system and then develop two non-deterministic oracles. As a by-product we will also create a static, minimally swapping oracle, resolving a previously open issue (Nivre et al., 2009).

Empirically we will evaluate the system with latent transitions on a range of 10 treebanks with varying degree of non-projectivity, using both a greedy, classifier-based parser and a beam search parser. While the greedy case has previously been explored for other transition systems (Goldberg and Nivre, 2012; Goldberg et al., 2014, *inter alia*; see Section 4.6.3), we are the first to make this comparison for a beam search parser. Interest-

[1] This statement primarily applies to beam search parsers. See Section 4.6.3 for a discussion on dynamic oracles for greedy parsers.

ingly, while latent structure has been demonstrated to be beneficial to greedy parsers – a result we also corroborate with this system – it appears to play a lesser role in the beam search setting, where latent and canonical sequences roughly are on par with each other.

4.2 Transition-based Dependency Parsing

This section provides an introduction to the topic of dependency parsing starting with a high-level task definition of dependency parsing. This definition will also introduce the concept of non-projectivity, a particular property of dependency trees. We will then review transition-based parsing, in particular the systems ArcStandard and its extension SwapStandard. A more general introduction to dependency grammar and dependency parsing can be found in the book by Kübler et al. (2009).

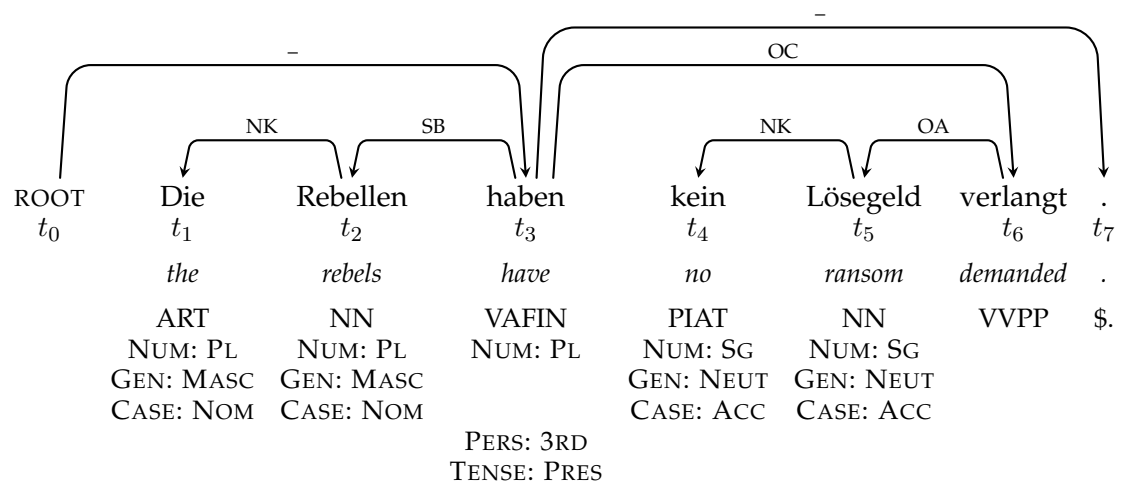
4.2.1 Task Definition

Formally, we define a dependency graph to be a pair $G = \langle S, A \rangle$ where S denotes a sentence, following the definition given in the previous chapter (cf. Section 3.2.1) and $A \subset S \times L \times S$ is a set of arcs, where L denotes the set of arc labels according to the training data. However, the definition of S differs in one regard to what we saw in the previous chapter: In addition to the words of the sentence (numbered t_1 through t_n), we introduce a token t_0 that denotes an artificial ROOT token. A governor-modifier pair $\langle t_i, l, t_j \rangle \in A$ corresponds to an arc between a governor t_i and a modifier t_j with the label $l \in L$. We will sometimes use the more compact notation $t_i \xrightarrow{l} t_j$ to denote the same arc, possibly omitting the label when it is not relevant. To signify the linear order in the input sentence, we will use the $\prec$ symbol. Additionally, we refer to the transitive-reflexive closure of the governor relation as the *dominance* relation and denote by $t_i \xrightarrow{*} t_j$ that t_i dominates t_j, i.e., that there is a (possibly empty, in the case when $t_i = t_j$) directed path from t_i to t_j. Finally, we postulate the following constraints on a dependency graph:

1. **Root constraint**: The first token t_0 of a sentence S is a root and there is no other token $t \in S$ such that $t \rightarrow t_0$;

2. **Connectedness constraint**: The root dominates all tokens of the sentence, i.e., $t_0 \xrightarrow{*} t$ for any $t \in S$;

3. **Single-governor constraint**: Each token in a sentence has at most one governor.



"The rebels have not demanded any ransom."

Figure 4.1: An example dependency tree. The figure shows the output structure $\mathbf{y}$ as a tree on top and the input $\mathbf{x}$ underneath. Our translation is also included underneath, both token-wise and a translation of the full sentence. The input also includes atomic features on the tokens, such as part-of-speech tags.

It is easy to see that a dependency graph G that subscribes to the above properties is, in fact, a tree (as a consequence of constraints 2. and 3.). While there are some theories for dependency syntax that do not require dependency graphs to be trees, the predominant approach in the literature, as well as the approach we are taking, make this assumption. We will therefore adopt the term *dependency tree* for the remainder of this dissertation.

To put these definitions into context, consider the example dependency tree in Figure 4.1. The example is taken from the German TIGER corpus (Brants et al., 2002) after conversion to dependencies (Seeker and Kuhn, 2012). The figure shows the input words connected by arcs above. The constraints we impose can easily be confirmed – (1) The first token is ROOT and it has no incoming arcs; (2) There is a directed path from ROOT to every other token in the sentences; and (3) every token except the root has exactly one incoming arc, i.e., its governor. A concrete example of the dominance relation is how *haben* dominates *Die*, i.e., $t_3 \overset{*}{\rightarrow} t_1$ through the arcs $(t_3 \overset{\text{SB}}{\rightarrow} t_2, t_2 \overset{\text{NK}}{\rightarrow} t_1)$.

The example in Figure 4.1 also contains some examples of atomic token-level features from the input $\mathbf{x}$, e.g., part-of-speech tags and some morphological features. For instance, *Die* has the part-of-speech tag ART, for article, whereas the main verb *haben* is tagged as VAFIN, for finite auxiliary verb. Some examples of the additional morphological features are the gender annotations on the nouns and determiners (masculine and neuter);

grammatical case on the nouns and determiners (nominative and accusative); as well as person and tense on the main verb (third person, present tense).

In general, we define the input structure $\mathbf{x}$ following the token definition from Section 3.2.1, where the token is equipped with a set of feature values $f_1, ..., f_k$. In this case this includes the part-of-speech tags and morphological features (which are typically treebank- and language-specific). Similarly, a sentence S is a set of tokens, and this is the full input, i.e., $\mathbf{x} = S$. Following the definition of dependency trees above, the expected output is a set of arcs, i.e., $\mathbf{y} = A$ where A is the set of arcs spanning the words of the input $\mathbf{x}$.

4.2.2 Non-projectivity

A common challenge for parsing algorithms is the treatment of non-projective dependency trees. Informally, a dependency tree is non-projective if it cannot be drawn without crossing arcs in the plane above the tokens.[2] More formally, projectivity is first and foremost a property of an arc (in context, i.e., in a tree). A dependency tree is said to be projective if all its arcs are projective, otherwise it is non-projective. A dependency arc $h \rightarrow d$, where $h \prec d$, is projective if

$$\forall t,\ h \prec t \prec d :\ h \overset{*}{\rightarrow} t.$$

The case where $d \prec h$ is analogous. More verbosely, the definition requires that every token occurring between the governor and modifier also is dominated by the governor. This implies that for projective arcs, the subtree under the governing token constitutes a continuous span dominated by that token.

An example of a non-projective tree is displayed in Figure 4.2. The example sentence displays one of the typical linguistic phenomena that cause non-projective arcs, known as wh-movement. The interrogative *what* is a prepositional object of *after*, resulting in an arc between the two tokens. But the arc does not dominate all the intervening tokens and is therefore non-projective.

Non-projectivity has received considerable attention in the literature, both from a linguistic and a mathematical perspective. There are many equivalent definitions of projectivity, see, e.g., Havelka (2007) for a longer discussion. As it turns out, projective trees

[2]The graphical intuition of projectivity is related, but not identical, to the notion of planar graphs from graph theory. Planar graphs in graph theory allow for edges/arcs to be drawn anywhere in the plane, whereas projective dependency tress require the arcs to be drawn *above* the tokens, i.e., the nodes, after they have been laid out in sequential (linear) order.

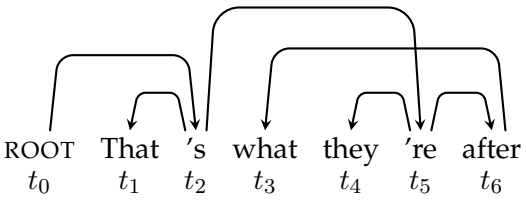


Figure 4.2: A non-projective dependency tree. The arc between t_6 and t_3 is non-projective as t_6 does not dominate any of the intervening tokens.

bear a very close relationship to context-free grammars and they can be shown to be weakly equivalent (Hays, 1964). Subclasses of non-projective arcs and their empirical coverage in annotated data sets have also been analyzed (Kuhlmann and Nivre, 2006; Havelka, 2007). In terms of transition-based parsing, early systems such as ArcStandard (Nivre, 2004) and ArcEager (Nivre, 2003) were entirely restricted to projective dependency trees. More recently several transition systems have been developed that can handle certain subclasses of non-projective trees (Attardi, 2006; Gómez-Rodríguez and Nivre, 2010; Pitler and McDonald, 2015, *inter alia*).

From a linguistic perspective, English typically exhibits rather limited amounts of non-projective arcs, the most prevalent underlying phenomenon being wh-movement. While English has relatively fixed word order, languages where word order is more free, e.g., Czech, German, or Hungarian, tend to allow for more non-projective constructions. In Section 4.4 we provide an empirical analysis of the amount of non-projectivity in the data sets we use. This analysis suggests that non-projective constructions are indeed particularly infrequent in English and that they are more prevalent in the other languages mentioned above.

4.2.3 Transition System

We now turn to the transition systems ArcStandard and SwapStandard. Both systems process an input sentence sequentially from left to right. A state $s = (\Sigma, B, A)$ comprises three data structures: an input buffer, denoted B, a stack of partially processed tokens, Σ, and a set of arcs constructed so far, A. In the initial state, the input tokens are all stored in linear order on the buffer, the stack consists of the ROOT, and the arc set is empty. That is, $s_0 = ([t_0], [t_1, t_2, ...], \emptyset)$. A terminal state is a state where the buffer is empty and the stack contains only the ROOT. With respect to the framework from Chapter 2, the *decisions* are the *transitions* of the system which operate on a state and return a new state. The simplest transition is Shift which moves a tokens from the buffer onto the stack. Arcs

Transition				Preconditions
Shift	$(\sigma, b_0\|\beta, A)$	$\Rightarrow$	$(\sigma\|b_0, \beta, A)$	
LeftArc$_L$	$(\sigma\|u_1\|u_0, B, A)$	$\Rightarrow$	$(\sigma\|u_0, B, A \cup \{u_0 \overset{L}{\rightarrow} u_1\})$	$u_1 \neq t_0$
RightArc$_L$	$(\sigma\|u_1\|u_0, B, A)$	$\Rightarrow$	$(\sigma\|u_1, B, A \cup \{u_1 \overset{L}{\rightarrow} u_0\})$	
Swap	$(\sigma\|u_1\|u_0, B, A)$	$\Rightarrow$	$(\sigma\|u_0, u_1\|B, A)$	$u_1 \neq t_0 \wedge u_1 \prec u_0$

Figure 4.3: Transitions of the ArcStandard and SwapStandard systems. The Swap transition is only used by the SwapStandard system.

can be constructed between the two topmost tokens on the stack through the transitions LeftArc and RightArc. The only difference between the ArcStandard and SwapStandard is the addition of a fourth transition, Swap, that can be used to reorder the input tokens. This enables SwapStandard to construct non-projective arcs.

A schematic view of the transitions is shown in Figure 4.3. The operator | delineates items from the buffer and stack, and is taken to be right- and left-associative, respectively. That is, a stack $\sigma|u_1|u_0$, denotes a stack with the topmost item u_0, second top-most u_1, and an arbitrarily-sized tail σ. Similarly, a buffer $b_0|\beta$ denotes a buffer with the first item b_0, and an arbitrarily-sized tail β. The table has the three common transitions at the top, and the Swap transition underneath in order to emphasize the distinction between the two systems. Note that the LeftArc and RightArc transitions are parametrized for arc labels, i.e., they each correspond to a family of transitions that share the same semantics except for the label they introduce. The LeftArc transition has a special precondition, requiring the second top-most item on the stack *not to be* t_0, i.e., the ROOT. If LeftArc would be permitted in such a state, it would attach ROOT as a modifier, violating the root constraint, i.e., that ROOT has no governor. The Swap transition removes the second top-most item from the stack and inserts it as the front of the buffer. This transition comes with two preconditions: First, it cannot be applied if the ROOT is the second top-most item in the stack. Since arcs originating at ROOT always are projective, it never makes sense to swap this token. Second, Swap is not allowed if the two top-most tokens on the stack are not in their linear order. This second precondition prevents the system from entering a loop where tokens are repeatedly swapped back and forth between the stack and buffer using intermediate Shift transitions.

As for complexity, the ArcStandard system requires $2n$ transitions for a sentence of n tokens. This is easy to realize: Every token will be shifted onto the buffer once, and attached to a governor through a LeftArc or RightArc once. In the SwapStandard system the premise is the same, except that Swap needs to be taken into consideration as well. In the simplest case, SwapStandard parses a sentence without applying a single Swap

transition. Then the situation is the same as for ArcStandard. When a Swap transition is applied, however, this causes a token to move back from the stack onto the buffer, requiring an additional Shift. In particular, for the SwapStandard system the number of transitions required is $2n + 2k$, where k denotes the number of swaps. Theoretically the number of swaps can be quadratic in the length of the sentence, potentially rendering the transition system non-linear in terms of sentence length. Nivre (2009) however shows that in practice this does not occur and that the system empirically behaves linearly with respect to sentence length.

4.3 Oracles and Ambiguity

We now turn to the question of the oracle function. Traditionally, transition-based parsers have been trained using what has come to be called a *static oracle*. A static oracle derives a single transition sequence for a given sentence. Such a derivation can be carried out "offline", before training commences. Recall, however, that such an oracle is not sufficient for our means – as we need to capture the spurious ambiguity of the system. That is, we need to define a *non-deterministic oracle* that captures all the possible options during a derivation.

Nevertheless, this section starts out by defining static oracles, since they also constitute the basis for non-deterministic oracles. Moreover, static oracles will act as the baseline in the experimental evaluation. We will then continue with mapping out the spurious ambiguities in the SwapStandard system, followed by the definition of a set of non-deterministic oracles. Finally, we will define an optimally swapping static oracle.

Throughout this chapter we will be using the terms *permissible* and *applicable* transitions. Permissible has the same meaning as the function PERMISSIBLE from Chapter 2, i.e., it returns all transitions that can be applied in a given state, irrespective of whether they can derive the correct output structure. Applicable transitions are the subset of those that can derive a given output structure. That is, they correspond to the ORACLE function from Chapter 2.

4.3.1 Static Oracles

Recall that the job of the oracle is to create a sequence of transitions that can derive a given dependency tree. To this end, an oracle takes a current state and a dependency tree corresponding to the sentence as input and returns an applicable transition. Repeatedly applying this oracle, starting from an initial state until it reaches a terminal one will result

Algorithm 4.1 Generic static oracle

Input: State s, dependency tree $\mathbf{y}$

Output: The next applicable transition, given s and $\mathbf{y}$

```
1: function NEXTTRANSITION(s, y)
2:     if CANLEFTARC(s, y) then
3:         return LeftArc
4:     else if CANRIGHTARC(s, y) then
5:         return RightArc
6:     else if CANSWAP(s, y) then
7:         return Swap
8:     else
9:         return Shift
```

in a transition sequence for a given sentence and tree. A generic view on static oracles is given in Algorithm 4.1. A number of predicates are sequentially tested in a pre-defined order, and the action matching the first true predicate is returned. If all else fails, Shift is the fallback option. The semantics of the predicates for LeftArc and RightArc are rather straightforward: if the two top tokens should be connected through an arc *and* the modifier among the two has already collected all its own modifiers, then the corresponding transition can be applied.

We will shortly turn to the Swap transition, but we first note that if the oracle is applied to a projective tree, the test for Swap (lines 6 and 7) can be skipped. In this case, the oracle is de facto an oracle for the ArcStandard system. An application of the oracle to the example sentence from Figure 4.1 is shown in Figure 4.4. To ease the interpretation of the derivation, the figure uses a slightly different notation for tokens on the stack and buffer, where the surface forms of the words are used and their indices in the sentence are used as subscripts. Every row denotes a state, and the stack and buffer on each row correspond to the state *after* application of the transition on the same row.

Testing for Swap is a bit more complicated and requires an auxiliary ordering of the input tokens called the **projective order**. The projective order is obtained by making an in-order traversal over the nodes in the dependency tree, and we will denote this ordering by $\prec_{\text{PROJ}}$. If the tokens in the input sentence are rearranged according to the projective order, the corresponding dependency tree becomes projective. For instance, the non-projective example from Figure 4.2 is given in Figure 4.5 with the tokens reordered according to the projective order.

Nivre (2009) defines Swap to be applicable when the two top-most tokens on the

State	Transition	Stack	Buffer	Arc Added
s_0		[ROOT$_0$]	[Die$_1$, ..., .$_7$]	
s_1	Shift	[ROOT$_0$, Die$_1$]	[Rebellen$_2$, ..., .$_7$]	
s_2	Shift	[ROOT$_0$, Die$_1$, Rebellen$_2$]	[haben$_3$, ..., .$_7$]	
s_3	LeftArc$_{\text{NK}}$	[ROOT$_0$, Rebellen$_2$]	[haben$_3$, ..., .$_7$]	Rebellen$_2 \overset{\text{NK}}{\rightarrow}$ Die$_1$
s_4	Shift	[ROOT$_0$, Rebellen$_2$, haben$_3$]	[kein$_4$, ..., .$_7$]	
s_5	LeftArc$_{\text{SB}}$	[ROOT$_0$, haben$_3$]	[kein$_4$, ..., .$_7$]	haben$_3 \overset{\text{SB}}{\rightarrow}$ Rebellen$_2$
s_6	Shift	[ROOT$_0$, haben$_3$, kein$_4$]	[Lösegeld$_5$, ..., .$_7$]	
s_7	Shift	[ROOT$_0$, ..., kein$_4$, Lösegeld$_5$]	[verlangt$_6$, .$_7$]	
s_8	LeftArc$_{\text{NK}}$	[ROOT$_0$, haben$_3$, Lösegeld$_5$]	[verlangt$_6$, .$_7$]	Lösegeld$_5 \overset{\text{NK}}{\rightarrow}$ kein$_4$
s_9	Shift	[ROOT$_0$, ..., Lösegeld$_5$, verlangt$_6$]	[.$_7$]	
s_{10}	LeftArc$_{\text{OA}}$	[ROOT$_0$, haben$_3$, verlangt$_6$]	[.$_7$]	verlangt$_6 \overset{\text{OA}}{\rightarrow}$ Lösegeld$_5$
s_{11}	RightArc$_{\text{OC}}$	[ROOT$_0$, haben$_3$]	[.$_7$]	haben$_3 \overset{\text{OC}}{\rightarrow}$ verlangt$_6$
s_{12}	Shift	[ROOT$_0$, haben$_3$, .$_7$]	[ ]	
s_{13}	RightArc$_{-}$	[ROOT$_0$, haben$_3$]	[ ]	haben$_3 \overset{-}{\rightarrow}$.$_7$
s_{14}	RightArc$_{-}$	[ROOT$_0$]	[ ]	ROOT$_0 \overset{-}{\rightarrow}$ haben$_3$

Figure 4.4: Oracle derivation for parsing the sentence from Figure 4.1.

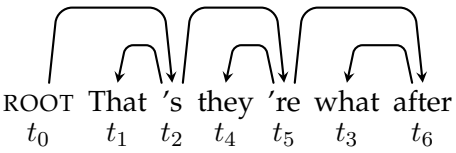


Figure 4.5: The dependency tree from Figure 4.2 with tokens reordered according to the projective order.

stack are in reverse projective order. That is, when $u_0 \prec_{\text{PROJ}} u_1$. In the example from Figure 4.2, this means that Swap is applicable when u_0 is either of *they* or *'re* and u_1 is *what*. An example derivation using Algorithm 4.1 on the sentence from Figure 4.2 is shown in Figure 4.6. The derivation starts out by applying two Shift and then a LeftArc, constructing the first arc between *That* and *'s*. It then continues to Shift *what* onto the stack. However, since *what* belongs further to the right in the projective order, the system now needs to start using the Swap transition to move *what* closer to its governor *after*. This is accomplished by a sequence of Shift and Swap transitions, after which *what* can be attached to its governor.

While this oracle is able to produce a transition sequence for any non-projective sentence, it is very generous with the amount of Swap transitions. In fact, the example sentence just seen can be successfully parsed using only a single Swap transition. The primary observation is that the oracle applies Swap as soon as it is applicable, even though a Shift transition may also be applicable. We therefore call this oracle EAGER. Drawing

State	Transition	Stack	Buffer	Arc Added
s_0		[ROOT$_0$]	[That$_1$, ..., after$_6$]	
s_1	Shift	[ROOT$_0$, That$_1$]	['s$_2$, ..., after$_6$]	
s_2	Shift	[ROOT$_0$, That$_1$, 's$_2$]	[what$_3$, ..., after$_6$]	
s_3	LeftArc$_{\text{NSUBJ}}$	[ROOT$_0$, 's$_2$]	[what$_3$, ..., after$_6$]	's$_2$ $\overset{\text{NSUBJ}}{\rightarrow}$ That$_1$
s_4	Shift	[ROOT$_0$, 's$_2$, what$_3$]	[they$_4$, ..., after$_6$]	
s_5	Shift	[ROOT$_0$, ..., what$_3$, they$_4$]	['re$_5$, after$_6$]	
s_6	Swap	[ROOT$_0$, 's$_2$, they$_4$]	[what$_3$, ..., after$_6$]	
s_7	Shift	[ROOT$_0$, ..., they$_4$, what$_3$]	['re$_5$, after$_6$]	
s_8	Shift	[ROOT$_0$, ..., what$_3$, 're$_5$]	[after$_6$]	
s_9	Swap	[ROOT$_0$, ..., they$_4$, 're$_5$]	[what$_3$, after$_6$]	
s_{10}	LeftArc$_{\text{NSUBJ}}$	[ROOT$_0$, 's$_2$, 're$_5$]	[what$_3$, after$_6$]	're$_5$ $\overset{\text{NSUBJ}}{\rightarrow}$ they$_4$
s_{11}	Shift	[ROOT$_0$, ..., 're$_5$, what$_3$]	[after$_6$]	
s_{12}	Shift	[ROOT$_0$, ..., what$_3$, after$_6$]	[ ]	
s_{13}	LeftArc$_{\text{POBJ}}$	[ROOT$_0$, ..., 're$_5$, after$_6$]	[ ]	after$_6$ $\overset{\text{POBJ}}{\rightarrow}$ what$_3$
s_{14}	RightArc$_{\text{PREP}}$	[ROOT$_0$, 's$_2$, 're$_5$]	[ ]	're$_5$ $\overset{\text{PREP}}{\rightarrow}$ after$_6$
s_{15}	RightArc$_{\text{CCOMP}}$	[ROOT$_0$, 's$_2$]	[ ]	's$_2$ $\overset{\text{CCOMP}}{\rightarrow}$ 're$_5$
s_{16}	RightArc$_{\text{ROOT}}$	[ROOT$_0$]	[ ]	ROOT$_0$ $\overset{\text{ROOT}}{\rightarrow}$'s$_2$

Figure 4.6: Example oracle derivation for the dependency tree from Figure 4.2 using the EAGER oracle.

upon this observation, Nivre et al. (2009) present an improved oracle that considerably reduces the number of swaps. Their oracle is based on the same basic structure as the one in Algorithm 4.1, but the semantics of CANSWAP are redefined. The oracle relies on a notion of **maximal projective components** (MPCs). In addition to requiring that u_0 and u_1 appear in the wrong order with respect to the projective order, this oracle also requires u_0 and b_0 not to be part of the same MPCs. MPCs are defined as the connected components, i.e., the resulting subtrees obtained by running the oracle parser without Swap until it hits a dead end. Nivre et al. (2009) show empirically that this oracle substantially reduces the number of Swap transitions, sometimes by up to 80%. The corresponding derivation for the non-projective example from Figure 4.2 is listed in Figure 4.7. This oracle does not admit the first Swap transition from EAGER. Instead, it continues by shifting *'re* and then attaching *they* through a LeftArc. That is, the use of the Swap transition is postponed and the two intervening tokens between the modifier and governor of the non-projective arc are reduced to a subtree, removing the need for one extra Swap and Shift. As this oracle tries to postpone Swap transitions when possible, we will refer to it as LAZY.

4.3.2 Spurious Ambiguity

Although the LAZY oracle is much more efficient than EAGER, Nivre et al. (2009) note that there are still situations where it uses more Swap transitions than necessary. We

State	Transition	Stack	Buffer	Arc Added
s_0		[ROOT$_0$]	[That$_1$, ..., after$_6$]	
s_1	Shift	[ROOT$_0$, That$_1$]	['s$_2$, ..., after$_6$]	
s_2	Shift	[ROOT$_0$, That$_1$, 's$_2$]	[what$_3$, ..., after$_6$]	
s_3	LeftArc$_{\text{NSUBJ}}$	[ROOT$_0$, 's$_2$]	[what$_3$, ..., after$_6$]	's$_2$ $\overset{\text{NSUBJ}}{\rightarrow}$ That$_1$
s_4	Shift	[ROOT$_0$, 's$_2$, what$_3$]	[they$_4$, ..., after$_6$]	
s_5	Shift	[ROOT$_0$, ..., what$_3$, they$_4$]	['re$_5$, after$_6$]	
s_6	Shift	[ROOT$_0$, ..., they$_4$, 're$_5$]	[after$_6$]	
s_7	LeftArc$_{\text{NSUBJ}}$	[ROOT$_0$, ..., what$_3$, 're$_5$]	[after$_6$]	're$_5$ $\overset{\text{NSUBJ}}{\rightarrow}$ they$_4$
s_8	Swap	[ROOT$_0$, 's$_2$, 're$_5$]	[what$_3$, after$_6$]	
s_9	Shift	[ROOT$_0$, ..., 're$_5$, what$_3$]	[after$_6$]	
s_{10}	Shift	[ROOT$_0$, ..., what$_3$, after$_6$]	[ ]	
s_{11}	LeftArc$_{\text{POBJ}}$	[ROOT$_0$, ..., 're$_5$, after$_6$]	[ ]	after$_6$ $\overset{\text{POBJ}}{\rightarrow}$ what$_3$
s_{12}	RightArc$_{\text{PREP}}$	[ROOT$_0$, 's$_2$, 're$_5$]	[ ]	're$_5$ $\overset{\text{PREP}}{\rightarrow}$ after$_6$
s_{13}	RightArc$_{\text{CCOMP}}$	[ROOT$_0$, 's$_2$]	[ ]	's$_2$ $\overset{\text{CCOMP}}{\rightarrow}$ 're$_5$
s_{14}	RightArc$_{\text{ROOT}}$	[ROOT$_0$]	[ ]	ROOT$_0$ $\overset{\text{ROOT}}{\rightarrow}$'s$_2$

Figure 4.7: Example oracle derivation for the dependency tree from Figure 4.2 using the LAZY oracle

will shortly resolve this issue and present an optimal oracle that requires the minimum number of Swap transitions, but before that we carry out a careful analysis of spurious ambiguities, i.e., the situations where more than one transition is applicable.

By design, the oracle algorithm selects among the possible transitions using a predefined order, i.e., the order of the tests in the if-clauses (cf. Algorithm 4.1). This procedure yields a deterministic sequence of transitions for any dependency tree. This sequence is not necessarily the only correct one for a given dependency tree. In fact, most dependency trees seen in standard treebanks can be derived from more than one sequence of transitions. Different sequences always start and end with the same transitions, however, somewhere along the way a fork point occurs where more than one transition can be applied while still keeping the desired dependency tree reachable. We call these fork points *spurious ambiguities* as they all lead to the same tree. We have already seen one such ambiguity above when we compared EAGER and LAZY, namely the Shift-Swap ambiguity. Below we review the remaining spurious ambiguities of the ArcStandard and SwapStandard systems and give examples of each kind.

Shift-LeftArc ambiguity. While the canonical way of constructing the dependency tree is to attach left modifiers as early as possible, these decisions can sometimes be delayed. Consider the example sentence on the left in Figure 4.8 and the main verb *likes*. Here, the left modifier of *likes* need not be attached before the right. The example thus has two possible transition sequences that create the given dependency tree. To keep figures

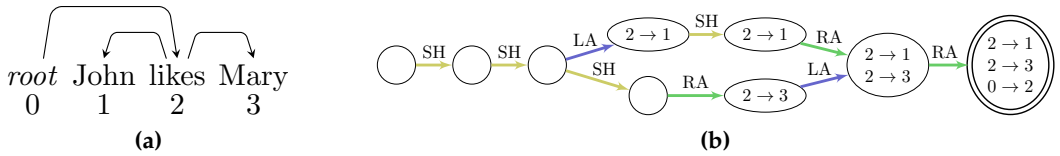


Figure 4.8: A dependency tree which exhibits the Shift-LeftArc ambiguity in (a) and a graph that encodes the two alternative transition sequences in (b).

concise, we illustrate these sequences in the form of a search graph (and not a search tree), as depicted on the right in Figure 4.8. Nodes in the graph correspond to parser states and arcs between them to transitions (color-coded for different transitions). The initial state is on the left, and the terminal state is on the right (indicated by a double circle). The text in the nodes indicate the arcs that have been constructed thus far.[3] The Shift-LeftArc ambiguity gives rise to the fork point where there are two parallel paths, corresponding to early and late attachments of the left modifier. In both cases the initial transitions (two shifts) as well as the last transition (the RightArc for the root attachment) is the same for both sequences. This ambiguity is not specific to SwapStandard, but also occurs in the projective ArcStandard system. In the projective case, this type of ambiguity always arises when the parser can make a LeftArc attachment but b_0 is dominated by u_0.

Shift-RightArc ambiguity. While the Shift-RightArc ambiguity is not possible in the plain ArcStandard system the introduction of Swap enables this ambiguity. In the ArcStandard system, applying a Shift when an RightArc is applicable results in "burying" tokens on the stack such that they are irretrievable and cannot be attached to their modifiers. Since the Swap transition moves tokens out of the stack and back onto the buffer, it is sometimes possible to recover these buried tokens and thus do the RightArc at a later point.

Consider the German sentence in Figure 4.9. The static oracle would parse this sentence by first applying three Shift followed by a RightArc, attaching *sie* to *wurde*. However, the projective order of this sentence is *Ausgelöst* $\prec_{\text{Proj}}$ *durch* $\prec_{\text{Proj}}$ *Intel* $\prec_{\text{Proj}}$ *wurde* $\prec_{\text{Proj}}$ *sie*. The system is thus able to delay the RightArc transition, do another Shift and then swap *wurde* and *sie* past *durch* and handle this attachment later. The graph in Figure 4.10 shows all ambiguities for this sentence. The first Shift-RightArc ambiguity is highlighted by a red

[3] While the nodes only display the arc set so far, the merge points in the graph truly correspond to equivalent states where the stack, buffer, and arc set are all identical. However, the scores of different paths through the graph to the same state may differ.

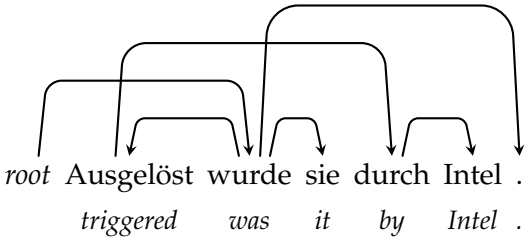


Figure 4.9: A non-projective German sentence. The arc from *Ausgelöst* to *durch* is non-projective since *Ausgelöst* does not dominate the intervening tokens.

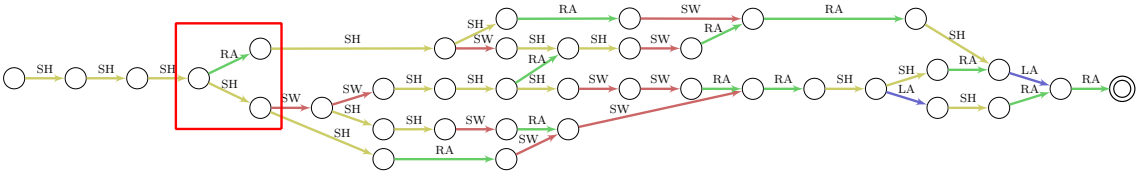


Figure 4.10: A graph encoding the different transition sequences for the sentence from Figure 4.9.

box. To keep the figure reasonably sized, the arcs obtained are omitted from the nodes in the graph.

In comparison to the Shift-LeftArc ambiguity seen earlier, the Shift-RightArc ambiguity always relies on additional Swap transitions and thus leads to longer transition sequences. Note that the same logic also holds for LeftArc – sometimes both the governor and modifier of a LeftArc transition can be swapped out, creating a second kind of Shift-LeftArc ambiguity in addition to the one we discussed previously.

4.3.3 Non-deterministic Oracles

So far we have only discussed how to create canonical sequences using static oracles. In order to construct latent transition sequences we have to create a non-deterministic oracle. So far we have seen three types of spurious ambiguities. It is easy to convince oneself that no other ambiguities are possible – by sheer enumeration of the possible pairs of transitions, any other ambiguities would involve one of the pairs LeftArc-RightArc, LeftArc-Swap, or RightArc-Swap. A LeftArc-RightArc ambiguity would only be possible if u_0 is the governor of u_1 and u_1 is the governor of u_0 which implies that the tree has a cycle (and is therefore not a tree; thus a contradiction). The LeftArc-Swap and RightArc-Swap ambiguities would swap a modifier past its governor, or a governor past its modifier. This would violate the projective order and is therefore also not allowed.

Algorithm 4.2 Shift applicable

Input: State s, dependency tree $\mathbf{y}$

Output: `true` if Shift is applicable, `false` otherwise

```
1:  function CanShift(s, y)
2:      if |s.B| = 0 then                          ▷ Not possible if buffer is empty
3:          return false
4:      s = ApplyShift(s)                          ▷ Initial shift
5:      while ¬Terminal(s) do                      ▷ Loop until terminal state or dead end
6:          if CanLeftArc(s, y) then
7:              s = ApplyLeftArc(s)
8:          else if CanRightArc(s, y) then
9:              s = ApplyRightArc(s)
10:         else if CanSwap_Eager(s, y) then
11:             s = ApplySwap(s)
12:         else if |s.B| > 0 then                 ▷ Shift if buffer is not empty
13:             s = ApplyShift(s)
14:         else
15:             return false                       ▷ Hit dead end
16:     return true
```

The key question that needs to be resolved in order to construct a non-deterministic oracle is when a Shift transition can be applied. The static oracle defines when the other three transitions can be applied, but treats Shift as a fallback when no other transitions are possible. So when is Shift applicable? One way to find out is to try a Shift and see if there is any way to recover the full parse. This procedure is described in Algorithm 4.2. The algorithm applies an initial shift (line 4) and uses the static EAGER oracle from then on (lines 5 to 15). If the parser can recover the correct parse, then Shift is applicable. If not, the parser will eventually end up in a dead end where the buffer is empty, the stack contains several items, but no other transitions are applicable (line 14).

The worst case runtime of this algorithm is $O(n^2)$, although it is enough to halt the search when the stack has been reduced to only two tokens (one being root), since from that point on the gold tree has to be recoverable. Although this might seem costly since the procedure will be called many times within a single sentence, in practice we observed that applying Algorithm 4.2 during training had negligible effect on overall training time.

The reason this algorithm works is that after applying the initial Shift, it prefers all the other transitions over additional Shift transitions. The other transitions are all taking clear steps towards avoiding dead ends, either by introducing arcs (removing tokens from the system) or by applying swaps (permuting the words in the direction of the projective

order by moving tokens back from the stack onto the buffer).

The procedure outlined in Algorithm 4.2 allows us to construct a non-deterministic oracle for the SwapStandard system. Specifically, whenever either of LeftArc, RightArc, or Swap are applicable, the oracle also checks if Shift is applicable and returns the *set* of all applicable transitions. If neither of LeftArc, RightArc, or Swap are applicable, then a set containing only Shift is returned. This oracle allows for all possible ambiguities and we refer to it as ND-ALL.

Since the LeftArc and RightArc transitions introduce new arcs, thereby adding more structure that can be used for feature extraction, it could be argued that a parser could profit from having a more eager treatment of arc attachments and that the Shift-LeftArc and Shift-RightArc ambiguities should be avoided. As for the Shift-Swap ambiguity, we have seen that there is a continuum of how eagerly Swap transitions should be applied, ranging from EAGER to LAZY (and beyond, as we will see shortly). The static oracles apply Swap according to the predefined rules that are primarily grounded in tree structural characteristics. These rules may not be the most motivated from a linguistic perspective, and there could be a more systematic treatment of swaps lying somewhere in between that is easier to learn. In order to investigate whether the parser is able to learn better such patterns latently, we also construct an oracle that only permits the Shift-Swap ambiguity but no others. We will refer to this oracle as ND-SW.

4.3.4 Optimally Swapping Static Oracle

We previously discussed how the LAZY oracle considerably reduces the number of Swap transitions compared to EAGER. But this oracle still does not always yield the minimal number of swaps for a given dependency tree. Given the possibility to tell when a Shift is applicable (Algorithm 4.2), we can construct search graphs as those shown above (Figures 4.8 and 4.10) for any dependency tree. By searching such a graph for the shortest path from the initial state to the terminal state, the shortest possible transition sequence can be found. As this oracle minimizes the number of Swap transitions, we refer to it as MINIMAL.[4]

This procedure cannot be formalized as concisely as Algorithm 4.1 and relies on searching the corresponding graph. But it should be noted that when a static oracle is used for training, the transition sequence for each sentence only needs to be computed once before training.

[4]It should be noted that in certain cases the number of Swap transitions can be reduced even further by swapping tokens that are already in the projective order. The MINIMAL oracle ensures that the number of Swap transitions is minimal while still respecting the projective order.

The graphs can grow extremely large, to the point where the graph of a single sentence cannot be kept in main memory.[5] A depth-first search that keeps a stack of fork points in memory circumvents this problem. After reaching the terminal state the first time, only the path chosen and the number of transitions need to be remembered. Any further paths that have not reached the terminal state after as many transitions as the currently seen shortest path can be terminated immediately. While this oracle is clearly slower than the other static ones, we found that the extra overhead is negligible in comparison to overall training time.

4.4 Experimental Setup

This section will outline the experimental environment used for evaluation. We start by describing data sets and evaluation metrics. We then continue by discussing the instantiation of the parser using the framework from Chapter 2.

4.4.1 Data Sets

We evaluate the oracles on ten treebanks. Specifically, we use the nine treebanks from the SPMRL 2014 Shared Task (Seddah et al., 2014), comprising Arabic, Basque, French, German, Hebrew, Hungarian, Korean, Polish, and Swedish. For these treebanks we use the splits into training, development and test sets provided by the Shared Task organizers. Additionally, we use the English Penn Treebank (Marcus et al., 1993) converted to Stanford dependencies (de Marneffe et al., 2006) with the standard split, sections 2-21 for training, section 24 for development, and section 23 for test.

A breakdown of the characteristics of the training sets of each treebank is shown in Table 4.1. The table includes the total number of sentences and the percentage of projective sentences. It also shows the total number of swap transitions required by EAGER, and the reduction of swaps of LAZY and MINIMAL relative to EAGER. For instance, in the Arabic treebank 97.32% of the sentences are projective and the LAZY and MINIMAL reduce the number of swaps by 80.59% and 80.79%, respectively. For about half the treebanks LAZY is already minimal and we exclude MINIMAL from the empirical evaluation.

Korean is the only strictly projective treebank, although some of the treebanks have very few non-projective arcs in their training sets, particularly Hebrew and French. This means that the number of Shift-Swap ambiguities considered by the non-deterministic

[5] Even with 256gb of main memory we were unable to keep some of the graphs of the training sets in memory despite an efficient implementation.

	# Sentences	% Projective Sentences	# EAGER Swaps	% Reduced Swap (LAZY)	% Reduced Swap (MINIMAL)	% Sentences w/ unique Transition sequences
Arabic	15,762	97.32%	6,481	80.59%	80.79%	9.94%
Basque	7,577	94.71%	984	53.46%	–	1.06%
English	39,832	99.90%	146	71.92%	–	1.31%
German	40,472	67.23%	155,041	75.09%	83.88%	7.81%
French	14,759	99.97%	6	16.67%	–	2.66%
Hebrew	5,000	99.82%	12	8.33%	–	2.82%
Hungarian	8,146	87.75%	3,654	51.07%	54.24%	10.25%
Korean	23,010	100%	0	–	–	0.27%
Polish	6,578	99.54%	91	59.34%	–	10.57%
Swedish	5,000	93.62%	2,062	75.90%	77.79%	7.28%

Table 4.1: Data set statistics from the training sets showing the amount of non-projectivity in the treebanks, how many swaps the different oracles use, and the proportion of sentences where there is no spurious ambiguity.

oracles during training is extremely small. The ND-SW oracle thus exhibits a very tiny amount of spurious ambiguity in these cases. Nevertheless, ND-ALL will still consider the Shift-LeftArc ambiguity. The last column of Table 4.1 gives the percentage of sentences that exhibit no spurious ambiguity under the ND-ALL oracle (or, equivalently, the percentage that has a single unique transition sequence). This fraction ranges between almost 0% and up to about 10%, which means that there are indeed plenty of spurious ambiguities in the training data.

We adopt a realistic evaluation setting and use predicted part-of-speech tags and morphological features. Specifically, we use MarMoT (Mueller et al., 2013), a state-of-the-art CRF tagger that jointly predicts part-of-speech tags and morphology. We train the parsers on 10-fold jackknifed training data. For the development and test sets the tagger is trained on the full training set.

4.4.2 Evaluation Metrics

The evaluation metric we use is the Labeled Attachment Score (LAS). This is computed as the percentage of tokens that received the correct governor and arc label. While there is also an unlabeled version of this metric, where only the tree structure counts, we believe this is not an appropriate metric since it ignores part of the output structure.

4.4.3 Instantiation

We implement a parser based on SwapStandard using the framework from Chapter 2. Some of the functions required by the framework have partially been covered earlier in

this chapter. In particular, PERMISSIBLE and ORACLE have already been discussed at length earlier. PERMISSIBLE is defined as those transitions that satisfy the preconditions shown in Figure 4.3. As for ORACLE, we have defined number of both static and non-deterministic oracles in the previous section. We should, however, remind the reader that the transitions LeftArc and RightArc each have stood in place for a family of transitions, each corresponding to a different arc label. Whenever either of these arc-inducing transitions are permissible, the whole family is returned by PERMISSIBLE. For ORACLE, obviously only the arc-inducing transition with the correct arc label is returned.[6]

The state representation we use is straightforward. It implements the abstract requirements from Chapter 2 – score of state, backpointer to previous state, as well as decision (transition) taken. Additionally, they keep a representation of the task-specific data structures – the stack, the buffer, and the current arc set. In practice we did not implement the task-specific data structures in the verbose way as they have been defined here. Rather, we follow Goldberg et al. (2013) for an efficient implementation that allows for constant time copying of the state.

We will experiment with both a greedy and a beam search parser. The greedy parser uses a locally normalized classifier trained using regular perceptron updates (i.e., setting the passive-aggressive weight τ to 1, cf. Section 2.4) and only the next transition is left latent. The beam search parser is trained with the globally normalized structured perceptron using the passive-aggressive updates and a structured loss function (defined below). The beam size is set to 20 and the update method used is max-violation and kept fixed throughout. The number of training epochs was not kept fixed, but rather tuned for individual experiments, to be discussed in the next section.

The last bits of the framework that needs to be discussed is the feature extraction function and the loss function. They are discussed sequentially below.

Feature Extraction. The feature extraction function $\phi(\mathbf{x}, s, d)$ acts on a combination of a state s and transition d and returns a high-dimensional vector representation of this pair, using the information contained in the sentence representation $\mathbf{x}$ as well as the structural context from the current state. As input to the parser, $\mathbf{x}$ is assumed to contain the atomic token-level annotations such as part-of-speech tags and morphological features. ϕ will thus have access to this information when encoding a state-decision pair.

In a broad sense, ϕ will extract information based on the top-most tokens in the stack as well as the front of the buffer. In addition, it can also access tokens in the partial sub-

[6]We remind the reader again that for the sake of reproducibility the implementation used for the experiments is available on the author's website.

trees rooted under the stack tokens and, when applicable, partial subtrees of the buffer tokens.[7] For instance, the surface forms, part-of-speech tags, and morphological features on the three top-most tokens on the stack as well as the three first items on the buffer might be extracted. Each position in stack and buffer of course corresponds to different feature templates in order to distinguish the role of these.

We have designed the feature extraction function used in our implementation by hand although most templates have been drawn from previous work (Zhang and Nivre, 2011; Bohnet and Nivre, 2012; Bohnet et al., 2013). Additionally, manual experimentation over a large number of data sets (including, but not limited to, the ones used for evaluation in this chapter) have yielded additional templates that have been added after empirical experiments. We emphasize that we never evaluated these additional templates on test sets during development. More generally, we also did not do it in a selective manner with respect to data sets, transition system, update methods, or oracles. Rather, they are the result of quasi-random explorations over a long period of time as the implementation was developed. We thus argue that the feature extraction function should not be expected to be particularly biased in favor of any certain combination of the aforementioned configurational parameters.

Loss function. The purpose of the loss function is to reflect the relative (in-)correctness of a (partial) prediction to the correct sequence of transitions. Equation 4.1 shows the loss function we use at a high level. The loss incurred can be broken down into penalties from the two arc-inducing transitions plus a penalty from swaps.

$$\textsc{Loss}(\tilde{\mathbf{z}}, \hat{\mathbf{z}}) = \textsc{LeftArcLoss}(\tilde{\mathbf{z}}, \hat{\mathbf{z}}) + \textsc{RightArcLoss}(\tilde{\mathbf{z}}, \hat{\mathbf{z}}) + \textsc{SwapLoss}(\tilde{\mathbf{z}}, \hat{\mathbf{z}}) \quad (4.1)$$

The loss for LeftArc transitions is computed as the number of such transitions that created an arc, irrespective of arc label, that are not present in the correct output tree. Additionally, LeftArc transitions that created a correct arc, but with the wrong label, are penalized with one half. The case for RightArc is identical. The motivation for the half penalty is that many features are drawn from the partially constructed dependency tree. An arc connecting a correct governor-modifier pair will thus provide useful context even in the situation where the arc was assigned the wrong label.

[7] Since arc-inducing operations always are carried out on the stack, in the ArcStandard system tokens on the buffer by definition cannot have any modifiers. However, with the Swap transition it is possible that a token that has already collected some modifiers is swapped back onto the buffer.

For Swap, we count the number of false positive and false negative swaps. A false negative is a pair of tokens that were swapped in the correct sequence, but not in the prediction. A false positive is a pair of tokens that were swapped in the prediction, but not in the correct sequence.

Somewhat conspicuously, there is no penalty related to Shift. This can be motivated by the fact that Shift does not actually create any output structure as it only moves tokens between the buffer and stack. More importantly, however, erroneous Shift transitions will one way or another cause subsequent erroneous transitions that will be covered by the loss function. Similarly, these losses do not penalize correct transitions taking place at the wrong place in the sequence. For instance, a RightArc transition can correctly attach a modifier to its governor even though the modifier has not yet collected its own modifiers. Since the transition will remove the modifier from the stack, it will no longer be able to attach its modifiers. Similar situations can arise for LeftArc and Swap. The argument here is the same as for not counting Shift towards the loss – these types of mistakes will inevitably incur more loss elsewhere in the sequence.[8]

4.5 Experimental Evaluation

In total we experiment with five different oracles. The three static ones, EAGER, LAZY, and MINIMAL, all use a single unique transition sequence for every sentence in the training data. The two non-deterministic oracles, ND-SW and ND-ALL, create latent transition sequences on the fly relying on the current parameters and may change across training iterations. We carry out these comparisons both for a greedy parser and a beam parser. In the greedy case this has been studied in the past (Goldberg and Nivre, 2012; Goldberg et al., 2014, *inter alia*), but not for the SwapStandard system.

One goal of the evaluation is of course to see if the non-deterministic oracles result in better parsers. An affirmative answer to this question has been given in the greedy case for other systems already (Goldberg and Nivre, 2012; Goldberg et al., 2014). But it is an open question whether this result carries over to the SwapStandard system, and, perhaps more importantly, whether it carries over to the beam search setting. Another important question is how the different static and non-deterministic oracles compare to each other. In the static case, it is tempting to believe that sequences that use fewer Swap transitions are better, but whether that is the case is not known. Nivre et al. (2009) provide a partial answer, suggesting that LAZY outperforms EAGER, but their evaluation used different

[8]While the same argument can be used to argue against incurring any loss related to Swap, we found that penalizing erroneous Swap transitions empirically led to better results.

machine learning techniques, different treebanks, and they only report numbers on non-projective arcs and not full treebank results. Lastly, it is also an open question what is the difference in performance between the two non-deterministic oracles. With sufficient faith in the machine learning algorithm one could argue that ND-ALL, which allows all spurious ambiguities, ought to be better. On the other hand, as we introduced ND-SW we motivated it by the fact that it is appealing to build new structure as soon as possible since this should aid the feature representation of the model.

Since the parsers trained using the non-deterministic oracles rely on a latent sequence, they might require more training iterations before reaching good performance. Moreover, initial experiments on the development data indicated that the learning curves are not monotonically increasing. To test the main hypothesis we therefore tune the number of training iterations on the development sets for each oracle and treebank. That is, during training we test the current weights on the development set after each iteration. The best performing weights will then be saved and used for the final evaluation on the test set.

4.5.1 Greedy Parser

We begin the evaluation with the greedy parser. Table 4.2 shows the LAS scores of the greedy parser on the development sets after tuning the number of training epochs. When LAZY is already minimal, we omit MINIMAL since they would be identical (indicated by – in the table). For Korean, where the treebank is entirely projective, we only compare ND-ALL and EAGER, which thus reduces to comparing a static and a non-deterministic oracle for ArcStandard. It is also worth noting that the difference between EAGER and LAZY amounts to a single Swap for the Hebrew and French treebanks (cf. Table 4.1 in Section 4.4.1), suggesting that this should play a minuscule role. But the experimental results indicate that the difference sometimes amounts to more than 0.1% absolute. This tiny difference in transition sequences in the training data appears to have a butterfly effect during the online learning, such that a single different update changes the outcome of the resulting weight vector to this effect.

Overall, the non-deterministic oracles seem to outperform their static counterparts though. This can be seen in the average numbers, which show that both non-deterministic oracles on average are ahead of all static oracles. The experimental results indicate that, among the static oracles, the oracles that prefer fewer Swap transitions tend to be better, with slight exceptions for Hungarian and Swedish, where LAZY is a tiny bit ahead. Among the non-deterministic oracles, the ND-ALL oracle generally outperforms the ND-SW oracle, also here with small exceptions for Hungarian and Swedish.

	Static			Non-deterministic	
	EAGER	LAZY	MINIMAL	ND-SW	ND-ALL
Arabic	83.68	83.74	**83.76**	83.84	**84.12**
Basque	79.07	**79.40**	–	79.34	**79.59**
English	86.55	**86.70**	–	86.83	**87.57**
German	87.45	88.45	**88.81**	88.44	**88.76**
French	81.52	**81.57**	–	81.57	**81.99**
Hebrew	75.62	**75.75**	–	75.75	**76.18**
Hungarian	79.04	**79.18**	79.15	**79.17**	78.87
Korean	**84.92**	–	–	–	**85.06**
Polish	**80.18**	79.60	–	80.31	**80.47**
Swedish	72.53	**73.29**	73.08	**72.74**	72.72
Average	73.80	81.26	**81.27**	81.30	**81.53**

Table 4.2: Greedy results on development sets. The best non-deterministic and static oracles are bold. Cells marked with – denote cases where an oracle does not reduce the number of swaps compared to EAGER or LAZY.

4.5.2 Beam Parser

Having seen the positive results for the greedy parser, the open question is whether this carries over to a beam search setting. In the same manner as for greedy, we present a table comparing all oracles on the development sets after tuning the number of training epochs. The results are displayed in Table 4.3. Among the static oracles the trend remains the same – fewer swaps generally lead to higher accuracy. This is particularly pronounced for German, where the EAGER oracle is clearly behind. This correlates with the fact that the German treebank has a considerable amount of non-projective sentences and the fact that the reduction of Swap transitions from EAGER to LAZY and MINIMAL is rather high (cf. Table 4.1). Also for the non-deterministic oracles the picture remains roughly the same, i.e., the ND-ALL oracle is mostly ahead of ND-SW, although for Hungarian this is clearly not the case.

The answer to the main question, whether latent sequences lead to better parsers than canonical ones, is, however, less clear-cut for the beam search parser than for the greedy parser. In most cases the best static and non-deterministic oracles are very close. The main outlier here is Polish, where the ND-ALL oracle appears to be clearly ahead of the static ones. Considering the averages of each oracle alone, however, it is not obviously

	Static			Non-deterministic	
	EAGER	LAZY	MINIMAL	ND-SW	ND-ALL
Arabic	85.88	85.93	**85.96**	85.92	**86.10**
Basque	80.54	**80.87**	–	80.53	**80.88**
English	**88.96**	88.94	–	**89.08**	88.80
German	90.34	91.11	**91.18**	**91.12**	**91.12**
French	83.49	**83.65**	–	83.49	**83.66**
Hebrew	77.85	**77.99**	–	77.89	**78.00**
Hungarian	82.30	82.44	**82.52**	**82.44**	81.80
Korean	**85.30**	–	–	–	**85.18**
Polish	82.74	**82.99**	–	82.50	**83.98**
Swedish	75.01	75.25	**75.41**	**74.75**	74.60
Average	83.24	83.45	83.48	83.29	83.41

Table 4.3: Beam search results on development sets. The best non-deterministic and static oracles are bold. Cells marked with – denote cases where an oracle does not reduce the number of swaps compared to EAGER or LAZY.

the case that one of the non-deterministic oracles is better than a static one. Rather, the static oracles LAZY and MINIMAL obtain slightly higher average than the ND-ALL oracle, which is the better of the two non-deterministic oracles. This suggests that, in the case of beam search, the positive effect of non-deterministic oracles that we saw for the greedy parser vanishes.

4.5.3 Test Set Results

In the previous two tables we have seen development set results for the greedy and beam parsers. Recall that these numbers were in fact tuned on the very same development sets, and the numbers must thus be taken with a grain of salt. To make a final, fair comparison we compare the parsers trained with the best static and best non-deterministic oracles (i.e., corresponding to the bold numbers in Tables 4.2 and 4.3) on the test sets of each treebank in both a greedy and a beam setting. The results are shown in Table 4.4. The table includes the absolute numbers of the parsers trained with a static oracle and the relative difference to the parsers trained with a non-deterministic oracle.

	Greedy		Beam	
	Static	Non-deterministic	Static	Non-deterministic
Arabic	82.99	0.04	85.05	0.06
Basque	78.58	0.24	79.97	0.55
English	87.85	0.60†	90.35	0.13
German	84.22	0.03	87.53	-0.23
French	81.12	0.40†	83.10	-0.11
Hebrew	75.27	0.70	78.65	-0.39
Hungarian	81.45	0.22	83.60	0.08
Korean	84.52	0.30	85.03	0.09
Polish	79.10	1.33†	82.08	1.26†
Swedish	75.89	0.39	79.05	-0.07
Average	81.10	0.43	83.44	0.14

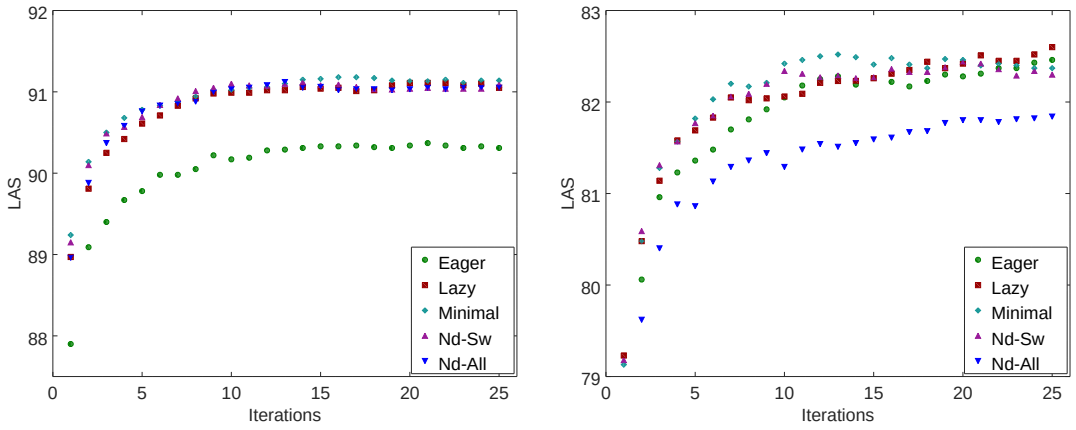
Table 4.4: Test set result comparing the best static and non-deterministic oracles for both greedy and beam. Significant differences at the $p < 0.01$ level are indicated by †.

For the greedy parser we see that all differences are positive. Moreover, in three cases the improvement is significant.[9] We thus conclude that, in the greedy case, non-deterministic oracles tend to either have a clear positive effect or a negligible effect compared to training with a static oracle.

For the beam search parser the results are more mixed. In about half the cases the non-deterministic oracle does slightly better than the static one, but in the other half it is the other way around. The only significant difference is the improvement of the non-deterministic oracle for Polish. Although we did not include it in the table of data set statistics, we note in passing that the Polish treebank has the shortest average sentence length of all treebanks we consider. Whether this plays a role in these results is an open question. The overall conclusion here is, nevertheless, that when a beam search parser is used, the choice between static and non-deterministic oracles is of lesser importance. That is, beam search appears to compensate for the added value of non-deterministic oracles that we saw in the greedy case.

Finally, if we take yet one more step back and consider the relationship between greedy and beam search, it is clear that the greedy parser generally is behind by more than two points on average. In some cases, e.g., Hebrew and Polish, the difference is more

[9] As in the previous chapter, we again use the Wilcoxon signed-rank test to test for significance.



(a) German **(b)** Hungarian

Figure 4.11: Learning curves of the different oracles for German and Hungarian. Taken on the development sets.

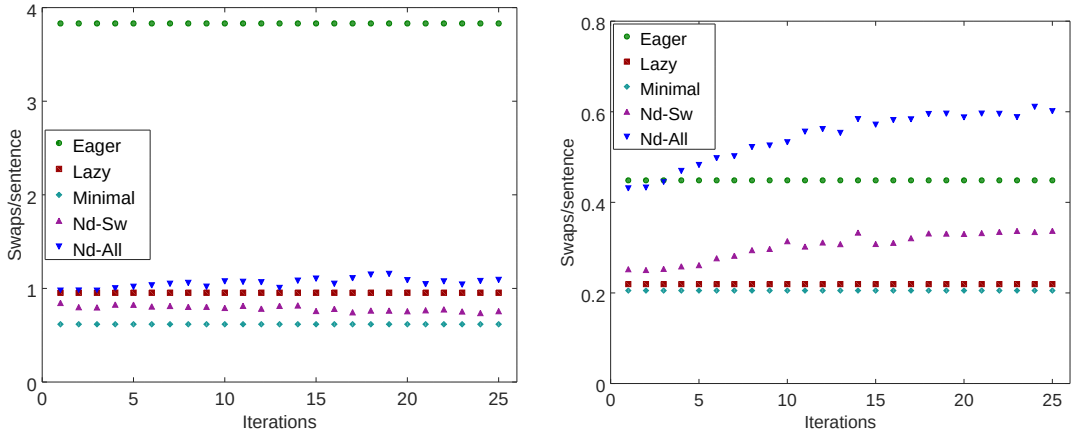
than 3 points. The smallest difference occurs for Korean, where the differences between the two parsers are remarkably small, yet in favor of the beam search parser. Nevertheless, this corroborates our general premise throughout this dissertation that the globally normalized structured models outperform their locally normalized counterparts.

4.5.4 Discussion

Experimentally we have seen that the non-deterministic oracles provide little if any improvement over their static counterparts for the beam search parsers. Differences between the data sets, such as language (which implies differences in word order, morphological richness, and so on), sentence length, and prevalence of non-projective arcs make it difficult to draw general conclusions about the role of these oracles at a high level. Instead we select German and Hungarian – the two treebanks that have the highest proportion of non-projective sentences (cf. Table 4.1) – where Swap and its spurious ambiguity plays a comparatively larger role and take a closer look at what is going on during training.

Figure 4.11 shows the learning curves of the beam search parser where accuracy is plotted against the training epoch for these two treebanks. The two plots paint a rather divergent picture, where EAGER is clearly under-performing for German, and ND-ALL is considerably worse than other oracles for Hungarian.

Similarly, Figure 4.12 contains two plots of the average number of Swap transitions per sentence for German and Hungarian as a function of the number of training itera-



(a) German **(b)** Hungarian

Figure 4.12: Average number of swaps per sentence during training for German and Hungarian.

tions. This number is calculated on the latent sequences that are being learned in the corresponding training iteration. The static oracles render straight lines since the transition sequences do not change between iterations, while the non-deterministic oracles do. Also here these two treebanks exhibit extreme behavior. In the case of German, the EAGER oracle is clearly applying Swap much more than any other oracle. The non-deterministic oracles tend to stay very close to the minimal number of Swap transitions. For Hungarian the picture is dramatically different. The ND-ALL oracle has a tendency to overswap and gradually applies more and more Swap transitions. These results bear a striking resemblance to those shown in the learning curves from Figure 4.11, where EAGER and ND-ALL are harmful for German and Hungarian, respectively. For most of the other treebanks the corresponding curves are much closer. Indeed, as the other treebanks exhibit considerably less non-projectivity, the amount of spurious ambiguity and choice of swaps is much more constrained.

But why do the non-deterministic oracles seem to be beneficial for the greedy parser but not for the beam search parser? One reason might be that the non-deterministic oracle provides greater diversity in the training data. This is the same effect that dynamic oracles achieve with training by exploration (these techniques are further discussed in Section 4.6.3), although it is less pronounced when only using a non-deterministic oracle. The beam search parser, on the other hand, is already exposed to many mistakes during training because of the global learning. Since the beam search parser actually does explore multiple possible transition sequences, it is probably also more lenient towards only seeing a single (static) sequence of transitions for every training instance.

4.6 Related Work

In the dependency parsing literature, the algorithmic approaches are broadly categorized into either transition-based or graph-based (Kübler et al., 2009). In contrast to transition-based approaches, graph-based parsers attempt to model the graph directly and globally search for the best tree according to a machine-learned scoring model. Classical work on transition-based (Nivre et al., 2006) and graph-based (McDonald et al., 2005) approaches illustrate very well the trade-offs between the two approaches: While the former offers efficient decoding and access to richer contextual features, it does so by using non-optimal search algorithms. Graph-based parsers, on the other hand, typically trade global search for a limited scope of features. Similarly, parsing algorithms that are restricted to projective trees tend to be more efficient and/or have access to a richer feature model than non-projective alternatives. For both graph-based and transition-based algorithms, considerable effort has gone into trying to bridge the gaps between expressivity in terms of features, efficiency in terms of time complexity, and coverage in terms of non-projectivity.

4.6.1 Graph-based Dependency Parsing

Before delving into the relevant work on transition-based parsing, we make a detour into graph-based parsing to illustrate the inherent trade-offs discussed above. For instance, as shown by McDonald et al. (2005), graph-based, non-projective parsing can be done in $O(n^2)$ time using the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967), which is a greedy recursive algorithm to find maximum spanning trees in directed graphs. However, the feature scope needs to be restricted to individual arcs, i.e., it is an arc-factored model and partial subtrees cannot be taken into consideration while scoring arcs. In fact, it has been shown that non-projective graph-based parsing is NP-hard if the scoring function is not arc-factored (McDonald and Pereira, 2006; McDonald and Satta, 2007).

Eisner's algorithm (Eisner, 1996), a graph-based algorithm based on dynamic programming, has been adapted to include features spanning pairs of arcs at the cost of $O(n^3)$ (McDonald and Pereira, 2006), three arcs at $O(n^4)$ (Koo and Collins, 2010), and so on (Ma and Zhao, 2012). With growing complexity, increasing the scope of features gradually becomes intractable, and adaptations of Eisner's algorithm that sacrifice the exactness of search while enabling richer features have also been proposed (Zhang and McDonald, 2012; Zhang et al., 2013b). Similarly, exact search can be given up in order to enable non-projective graph-based parsing algorithms that can access a richer structural

context using methods such as hill-climbing (McDonald and Pereira, 2006; Zhang et al., 2014c) or integer linear programming with approximate solvers (Koo et al., 2010; Martins et al., 2010, 2013). Finally, Pitler (2014) present an adaptation of Eisner's algorithm that makes a compromise between non-projectivity and access to rich features by defining a subset of non-projective trees. This way rich features can be accessed but the search problem can still be solved exactly in polynomial time.

4.6.2 Transition-based Dependency Parsing

Early work on transition-based dependency parsing was restricted to using greedy search and a locally normalized classifier to select the next transition (Yamada and Matsumoto, 2003; Nivre, 2004). Greedy transition-based parsers are attractive due to their efficiency. On the other hand, it is well known that search errors cause error propagation in locally normalized models and, consequently, lead to suboptimal parsing results. Beam search, the technique we use in this chapter (and dissertation) was first proposed for transition-based dependency parsing by Johansson and Nugues (2006, 2007b), however they had only limited success in terms of improvement. The reason was that they still relied on a locally trained classifier and only applied beam search at test time. Zhang and Clark (2008) were the first to couple it with globally normalized training through the structured perceptron with early updates. This combination has since become standard procedure and have resulted in many strong transition-based dependency parsers (Huang and Sagae, 2010; Zhang and Nivre, 2011; Bohnet and Kuhn, 2012). Moreover, due to the incrementality of transition-based parsing systems, this recipe has often been extended to also include multiple stages in a standard NLP pipeline, such as joint POS tagging and dependency parsing (Bohnet and Nivre, 2012; Hatori et al., 2011), or even joint tokenization, tagging, and parsing (Hatori et al., 2012). In the next chapter we will extend the machinery we have seen in this chapter to do joint sentence segmentation and parsing.

Another approach to coping with the search problem for transition-based dependency parsing is through dynamic programming. However, dynamic programming approaches are not known for all systems (in particular, not for SwapStandard) and the complexity of the dynamic program is a function of the complexity of the partially constructed subtrees that are used for the feature function. For instance, Huang and Sagae (2010) formulate a dynamic program for the ArcStandard system. However, the richness of their feature model prevents them from searching through it exhaustively. Instead, they also apply beam search to reduce the number of explored paths. More re-

cent work has leveraged the power of dense neural representations and removed features predicated on structural context during prediction and thus obtained competitive and exact search algorithms for several projective transition systems, including ArcEager and ArcHybrid (Shi et al., 2017) as well as some limited non-projective systems (Gómez-Rodríguez et al., 2018). There is, however, no known dynamic programming solution for the SwapStandard system.

4.6.3 Dynamic Oracles and Training With Exploration

Instead of trying to extend the scope of the search algorithm, one way to improve greedy parsers is to try to reduce the impact of error propagation. This has motivated the work on *dynamic oracles* (Goldberg and Nivre, 2012, 2013). Dynamic oracles should be regarded in the context of the static and non-deterministic oracles we have previously seen in this chapter. Recall that a static oracle provides a single transition sequence to derive a given dependency tree for a sentence. Non-deterministic oracles cover the spurious ambiguities and allow for all (or a subset of) the possible transition sequences. Dynamic oracles take this one step further, and provide a set of possible transitions given a state which has already deviated from any paths defined by a non-deterministic oracle. For the set of transitions, the dynamic oracle selects those transitions that could lead to a final tree with the *minimal* amount of attachment errors, subject to the current state.

Dynamic oracles were first proposed by Goldberg and Nivre (2012) for the ArcEager system. They also proposed the standard way of exploiting dynamic oracles for training locally normalized greedy parsers known as *training with exploration*. Here, the idea is that sometimes erroneous transitions are predicted during training. The dynamic oracle then comes into play by guiding the model towards the best possible tree (in terms of LAS), subject to the mistakes that have already been made. Similarly to the latent transition sequences used in our experiments, the next transition given the current state is left latent and the perceptron model under training gets to choose what is the best transition given the current state. These parsers thus retain their highly efficient runtime complexity, but improve the parsing results compared to when they are trained with a static oracle.

More recent work on dynamic oracles has primarily been concerned with developing dynamic oracles for other transition systems. Goldberg et al. (2014) present dynamic oracles for the ArcStandard system and the LR-spine parser (Sartorio et al., 2013). Dynamic oracles have also been defined for some transition systems that allow for a restricted amount of non-projectivity (Gómez-Rodríguez et al., 2014; Gómez-Rodríguez et al., 2018).

Gómez-Rodríguez and Fernández-González (2015) present a non-deterministic oracle for Covington's (2001) unrestricted parsing algorithm which could be argued to be transition-based (Nivre, 2008), but is strictly slower with an $O(n^2)$ time complexity with regard to the input.

Finally, as an alternative to dynamic oracles, recent work has also focused on developing approximate dynamic oracles using machine learning techniques. The basic idea is to use machine learning to try to decide what the latent transitions should be. This involves ideas such as searching for the best transitions in the presence of mistakes (Straka et al., 2015), or trying to directly learn a function that returns the highest possible mistake in the presence of mistakes (Le and Fokkens, 2017). Recently Yu et al. (2018) combined these ideas using reinforcement learning, where a function is learned using the gold standard tree as features.

4.7 Conclusion

In this chapter we have studied the SwapStandard transition system and analyzed its spurious ambiguities. This has enabled us to create new oracles that could be used for training parsers. We developed two non-deterministic oracles that can be used for learning latent transition sequences. One of these oracles provide the full space of all transition sequences, whereas the other was restricted to the specific ambiguity between Swap and Shift. Additionally, we solved the open problem of creating a static oracle that minimizes the number of Swap transitions required for non-projective parsing.

In terms of the framework from Chapter 2, the primary functionality applied in this chapter is the ability of using latent structures for learning. We abstained from an in-depth analysis of the different update methods, a topic that we saw in the previous chapter and will return to in the next one. However, at this point we can briefly mention that the update methods do not play an important role for sequences of the length seen in this chapter.

The primary question for the empirical evaluation was whether latent transition sequences can be used to improve the performance of a transition-based parser. We considered this question using both a greedy, classifier-based parser, which has previously been studied with positive results for other systems, and a beam search parser, where this question has previously not been regarded. The experimental results show that the answer to this question depends on whether greedy or beam search is used – in the greedy case, the latent transitions help, whereas for beam search the performance is roughly the

same as when using a static oracle. In a broad sense, the conclusion is that the non-deterministic oracles are never harmful compared to their static counterparts, although they sometimes also do not yield any improvements.

A secondary result of our experiments is a thorough comparison of all static and non-deterministic oracles. The general result could be summed up by saying that fewer swaps in the training sequences tends to improve the performance. This is apparent from the comparison between the static oracles EAGER and LAZY, corroborating and extending Nivre et al.'s (2009) results. While the static MINIMAL oracle theoretically reduces the number of swaps even more, the reduction is in practice rather small due to the nature of the treebanks and since the difference in swaps when moving from LAZY to MINIMAL is rather minor. The results from the experiments with non-deterministic oracles also points to he fact that fewer swaps leads to better performance. We saw this in the analysis of Hungarian, where the ND-ALL oracle had a tendency to overswap greatly, yielding more swaps and also worse results than the EAGER oracle.

Chapter 5

Joint Sentence Segmentation and Dependency Parsing

5.1 Introduction

In the previous chapter we studied the utility of non-deterministic oracles for transition-based dependency parsing. The novelty with respect to previous work was that we used the idea of latent structures for learning search-based transition-based dependency parsers. In this chapter we will look at another aspect that the framework from Chapter 2 is concerned with: the update methods required and their importance vis-à-vis the length of the sequences that need to be learned. We will extend the dependency parsing task to not just parse single sentences, but to parse a sequence of tokens (i.e., a document), where the beginnings and ends of the sentences are not known.

The default approach to parse documents is to build a pipeline of NLP components, solving a number of sub-tasks sequentially. Such a pipeline would start with a sentence boundary detector which splits the input document into sentences. Then, each sentence would be fed through a tokenizer followed by a part-of-speech tagger and morphological analyzer only after which the parser would step in. When working with carefully copy-edited text documents, sentence boundary detection can be viewed as a minor preprocessing task in such a pipeline, solvable with very high accuracy. However, when dealing with the output of automatic speech recognition or "noisier" texts such as blogs and emails, non-trivial sentence segmentation issues do occur. Dridan and Oepen (2013), for example, show that fully automatic preprocessing can result in considerable drops in parsing quality when moving from well-edited to less-edited text.

Two possible strategies to approach this problem are (i) to exploit other cues for sen-

tence boundaries, such as prosodic phrasing and intonation in speech (Kolář et al., 2006) or formatting cues in text documents (Read et al., 2012), and (ii) to emulate the human ability to exploit syntactic competence for segmentation. By coupling the prediction of sentence boundaries with syntax we will aim for the latter. The basic intuition is that segmentations that would give rise to suboptimal syntactic structures will also be more difficult to parse. Therefore, erroneous segmentations can be caught early during search and discarded in favor of segmentations where the syntactic structure receives a high score by the parsing model.

Our technical approach will be to extend the transition system from the previous chapter to predict sentence boundaries and syntax **jointly**. We will refine the transition system by a dedicated transition to label sentence boundaries and augment the states to keep track of this information. We characterize the necessary preconditions for the transition system in order to keep the resulting output well-formed.

Although this joint system and, consequently, the machine-learning problem are by and large similar to what we saw in the previous chapter, they differ strongly in terms of the length of the transition sequences. We instantiate the framework from Chapter 2 similarly as in the previous chapter, using both a greedy, classifier-based model as a baseline and a beam search parser as the contrastive system. We evaluate the update methods for the approximate search setting and find, similarly to the results on coreference resolution, that the update methods that discard training data are inadequate for this problem as they fail to outperform the baseline. However, when we apply DLaSO we find that the beam search parser outperforms the baseline.

From the computational linguistics perspective, the joint system allows us to test the utility of syntax when predicting sentence boundaries. We demonstrate empirically that syntactic information can make up to a large extent for missing or unreliable cues from punctuation. The joint system allows us to test the influence of syntactic information on the prediction of sentence boundaries as compared to a pipeline baseline where both tasks are performed independently of each other. With a thoughtful selection of data sets and baselines for the sentence segmentation problem, we are able to demonstrate that syntactic information is helpful for the task of sentence segmentation.

For our analysis, we use the Wall Street Journal as the standard benchmark set and as a representative for copy-edited text. We also use the Switchboard corpus of transcribed dialogues as a representative for data where punctuation cannot give clues to a sentence boundary predictor. Other types of data that may exhibit this property to varying degrees are web content data, e.g. forum posts or chat protocols, or (especially historical) manuscripts. While the Switchboard corpus gives us a realistic scenario for a setting with

unreliable punctuation, the syntactic complexity of telephone conversations is rather low compared to the Wall Street Journal. Therefore, as a controlled experiment for assessing how far syntactic competence alone can take us if we stop trusting punctuation and capitalization entirely, we also perform joint sentence boundary detection/parsing on a lower-cased, no-punctuation version of the Wall Street Journal.

5.2 Joint Transition-based Sentence Segmentation and Dependency Parsing

In this section we will define the task more formally and ground it in the notation and terminology from Chapter 2. We then go on to discuss the extended transition system after which a discussion of the oracle function follows.

5.2.1 Task Definition

The main difference between the tasks in this chapter and the previous is that now the input is a full document. Again, we reuse token and sentence definitions from previous chapters (cf. Sections 3.2.1 and 4.2.1), where a sentence is a set of tokens with associated atomic token-level features such as part-of-speech tags. In contrast to the previous chapter, however, the input $\mathbf{x}$ is no longer a sentence, but a document. Nevertheless, the input document can be regarded as a very long sentence, i.e., $\mathbf{x} = S_{\text{doc}}$ where all input tokens are part of the same (document-)sentence.

The expected output structure $\mathbf{y} = D$, where $D = (\langle S_1, A_1\rangle, \langle S_2, A_2\rangle, ..., \langle S_n, A_n\rangle)$ is a list of dependency trees over the tokens in the input. That is, the sentences S_i divide the input document S_{doc} into sub-sequences and construct a dependency tree for each S_i. The dependency trees A_i are subject to the same definitions and constraints as we saw in Section 4.2.1 where we defined the dependency parsing task. In addition to the three constraints for dependency trees we postulate one more constraint on the output:

4. **Continuity constraint**: For every dependency tree $\langle S, A\rangle \in D$, S spans a continuous sequence of tokens in S_{doc}.

This fairly obvious constraint requires that sentences in the output cannot be intertwined.

Figure 5.1 displays an example using the beginning of a document from the Switchboard corpus. For this particular example, which was chosen for its brevity, the task of

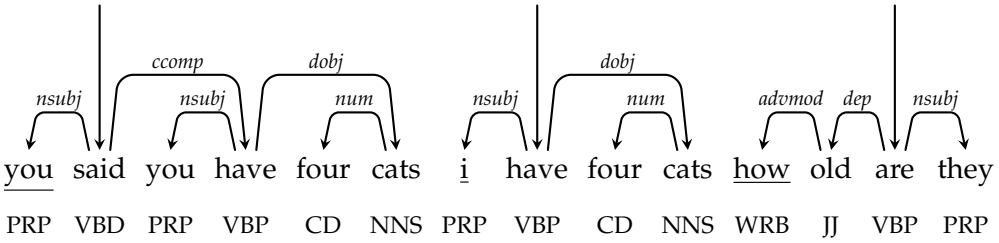


Figure 5.1: An example analysis of the first three sentences of a document taken from the Switchboard corpus. Tokens that start a sentence are underlined.

sentence boundary prediction could be solved easily with speaker information since the second sentence is from another speaker's turn. The interesting cases involve sentence segmentation within syntactically complex turns of a single speaker. Nevertheless, the example illustrates the expected input and output: Given the sequence of tokens and associated additional annotations as input (in this case morphological features are not included, since they are not present in the Switchboard data), the task is to produce the corresponding dependency trees and segment the input into sentences.

5.2.2 Transition System

The transition system we develop for joint sentence segmentation and dependency parsing is an extension of the SwapStandard system we saw in the previous chapter. First of all, we extend the state representation $s = (\Sigma, B, A, Q)$ to also include a list Q of sentence-initial tokens. That is, this list will holds tokens that have been flagged as the start of a new sentence.[1] The initial state $s_0 = ([t_0], [t_1, t_2, ...], \emptyset, [t_1])$ is correspondingly extended, and the very first input token is flagged as the start of a new sentence. We then introduce a new transition SentBound (for Sentence Boundary) that flags the next token on the buffer as the beginning of a new sentence. Finally, we augment the buffer representation to hold a boolean flag that indicates whether the buffer contains any tokens that have been shifted and subsequently swapped back. This flag equivalently also indicates whether the remaining contents of the buffer is a continuous sequence of tokens and is necessary to ensure the continuity constraint during parsing.

A full listing of the transitions of the system is shown in Figure 5.2. The transitions LeftArc, RightArc and Swap are identical to the basic SwapStandard system from the

[1] We note in passing that a list of sentence-final tokens would serve the same purpose. The decision to track sentence-initial tokens is arbitrary and, with certain revisions to the transitions, an alternative system that tracks sentence-final tokens could be constructed.

Transition				Preconditions
Shift	$(\sigma, b_0\|\beta, A, Q)$	$\Rightarrow$	$(\sigma\|b_0, \beta, A, Q)$	$b_0 \neq \text{LAST}(Q) \vee \|\sigma\| = 1 \vee \text{SWAPPED}(B)$
LeftArc$_\text{L}$	$(\sigma\|u_1\|u_0, B, A, Q)$	$\Rightarrow$	$(\sigma\|u_0, B, A \cup \{u_0 \overset{\text{L}}{\rightarrow} u_1\}, Q)$	$u_1 \neq t_0$
RightArc$_\text{L}$	$(\sigma\|u_1\|u_0, B, A, Q)$	$\Rightarrow$	$(\sigma\|u_1, B, A \cup \{u_1 \overset{\text{L}}{\rightarrow} u_0\}, Q)$	
Swap	$(\sigma\|u_1\|u_0, B, A, Q)$	$\Rightarrow$	$(\sigma\|u_0, u_1\|B, A, Q)$	$u_1 \neq t_0 \wedge u_1 \prec u_0$
SentBound	$(\sigma, b_0\|\beta, A, Q)$	$\Rightarrow$	$(\sigma, b_0\|\beta, A, Q \cup \{b_0\})$	$\neg\text{SWAPPED}(B) \wedge \text{LAST}(Q) \prec b_0$

Figure 5.2: Transition system. $\sigma|u_1|u_0$ denotes the stack with u_0 and u_1 on top, $b_0|\beta$ denotes the buffer with b_0 in front. $\text{LAST}(Q)$ denotes the most recent sentence boundary, and $\text{SWAPPED}(\beta)$ is true iff the buffer contains swapped items.

previous chapter. Shift has received a number of preconditions, which we will discuss shortly. The SentBound semantics are straightforward, simply adding b_0 to the list of sentence-initial tokens, although it comes with a couple of preconditions.

Before we describe the preconditions for Shift and SentBound in detail, we take a step back and explain the intuitive behavior of the system. The system processes the input like any other sentence. When it comes to a point where the first item on the buffer begins a new sentence a SentBound transition is applied, flagging the beginning of the next sentence. At this point, the current items on the stack need to be reduced before any new input can be shifted. If new tokens (belonging to the next sentence) could be shifted, arcs could be introduced between tokens from different sentences, which violates the continuity constraint. Once the stack has been reduced to only contain the ROOT, the dependency tree for that sentence is complete and the system can continue with Shift and process the next input.

Due to potential interactions with Swap, the preconditions for SentBound and Shift need to be carefully chosen. First of all, SentBound is only allowed when the buffer does not contain any swapped tokens. Otherwise b_0 would have to be part of the following sentence, even though tokens that succeed b_0 in the linear order are still on the stack, thus violating the continuity constraint. This precondition makes use of the SWAPPED(B) predicate which returns true if the buffer contains any swapped item. The system should also not be allowed to flag the same token as sentence-initial twice. Moreover, the system could possibly apply a SentBound followed by a Swap, which would push a token that linearly precedes the most recently flagged sentence boundary onto the buffer. However, after flagging a sentence-initial token, the goal is to reduce the stack contents fully before applying more SentBound transitions. Therefore, SentBound is only allowed when the next item on the buffer succeeds the most recent sentence-initial token. The predicate $\text{LAST}(Q)$ used in the preconditions of SentBound returns the most recently flagged sentence-initial token, and the precondition is that $\text{LAST}(Q) \prec b_0$.

While the intuition is that Shift should be blocked until a complete tree over the current sentence is done, the transition needs to be allowed if a Swap is applied after a SentBound transition. The possible situations can be broken down along the three following cases resulting in the preconditions shown in Figure 5.2:

- If $\text{LAST}(Q) \prec b_0$: The last predicted sentence boundary has already been shifted onto the stack. At this point, the system is building a new sentence and has not yet decided where it ends. Shift is therefore allowed.
- If $\text{LAST}(Q) \succ b_0$: This situation can only occur if the system predicted a sentence boundary and subsequently made a Swap. $\text{LAST}(Q)$ then denotes the end of the current sentence and is deeper in the buffer than b_0. Thus Shift is allowed since b_0 belongs to the current sentence.
- If $\text{LAST}(Q) = b_0$: The system must complete the current sentence by reducing the stack before it can continue shifting and Shift is generally not allowed, with two exceptions. (1) If the stack consists only of the root (i.e., $|\sigma| = 1$), the current sentence has been completed and the system is ready to begin parsing the next one. (2) If b_0 denotes the beginning of a new sentence, but it has been swapped back onto the buffer, then it belongs to the same sentence as the tokens currently on the stack.

Figure 5.3 displays a derivation of the sentences from the example in Figure 5.1. The system starts by constructing the tree like in a regular SwapStandard derivation. Then in state s_9, once the last token of the first sentence has been shifted onto the buffer, it applies a SentBound transition. The subsequent four transitions complete the tree of the first sentence and then, in state s_{14} the system continues with the next sentence. The procedure is repeated, with a SentBound transition occurring in state s_{19}, followed by finalization of the tree (state s_{22}). The last sentence is parsed as a regular SwapStandard derivation, without any SentBound transition since there is no following sentence.

Before ending the discussion of the transition system, it is worth considering it with respect to the task definition. The system we have just described uses a single ROOT token for each document, whereas the task definition supposes a ROOT for each sentence. Since mapping between the two is trivial, it is clear that this does not pose a problem in terms of delivering the proper output structure. The system also does not explicitly create subsequences of the input tokens corresponding to each sentence. Also here it is trivial to reconstruct such sequences given the list of sentence-initial tokens. In conclusion, the transition system is sufficient to derive the output structures we commit to in the task definition.

State	Transition	Stack	Buffer	Arc Added	SB Added
s_0		[ROOT$_0$]	[You$_1$, ..., they$_{14}$]		
s_1	Shift	[ROOT$_0$, You$_1$]	[said$_2$, ..., they$_{14}$]		
s_2	Shift	[ROOT$_0$, You$_1$, said$_2$]	[you$_3$, ..., they$_{14}$]		
s_3	LeftArc$_{\text{NSUBJ}}$	[ROOT$_0$, said$_2$]	[you$_3$, ..., they$_{14}$]	said$_2 \overset{\text{NSUBJ}}{\rightarrow}$ You$_1$	
s_4	Shift	[ROOT$_0$, said$_2$, you$_3$]	[have$_4$, ..., they$_{14}$]		
s_5	Shift	[ROOT$_0$, ..., you$_3$, have$_4$]	[four$_5$, ..., they$_{14}$]		
s_6	LeftArc$_{\text{NSUBJ}}$	[ROOT$_0$, said$_2$, have$_4$]	[four$_5$, ..., they$_{14}$]	have$_4 \overset{\text{NSUBJ}}{\rightarrow}$ you$_3$	
s_7	Shift	[ROOT$_0$, ..., have$_4$, four$_5$]	[cats$_6$, ..., they$_{14}$]		
s_8	Shift	[ROOT$_0$, ..., four$_5$, cats$_6$]	[I$_7$, ..., they$_{14}$]		
s_9	SentBound	[ROOT$_0$, ..., four$_5$, cats$_6$]	[I$_7$, ..., they$_{14}$]		I$_7$
s_{10}	LeftArc$_{\text{NUM}}$	[ROOT$_0$, ..., have$_4$, cats$_6$]	[I$_7$, ..., they$_{14}$]	cats$_6 \overset{\text{NUM}}{\rightarrow}$ four$_5$	
s_{11}	RightArc$_{\text{DOBJ}}$	[ROOT$_0$, said$_2$, have$_4$]	[I$_7$, ..., they$_{14}$]	have$_4 \overset{\text{DOBJ}}{\rightarrow}$ cats$_6$	
s_{12}	RightArc$_{\text{CCOMP}}$	[ROOT$_0$, said$_2$]	[I$_7$, ..., they$_{14}$]	said$_2 \overset{\text{CCOMP}}{\rightarrow}$ have$_4$	
s_{13}	RightArc$_{\text{ROOT}}$	[ROOT$_0$]	[I$_7$, ..., they$_{14}$]	ROOT$_0 \overset{\text{ROOT}}{\rightarrow}$ said$_2$	
s_{14}	Shift	[ROOT$_0$, I$_7$]	[have$_8$, ..., they$_{14}$]		
s_{15}	Shift	[ROOT$_0$, I$_7$, have$_8$]	[four$_9$, ..., they$_{14}$]		
s_{16}	LeftArc$_{\text{NSUBJ}}$	[ROOT$_0$, have$_8$]	[four$_9$, ..., they$_{14}$]	have$_8 \overset{\text{NSUBJ}}{\rightarrow}$ I$_7$	
s_{17}	Shift	[ROOT$_0$, have$_8$, four$_9$]	[cats$_{10}$, ..., they$_{14}$]		
s_{18}	Shift	[ROOT$_0$, ..., four$_9$, cats$_{10}$]	[How$_{11}$, ..., they$_{14}$]		
s_{19}	SentBound	[ROOT$_0$, have$_8$, cats$_{10}$]	[How$_{11}$, ..., they$_{14}$]		How$_{11}$
s_{20}	LeftArc$_{\text{NUM}}$	[ROOT$_0$, have$_8$, cats$_{10}$]	[How$_{11}$, ..., they$_{14}$]	cats$_{10} \overset{\text{NUM}}{\rightarrow}$ four$_9$	
s_{21}	RightArc$_{\text{DOBJ}}$	[ROOT$_0$, have$_8$]	[How$_{11}$, ..., they$_{14}$]	have$_8 \overset{\text{DOBJ}}{\rightarrow}$ cats$_{10}$	
s_{22}	RightArc$_{\text{ROOT}}$	[ROOT$_0$]	[How$_{11}$, ..., they$_{14}$]	ROOT$_0 \overset{\text{ROOT}}{\rightarrow}$ have$_8$	
s_{23}	Shift	[ROOT$_0$, How$_{11}$]	[old$_{12}$, ..., they$_{14}$]		
s_{24}	Shift	[ROOT$_0$, How$_{11}$, old$_{12}$]	[are$_{13}$, they$_{14}$]		
s_{25}	LeftArc$_{\text{ADVMOD}}$	[ROOT$_0$, old$_{12}$]	[are$_{13}$, they$_{14}$]	old$_{12} \overset{\text{ADVMOD}}{\rightarrow}$ How$_{11}$	
s_{26}	Shift	[ROOT$_0$, old$_{12}$, are$_{13}$]	[they$_{14}$]		
s_{27}	LeftArc$_{\text{DEP}}$	[ROOT$_0$, are$_{13}$]	[they$_{14}$]	are$_{13} \overset{\text{DEP}}{\rightarrow}$ old$_{12}$	
s_{28}	Shift	[ROOT$_0$, are$_{13}$, they$_{14}$]	[ ]		
s_{29}	RightArc$_{\text{NSUBJ}}$	[ROOT$_0$, are$_{13}$]	[ ]	are$_{13} \overset{\text{NSUBJ}}{\rightarrow}$ they$_{14}$	
s_{30}	RightArc$_{\text{ROOT}}$	[ROOT$_0$]	[ ]	ROOT$_0 \overset{\text{ROOT}}{\rightarrow}$ are$_{13}$	

Figure 5.3: Example derivation of the output from Figure 5.1.

5.2.3 Oracle

Since each sentence constitutes its own subtree under the root node, a regular static sentence-based oracle can be used to derive the oracle transition sequence for complete documents. Specifically, the sentence-based oracle can be applied to each sentence, and then the resulting sequences can be joined with a SentBound transition in between. A non-deterministic oracle for the sentences could equally well be used but given the results from the previous chapter, that static and non-deterministic oracles are equal, we abstain from this as it would only add further complexity without any obvious benefit. Specifically, we use the LAZY oracle (Nivre et al., 2009) from the previous chapter.

State	Transition	Stack	Buffer	Arc Added	SB Added
s_0		[ROOT$_0$]	[You$_1$, ..., they$_{14}$]		
s_1	Shift	[ROOT$_0$, You$_1$]	[said$_2$, ..., they$_{14}$]		
s_2	Shift	[ROOT$_0$, You$_1$, said$_2$]	[you$_3$, ..., they$_{14}$]		
s_3	LeftArc$_{\text{NSUBJ}}$	[ROOT$_0$, said$_2$]	[you$_3$, ..., they$_{14}$]	said$_2 \overset{\text{NSUBJ}}{\rightarrow}$ You$_1$	
s_4	Shift	[ROOT$_0$, said$_2$, you$_3$]	[have$_4$, ..., they$_{14}$]		
s_5	Shift	[ROOT$_0$, ..., you$_3$, have$_4$]	[four$_5$, ..., they$_{14}$]		
s_6	LeftArc$_{\text{NSUBJ}}$	[ROOT$_0$, said$_2$, have$_4$]	[four$_5$, ..., they$_{14}$]	have$_4 \overset{\text{NSUBJ}}{\rightarrow}$ you$_3$	
s_7	Shift	[ROOT$_0$, ..., have$_4$, four$_5$]	[cats$_6$, ..., they$_{14}$]		
s_8	Shift	[ROOT$_0$, ..., four$_5$, cats$_6$]	[I$_7$, ..., they$_{14}$]		
s_9	LeftArc$_{\text{NUM}}$	[ROOT$_0$, ..., have$_4$, cats$_6$]	[I$_7$, ..., they$_{14}$]	cats$_6 \overset{\text{NUM}}{\rightarrow}$ four$_5$	
s_{10}	RightArc$_{\text{DOBJ}}$	[ROOT$_0$, said$_2$, have$_4$]	[I$_7$, ..., they$_{14}$]	have$_4 \overset{\text{DOBJ}}{\rightarrow}$ cats$_6$	
s_{11}	RightArc$_{\text{CCOMP}}$	[ROOT$_0$, said$_2$]	[I$_7$, ..., they$_{14}$]	said$_2 \overset{\text{CCOMP}}{\rightarrow}$ have$_4$	
s_{12}	RightArc$_{\text{ROOT}}$	[ROOT$_0$]	[I$_7$, ..., they$_{14}$]	ROOT$_0 \overset{\text{ROOT}}{\rightarrow}$ said$_2$	
s_{13}	SentBound	[ROOT$_0$]	[I$_7$, ..., they$_{14}$]		I$_7$
s_{14}	Shift	[ROOT$_0$, I$_7$]	[have$_8$, ..., they$_{14}$]		
s_{15}	Shift	[ROOT$_0$, I$_7$, have$_8$]	[four$_9$, ..., they$_{14}$]		
s_{16}	LeftArc$_{\text{NSUBJ}}$	[ROOT$_0$, have$_8$]	[four$_9$, ..., they$_{14}$]	have$_8 \overset{\text{NSUBJ}}{\rightarrow}$ I$_7$	
s_{17}	Shift	[ROOT$_0$, have$_8$, four$_9$]	[cats$_{10}$, ..., they$_{14}$]		
s_{18}	Shift	[ROOT$_0$, ..., four$_9$, cats$_{10}$]	[How$_{11}$, ..., they$_{14}$]		
s_{19}	LeftArc$_{\text{NUM}}$	[ROOT$_0$, have$_8$, cats$_{10}$]	[How$_{11}$, ..., they$_{14}$]	cats$_{10} \overset{\text{NUM}}{\rightarrow}$ four$_9$	
s_{20}	RightArc$_{\text{DOBJ}}$	[ROOT$_0$, have$_8$]	[How$_{11}$, ..., they$_{14}$]	have$_8 \overset{\text{DOBJ}}{\rightarrow}$ cats$_{10}$	
s_{21}	RightArc$_{\text{ROOT}}$	[ROOT$_0$]	[How$_{11}$, ..., they$_{14}$]	ROOT$_0 \overset{\text{ROOT}}{\rightarrow}$ have$_8$	
s_{22}	SentBound	[ROOT$_0$]	[How$_{11}$, ..., they$_{14}$]		How$_{11}$
s_{23}	Shift	[ROOT$_0$, How$_{11}$]	[old$_{12}$, ..., they$_{14}$]		
s_{24}	Shift	[ROOT$_0$, How$_{11}$, old$_{12}$]	[are$_{13}$, they$_{14}$]		
s_{25}	LeftArc$_{\text{ADVMOD}}$	[ROOT$_0$, old$_{12}$]	[are$_{13}$, they$_{14}$]	old$_{12} \overset{\text{ADVMOD}}{\rightarrow}$ How$_{11}$	
s_{26}	Shift	[ROOT$_0$, old$_{12}$, are$_{13}$]	[they$_{14}$]		
s_{27}	LeftArc$_{\text{DEP}}$	[ROOT$_0$, are$_{13}$]	[they$_{14}$]	are$_{13} \overset{\text{DEP}}{\rightarrow}$ old$_{12}$	
s_{28}	Shift	[ROOT$_0$, are$_{13}$, they$_{14}$]	[ ]		
s_{29}	RightArc$_{\text{NSUBJ}}$	[ROOT$_0$, are$_{13}$]	[ ]	are$_{13} \overset{\text{NSUBJ}}{\rightarrow}$ they$_{14}$	
s_{30}	RightArc$_{\text{ROOT}}$	[ROOT$_0$]	[ ]	ROOT$_0 \overset{\text{ROOT}}{\rightarrow}$ are$_{13}$	

Figure 5.4: Example derivation of the output from Figure 5.1. Here the SentBound transition is applied as late as possible.

It should be noted that there is a spurious ambiguity as to when the SentBound transition can be applied. Contrary to the example from Figure 5.3, where SentBound was applied as *early* as possible, the method just described implies that SentBound is applied after the complete tree of the current sentence has been constructed. That is, as *late* as possible. For completeness, Figure 5.4 shows the alternative derivation, where SentBound is applied as late as possible. As compared to the previous derivation, the SentBound transitions are now applied later, when the stack solely holds the ROOT token.

Of course, the SentBound transition can also correctly be applied at any point in between, assuming that there are no swapped items on the buffer. The location of

SentBound can also be left latent and diverted to the machine learning algorithm. During implementation of the system we compared these three options and found that the difference in accuracy was generally rather small, albeit with a slight advantage for the early strategy. This is also intuitively more appealing, as it puts the system in the state where Shift is blocked as early as possible. We thus decided for a static early oracle for the experimental evaluation that is carried out later in this chapter.

5.3 Experimental Setup

In this section we will discuss the experimental setup. As usual, we begin with data sets and evaluation metrics. We then go on to outline the instantiation of the framework from Chapter 2. Before concluding this section, we also have a discussion on baselines and pipelines that will be used for the experiments.

5.3.1 Data Sets

We experiment with two parts of the English Penn Treebank (Marcus et al., 1993). We use the Wall Street Journal (WSJ) as an example of copy-edited newspaper-quality texts with proper punctuation and capitalized sentences. We also use the Switchboard portion which consists of (transcribed) telephone conversations between strangers. Following previous work on Switchboard we lowercase all text and remove punctuation and disfluency markups in order to emulate a more realistic setting.

We use sections 2-21 of the WSJ for training, 24 as development set and 23 as test set. For Switchboard we follow Charniak and Johnson (2001), using subdirectories 2 and 3 for training and splitting directory 4 into test and development sets. We convert both data sets to Stanford dependencies with the Stanford dependency converter (de Marneffe et al., 2006). We predict part-of-speech tags with the CRF tagger MarMoT (Mueller et al., 2013) and annotate the training sets via 10-fold jackknifing. Depending on the experimental scenario, to be discussed in Section 5.3.4, we use MarMoT in two different settings – standard sentence-level where we train and apply it on sentences, and document-level where a whole document is fed to the tagger, implicitly treating it as a single very long sentence. Table 5.1 shows an evaluation of the part-of-speech tagger. For WSJ both levels – sentences and documents – are POS tagged with almost identical accuracy. For Switchboard, which has much fewer training documents than the WSJ (1875 compared to 496), accuracy drops slightly on the document-level.

	WSJ		Switchboard	
	Train	Dev	Train	Dev
Sentence	97.31	97.21	95.92	96.17
Document	97.31	97.20	95.74	95.98

Table 5.1: Part-of-speech tagging accuracies on development sets when tagging either sentences in isolation or full documents using MarMoT.

The two data sets are quite different in nature. The WSJ is copy-edited newspaper articles with relatively long sentences (roughly 25 tokens per sentence on average) whereas Switchboard has considerably shorter (10 tokens per sentence on average). Since the sentence boundary detection problem is almost trivial for WSJ, Switchboard could be considered more appropriate for the joint system. On the other hand, with much shorter sentences, Switchboard does not offer as much syntactic complexity as WSJ. We therefore introduce a third data set: A version of the WSJ where we removed all punctuation and lower-cased all words, effectively making it identical to the Switchboard setting (denoted WSJ*). Although the experimental setting is rather artificial, WSJ* offers an additional perspective on the joint system where syntactic complexity is high, but the standard orthographic clues are not available for sentence segmentation.

5.3.2 Evaluation Metrics

To evaluate syntax we use the LAS metric that was defined in the previous chapter, i.e., the percentage of tokens that received their correct governor and arc label. For sentence boundaries we use F_1-measure over *tokens that start sentences*. That is, a true positive is defined as a token that was correctly predicted to begin a new sentence. False positives are erroneously predicted sentence-starts, and false negatives are sentence-starts that were not introduced. Precision and recall can then be computed following standard procedure and combined to form the F_1.

5.3.3 Instantiation

Just like the transition system for joint sentence segmentation and dependency parsing is an extension of the system seen in the previous chapter, so is the instantiation. With respect to the framework from Chapter 2, a number of the required functions are obvious given the discussion earlier in this and the previous chapter. Specifically, the functions PERMISSIBLE and ORACLE were discussed in Section 5.2. As for update methods and

training epochs, these will be varied and evaluated in the next section where the experiments are presented. We will use a default beam size of 20, but will make it explicit when we vary it for certain experiments. Two functions that require a bit more description, the loss function and the feature extraction function, are discussed in detail below.[2]

Loss Function. The loss function extends the loss function used for the SwapStandard system by adding loss on erroneous SentBound transitions. Equation 5.1 shows this at a high level, where $\text{LOSS}_{\text{SwapStandard}}$ denotes the loss function from the previous chapter.

$$\text{LOSS}(\tilde{\mathbf{z}}, \hat{\mathbf{z}}) = \text{LOSS}_{\text{SwapStandard}}(\tilde{\mathbf{z}}, \hat{\mathbf{z}}) + 2 \cdot \text{SENTBOUNDLOSS}(\tilde{\mathbf{z}}, \hat{\mathbf{z}}) \tag{5.1}$$

The loss on SentBound transitions is computed as the sum of false negative and false positive sentence-initial tokens in the prediction. That is, the tokens that were erroneously flagged as beginning a new sentence (false positives) and the ones that were not flagged although they begin a new sentence in the training data (false negatives). Since the sentence boundary transitions are relatively infrequent compared to the other transitions its contribution to the total loss is rather small. We therefore scale the sum by a factor of two in Equation 5.1.

Feature Extraction. The feature extraction function is, again, an extension of the function from the sentence-based parser seen in the previous chapter. We added two types of feature templates that are particularly relevant to the sentence segmentation task. First of all, the original feature set does not include linear context on the first item in the buffer. This means that a punctuation symbol that ends a sentence is not visible to the feature extraction function once it has been attached to its governor. In such a situation, where the next token on the buffer also begins a new sentence, the obvious clue for predicting a sentence boundary is no longer visible to the feature function. We therefore added templates predicated on the linear context of the next item in the buffer, extracting surface forms and part-of-speech tags.

The second addition to the feature set is meant to target the typical orthographic clues used in copy-edited texts. Here, surface forms are reduced to a *shape* representation that abstracts surface forms into the sequence of uppercase, lowercase, numerical, and punctuation characters. These features primarily help capturing the fact that next sentences tend to begin with uppercase letters in properly edited text.

[2]To facilitate transparency and reproducibility we point out once again that the implementation used for the experiments is available on the author's website.

5.3.4 Baselines and Pipelines

For sentence boundary detection we briefly consider two established baselines from the literature: The rule-based tokenizer from the Stanford CoreNLP[3] (Manning et al., 2014), denoted CORENLP, and the sentence boundary detector from OpenNLP,[4] denoted OPENNLP. The latter is based on a binary maximum entropy classifier and classifies whether a token, or more specifically, a punctuation symbol marks the end of a sentence. More generally, CORENLP and OPENNLP only target punctuation marks and are therefore inapplicable to data like Switchboard that does not include punctuation. In such cases CRF taggers are commonly selected as baselines, either in the context of sentence segmentation task (Evang et al., 2013) or also for the related punctuation prediction task (Zhang et al., 2013a). We therefore introduce a third baseline using MarMoT (denoted MARMOT). During preliminary experiments we tested MARMOT in two settings: In the first setting, tokens were labeled with a binary scheme that indicated whether a token starts a new sentence or not. In the second setting, we augmented the binary labels with the part-of-speech from the training data, thus jointly predicting sentence boundaries and part-of-speech tags. As the second setting performed slightly better, this is the one we selected for the experiments.

Table 5.2 shows the accuracies of these three baseline systems on the development sets. There are two main points to take away from the table: First, predicting sentence boundaries for WSJ* and the Switchboard dataset is a much more difficult task than for well-formatted text like the WSJ. Second, MARMOT is a strong baseline on the data set (WSJ) where we can compare it to the state-of-the-art. We thus conclude that we can proceed with the experiments in the next section using MARMOT as a baseline for sentence boundary detection.

	WSJ	Switchboard	WSJ*
OPENNLP	98.09	–	–
CORENLP	98.60	–	–
MARMOT	98.21	71.78	52.82

Table 5.2: Results (F_1) for baselines for sentence boundary detection on development sets.

Although we can compare the joint system with the MARMOT baseline in terms of

[3] We use version 3.5.2 of CoreNLP.

[4] `http://opennlp.apache.org`; We use version 1.5.0. Additionally, we found that 70% of the errors on the development set were due to incorrectly introduced sentence boundaries before quotation marks. We therefore postprocessed the output and attach quotation marks back to previous sentences.

sentence boundary prediction, such a comparison does not isolate the effect of syntax for sentence boundary detection. If the joint system performs better than the baselines, it is not clear whether this effect stems from syntax or whether it stems from differences in the machine-learning method (e.g., perceptron vs. CRF) and/or algorithmic framework (sequence labeling vs. transition system). To properly measure the effect of syntactic information on the sentence boundary detection task, we therefore introduce an additional baseline that is based on the joint system. The important difference is that when we trained it we replaced the gold-standard trees with trivial trees that connect the last token of each sentence to the root node, and everything in between as a left-branching chain. We dub this setting NOSYNTAX and it will allow us to use exactly the same machine learning for a fair comparison between a system that has access to syntax and one without.

For parsing, the obvious baseline is to run the parser without the sentence segmentation option (i.e., falling back to the SwapStandard system) on pre-segmented sentences from either of the sentence boundary detection baselines. The parser is then trained on the single sentences in the training data as we did in Chapter 4. These baselines can then be compared to the joint system (denoted JOINT) in terms of parsing accuracy. A third option is to use the joint model only as a sentence segmenter, and then apply a sentence-based parser on the output of the joint model as in the case for the baseline (denoted JOINT-REPARSED). Finally, to asses the role of erroneous sentence segmentation for parsing we will also consider feeding gold standard sentence boundaries to the parser in order to get an upper bound on the parsing performance (denoted GOLD).

Figure 5.5 summarizes the different pipelines that we will experiment with from start (raw documents) to end (sentences with dependency trees). The input is first fed into the document-based part-of-speech tagger, which is then fed to the NOSYNTAX baseline and the JOINT system. MARMOT does not use part-of-speech tags when it acts as a sentence boundary detection baseline. After sentence segmentation, the sentences are fed through vanilla part-of-speech tagging and parsing steps. For these cases, the part-of-speech tagger was trained in the normal way on sentences, and the parser was trained on sentences that were jack-knifed for part-of-speech tags, using the default setup from the previous chapter (i.e., using a beam size 20 and max-violation updates).

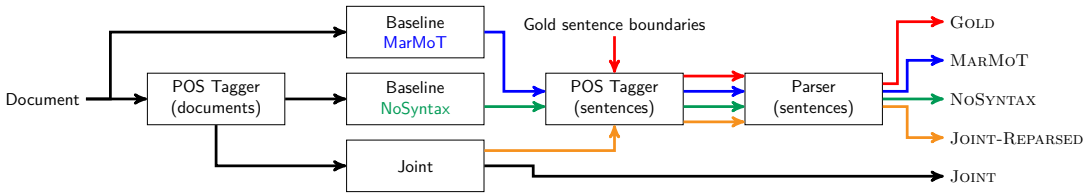


Figure 5.5: Overview of the different pipelines used in the experimental evaluation.

5.4 Experimental Evaluation

We now turn to the experiments in this chapter. From the machine learning perspective the main question is how the beam search parser fares in comparison to the greedy parser with regard to the different update types. We will then look more closely into the role of syntax for the task of sentence segmentation, by comparing the joint system to a set of baselines that do not utilize syntax.

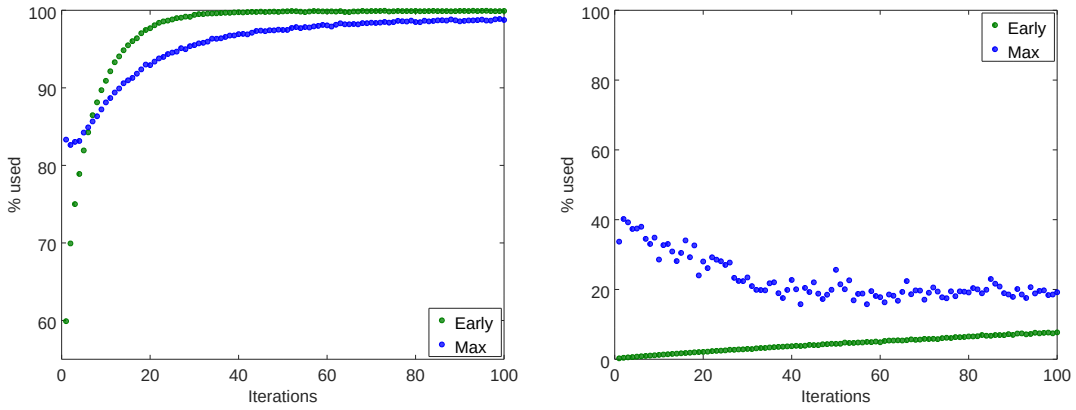
5.4.1 Update Methods

In comparison to a regular transition-based parser for sentences, the joint parser faces considerably longer transition sequences. As for the update methods we discussed in Chapter 2, early and max-violation updates do not commit to using the full sequences during training. In standard sentence-level tasks such as part-of-speech tagging or sentence-based dependency parsing these updates suffice and reasonably quickly reach a level where all or almost all of the training sequences are used for training. An entire document, however, may be composed of tens or hundreds of sentences, leading to transition sequences that are orders of magnitude longer.[5]

To illustrate the difference between sentence-level and document-level parsing Figure 5.6 displays plots of the average percentage of the gold training sequences that are being used as training progresses on the Switchboard training set.

The left plot shows the parser trained on sentences, the right one when it is trained on documents (where it also has to predict sentence boundaries). On the sentence level we see that both update strategies quite quickly reach close to 100%, i.e., they see more or less complete transition sequences during training. On the document level the picture is considerably different. The average length of seen transition sequences never even

[5] As noted in Chapter 4, the transition system requires a minimum of $2n$ transitions to parse a sentence of length n. In the Switchboard training set the average number of tokens per sentence is 9.97, while the average number per document is about 1300, translating into a minimum of 20 and 2600 transitions per training instance, respectively.



(a) sentences **(b)** documents

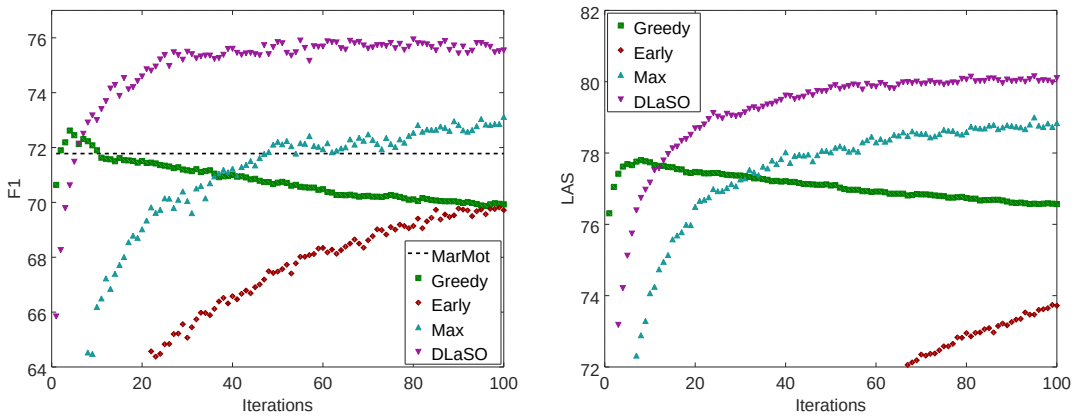
Figure 5.6: Average length of training sequences used during training on Switchboard for early update and max violation. Shown for a parser trained on sentences only in (a) and on full documents in (b).

goes above 50%. In other words, more than half of the training data is never used. Early update tends to exhibit a slow increase over time, presumably because the parser sees a bit more of every training instance at every iteration and therefore advances. However, max violation starts off using much more training data than early update, but then drops and settles around 20%. This illustrates the fact that max violation does not commit to exploiting more training data, but rather selects the update which constitutes the maximum violation irrespective of how much of the instance is being used.

Empirically, even though the percentage of used training data decreases over iterations, max violation is still profiting from more iterations in the document-level task. Figure 5.7 shows learning curves of the different training algorithms where sentence boundary F_1 and parsing accuracy LAS are plotted as a function of training iterations. The plots display performance for early update, max-violation, and DLaSO updates. For reference, we also include a greedy version of the parser. The greedy parser uses a plain averaged perceptron classifier that is trained on all the training data. The straight dashed line corresponds to the MARMOT baseline.

While the greedy parser, DLaSO, and the MARMOT baseline all exploit the full training data during training, it now becomes clear that early update and max-violation suffer from not doing so. DLaSO reaches a plateau rather quickly, whereas, even after 100 iterations, early update and max-violation perform considerably worse.[6] We also see that the

[6]We cut the x-axis at 100 iterations for simplicity. Although early and max-violation still seem to be growing, the main result remains. Even after 200 iterations DLaSO outperforms them by two points absolute.



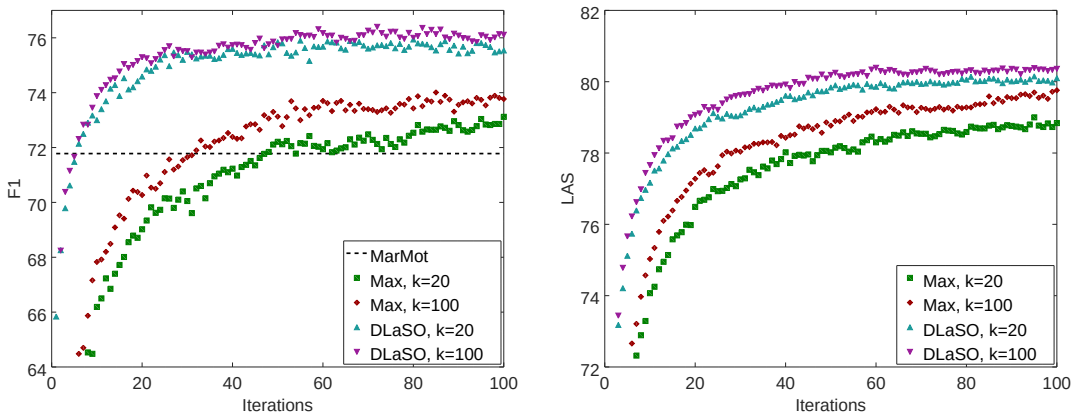
(a) sentence boundary detection **(b)** parsing

Figure 5.7: Performance of different update strategies on the Switchboard development set. Sentence boundary F_1 is shown in (a) and parsing LAS is shown in (b).

greedy parser quickly reaches an optimum and then starts degrading, presumably due to overfitting. It is noteworthy that max-violation needs about 40 to 60 iterations until it reaches a level similar to the greedy parsers optimal value. This effect is quite different from single sentence parsing scenarios, where it is known that beam search parsers easily outperform the greedy counterparts, requiring not nearly as many training iterations.

Increasing the beam size. Intuitively, using a bigger beam size might alleviate the problem of discarded training data and enable max-violation to exploit more training data. Figure 5.8 shows the sentence boundary F_1 and the LAS score for parsing as a function of training iterations for the default beam size 20 and a larger beam size of 100, comparing DLaSO and max-violation. The general impression is the same as when we compared beam sizes for coreference resolution (cf. Section 3.4) – a greater beam size improves the results slightly, but it does not change the internal ranking between the update methods. This improvement from increasing the beam size is more pronounced for max-violation than for DLaSO, but it is nonetheless there also for DLaSO. The parsing results might suggest that max-violation with beam size 100 would catch up with DLaSO after more than 100 iterations, but this is not the case. At around 100 iterations max-violation levels out and continues as a rather flat curve, never overtaking the DLaSO systems.[7] In theory a beam size orders of magnitude greater may rival DLaSO but as the beam size directly influences the time complexity of the parser, this is not a viable option.

[7] Again, we cut the x-axis at 100 iterations for brevity.



(a) sentence boundary detection **(b)** parsing

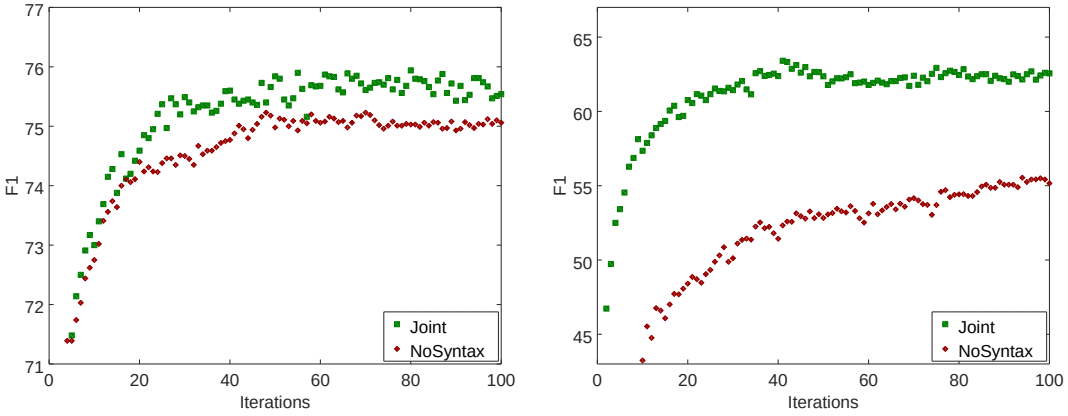
Figure 5.8: The effect of increasing beam size from 20 to 100 on the Switchboard development set. Sentence boundary F_1 is shown in (a) and parsing LAS is shown in (b).

5.4.2 Sentence Boundary Detection

One of the underlying assumptions of the joint model is our expectation that access to syntactic information should support the model in finding the sentence boundaries. The plots in the previous subsection suggest that on the Switchboard development set the joint parser, when trained using the appropriate update method, outperforms the MARMOT baseline by a big margin in terms of sentence boundary F_1 (cf. Figures 5.7 and 5.8).

However, as mentioned previously, we cannot rule out that the effect comes from syntax alone since they are inherently different along several axes. Additionally, we have argued that the syntactic complexity in Switchboard is rather low and therefore introduced the WSJ* data set, i.e., the WSJ where we removed all punctuation and lowercased all words. Figure 5.9 shows sentence boundary F_1 comparing the two versions of the transition system – NOSYNTAX and JOINT – on the development sets over training iterations when training with and without access to syntax. On both data sets, the system with access to syntax stays consistently above the other systems. The striking difference between the data sets is that syntactic information has a much bigger impact on WSJ* than on Switchboard, which we attribute to the higher syntactic complexity of newswire text. Overall the comparison indicates clearly that syntactic structure provides useful information to the task of sentence boundary detection.

Finally, we make an evaluation using the test sets of each data set, comparing not only with NOSYNTAX, but also with MARMOT. We optimize the number of iterations on



(a) Switchboard **(b)** WSJ*

Figure 5.9: The effect of syntactic information on sentence boundary prediction shown by comparing JOINT and NOSYNTAX. Evaluated on the developments sets for Switchboard in (a) and WSJ* in (b).

the development sets: for the joint model we take the iteration with the highest average between F_1 and LAS, NOSYNTAX is tuned according to F_1. Table 5.3 gives the performance of the sentence boundary detectors on test sets. We test for significance using the Wilcoxon signed-rank test with $p < 0.01$. † and ‡ denote significant increases over MARMOT and NOSYNTAX, respectively.

	WSJ	Switchboard	WSJ*
MARMOT	97.64	71.87	53.02
NOSYNTAX	98.21	76.31†	55.15
JOINT	98.21	76.65†	65.34†‡

Table 5.3: Sentence boundary detection results (F_1) on test sets.

On WSJ all systems are close to 98 and these high numbers once again affirm that the task of segmenting newspaper-quality text does not leave much space for improvement. Although the parsing models outperform MARMOT, the improvements in F_1 are not significant. In contrast, all systems fare considerably worse on WSJ* which confirms that the orthographic clues in newspaper text suffice to segment the sentences properly. Although NOSYNTAX outperforms MARMOT, the difference is not significant. However, when real syntax is used (JOINT) we see a huge improvement in F_1 – 10 points absolute – which is significantly better than both NOSYNTAX and MARMOT.

On Switchboard MARMOT is much lower and both parsing models outperform it

significantly. Surprisingly the NOSYNTAX system achieves a very high result beating the baseline significantly by almost 4.5 points. The usage of syntax in the JOINT model raises this gain to 4.8 points.

All in all, the results suggest that (i) the transition system is at least as strong, and sometimes stronger, than an off-the-shelf sequence labeler for sentence boundary detection. Whether this is because of the features, machine-learning algorithm (perceptron vs. CRF), or the algorithmic treatment (beam search transition system vs. sequence labeling) is not obvious and would require further inquiry. Moreover, (ii) when dealing with properly copy-edited text, sentence boundary detection is essentially trivial. However, (iii) when dealing with sentences that are sufficiently syntactically complex (as in WSJ*, but not in Switchboard), syntax *does* provide additional information that improves sentence segmentation.

5.4.3 Parsing Results

In order to evaluate the joint model on the parsing task separately we compare it to pipeline setups. In addition to a pipeline with the baseline sentence boundary detectors – i.e., MARMOT and NOSYNTAX – we include two more segmentations. First, gold sentence boundaries provide an upper bound to the parsing performance when sentence boundaries do not pose a problem (denoted GOLD). Second, we also build a pipeline where we use JOINT only as a sentence segmenter and then parse once again (denoted JOINT-REPARSED).

Table 5.4 shows the results on the test sets. Again, we test for significance using the Wilcoxon signed-rank test with $p < 0.01$. † and ‡ denote significant increases over MARMOT and NOSYNTAX, respectively. Additionally, * denotes significant increases over JOINT. For WSJ, where sentence segmentation is almost trivial, we see only minor drops in LAS between GOLD and the systems that use predicted sentence boundaries. Among the systems that use predicted boundaries, no differences are significant.

For WSJ* and Switchboard the picture is much different. Compared to GOLD, all systems suffer considerable drops in accuracy which asserts that errors from the sentence boundary detection task propagate to the parser and worsen the parser accuracy. On Switchboard the scenarios where a parser is involved in sentence segmentation (including NOSYNTAX) result in significantly better results than when MARMOT is responsible for sentence segmentation. The best result is obtained after reparsing and this is also significantly better than any other system. Although there is a slight drop in accuracy between NOSYNTAX and JOINT, this difference is not significant.

	WSJ	Switchboard	WSJ*
GOLD	90.22	84.99	88.71
MARMOT	89.81	78.93	83.37
NOSYNTAX	89.95	80.30†	83.61
JOINT	89.71	79.97†	85.66†‡
JOINT-REPARSED	89.93	80.61$^{\dagger\ddagger *}$	85.38†‡

Table 5.4: Parsing results (LAS) on test sets for different sentence boundaries.

The results on WSJ* show that not only does syntax help to improve sentence segmentation, it does so to a degree that parsing results deteriorate when simpler sentence boundary detectors are used. Here, both JOINT and JOINT-REPARSED obtain significantly better parsing accuracies than the systems that do not have access to syntax during sentence boundary prediction. Although JOINT-REPARSED performs a bit worse, the difference compared to JOINT is not significant.

5.5 Related Work

The work presented in this chapter spans several domains of NLP. The connections to (transition-based) dependency parsing are obvious, and we refer to the related work from the previous chapter for a discussion on that. As previously stated, the sentence segmentation problem is often regarded as trivial when it comes to properly copy-edited text but becomes more relevant in the context of speech recognition or processing of user-generated content on the web. Historical manuscripts often do not use modern conventions in terms of capitalization and punctuation and here sentence segmentation is sometimes a tough challenge (Haug et al., 2009; Eckhoff et al., 2018). In fact, in the CoNLL 2017 Shared Task (Zeman et al., 2017) on multi-lingual end-to-end dependency parsing, a number of data sets fit this description particularly well (e.g., treebanks for Ancient Greek, Old Church Slavonic, Gothic, and Latin). While predicted sentence boundaries were provided by the shared task organizers, for a select number of data sets we applied the system from this chapter and obtained the best results on sentence segmentation overall (Björkelund et al., 2017).[8]

[8] As for a textual example for historical manuscripts, we remind the reader about the Bible excerpt in Figure 1.5 in Chapter 1.

5.5.1 Sentence Segmentation and Punctuation Prediction

Previous work on sentence segmentation often assumed edited text with proper punctuation and capitalization. The task is then often reduced to *period disambiguation* or *punctuation disambiguation*, where the challenge boils down to deciding whether a given period (or other punctuation symbol) denotes the end of a sentence or not. In this case, the task is sometimes tackled in rule-based manners, such as the Stanford CoreNLP (Manning et al., 2014), which we used as a baseline. A classical machine learning approach is MxTerminator (Reynar and Ratnaparkhi, 1997) which uses a maximum entropy model to classify punctuation as either ending a sentence or not. The OpenNLP baseline we used is a re-implementation of this approach. A third option is unsupervised approaches such as Kiss and Strunk (2006) who explicitly target full stops based on statistics from large corpora. Similarly, Schmid (2000) gathers large statistics on full stops and identifies likely abbreviations that do not end a sentence. In the more general case, where any token can end a sentence, neither of these methods are, however, applicable as they focus on punctuation symbols. The standard approach is then to use sequence labelers (Evang et al., 2013; Dridan and Oepen, 2013), which is the same approach that we used for the MarMoT baseline. Read et al. (2012) provide a broad survey of methodology and tools of both paradigms.

When it comes to non-edited text, Read et al. (2012) show that the performance of state-of-the-art sentence boundary detectors drops when moving from processing newspaper-quality texts into user-generated content. Dridan and Oepen (2013) point out the importance of shifting towards practical use cases and evaluating the end-to-end performance, starting with finding sentence boundaries up to higher-level tasks like parsing. Additionally, the sentence segmentation problem has frequently come up in literature on speech recognition. Since speech recognizers typically do not produce punctuation, the related task *punctuation prediction* has been extensively studied (Matusov et al., 2006; Lu and Ng, 2010; Cho et al., 2012). The overall layout is that a speech recognizer produces a sequence of words, and then the next step in a pipeline inserts punctuation, such that further processing can be carried out. The underlying motivation here is that steps involved in further processing are trained on properly edited texts that includes punctuation, and the punctuation therefore must be inserted.

Perhaps the work most similar to ours is that of Zhang et al. (2013a). They present a projective transition-based parser that jointly predicts punctuation and syntax, with the intended use on spoken transcripts. Their ArcEager transition system (Nivre, 2003) includes an additional transition that introduces punctuation similar to our SentBound

transition. They also use beam search but with early updates and, unlike us, circumvent the problem of long training sequences by chopping up the training data into pseudo-documents of at most 10 sentences. Our analysis comparing early updates and DLaSO suggests that their solution works because the training instances sufficiently small.

5.5.2 Joint Models

Pipelines of NLP components are common both for practical applications and in the research literature. Pipelines are attractive because components can be built and evaluated in isolation. Nevertheless, the interaction between the different levels in a pipeline is unavoidable and constructing two levels of annotation as separate steps in a pipeline means that information cannot flow between the two levels. In practice, such a separation has two negative implications: On the one hand, errors happening at an early stage in the pipeline will propagate and provide problematic input to subsequent steps. This became evident in this chapter as parsing results deteriorated when weak baselines were used for the sentence segmentation. On the other hand, the levels that are predicted later in the pipeline are unavailable to the earlier steps, which was apparent in this chapter when sentence boundaries were predicted jointly with syntax.

Joint models, NLP components that solve two or more steps of a pipeline simultaneously, are the commonly proposed remedy to these problems. Joint models are appealing because information can flow between the levels at prediction time and often lead to overall better models, especially for the lower level tasks that suddenly have access to syntactic information. Since coupling two tasks by definition inflates the search space, joint models come at a cost. Either in terms of increased computational complexity with regard to space and/or time or by sacrificing the scope of features in the joint output structure. In some cases joint models also require considerable adaptations of basic algorithms that can be used to solve the component tasks individually.

Transition-based joint models have recently received considerable attention in the literature. They are appealing due to their incremental approach and their low theoretical complexity. Transition systems that jointly carry out part-of-speech tagging and parsing (Hatori et al., 2011; Bohnet and Nivre, 2012), or even tokenization, tagging and parsing (Zhang et al., 2013c, 2014b), have been demonstrated. Joint models are however not restricted to transition-based dependency parsing. Constituency parsers traditionally predict part-of-speech tags as part of the parsing process, and also graph-based dependency parsers have been extended to joint modeling. For instance, Seeker and Kuhn (2013) present a graph-based dependency parser that predicts the syntactic structure jointly

with part-of-speech and morphology. Seeker and Çetinoğlu (2015) extend this further and additionally deal with tokenization problem in Hebrew and Turkish, languages (or rather, scripts) where the segmentation problem at the token level also is challenging.

5.6 Conclusion

In this chapter we have extended the SwapStandard transition system to also carry out sentence segmentation. By adding a new transition to flag the beginning of a sentence, the transition system can process documents rather than sentences and output a list of sentences coupled with their own dependency trees. We described the semantics of this transition system along with its preconditions. We introduced preconditions for Shift which requires special attention as it may interact with the Swap transition at the boundaries of sentences.

When it comes to the machine learning framework this chapter was concerned with the role of the update methods. We showed that, unlike in the case where only sentences are parsed, early and max-violation updates are inadequate for training the joint system as they fail to outperform a greedy baseline. The reason is that the transition sequences grow very large and that these update methods therefore are unable to capitalize on all the training data. While a larger beam size potentially could compensate for this, we demonstrated that this is not the case, at least not if the beam size is kept somewhat reasonable.

We would like to emphasize that the contrast between the sentence-based parser, seen in the previous chapter, and the document parser seen in this chapter is mostly related to the length of the training instances. In general, it could be argued that the parsing problems encountered in this and the previous chapter are roughly of the same difficulty. But the fact that the training instances are much longer implies that the document parser requires update methods that exploit all available training data. This suggests that it is not just the inherent difficulty, or hardness, of a task that governs which update methods are required for training the perceptron with approximate search. In addition to the hardness, the length of the training instances must be taken into account when selecting among available update.

From the perspective of computational linguistics we have shown that syntax can be beneficial for sentence boundary detection. In order to demonstrate this effect we relied on two standard English data sets – the Wall Street Journal, an example of properly edited newswire text, and Switchboard, a corpus of transcribed telephone conversations.

Since the Wall Street Journal includes standard orthographic clues such as punctuation and capitalization sentence segmentation is more or less trivial. We therefore also modified this data set by lower-casing and deleting sentence-final punctuation, thus removing the obvious clues. In this setting, we have shown experimentally that the joint system indeed outperforms the other sentence boundary detection baselines. We can thus conclude that, under the right circumstances, *syntax is helpful for sentence segmentation*. In that experiment, the difference was so pronounced that it carries over to the results for parsing, i.e., with weaker sentence segmentation the parsing quality also degrades significantly.

For the Switchboard corpus the results are not as clear-cut. While the joint system outperforms the CRF baseline in terms of sentence segmentation, this result comes primarily from a stronger feature set in the joint system. We arrived at this conclusion by running the joint system in an artificial setting, where the syntax trees were replaced with trivial trees, allowing us to compare the situations when syntax is present or not. The results indicate that comparing the joint system with or without syntax on Switchboard makes almost no difference. We attribute this fact to the low syntactic complexity in Switchboard, where the sentences tend to be rather short and simple.

Chapter 6

Conclusion

This dissertation has focused on *structured prediction* in the Natural Language Processing domain. The tasks we have focused on – coreference resolution, dependency parsing, and joint sentence segmentation and dependency parsing – are typical examples of structured NLP tasks that are active topics of research. We have argued for handling these tasks with a **structured** model, where a model is *globally* trained to predict a complete coreference assignment or dependency tree, as opposed to using a simpler, *locally* trained classifier. One obvious, but nevertheless important, advantage of a structured approach is that the model is biased towards the overall output structure rather than focusing on making the best local decision. Single local decisions may be good, but when they are wrong they tend to have a cascading effect known as error propagation. Another advantage of structured models is that they can exploit structural context for feature extraction in a realistic setting. That is, although a classifier could be trained to use some structural context, this is typically accomplished by providing the classifier with gold standard features from the structural context. This has a tendency of exacerbating error propagation as it does not properly emulate the test time scenario.

In addition, we have dealt with tasks where prediction takes place in a larger prediction space than the actual output space. In the coreference case we predict trees that are subsequently mapped to coreference partitions, whereas for dependency parsing and joint sentence segmentation and dependency parsing we predict transition sequences that are subsequently mapped to the corresponding output structures. Since the relationship between prediction and output space is many-to-one, this raises the core question of which prediction structure to learn. The easiest way to handle this issue is to define a set of rules that creates a canonical prediction structure and use this as the ground truth for learning. As an example, for coreference resolution this would amount to deciding a

canonical antecedent for each coreferent mention. We have in this dissertation, however, taken a different approach where the ground truth is left **latent** and the machine-learning algorithm decides which prediction structure to learn. In the coreference example, this implies that the antecedents of coreferent mentions are underspecified and defined by the machine-learning model. The intuition is that the latent structures should be easier to learn and generalize better. While this tactic has been shown very successful for coreference resolution, transition-based dependency parsing has typically relied on the easier solution with a single canonical prediction structure.

The third theme that runs throughout this dissertation is **search**. Search is relevant not only at test time, but also in order to create latent representations for learning. In order not to impose any restrictions on the expressivity of feature functions, we must surrender to the fact that the search problem – i.e., finding the highest scoring prediction (or latent) structure – cannot be solved exactly. To this end, we have applied beam search as the approximate search method. The combination of beam search with the structured perceptron has been used successfully for many diverse problems in the NLP domain. It has, however, also been demonstrated that particular update methods are required to combat search errors during learning and multiple update methods have been proposed (Collins and Roark, 2004; Daumé III and Marcu, 2005; Huang et al., 2012; Björkelund and Kuhn, 2014).

6.1 Contributions

In this dissertation we have formalized a learning and search procedure into an abstract framework. This framework comes with knobs and switches, e.g., update methods and hyperparameters such as beam size. We have then instantiated this framework for each of the NLP tasks that we deal with. Depending on the task and based on previous work, we have chosen to turn and flip some knobs and switches for the different tasks. This has enabled a careful and systematic analysis of these methods using the NLP tasks as examples that exhibit certain intrinsic properties that may lend them more or less amenable to these techniques. Along the way we have also had to develop new ideas, both with regard to learning in general, as well as with regard to the specific tasks. The work we have presented thus sits at the intersection between machine learning and NLP. We discuss these aspects separately below.

Contributions with respect to NLP. In Chapter 3 on **coreference resolution**, we started from Fernandes et al.'s (2012) strong latent tree model. This model was very effective and

established a new state of the art when it was presented. It uses the perceptron to derive latent structure but since features are restricted to two mentions at a time (i.e., it is an *arc-factored* model), it has poor access to contextual features in the prediction. This has the advantage that the search problem can be solved exactly, but the limited scope of features prevents the model from accessing features on other, partially processed mentions in an input document when considering a pair of mentions. By applying beam search and the structured perceptron, we have shown how this model is compliant to include features that can exploit a greater portion of the prediction structure. Instantiated in our framework, an arc-factored model (such as our arc-factored baseline) can be seen as a special case of the more general entity-mention model, where a mention is compared to the full partial structure that has so far been predicted. This enabled us to make controlled experiments that empirically demonstrate that the more feature-rich entity-mention model outperforms the feature-poor mention-pair model for three data sets.

Latent structure has received less attention in the **dependency parsing** community, a topic we picked up in Chapter 4. For classifier-based greedy parsers, dynamic oracles have been designed and used to show promising results (Goldberg and Nivre, 2013, *inter alia*). But greedy parsers are known to underperform compared to their search-based counterparts, and latent transition sequences has previously not been considered for beam search parsers. Moreover, for the particular SwapStandard system (Nivre, 2009) we considered, no dynamic or non-deterministic oracle was previously known. We therefore designed a non-deterministic oracle for SwapStandard enabling us to instantiate the learning framework. The empirical results, carried out over ten different treebanks, corroborate previous findings that non-deterministic oracles can improve greedy dependency parsers (Goldberg and Nivre, 2013). In the beam search setting, however, latent transition sequences by and large perform on the same level as fixed canonical sequences. That is, the role of latent structure for dependency parsing is dependent on the choice between greedy, locally normalized models on the one hand, vs. globally normalized models with beam search inference on the other. Put somewhat differently, globally normalized beam search parsers seem to be more lenient towards training with a single ground truth than their greedy counterparts.

The third task covered in this dissertation (Chapter 5) was **joint sentence segmentation and dependency parsing**. We approached this task using the transition-based paradigm again and extended the SwapStandard system to also predict sentence boundaries. This extension required careful handling of the preconditions for the transitions (i.e., defining the PERMISSIBLE function) to ensure not just that the sentences are continuous sequences of tokens but also that each dependency tree covers all and only all tokens

of its associated sentence. A major motivation for exploring such a transition system was to test the hypothesis that syntax can be helpful to sentence segmentation. The experimental setup required a thoughtful design of baselines in order to isolate this effect. Moreover, the data sets had to be chosen carefully, as sentence segmentation in properly copy-edited text is next to trivial. The empirical evaluation confirmed the hypothesis and showed that, in the absence of standard orthographic clues, the transition system outperforms the baselines, thereby demonstrating that syntax can be helpful for sentence segmentation.

Contributions with respect to machine learning. In this dissertation we have strung together the combination of beam search with the structured perceptron into a joint abstract framework, as defined in Chapter 2. While some of the bits and pieces of this framework are not novel in themselves, the abstraction has enabled us to carry out systematic experiments that evaluate certain properties in a broader perspective.

Using the NLP tasks we have utilized the framework to inspect particular corner cases of the learning machinery. The selection of tasks has allowed us to move along certain boundaries that have previously not been crossed in the literature. Specifically, in Chapter 3, where we dealt with coreference resolution, we considered the case of moving from **exact search** to **approximate search**, trading the exactness of search for richer features. The empirical results demonstrate that the richer feature set are only able to outperform the arc-factored baseline when learning is carried out using appropriate update methods. That is, although richer features should intuitively improve the performance of a machine-learning system, the move to approximate search may counteract this advantage unless the training regime is appropriately adapted as well.

In Chapter 4, devoted to transition-based dependency parsing, we explored the behavior of the learning machinery along another boundary. Concretely, we moved from learning **canonical, uniquely defined transition sequences** to **latent sequences**. As we compared a greedy classifier-based baseline to a beam-search system, we found that latent structure plays different roles in the two settings – in the greedy case it improves performance, whereas the beam-search parser does not profit (nor suffer) in any considerable manner compared to canonical sequences. This suggests that the advantage of latent structures is consumed by the advantage of richer search methods for this task. More generally, this reminds us that we should not assume that seemingly orthogonal improvements to a single baseline system combine additively in terms of performance.

The empirical evaluation in Chapter 5, where we tackled joint sentence segmentation and dependency parsing, can also be seen as a way of moving across a boundary. Here,

we moved from the **moderately-sized sequences** in Chapter 4 to **very long** ones. Compared to a greedy baseline, applying beam-search to the transition system may lead to inferior results. As was the case for coreference, the update methods must be chosen with care in order to profit from the increased power of a structured search-based model. If we assume for a moment that sentence segmentation is a fairly easy problem, then the primary difference between the two tasks of joint sentence segmentation and dependency parsing and dependency parsing lies in the length of the sequences. At a higher level the lesson here is that the length of instances must be taken into consideration when making choices about learning and decoding.

In a meta-perspective, however, one of the most important issues we have uncovered using this framework is the role of **discarded training** data when training with approximate search. As Huang et al. (2012) proposed max-violation (and other) updates they observed that the proposed updates tend to converge faster than early updates. As we have shown in Chapters 3 and 5, it is clear that for some tasks the rate of discarded training data can be so severe that the richer models with approximate search fail to outperform simpler baselines. We have also demonstrated that this problem cannot straightforwardly be handled by increasing the beam size. Instead, we employed update methods that do not discard any training data and provided experimental results that indicate that these are required. LaSO (Daumé III and Marcu, 2005) fits this description, but in Chapter 3 we argued that LaSO-style updates, which provide feedback within training instances, poorly mimic the setting encountered at test time, where no such feedback is available. We therefore proposed the DLaSO updates. Empirically we have shown that DLaSO is at least as strong as LaSO and that it is able to outperform the simpler baselines considered in Chapters 3 and 5.

6.2 The Bigger Picture

As we are concluding this dissertation we would like to take a moment and regard the results contained herein in a broader perspective. What do the ideas and empirical results we have seen imply when viewed from a higher vantage point with respect to structured prediction in NLP? And, finally, given the recent rise of deep learning in NLP, how does the work in this dissertation relate to neural networks? We adress these questions below.

Learning with approximate search. The framework from Chapter 2 which we have instantiated in the content chapters of this dissertation has allowed us to apply the same learning machinery to multiple problems from the NLP domain. We have seen how

"simpler" update methods, such as early update, have had to give way for "stronger" methods such as max-violation and DLaSO. One of the core issues we have identified is the role of discarded training data. There are two fundamental aspects of a learning problem to take into consideration with regard to discarded training data. First, what is the length of an instance and what do the branching points in the search tree look like. Or, in other words, what is the **size** of the space of possible predictions. A smaller space or a shorter sequence reduces the amount of discarded training data. Second, the inherent difficulty, or **hardness**, of a task, subject to whatever features the model is offered. The easier a task is to learn, the less likely a correct solution is to fall of the beam during beam search.

Given these two parameters, we speculate that the relationship between (1) size-hardness of instances and (2) the choice of update methods can be visualized as in Figure 6.1. The plot shows a hypothetical view of the relationship between instance size (x-axis) and the hardness of a task (y-axis). The shaded areas are meant to illustrate the strength of the update methods. For some tasks, **standard** perceptron updates, i.e., full updates after a round of beam search, with or without search errors, seem to suffice. This is displayed as a strong pink shading in the bottom left corner of the figure. Example tasks include part-of-speech tagging (Huang et al., 2012) and Chinese word segmentation (Zhang and Clark, 2007). These problems appear to be short enough and easy enough that the tasks can be learned sufficiently well even in the presence of potential search errors despite the fact that standard updates may be invalid.

If the task difficulty and/or instance size is stepped up a notch, invalid updates are detrimental to performance. Huang et al. (2012) show this difference when comparing standard (potentially invalid) and early, max-violation, and latest updates for transition-based dependency parsing. As mentioned several times throughout this dissertation (Section 2.5 in particular), beam search with these types of updates have been successfully applied to several tasks, and we are inclined to believe that, at least in some of these cases, standard updates would have underperformed as well (i.e., as demonstrated by Huang et al. (2012) for transition-based dependency parsing). These tasks belong to a set where **partial** updates such as early, max-violation or latest, are strong enough to learn good predictors.

Finally, as we have shown in this dissertation, for some tasks even partial updates are deficient, and update methods that utilize *all training data* through **multiple partial** updates are required. LaSO or DLaSO are the updates that fit this description. The empirical evidence stems from Chapters 3 and 5, where we saw a major discrepancy between models using partial updates and multiple partial ones.

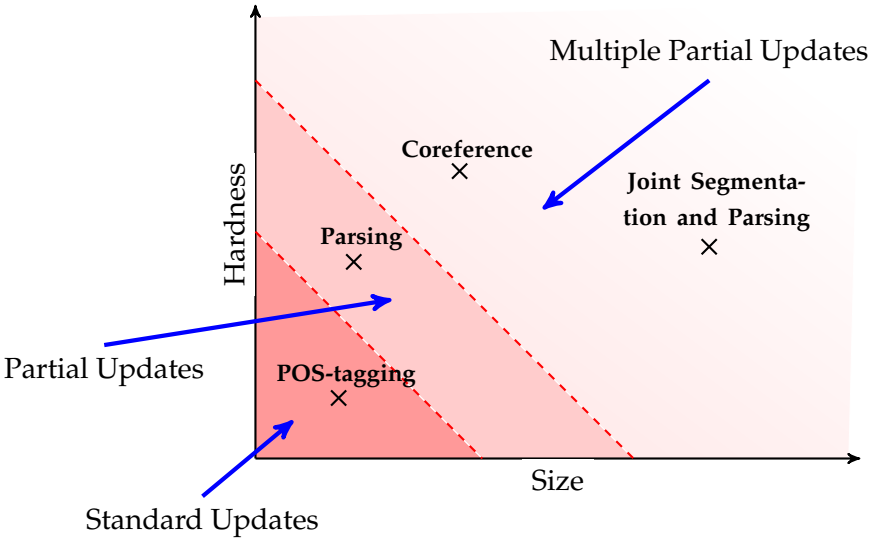


Figure 6.1: Hypothetical layout of the relationship between hardness of tasks and instance size vs. the update types required to learn strong predictors.

To summarize these observations, there seems to be a continuum of more or less difficult tasks, with relatively shorter or longer decision sequences that require a variable sophistication in terms of update types. Easier and shorter problems can get away with standard updates, intermediate problems require partial updates, whereas the toughest ones need to be treated with multiple partial updates. We should emphasize once again that this claim is somewhat speculative and grounded in empirical observations from this dissertation and previous work. Obviously, carrying out similar analyses on related tasks and mapping out their location in the plot from Figure 6.1 would contribute to our understanding of this continuum. Additionally, although we have empirically demonstrated the utility of DLaSO, it remains an open issue to consolidate this type of update with the convergence proofs known for the other updates in order to prove any theoretical guarantees on this method.

What about Neural models? Throughout this dissertation we have consciously abstained from discussing the recent rise of deep learning and neural networks in the NLP community. We are now ready to add a some final remarks on how this relates to the work presented in this dissertation. The upsurge of neural models has triggered great advances in NLP, both in terms of pushing the state of the art for established tasks forward, but we have also seen a dramatic increase in the use of techniques such as multi-task training, embeddings of words and characters, as well as multi-modal models.

Let us consider the task of dependency parsing, which can be regarded as one of the most competitive topics in the research community. One of the running themes in this dissertation has been the hunt for greater feature scope and clever search methods. For transition-based dependency parsing the move from greedy to beam search is a clear winner (cf. Chapter 4). For graph-based parsers, as we discussed briefly in Section 4.6, considerable effort has been made to increase the scope of these parsers to a wider contextual window, starting from arc-factored models ranging to models that see a wide neighborhood when scoring arcs. The neural revolution has brought considerable improvements to the state of the art of dependency parsing, and these days essentially all papers on the topic are based on neural models. Nevertheless, in contrast to the previous discussion on expressiveness of parser models and search, some of the strongest parsers presented in recent years are based on greedy search with a very limited scope of structural context for features. This includes both transition-based models (Kiperwasser and Goldberg, 2016; de Lhoneux et al., 2017), where greedy inference utilizing a very small neighborhood of features are used, as well as graph-based approaches (Kiperwasser and Goldberg, 2016; Dozat and Manning, 2017), that have returned to simple arc-factored models, where features are only drawn from two words. However, it should be noted that these models include representations derived through bi-directional long short-term memory (bi-LSTM) embeddings of the input and that these, by means of their impressive ability to encode and distill relevant information from sequential data, most likely make up for the lack of structural context to some degree.

The discussion above raises the question of how structured models with approximate search relate to neural models. First of all, we should say that the jury is still out – as a hot topic of research, further advances are bound to follow. However, given the recent improvements brought by neural models to various tasks in NLP, as well as in other ML-driven disciplines, neural models appear to have an inherent edge over traditional models in the first place. Most likely we are currently in a phase of sudden exploration and adaptation of these models, thereby yielding large improvements relatively quickly. However, we expect that over time the improvements from neural modeling alone will reach a certain plateau and we suspect that the fundamental issues of feature scope and search will return to scrutiny once the rapid advances of deep learning begin to slow down.

That said, research is still progressing on integrating neural methods with beam search. Recent work on transition-based dependency parsing replace the structured perceptron from our work with a feed-forward neural network and use beam search with a neural model that is globally normalized (Zhou et al., 2015). Andor et al. (2016) improved

the training regime of this approach and provide a proof that globally normalized models are strictly more expressive than locally normalized ones. Similar methods have been applied to incremental (transition-based) phrase-structure parsing (Watanabe and Sumita, 2015) and (transition-based) word segmentation (Zhang et al., 2016). Finally, and perhaps most interestingly, Wiseman and Rush (2016) present a seq2seq neural model that uses beam search and train their model using DLaSO updates as they found early updates to underperform. In this light, we conclude this dissertation by stating that, amid the current paradigm shift towards neural models, we are confident that the role of search and contextual features will remain vital for future research in our domain.

Bibliography

Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., Petrov, S., and Collins, M. (2016). Globally Normalized Transition-Based Neural Networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany. Association for Computational Linguistics. Cited on page 140.

Aone, C. and Bennett, S. W. (1995). Evaluating Automated and Manual Acquisition of Anaphora Resolution Strategies. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 122–129, Cambridge, Massachusetts, USA. Association for Computational Linguistics. Cited on page 72.

Asher, N. and Lascarides, A. (1998). Bridging. *Journal of Semantics*, 15:83–113. Cited on page 74.

Attardi, G. (2006). Experiments with a Multilanguage Non-Projective Dependency Parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 166–170, New York City, New York, USA. Association for Computational Linguistics. Cited on page 82.

Auli, M. and Lopez, A. (2011). Efficient CCG Parsing: A* versus Adaptive Supertagging. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1577–1585, Portland, Oregon, USA. Association for Computational Linguistics. Cited on page 45.

Bagga, A. and Baldwin, B. (1998a). Algorithms for Scoring Coreference Chains. In *In The First International Conference on Language Resources and Evaluation Workshop on Linguistics Coreference*, pages 563–566. Cited on pages 59 and 60.

Bagga, A. and Baldwin, B. (1998b). Entity-Based Cross-Document Coreferencing Using the Vector Space Model. In *36th Annual Meeting of the Association for Computational*

Linguistics and 17th International Conference on Computational Linguistics, Volume 1. Cited on page 74.

Barzilay, R. and Lapata, M. (2008). Modeling Local Coherence: An Entity-Based Approach. *Computational Linguistics*, 34(1):1–34. Cited on page 63.

Baumann, S. and Riester, A. (2012). Referential and Lexical Givenness: Semantic, Prosodic and Cognitive Aspects. In Elordieta, G. and Prieto, P., editors, *Prosody and Meaning*, volume 25 of *Interface Explorations*, pages 119–162. Mouton de Gruyter, Berlin, Germany. Cited on page 74.

Baumann, S. and Riester, A. (2013). Coreference, lexical givenness and prosody in German. *Lingua*, 136:16 – 37. SI: Information Structure Triggers. Cited on page 74.

Bengtson, E. and Roth, D. (2008). Understanding the Value of Features for Coreference Resolution. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 294–303, Honolulu, Hawaii, USA. Association for Computational Linguistics. Cited on pages 6 and 72.

Bergsma, S. and Lin, D. (2006). Bootstrapping Path-Based Pronoun Resolution. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 33–40, Sydney, Australia. Association for Computational Linguistics. Cited on pages 59 and 62.

Björkelund, A., Çetinoğlu, Ö., Faleńska, A., Farkas, R., Müller, T., Seeker, W., and Szántó, Z. (2014). The IMS-Wrocław-Szeged-CIS Entry at the SPMRL 2014 Shared Task: Reranking and Morphosyntax meet Unlabeled Data. In *Notes of the SPMRL 2014 Shared Task on Parsing Morphologically-Rich Languages*, Dublin, Ireland.

Björkelund, A., Cetinoglu, O., Farkas, R., Mueller, T., and Seeker, W. (2013). (Re)ranking Meets Morphosyntax: State-of-the-art Results from the SPMRL 2013 Shared Task. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 135–145. Association for Computational Linguistics.

Björkelund, A., Eckart, K., Riester, A., Schauffler, N., and Schweitzer, K. (2014). The Extended DIRNDL Corpus as a Resource for Coreference and Bridging Resolution. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*. European Language Resources Association (ELRA).

Björkelund, A., Faleńska, A., Seeker, W., and Kuhn, J. (2016). How to Train Dependency Parsers with Inexact Search for Joint Sentence Boundary Detection and Parsing of Entire Documents. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1924–1934, Berlin, Germany. Association for Computational Linguistics.

Björkelund, A., Falenska, A., Yu, X., and Kuhn, J. (2017). IMS at the CoNLL 2017 UD Shared Task: CRFs and Perceptrons Meet Neural Networks. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 40–51. Association for Computational Linguistics. Cited on page 128.

Björkelund, A. and Farkas, R. (2012). Data-driven Multilingual Coreference Resolution using Resolver Stacking. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 49–55. Association for Computational Linguistics. Cited on pages 64 and 72.

Björkelund, A. and Kuhn, J. (2012a). Comparing Non-projective Strategies for Labeled Graph-Based Dependency Parsing. In *Proceedings of COLING 2012: Posters*, pages 135–144. The COLING 2012 Organizing Committee.

Björkelund, A. and Kuhn, J. (2012b). Phrase Structures and Dependencies for End-to-End Coreference Resolution. In *Proceedings of COLING 2012: Posters*, pages 145–154. The COLING 2012 Organizing Committee.

Björkelund, A. and Kuhn, J. (2014). Learning Structured Perceptrons for Coreference Resolution with Latent Antecedents and Non-local Features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 47–57, Baltimore, Maryland, USA. Association for Computational Linguistics. Cited on pages 4, 39, 50, 71, 75, and 134.

Björkelund, A. and Nivre, J. (2015). Non-Deterministic Oracles for Unrestricted Non-Projective Transition-Based Dependency Parsing. In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 76–86, Bilbao, Spain. Association for Computational Linguistics.

Björkelund, A. and Nugues, P. (2011). Exploring Lexicalized Features for Coreference Resolution. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, pages 45–50. Cited on pages 62 and 72.

Block, H. D. (1962). The Perceptron: A Model for Brain Functioning. *Reviews of Modern Physics*, 34:123–135. Cited on page 43.

Blum, M., Floyd, R. W., Pratt, V., Rivest, R. L., and Tarjan, R. E. (1973). Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448 – 461. Cited on page 31.

Bohnet, B. (2010). Very High Accuracy and Fast Dependency Parsing is not a Contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97, Beijing, China. Coling 2010 Organizing Committee. Cited on page 25.

Bohnet, B., Björkelund, A., Kuhn, J., Seeker, W., and Zarriess, S. (2012). Generating Non-Projective Word Order in Statistical Linearization. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 928–939. Association for Computational Linguistics. Cited on page 43.

Bohnet, B. and Kuhn, J. (2012). The Best of BothWorlds – 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*, pages 77–87, Avignon, France. Association for Computational Linguistics. Cited on page 105.

Bohnet, B., Mille, S., Favre, B., and Wanner, L. (2011). <StuMaBa >: From Deep Representation to Surface. In *Proceedings of the Generation Challenges Session at the 13th European Workshop on Natural Language Generation*, pages 232–235, Nancy, France. Association for Computational Linguistics. Cited on page 43.

Bohnet, B. and Nivre, J. (2012). A Transition-Based System for Joint Part-of-Speech Tagging and Labeled Non-Projective Dependency Parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465, Jeju Island, Korea. Association for Computational Linguistics. Cited on pages 78, 96, 105, and 130.

Bohnet, B., Nivre, J., Boguslavsky, I., Farkas, R., Ginter, F., and Hajič, J. (2013). Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics*, 1:415–428. Cited on pages 78 and 96.

Bottou, L. (2004). Stochastic Learning. In Bousquet, O. and von Luxburg, U., editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer Verlag, Berlin, Germany. Cited on page 44.

Boyd, A., Gegg-Harrison, W., and Byron, D. (2005). Identifying Non-Referential it: A Machine Learning Approach Incorporating Linguistically Motivated Patterns. In *Proceed-*

ings of the ACL Workshop on Feature Engineering for Machine Learning in Natural Language Processing, pages 40–47. Cited on page 55.

Brants, S., Dipper, S., Hansen-Shirra, S., Lezius, W., and Smith, G. (2002). The TIGER treebank. In *Proceedings of the 1st Workshop on Treebanks and Linguistic Theories*, pages 24–41, Sozopol, Bulgaria. Cited on page 80.

Cahill, A. and Riester, A. (2012). Automatically Acquiring Fine-Grained Information Status Distinctions in German. In *Proceedings of the 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 232–236, Seoul, South Korea. Association for Computational Linguistics. Cited on page 74.

Cai, J. and Strube, M. (2010). Evaluation Metrics For End-to-End Coreference Resolution Systems. In *Proceedings of the SIGDIAL 2010 Conference*, pages 28–36, Tokyo, Japan. Association for Computational Linguistics. Cited on page 59.

Chang, K.-W., Samdani, R., and Roth, D. (2013). A Constrained Latent Variable Model for Coreference Resolution. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 601–612, Seattle, Washington, USA. Association for Computational Linguistics. Cited on pages 45 and 73.

Chang, K.-W., Samdani, R., Rozovskaya, A., Sammons, M., and Roth, D. (2012). Illinois-Coref: The UI System in the CoNLL-2012 Shared Task. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 113–117, Jeju Island, Korea. Association for Computational Linguistics. Cited on page 72.

Charniak, E. and Elsner, M. (2009). EM Works for Pronoun Anaphora Resolution. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 148–156, Athens, Greece. Association for Computational Linguistics. Cited on page 73.

Charniak, E. and Johnson, M. (2001). Edit Detection and Parsing for Transcribed Speech. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics on Language Technologies*, NAACL '01, pages 1–9, Stroudsburg, Pennsylvania, USA. Association for Computational Linguistics. Cited on page 117.

Cherry, C. and Bergsma, S. (2005). An Expectation Maximization Approach to Pronoun Resolution. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 88–95, Ann Arbor, Michigan, USA. Association for Computational Linguistics. Cited on page 73.

Cho, E., Niehues, J., and Waibel, A. (2012). Segmentation and punctuation prediction in speech language translation using a monolingual translation system. In *International Workshop on Spoken Language Translation (IWSLT) 2012*. Cited on page 129.

Choi, J. D. and Palmer, M. (2010). Robust Constituent-to-Dependency Conversion for English. In *Proceedings of 9th Treebanks and Linguistic Theories Workshop (TLT)*, pages 55–66. Cited on page 54.

Chomsky, N. (1981). *Lectures on Government and Binding*. Foris Publications, Dordrecht. Cited on page 62.

Chu, Y. and Liu, T. (1965). On the shortest aborescence of a directed graph. *Science Sinica*, 14:1396–1400. Cited on pages 50 and 104.

Clark, K. and Manning, C. D. (2016). Deep Reinforcement Learning for Mention-Ranking Coreference Models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2256–2262, Austin, Texas, USA. Association for Computational Linguistics. Cited on page 71.

Collins, M. (2002). Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8. Association for Computational Linguistics. Cited on pages 3, 32, 34, and 43.

Collins, M. and Roark, B. (2004). Incremental Parsing with the Perceptron Algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 111–118, Barcelona, Spain. Cited on pages 3, 9, 35, 75, and 134.

Collins, M. J. (1999). *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania. Cited on page 54.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. The MIT Press, 2nd edition. Cited on pages 30 and 31.

Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, Athens, Georgia, USA. Cited on page 107.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online Passive–Aggressive Algorithms. *Journal of Machine Learning Reseach*, 7:551–585. Cited on pages 32, 34, and 43.

Cucerzan, S. (2007). Large-Scale Named Entity Disambiguation Based on Wikipedia Data. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 708–716, Prague, Czech Republic. Association for Computational Linguistics. Cited on page 75.

Culotta, A., Wick, M., and McCallum, A. (2007). First-Order Probabilistic Models for Coreference Resolution. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 81–88, Rochester, New York, USA. Association for Computational Linguistics. Cited on page 63.

Daumé III, H. (2006). *Practical Structured Learning Techniques for Natural Language Processing*. PhD thesis, University of Southern California. Cited on page 34.

Daumé III, H., Langford, J., and Marcu, D. (2009). Search-based structured prediction. *Machine Learning*, 75(3):297–325. Cited on page 46.

Daumé III, H. and Marcu, D. (2005). Learning as search optimization: approximate large margin methods for structured prediction. In *ICML*, pages 169–176. Cited on pages 4, 10, 15, 38, 75, 134, and 137.

de Lhoneux, M., Stymne, S., and Nivre, J. (2017). Arc-Hybrid Non-Projective Dependency Parsing with a Static-Dynamic Oracle. In *Proceedings of the 15th International Conference on Parsing Technologies*, pages 99–104. Association for Computational Linguistics. Cited on page 140.

de Marneffe, M.-C., MacCartney, B., and Manning, C. D. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, pages 449–454, Genoa, Italy. Cited on pages 54, 93, and 117.

Deemter, K. v. and Kibble, R. (2000). Squibs and Discussions: On Coreferring: Coreference in MUC and Related Annotation Schemes. *Computational Linguistics*, 26(4). Cited on page 74.

Denis, P. and Baldridge, J. (2007). Joint Determination of Anaphoricity and Coreference Resolution using Integer Programming. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 236–243, Rochester, New York, USA. Association for Computational Linguistics. Cited on page 74.

Denis, P. and Baldridge, J. (2009). Global Joint Models for Coreference Resolution and Named Entity Classification. In *Procesamiento del Lenguaje Natural 42*, pages 87–96, Barcelona, Spain. Cited on page 59.

Dozat, T. and Manning, C. D. (2017). Deep biaffine attention for neural dependency parsing. In *Proceedings of the 5th International Conference on Learning Representations*, Touloun, France. Cited on page 140.

Dridan, R. and Oepen, S. (2013). Document Parsing: Towards Realistic Syntactic Analysis. In *Proceedings of the 13th International Conference on Parsing Technologies*, Nara, Japan. Cited on pages 109 and 129.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Reseach*, 12:2121–2159. Cited on page 44.

Durrett, G. and Klein, D. (2013). Easy Victories and Uphill Battles in Coreference Resolution. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1971–1982, Seattle, Washington, USA. Association for Computational Linguistics. Cited on pages 15, 45, 62, and 73.

Durrett, G. and Klein, D. (2014). A Joint Model for Entity Analysis: Coreference, Typing, and Linking. *Transactions of the Association for Computational Linguistics*, 2:477–490. Cited on page 75.

Eckhoff, H., Bech, K., Bouma, G., Eide, K., Haug, D., Haugen, O. E., and Jøhndal, M. (2018). The PROIEL treebank family: a standard for early attestations of Indo-European languages. *Language Resources and Evaluation*, 52(1):29–65. Cited on page 128.

Edelkamp, S. and Schroedl, S. (2010). *Heuristic Search: Theory and Applications*. Morgan Kaufmann Publishers Inc., San Francisco, California, USA. Cited on page 44.

Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71(B):233–240. Cited on pages 50 and 104.

Eisner, J. (1996). Three New Probabilistic Models for Dependency Parsing: An Exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (Coling 1996)*, pages 340–345, Copenhagen, Denmark. Cited on page 104.

Evang, K., Basile, V., Chrupała, G., and Bos, J. (2013). Elephant: Sequence Labeling for Word and Sentence Segmentation. In *Proceedings of the 2013 Conference on Empirical*

Methods in Natural Language Processing, pages 1422–1426, Seattle, Washington, USA. Association for Computational Linguistics. Cited on pages 120 and 129.

Faleńska, A., Björkelund, A., Çetinoğlu, Ö., and Seeker, W. (2015). Stacking or Supertagging for Dependency Parsing – What's the Difference? In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 118–129. Association for Computational Linguistics.

Fernandes, E., dos Santos, C., and Milidiú, R. (2012). Latent Structure Perceptron with Feature Induction for Unrestricted Coreference Resolution. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 41–48, Jeju Island, Korea. Association for Computational Linguistics. Cited on pages 3, 4, 6, 7, 15, 45, 49, 50, 61, 72, 75, and 134.

Freund, Y. and Schapire, R. E. (1999). Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37(3):277–296. Cited on pages 34 and 43.

Ganchev, K. and Dredze, M. (2008). Small Statistical Models by Random Feature Mixing. In *Proceedings of the ACL-08: HLT Workshop on Mobile Language Processing*, pages 19–20, Columbus, Ohio, USA. Association for Computational Linguistics. Cited on page 25.

Gildea, D. and Jurafsky, D. (2002). Automatic Labeling of Semantic Roles. *Computational Linguistics*, 28(3):245–288. Cited on page 62.

Goldberg, Y. and Elhadad, M. (2010). An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750, Los Angeles, California, USA. Association for Computational Linguistics. Cited on page 46.

Goldberg, Y. and Nivre, J. (2012). A Dynamic Oracle for Arc-Eager Dependency Parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India. The COLING 2012 Organizing Committee. Cited on pages 46, 78, 97, and 106.

Goldberg, Y. and Nivre, J. (2013). Training Deterministic Parsers with Non-Deterministic Oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414. Cited on pages 46, 78, 106, and 135.

Goldberg, Y., Sartorio, F., and Satta, G. (2014). A Tabular Method for Dynamic Oracles in Transition-Based Parsing. *Transactions of the Association for Computational Linguistics*, 2:119–130. Cited on pages 78, 97, and 106.

Goldberg, Y., Zhao, K., and Huang, L. (2013). Efficient Implementation of Beam-Search Incremental Parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 628–633, Sofia, Bulgaria. Association for Computational Linguistics. Cited on pages 31 and 95.

Gómez-Rodríguez, C. and Fernández-González, D. (2015). An Efficient Dynamic Oracle for Unrestricted Non-Projective Parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 256–261, Beijing, China. Association for Computational Linguistics. Cited on page 107.

Gómez-Rodríguez, C. and Nivre, J. (2010). A Transition-Based Parser for 2-Planar Dependency Structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1492–1501, Uppsala, Sweden. Association for Computational Linguistics. Cited on page 82.

Gómez-Rodríguez, C., Sartorio, F., and Satta, G. (2014). A Polynomial-Time Dynamic Oracle for Non-Projective Dependency Parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 917–927, Doha, Qatar. Association for Computational Linguistics. Cited on page 106.

Gómez-Rodríguez, C., Shi, T., and Lee, L. (2018). Global Transition-based Non-projective Dependency Parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2664–2675. Association for Computational Linguistics. Cited on page 106.

Gooi, C. H. and Allan, J. (2004). Cross-Document Coreference on a Large Scale Corpus. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*. Cited on page 74.

Grosz, B. J., Joshi, A. K., and Weinstein, S. (1983). PROVIDING A UNIFIED ACCOUNT OF DEFINITE NOUN PHRASES IN DISCOURSE. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 44–50, Cambridge, Massachusetts, USA. Association for Computational Linguistics. Cited on page 63.

Grosz, B. J., Weinstein, S., and Joshi, A. K. (1995). Centering: A Framework for Modeling the Local Coherence of Discourse. *Computational Linguistics*, 21(2). Cited on page 63.

Hajishirzi, H., Zilles, L., Weld, D. S., and Zettlemoyer, L. (2013). Joint Coreference Resolution and Named-Entity Linking with Multi-Pass Sieves. In *Proceedings of the 2013*

Conference on Empirical Methods in Natural Language Processing, pages 289–299, Seattle, Washington, USA. Association for Computational Linguistics. Cited on page 75.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107. Cited on page 44.

Hatori, J., Matsuzaki, T., Miyao, Y., and Tsujii, J. (2011). Incremental Joint POS Tagging and Dependency Parsing in Chinese. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 1216–1224, Chiang Mai, Thailand. Asian Federation of Natural Language Processing. Cited on pages 105 and 130.

Hatori, J., Matsuzaki, T., Miyao, Y., and Tsujii, J. (2012). Incremental Joint Approach to Word Segmentation, POS Tagging, and Dependency Parsing in Chinese. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1045–1053, Jeju Island, Korea. Association for Computational Linguistics. Cited on page 105.

Haug, D., Eckhoff, H., Majer, M., and Welo, E. (2009). Breaking down and putting back together: analysis and synthesis of New Testament Greek. *Journal of Greek Linguistics*, 9(1):56–92. Cited on page 128.

Haug, D. T. T. and Jøhndal, M. (2008). Creating a Parallel Treebank of the Old Indo-European Bible Translations. In *Proceedings of the Second Workshop on Language Technology for Cultural Heritage Data (LaTeCH 2008)*, pages 27–34, Marrakech, Morocco. Cited on page 13.

Havelka, J. (2007). Beyond Projectivity: Multilingual Evaluation of Constraints and Measures on Non-Projective Structures. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 608–615, Prague, Czech Republic. Association for Computational Linguistics. Cited on pages 81 and 82.

Hays, D. G. (1964). Dependency theory: A formalism and some observations. *Language*, 40(4):511–525. Cited on page 82.

Hobbs, J. R. (1978). Resolving pronoun references. *Lingua*, 44(4):311 – 338. Cited on page 73.

Hou, Y., Markert, K., and Strube, M. (2013). Global Inference for Bridging Anaphora Resolution. In *Proceedings of the 2013 Conference of the North American Chapter of the*

Association for Computational Linguistics: Human Language Technologies, pages 907–917, Atlanta, Georgia, USA. Association for Computational Linguistics. Cited on page 74.

Huang, L. (2008). Forest Reranking: Discriminative Parsing with Non-Local Features. In *Proceedings of ACL-08: HLT*, pages 586–594, Columbus, Ohio, USA. Association for Computational Linguistics. Cited on page 43.

Huang, L., Fayong, S., and Guo, Y. (2012). Structured Perceptron with Inexact Search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151, Montréal, Quebec, Canada. Association for Computational Linguistics. Cited on pages 3, 10, 35, 37, 43, 75, 134, 137, and 138.

Huang, L. and Sagae, K. (2010). Dynamic Programming for Linear-Time Incremental Parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086, Uppsala, Sweden. Association for Computational Linguistics. Cited on page 105.

Hudson, R. A. (1984). *Word Grammar*. Basil Blackwell, Oxford, UK. Cited on page 10.

Johansson, R. and Nugues, P. (2006). Investigating Multilingual Dependency Parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 206–210. Association for Computational Linguistics. Cited on pages 44 and 105.

Johansson, R. and Nugues, P. (2007a). Extended Constituent-to-Dependency Conversion for English. In Nivre, J., Kaalep, H.-J., Muischnek, K., and Koit, M., editors, *NODALIDA 2007 Conference Proceedings*, pages 105–112, Tartu, Estonia. Cited on page 54.

Johansson, R. and Nugues, P. (2007b). Incremental Dependency Parsing Using Online Learning. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 1134–1138, Prague, Czech Republic. Association for Computational Linguistics. Cited on pages 44 and 105.

Jurafsky, D. and Martin, J. H. (2009). *Speech and language processing*, volume 2. Pearson London. Cited on page 53.

Kiperwasser, E. and Goldberg, Y. (2016). Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *Transactions of the Association for Computational Linguistics*, 4:313–327. Cited on page 140.

Kiss, T. and Strunk, J. (2006). Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525. Cited on page 129.

Klein, D. and Manning, C. D. (2002). Fast exact inference with a factored model for natural language parsing. In *Advances in neural information processing systems*, pages 3–10. Cited on page 45.

Klein, D. and Manning, C. D. (2003). A* Parsing: Fast Exact Viterbi Parse Selection. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*. Cited on page 45.

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical Phrase-Based Translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*. Cited on page 44.

Kolář, J., Shriberg, E., and Liu, Y. (2006). Using prosody for automatic sentence segmentation of multi-party meetings. In *Text, Speech and Dialogue*, pages 629–636. Springer. Cited on page 110.

Koo, T. and Collins, M. (2010). Efficient Third-Order Dependency Parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden. Association for Computational Linguistics. Cited on page 104.

Koo, T., Rush, A. M., Collins, M., Jaakkola, T., and Sontag, D. (2010). Dual Decomposition for Parsing with Non-Projective Head Automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298, Cambridge, Massachusetts, USA. Association for Computational Linguistics. Cited on page 105.

Kübler, S., McDonald, R., and Nivre, J. (2009). *Dependency Parsing*. Morgan and Claypool. [Pedagogical]. Cited on pages 79 and 104.

Kuhlmann, M. and Nivre, J. (2006). Mildly Non-Projective Dependency Structures. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 507–514, Sydney, Australia. Association for Computational Linguistics. Cited on page 82.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97. Cited on page 60.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the*

Eighteenth International Conference on Machine Learning, pages 282–289, Williamstown, Massachusetts, USA. Cited on page 43.

Lappin, S. and Leass, H. J. (1994). An Algorithm for Pronominal Anaphora Resolution. *Computational Linguistics*, 20(4). Cited on page 73.

Lassalle, E. and Denis, P. (2011). Leveraging Different Meronym Discovery Methods for Bridging Resolution in French. In Hendrickx, I., Lalitha Devi, S., Branco, A., and Mitkov, R., editors, *Anaphora Processing and Applications*, pages 35–46, Berlin, Heidelberg. Springer Berlin Heidelberg. Cited on page 74.

Lassalle, E. and Denis, P. (2015). Joint Anaphoricity Detection and Coreference Resolution with Constrained Latent Structures. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 2274–2280. AAAI Press. Cited on page 74.

Le, M. and Fokkens, A. (2017). Tackling Error Propagation through Reinforcement Learning: A Case of Greedy Dependency Parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 677–687, Valencia, Spain. Association for Computational Linguistics. Cited on page 107.

Lee, K., He, L., Lewis, M., and Zettlemoyer, L. (2017). End-to-end Neural Coreference Resolution. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 188–197, Copenhagen, Denmark. Association for Computational Linguistics. Cited on page 71.

Li, Q., Ji, H., and Huang, L. (2013). Joint Event Extraction via Structured Prediction with Global Features. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 73–82, Sofia, Bulgaria. Association for Computational Linguistics. Cited on page 43.

Lowerre, B. T. (1976). *The HARPY Speech Recognition System*. PhD thesis, Carnegie-Mellon University. Cited on page 44.

Lu, W. and Ng, H. T. (2010). Better Punctuation Prediction with Dynamic Conditional Random Fields. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 177–186, Cambridge, Massachusetts, USA. Association for Computational Linguistics. Cited on page 129.

Luo, X. (2005). On Coreference Resolution Performance Metrics. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 25–32, Vancouver, British Columbia, Canada. Association for Computational Linguistics. Cited on pages 59, 60, and 61.

Luo, X., Ittycheriah, A., Jing, H., Kambhatla, N., and Roukos, S. (2004). A Mention-Synchronous Coreference Resolution Algorithm Based On the Bell Tree. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics*, pages 135–142, Barcelona, Spain. Cited on pages 44, 63, and 72.

Luo, X. and Pradhan, S. (2016). Evaluation Metrics. In Poesio, M., Stuckardt, R., and Versley, Y., editors, *Anaphora Resolution: Algorithms, Resources, and Applications*, pages 141–163. Springer Berlin Heidelberg, Berlin, Heidelberg. Cited on page 59.

Ma, X. and Zhao, H. (2012). Fourth-Order Dependency Parsing. In *Proceedings of COLING 2012: Posters*, pages 785–796, Mumbai, India. The COLING 2012 Organizing Committee. Cited on page 104.

Maes, F., Denoyer, L., and Gallinari, P. (2009). Structured prediction with reinforcement learning. *Machine Learning*, 77(2):271. Cited on page 46.

Magerman, D. M. (1994). *Natural Language Parsing as Statistical Pattern Recognition*. PhD thesis, Stanford University. Cited on page 54.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60. Cited on pages 120 and 129.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330. Cited on pages 93 and 117.

Markert, K., Hou, Y., and Strube, M. (2012). Collective Classification for Fine-grained Information Status. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 795–804, Jeju Island, Korea. Association for Computational Linguistics. Cited on page 74.

Martins, A., Almeida, M., and Smith, N. A. (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), pages 617–622, Sofia, Bulgaria. Association for Computational Linguistics. Cited on page 105.

Martins, A., Smith, N., Xing, E., Aguiar, P., and Figueiredo, M. (2010). Turbo Parsers: Dependency Parsing by Approximate Variational Inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 34–44, Cambridge, Massachusetts, USA. Association for Computational Linguistics. Cited on page 105.

Martschat, S. and Strube, M. (2015). Latent Structures for Coreference Resolution. *Transactions of the Association for Computational Linguistics*, 3:405–418. Cited on page 73.

Matusov, E., Mauser, A., and Ney, H. (2006). Automatic sentence segmentation and punctuation prediction for spoken language translation. In *Proceedings of IWSLT*, pages 158–165. Cited on page 129.

McCallum, A. K. (2002). MALLET: A Machine Learning for Language Toolkit. http://mallet.cs.umass.edu. Cited on page 55.

McCarthy, J. F. and Lehnert, W. G. (1995). Using Decision Trees for Coreference Resolution. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1050–1055. Cited on page 72.

McDonald, R. and Pereira, F. (2006). Online Learning of Approximate Dependency Parsing Algorithms. In *Proceedings of the 11th Conference of the European Chapter of the ACL (EACL 2006)*, pages 81–88, Trento, Italy. Association for Computational Linguistics. Cited on pages 104 and 105.

McDonald, R., Pereira, F., Ribarov, K., and Hajic, J. (2005). Non-Projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada. Association for Computational Linguistics. Cited on page 104.

McDonald, R. and Satta, G. (2007). On the Complexity of Non-Projective Data-Driven Dependency Parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 121–132, Prague, Czech Republic. Association for Computational Linguistics. Cited on page 104.

Mel'čuk, I. (1988). *Dependency Syntax: Theory and Practice*. State University Press of New York, SUNY serie edition. Cited on page 10.

Mitkov, R. (2002). *Anaphora resolution*. Longman, London, UK. Cited on page 72.

Morton, T. S. (2000). Coreference for NLP Applications. In *Proceedings of the 38th Meeting of the Association for Computational Linguistics*, Hong Kong, China. Cited on page 73.

Mueller, T., Schmid, H., and Schütze, H. (2013). Efficient Higher-Order CRFs for Morphological Tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332, Seattle, Washington, USA. Association for Computational Linguistics. Cited on pages 94 and 117.

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38. Cited on page 60.

Neubig, G., Watanabe, T., Mori, S., and Kawahara, T. (2012). Machine Translation without Words through Substring Alignment. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 165–174, Jeju Island, Korea. Association for Computational Linguistics. Cited on page 45.

Ng, V. (2009). Graph-Cut-Based Anaphoricity Determination for Coreference Resolution. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 575–583, Boulder, Colorado, USA. Association for Computational Linguistics. Cited on page 74.

Ng, V. (2010). Supervised Noun Phrase Coreference Research: The First Fifteen Years. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1396–1411, Uppsala, Sweden. Association for Computational Linguistics. Cited on pages 49 and 72.

Ng, V. (2017). Machine Learning for Entity Coreference Resolution: A Retrospective Look at Two Decades of Research. In *Proceedings of AAAI Conference on Artificial Intelligence*. Cited on page 72.

Ng, V. and Cardie, C. (2002a). Identifying Anaphoric and Non-Anaphoric Noun Phrases to Improve Coreference Resolution. In *COLING 2002: The 19th International Conference on Computational Linguistics*. Cited on page 73.

Ng, V. and Cardie, C. (2002b). Improving Machine Learning Approaches to Coreference Resolution. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics. Cited on pages 2, 7, 15, 62, 69, 72, and 73.

Nissim, M. (2006). Learning Information Status of Discourse Entities. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 94–102. Association for Computational Linguistics. Cited on page 74.

Nissim, M., Dingare, S., Carletta, J., and Steedman, M. (2004). An Annotation Scheme for Information Status in Dialogue. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*. European Language Resources Association (ELRA). Cited on page 74.

Nivre, J. (2003). An Efficient Algorithm for Projective Dependency Parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160. Cited on pages 82 and 129.

Nivre, J. (2004). Incrementality in Deterministic Dependency Parsing. In Keller, F., Clark, S., Crocker, M., and Steedman, M., editors, *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain. Association for Computational Linguistics. Cited on pages 78, 82, and 105.

Nivre, J. (2005). Dependency Grammar and Dependency Parsing. Technical Report MSI 05133, Växjö University, School of Mathematics and Systems Engineering. Cited on page 10.

Nivre, J. (2008). Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, 34(4):513–553. Cited on pages 2 and 107.

Nivre, J. (2009). Non-Projective Dependency Parsing in Expected Linear Time. 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*, pages 351–359, Suntec, Singapore. Association for Computational Linguistics. Cited on pages 4, 16, 78, 84, 85, and 135.

Nivre, J., De Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C. D., McDonald, R. T., Petrov, S., Pyysalo, S., Silveira, N., et al. (2016). Universal Dependencies v1: A Multilingual Treebank Collection. In *LREC*. Cited on page 11.

Nivre, J., Hall, J., Nilsson, J., Eryiğit, G., and Marinov, S. (2006). Labeled Pseudo-Projective Dependency Parsing with Support Vector Machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 221–225, New York City, New York, USA. Association for Computational Linguistics. Cited on page 104.

Nivre, J., Kuhlmann, M., and Hall, J. (2009). An Improved Oracle for Dependency Parsing with Online Reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76, Paris, France. Association for Computational Linguistics. Cited on pages 78, 87, 97, 108, and 115.

Novikoff, A. B. J. (1963). On convergence proofs for perceptrons. Technical report, Stanford Research Institute. Cited on page 43.

Pauls, A. and Klein, D. (2009). K-Best A* 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*, pages 958–966, Suntec, Singapore. Association for Computational Linguistics. Cited on page 45.

Pauls, A., Klein, D., and Quirk, C. (2010). Top-Down K-Best A* Parsing. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 200–204, Uppsala, Sweden. Association for Computational Linguistics. Cited on page 45.

Pitler, E. (2014). A Crossing-Sensitive Third-Order Factorization for Dependency Parsing. *Transactions of the Association for Computational Linguistics*, 2:41–54. Cited on page 105.

Pitler, E. and McDonald, R. (2015). A Linear-Time Transition System for Crossing Interval Trees. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 662–671, Denver, Colorado, USA. Association for Computational Linguistics. Cited on page 82.

Poesio, M., Stuckardt, R., and Versley, Y. (2016). *Anaphora Resolution: Algorithms, Resources, and Applications*. Springer Publishing Company, Incorporated, 1st edition. Cited on page 72.

Poesio, M., Uryupina, O., Vieira, R., Alexandrov-Kabadjov, M., and Goulart, R. (2004). Discourse-New Detectors for Definite Description Resolution: A Survey and a Preliminary Proposal. In *Proceedings of the Conference on Reference Resolution and Its Applications*. Cited on page 73.

Poesio, M. and Vieira, R. (1998). A Corpus-Based Investigation of Definite Description Use. *Computational Linguistics*, 24(2):183–216. Cited on page 74.

Pradhan, S., Luo, X., Recasens, M., Hovy, E., Ng, V., and Strube, M. (2014). Scoring Coreference Partitions of Predicted Mentions: A Reference Implementation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short*

Papers), pages 30–35, Baltimore, Maryland, USA. Association for Computational Linguistics. Cited on page 59.

Pradhan, S., Moschitti, A., Xue, N., Ng, H. T., Björkelund, A., Uryupina, O., Zhang, Y., and Zhong, Z. (2013). Towards Robust Linguistic Analysis using OntoNotes. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 143–152. Association for Computational Linguistics.

Pradhan, S., Moschitti, A., Xue, N., Uryupina, O., and Zhang, Y. (2012). CoNLL-2012 Shared Task: Modeling Multilingual Unrestricted Coreference in OntoNotes. In *Joint Conference on EMNLP and CoNLL – Shared Task*, pages 1–40, Jeju Island, Korea. Association for Computational Linguistics. Cited on pages 49, 58, and 59.

Pradhan, S., Ramshaw, L., Marcus, M., Palmer, M., Weischedel, R., and Xue, N. (2011). CoNLL-2011 Shared Task: Modeling Unrestricted Coreference in OntoNotes. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–27, Portland, Oregon, USA. Association for Computational Linguistics. Cited on page 59.

Prince, E. F. (1981). Toward a taxonomy of given-new information. In Cole, P., editor, *Radical Pragmatics*, volume 14, pages 223–255. Academic Press, New York. Cited on pages 63 and 74.

Rahman, A. and Ng, V. (2009). Supervised Models for Coreference Resolution. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 968–977, Suntec, Singapore. Association for Computational Linguistics. Cited on pages 15, 59, 62, 63, and 72.

Rahman, A. and Ng, V. (2011a). Coreference Resolution with World Knowledge. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 814–824, Portland, Oregon, USA. Association for Computational Linguistics. Cited on page 62.

Rahman, A. and Ng, V. (2011b). Learning the Information Status of Noun Phrases in Spoken Dialogues. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1069–1080, Edinburgh, Scotland, UK. Association for Computational Linguistics. Cited on page 74.

Rahman, A. and Ng, V. (2011c). Narrowing the Modeling Gap: A Cluster-ranking Approach to Coreference Resolution. *Journal of Artificial Intelligence Research*, 40(1):469–521. Cited on pages 72 and 73.

Rao, D., McNamee, P., and Dredze, M. (2010). Streaming Cross Document Entity Coreference Resolution. In *Coling 2010: Posters*, pages 1050–1058. Coling 2010 Organizing Committee. Cited on page 74.

Ratinov, L. and Roth, D. (2009). Design Challenges and Misconceptions in Named Entity Recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, pages 147–155, Boulder, Colorado, USA. Association for Computational Linguistics. Cited on page 43.

Read, J., Dridan, R., Oepen, S., and Solberg, L. J. (2012). Sentence Boundary Detection: A Long Solved Problem? In *Proceedings of COLING 2012: Posters*, pages 985–994, Mumbai, India. The COLING 2012 Organizing Committee. Cited on pages 110 and 129.

Reddy, R. (1977). SPEECH UNDERSTANDING SYSTEMS: Summary of Results of the Five-Year Research Effort at Carnegie-Mellon. Technical report, Carnegie-Mellon University, Department of Computer Science. Cited on page 44.

Reynar, J. C. and Ratnaparkhi, A. (1997). A Maximum Entropy Approach to Identifying Sentence Boundaries. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 16–19, Washington, Distric of Columbia, USA. Association for Computational Linguistics. Cited on page 129.

Richardson, K. and Kuhn, J. (2016). Learning to Make Inferences in a Semantic Parsing Task. *Transactions of the Association for Computational Linguistics*, 4:155–168. Cited on page 45.

Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6):386–408. Cited on pages 3, 32, and 43.

Rösiger, I., Riester, A., and Kuhn, J. (2018). Bridging resolution: Task definition, corpus resources and rule-based experiments. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3516–3528. Association for Computational Linguistics. Cited on page 74.

Rösiger, I. and Teufel, S. (2014). Resolving Coreferent and Associative Noun Phrases in Scientific Text. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 45–55, Gothenburg, Sweden. Association for Computational Linguistics. Cited on page 74.

Ross, S., Gordon, G. J., and Bagnell, J. A. (2011). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *AISTATS*, pages 627–635. Cited on page 46.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence, A Modern Approach*. Prentice Hall, Upper Saddle River, 2nd edition. Cited on pages 25 and 44.

Sartorio, F., Satta, G., and Nivre, J. (2013). A Transition-Based Dependency Parser Using a Dynamic Parsing Strategy. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 135–144, Sofia, Bulgaria. Association for Computational Linguistics. Cited on page 106.

Schmid, H. (2000). Unsupervised Learning of Period Disambiguation for Tokenisation. Technical report, IMS, University of Stuttgart. Cited on page 129.

Seddah, D., Kübler, S., and Tsarfaty, R. (2014). Introducing the SPMRL 2014 Shared Task on Parsing Morphologically-rich Languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 103–109, Dublin, Ireland. Dublin City University. Cited on page 93.

Seeker, W. and Çetinoğlu, Ö. (2015). A Graph-based Lattice Dependency Parser for Joint Morphological Segmentation and Syntactic Analysis. *Transactions of the Association for Computational Linguistics*, 3:359–373. Cited on page 131.

Seeker, W. and Kuhn, J. (2012). Making Ellipses Explicit in Dependency Conversion for a German Treebank. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*. European Language Resources Association (ELRA). Cited on page 80.

Seeker, W. and Kuhn, J. (2013). Morphological and Syntactic Case in Statistical Dependency Parsing. *Computational Linguistics*, 39:23–55. Cited on page 130.

Sgall, P., Hajičová, E., and Panevová, J. (1986). *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. Dordrecht:Reidel Publishing Company and Prague:Academia. Cited on page 10.

Shen, L. and Joshi, A. (2008). LTAG Dependency Parsing with Bidirectional Incremental Construction. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 495–504, Honolulu, Hawaii, USA. Association for Computational Linguistics. Cited on page 46.

Shen, L., Satta, G., and Joshi, A. (2007). Guided Learning for Bidirectional Sequence Classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 760–767, Prague, Czech Republic. Association for Computational Linguistics. Cited on page 46.

Shi, T., Huang, L., and Lee, L. (2017). Fast(er) Exact Decoding and Global Training for Transition-Based Dependency Parsing via a Minimal Feature Set. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 12–23, Copenhagen, Denmark. Association for Computational Linguistics. Cited on page 106.

Smith, N. A. (2011). *Linguistic Structure Prediction*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool. Cited on page 1.

Soon, W. M., Ng, H. T., and Lim, D. C. Y. (2001). A Machine Learning Approach to Coreference Resolution of Noun Phrases. *Computational Linguistics*, 27(4):521–544. Cited on pages 2, 5, 7, 15, 62, 63, 69, and 72.

Stoyanov, V. and Eisner, J. (2012). Easy-first Coreference Resolution. In *Proceedings of COLING 2012*, pages 2519–2534, Mumbai, India. Cited on page 46.

Stoyanov, V., Gilbert, N., Cardie, C., and Riloff, E. (2009). Conundrums in Noun Phrase Coreference Resolution: Making Sense of the State-of-the-Art. 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*, pages 656–664, Suntec, Singapore. Association for Computational Linguistics. Cited on pages 59 and 72.

Straka, M., Hajic, J., Straková, J., and Hajic jr, J. (2015). Parsing universal dependency treebanks using neural networks and search-based oracle. In *International Workshop on Treebanks and Linguistic Theories (TLT14)*, pages 208–220. Cited on page 107.

Sun, X., Matsuzaki, T., Okanohara, D., and Tsujii, J. (2009). Latent Variable Perceptron Algorithm for Structured Classification. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 1236–1242. Cited on page 45.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts, USA. Cited on page 46.

Tarjan, R. E. (1977). Finding optimum branchings. *Networks*, 7(1):25–35. Cited on page 50.

Taskar, B., Guestrin, C., and Koller, D. (2004). Max-margin Markov networks. In *Advances in neural information processing systems*, pages 25–32, Vancouver, Canada. Cited on page 43.

Tillmann, C. and Ney, H. (2003). Word Reordering and a Dynamic Programming Beam Search Algorithm for Statistical Machine Translation. *Computational Linguistics*, 29(1). Cited on page 44.

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of machine learning research*, 6(Sep):1453–1484. Cited on page 43.

Tsuruoka, Y., Tsujii, J., and Ananiadou, S. (2009). Stochastic Gradient Descent Training for L1-regularized Log-linear Models with Cumulative Penalty. 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*, pages 477–485, Suntec, Singapore. Association for Computational Linguistics. Cited on page 44.

Uryupina, O. (2003). High-precision Identification of Discourse New and Unique Noun Phrases. In *Proceedings of the ACL Student Research Workshop*, pages 80–86. Cited on page 73.

Versley, Y. and Björkelund, A. (2016). Off-the-Shelf Tools. In Poesio, M., Stuckardt, R., and Versley, Y., editors, *Anaphora Resolution: Algorithms, Resources, and Applications*, pages 237–266. Springer Berlin Heidelberg, Berlin, Heidelberg. Cited on page 72.

Versley, Y., Ponzetto, S., Poesio, M., Eidelman, V., Jern, A., Smith, J., Yang, X., and Moschitti, A. (2008). BART: A modular toolkit for coreference resolution. In *LREC 2008*. Cited on page 72.

Vilain, M., Burger, J., Aberdeen, J., Connolly, D., and Hirschman, L. (1995). A model theoretic coreference scoring scheme. In *Proceedings MUC-6*, pages 45–52, Columbia, Maryland. Cited on page 59.

Watanabe, T. and Sumita, E. (2015). Transition-based Neural Constituent Parsing. 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), pages 1169–1179, Beijing, China. Association for Computational Linguistics. Cited on page 141.

Weinberger, K., Dasgupta, A., Langford, J., Smola, A., and Attenberg, J. (2009). Feature Hashing for Large Scale Multitask Learning. In Bottou, L. and Littman, M., editors, *Proceedings of the 26th International Conference on Machine Learning*, pages 1113–1120, Montreal. Omnipress. Cited on page 25.

Weischedel, R., Hovy, E., Marcus, M., Palmer, M., Belvin, R., Pradhan, S., Ramshaw, L., and Xue, N. (2011). OntoNotes: A Large Training Corpus for Enhanced Processing. In Olive, J., Christianson, C., and McCary, J., editors, *Handbook of Natural Language Processing and Machine Translation: DARPA Global Autonomous Language Exploitation*. Springer. Cited on page 58.

Wiseman, S. and Rush, A. M. (2016). Sequence-to-Sequence Learning as Beam-Search Optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306, Austin, Texas, USA. Association for Computational Linguistics. Cited on page 141.

Wiseman, S., Rush, A. M., and Shieber, S. M. (2016). Learning Global Features for Coreference Resolution. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 994–1004, San Diego, California, USA. Association for Computational Linguistics. Cited on page 71.

Yamada, H. and Matsumoto, Y. (2003). Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*, pages 195–206. Cited on pages 77 and 105.

Yang, X., Su, J., Lang, J., Tan, C. L., Liu, T., and Li, S. (2008). An Entity-Mention Model for Coreference Resolution with Inductive Logic Programming. In *Proceedings of ACL-08: HLT*, pages 843–851, Columbus, Ohio, USA. Association for Computational Linguistics. Cited on page 72.

Yang, X., Su, J., and Tan, C. L. (2006). Kernel-Based Pronoun Resolution with Structured Syntactic Knowledge. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 41–48, Sydney, Australia. Association for Computational Linguistics. Cited on page 62.

Yu, C.-N. and Joachims, T. (2009). Learning Structural SVMs with Latent Variables. In *International Conference on Machine Learning (ICML)*. Cited on pages 45 and 72.

Yu, H., Huang, L., Mi, H., and Zhao, K. (2013). Max-Violation Perceptron and Forced Decoding for Scalable MT Training. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1112–1123, Seattle, Washington, USA. Association for Computational Linguistics. Cited on page 43.

Yu, X., Vu, N. T., and Kuhn, J. (2018). Approximate Dynamic Oracle for Dependency Parsing with Reinforcement Learning. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 183–191. Association for Computational Linguistics. Cited on page 107.

Zeman, D., Popel, M., Straka, M., Hajic, J., Nivre, J., Ginter, F., Luotolahti, J., Pyysalo, S., Petrov, S., Potthast, M., Tyers, F., Badmaeva, E., Gokirmak, M., Nedoluzhko, A., Cinkova, S., Hajic jr., J., Hlavacova, J., Kettnerová, V., Uresova, Z., Kanerva, J., Ojala, S., Missilä, A., Manning, C. D., Schuster, S., Reddy, S., Taji, D., Habash, N., Leung, H., de Marneffe, M.-C., Sanguinetti, M., Simi, M., Kanayama, H., dePaiva, V., Droganova, K., Martínez Alonso, H., Çöltekin, Ç., Sulubacak, U., Uszkoreit, H., Macketanz, V., Burchardt, A., Harris, K., Marheinecke, K., Rehm, G., Kayadelen, T., Attia, M., Elkahky, A., Yu, Z., Pitler, E., Lertpradit, S., Mandl, M., Kirchner, J., Alcalde, H. F., Strnadová, J., Banerjee, E., Manurung, R., Stella, A., Shimada, A., Kwak, S., Mendonca, G., Lando, T., Nitisaroj, R., and Li, J. (2017). CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19. Association for Computational Linguistics. Cited on page 128.

Zhang, D., Wu, S., Yang, N., and Li, M. (2013a). Punctuation Prediction with Transition-based Parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 752–760, Sofia, Bulgaria. Association for Computational Linguistics. Cited on pages 120 and 129.

Zhang, H. and Gildea, D. (2006). Efficient Search for Inversion Transduction Grammar. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 224–231, Sydney, Australia. Association for Computational Linguistics. Cited on page 45.

Zhang, H. and Gildea, D. (2008). Efficient Multi-Pass Decoding for Synchronous Context

Free Grammars. In *Proceedings of ACL-08: HLT*, pages 209–217, Columbus, Ohio, USA. Association for Computational Linguistics. Cited on page 45.

Zhang, H., Huang, L., Zhao, K., and McDonald, R. (2013b). Online Learning for Inexact Hypergraph Search. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 908–913, Seattle, Washington, USA. Association for Computational Linguistics. Cited on page 104.

Zhang, H. and McDonald, R. (2012). Generalized Higher-Order Dependency Parsing with Cube Pruning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 320–331, Jeju Island, Korea. Association for Computational Linguistics. Cited on page 104.

Zhang, K., Su, J., and Zhou, C. (2014a). Regularized Structured Perceptron: A Case Study on Chinese Word Segmentation, POS Tagging and Parsing. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 164–173, Gothenburg, Sweden. Association for Computational Linguistics. Cited on page 44.

Zhang, M., Zhang, Y., Che, W., and Liu, T. (2013c). Chinese Parsing Exploiting Characters. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 125–134, Sofia, Bulgaria. Association for Computational Linguistics. Cited on page 130.

Zhang, M., Zhang, Y., Che, W., and Liu, T. (2014b). Character-Level Chinese Dependency Parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1326–1336, Baltimore, Maryland, USA. Association for Computational Linguistics. Cited on page 130.

Zhang, M., Zhang, Y., and Fu, G. (2016). Transition-Based Neural Word Segmentation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 421–431, Berlin, Germany. Association for Computational Linguistics. Cited on page 141.

Zhang, Y. and Clark, S. (2007). Chinese Segmentation with a Word-Based Perceptron Algorithm. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 840–847, Prague, Czech Republic. Association for Computational Linguistics. Cited on pages 43 and 138.

Zhang, Y. and Clark, S. (2008). A Tale of Two Parsers: Investigating and Combining Graph-based and Transition-based Dependency Parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Honolulu, Hawaii, USA. Association for Computational Linguistics. Cited on pages 43 and 105.

Zhang, Y. and Clark, S. (2011). Syntactic Processing Using the Generalized Perceptron and Beam Search. *Computational Linguistics*, 37(1):105–151. Cited on page 54.

Zhang, Y., Lei, T., Barzilay, R., Jaakkola, T., and Globerson, A. (2014c). 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)*, pages 197–207, Baltimore, Maryland, USA. Association for Computational Linguistics. Cited on page 105.

Zhang, Y. and Nivre, J. (2011). Transition-based Dependency Parsing with Rich Non-local Features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA. Association for Computational Linguistics. Cited on pages 96 and 105.

Zhao, K. and Huang, L. (2015). Type-Driven Incremental Semantic Parsing with Polymorphism. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1416–1421, Denver, Colorado, USA. Association for Computational Linguistics. Cited on page 45.

Zhou, H., Zhang, Y., Huang, S., and Chen, J. (2015). A Neural Probabilistic Structured-Prediction Model for Transition-Based Dependency Parsing. 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)*, pages 1213–1222, Beijing, China. Association for Computational Linguistics. Cited on page 140.

Typeset January 22, 2019